

Қазақстан Республикасының Оқу-ағарту министрлігі
Министерство просвещения Республики Казахстан
«Тұран» университетінің колледжі
Колледж университета «Туран»

Мамандығы: 1305000 Ақпараттық жүйелер
Специальность: 1305000 Информационные системы

КУРСТЫҚ ЖҰМЫС
КУРСОВОЙ ПРОЕКТ

Пән бойынша/ Дисциплина: Основы алгоритмизации и
программирования

Тақырыбы / Тема: Реализация приложения “Ханойские башни”

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
РК ТУРАН 1305000 КР ПЗ

Орындаған топ оқушысы:
Выполнил обучающийся
группы: IS-202
Тарасов Владимир
Тексерген / Проверил(а):
Николаенкова Т.Е.

Алматы 2024



«ТУРАН» КОЛЛЕДЖИ КОЛЛЕДЖ «ТУРАН»

Курстык жұмыс

ТАПСЫРМАСЫ

ЗАДАНИЕ

на курсовую работу

Пәні /Дисциплина

Основы алгоритмизации и программирования

Тақырыбы /Тема: Реализация приложения "Ханойские башни"

Оқушының аты-жөні /Ф.И.О. обучающегося Тарасов Владимир Алексеевич

Мамандығы/Специальность Информационные системы

Курс /Группа IS-202

Оқушылардың аяқталған жұмысты өткізу уақыты

Срок сдачи обучающимся законченной работы 26.03.2024

1. Жұмысқа арналған бастапқы деректер:

Исходные данные к работе: Методические указания к курсовому проекту

Зерттелген тақырып бойынша ғылыми әдебиеттер тізімі

Список научной литературы по исследуемой теме

1. Троелсен, А. (2012). "Язык программирования C# 5.0 и платформа .NET 4.5." Санкт-Петербург: Питер.
2. Макконнелл, С. (2004). "Совершенный код. Мастер-класс." Санкт-Петербург: Питер.
3. Гриффитс, Э. (2007). "Программирование в C# 2005 для профессионалов." Москва: Издательский дом "Вильямс".

Өңдеуге жататын сұрақтар тізімі

Перечень вопросов, подлежащих разработке

Использование рекурсивного алгоритма для автоматического решения головоломок

Разработка приложения в Windows Forms на языке программирования C#

Разработка ПО

2. Тапсырма берілген күн

Дата выдачи задания 25.12.2023

Жетекші

Руководитель

Николаенкова Т.Е.

Тапсырма орындауға қабылданды



Задание принял к исполнению Тарасов В.А.

(Оқушының аты-жөні)

(ФИО обучающегося)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	4
ТЕОРИТИЧЕСКАЯ ЧАСТЬ.....	6
Используемые программы	6
Описание языка программирования.....	7
Способы структурирования программ.....	9
Средства обмена данными	10
Встроенные элементы.....	11
ПРАКТИЧЕСКАЯ ЧАСТЬ.....	13
Техническое задание.....	13
Структура информационного обеспечения.....	15
Структура программного обеспечения.....	16
Алгоритмы по созданию функций	20
Отладка программы и тестирование модулей.....	28
ЗАКЛЮЧЕНИЕ	29
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	31

					РК ТУРАН 1305000 КР ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Пояснительная записка курсового проекта	Лит.	Лист	Листов
Разраб.		Тарасов В.А.		26.03				
Провер.		Николаенкова Т.Е.		26.03			3	29
Реценз.						ТУРАН IS-202		
Н. Контр.								

ВВЕДЕНИЕ

В современном мире информационных технологий, где каждый день ставится перед научным и инженерным сообществом все новые и вызывающие вопросы задачи, изучение алгоритмов и структур данных остается одним из ключевых направлений развития программной инженерии.

Одним из классических и захватывающих примеров являются Ханойские башни. Цель настоящей курсовой работы заключается в исследовании и разработке программного комплекса, решающего задачу Ханойских башен. Эта задача, предложенная французским математиком Эдуардом Люка в 1883 году, продолжает привлекать внимание ученых и программистов своей необычной комбинаторной структурой.

В рамках данной работы будет представлен программный комплекс, способный эффективно решать Ханойские башни для различных параметров задачи. Актуальность данной темы обусловлена не только историческим контекстом, связанным с появлением задачи, но и ее применимостью в современных областях, таких как оптимизация процессов, анализ алгоритмов и обучение программированию.

В свете постоянного развития информационных технологий и повсеместного использования вычислительных методов, изучение Ханойских башен приобретает новый вектор значимости.

Курсовая работа посвящена разработке программного комплекса для решения задачи Ханойских башен, а также исследованию его применимости и эффективности в современном программном обеспечении.

Разработанный программный комплекс будет включать в себя реализацию алгоритмов для решения задачи Ханойских башен, а также графический интерфейс пользователя (GUI) для удобства взаимодействия с программой. Это позволит эффективно использовать программу не только для решения конкретной задачи, но и для изучения алгоритмов, их сравнения и оптимизации.

					ПК ТУРАН 1305000 КР ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Одним из ключевых моментов будет анализ эффективности различных алгоритмов решения задачи Ханойских башен в зависимости от размера и параметров задачи. Это позволит выявить оптимальные подходы к решению задачи в различных сценариях использования.

Кроме того, в рамках курсовой работы будет произведено исследование применимости решения задачи Ханойских башен в современных областях, таких как оптимизация процессов и обучение программированию. Это позволит выявить потенциальные области применения и значимость данной задачи в контексте современных информационных технологий.

Курсовая работа будет иметь целью не только разработку программного комплекса для решения задачи Ханойских башен, но и исследование её значимости и применимости в современном программном обеспечении и научных исследованиях. Это позволит расширить понимание и использование данной классической задачи в контексте современных информационных технологий и научных исследований.

Разработка программного комплекса для решения задачи Ханойских башен будет осуществляться в среде разработки, обеспечивающей удобную работу как с алгоритмами, так и с графическим интерфейсом пользователя (GUI).

Состояние поставленной задачи на момент получения задания – в процессе разработки.

Задачами данной курсовой работы будут:

- Разработка алгоритма для автоматического решения головоломки.
- Разработка проверки завершения игры пользователем.
- Корректное визуальное отображение игрового поля пользователю.
- Разработка пояснения с правилами для пользователя.
- Разработка средств контроля автоматического хода решения

головоломки пользователем.

ГЛАВА I. ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1. Используемые программы

Для разработки программного комплекса, решающего задачу Ханойских башен, были выбраны технологии, включающие в себя Windows Forms и язык программирования C#.

Windows Forms предоставляет набор инструментов для создания графического пользовательского интерфейса (GUI) в операционной системе Windows, что является актуальным для разработки приложений с интерактивным взаимодействием пользователя.

Windows Forms (WinForms) представляет собой библиотеку классов в .NET Framework для создания графического пользовательского интерфейса (GUI) в приложениях под операционную систему Windows. В контексте C# Windows Forms обеспечивает удобный способ создания интерфейса приложений с использованием различных элементов управления.

Windows Forms предоставляет широкий спектр элементов управления, таких как кнопки, текстовые поля, списки, таблицы, окна сообщений и многие другие. Эти элементы обеспечивают взаимодействие пользователя с приложением, а разработчик может легко настраивать их свойства и методы.

Событийная модель в Windows Forms позволяет отслеживать и обрабатывать действия пользователя и изменения в приложении. Реагируя на события, разработчик может определить логику выполнения определенных действий в ответ на взаимодействие пользователя.

Windows Forms предоставляет гибкие средства для создания пользовательского интерфейса с использованием различных макетов и контейнеров. Это позволяет эффективно организовывать элементы управления на форме приложения.

Обработка ввода и вывода Windows Forms упрощает обработку ввода пользователя, включая обработку клавиатурных и мышечных событий. Также

					ПК ТУРАН 1305000 КР ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

предоставляются средства для взаимодействия с внешними данными, файлами и другими источниками.

Windows Forms поддерживает рисование на форме с использованием различных графических инструментов. Это позволяет создавать пользовательские элементы интерфейса и реализовывать графические эффекты.

Windows Forms обеспечивает многозадачность и возможность работы с несколькими потоками, что важно для создания отзывчивых и эффективных приложений, особенно при выполнении длительных операций.

Синтаксис C# удобно интегрируется с WinForms, что обеспечивает естественный и читаемый код при создании логики обработки событий, управления элементами интерфейса и взаимодействия с другими компонентами приложения.

Visual Studio - основная среда разработки для C#, предоставляет удобные средства для отладки Windows Forms-приложений. Развертывание таких приложений также упрощено благодаря возможностям платформы .NET.

Windows Forms широко применяются для разработки настольных приложений под Windows, таких как управление базами данных, редакторы, игры, инструменты анализа данных и другие. Windows Forms предоставляет разработчикам C# удобное и мощное средство для создания графического пользовательского интерфейса в приложениях под операционную систему Windows. Сочетание C# и Windows Forms обеспечивает эффективное и удобное создание интерактивных приложений с привлекательным пользовательским интерфейсом.

1.2. Описание языка программирования

C# является полностью объектно-ориентированным языком, что означает, что все в нем является объектом. Объекты в C# обладают свойствами и методами, а программа, в целом, организуется вокруг взаимодействия объектов.

					ПК ТУРАН 1305000 КР ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

C# предназначен для создания приложений, которые будут выполняться на платформе .NET Framework. .NET Framework предоставляет среду выполнения, библиотеки классов и инфраструктуру для разработки и запуска разнообразных приложений, обеспечивая их совместимость. C# предоставляет разнообразные встроенные типы данных, такие как целочисленные, вещественные, символьные и строковые типы, а также возможность создания пользовательских типов.

C# был разработан с упором на удобочитаемость и простоту использования, что способствует быстрому освоению языка новичками. C# обеспечивает высокую производительность благодаря эффективной работе с памятью, многопоточности и другим оптимизациям. Язык предоставляет широкий спектр возможностей, включая поддержку различных парадигм программирования, включая процедурное, объектно-ориентированное и функциональное программирование.

C# поддерживает массивы для организации и хранения данных одного типа. Массивы предоставляют эффективные средства для обработки коллекций данных.

В C# имеются удобные средства программирования разветвляющихся и циклических процессов, такие как условные операторы и циклы.

Язык поддерживает создание методов (процедур) и функций, что обеспечивает модульность и повторное использование кода.

C# предоставляет стандартные структуры данных, такие как списки и словари, которые упрощают организацию и обработку данных в программе.

C# предоставляет разнообразные встроенные типы данных:

- Целочисленные типы: int, long, short, byte и другие.
- Вещественные типы: float, double.
- Символьный тип: char.
- Строковый тип: string.

Кроме того, язык предоставляет возможность создания пользовательских типов данных с использованием классов и структур.

					ПК ТУРАН 1305000 КР ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

В С# используются условные операторы if, else if, else для выполнения кода в зависимости от определенных условий.

Язык поддерживает циклы, такие как for, while и do-while, для повторения выполнения определенного блока кода.

1.3. Способы структурирования программ

За каждую форму отвечает отдельный класс, который наследуются от родительского Form, регистрирующего базовое расположение элементов на форме. Каждый элемент, располагаемый на форме, в свою очередь, имеет свой класс, имеющий необходимые методы для взаимодействия элемента интерфейса с пользователем или для его отображения, а также изъятия данных из изменяемых элементов.

Процедуры и функции для работы с формами и элементами:

— create_form - функция для создания новой формы. Принимает параметры, такие как размеры и заголовок формы. Возвращает экземпляр соответствующего класса формы.

— add_element_to_form - процедура для добавления элемента на форму. Принимает в качестве аргумента экземпляр элемента и позицию на форме.

— remove_element_from_form - процедура для удаления элемента с формы. Принимает элемент в качестве аргумента и удаляет его из формы.

— get_form_data - функция для извлечения данных из всех изменяемых элементов на форме. Возвращает словарь с данными, введенными пользователем.

Программные модули:

— form.py - модуль, содержащий базовый класс Form, от которого наследуются все формы. В этом модуле также могут содержаться вспомогательные функции для работы с формами.

— `elements.py` - модуль, содержащий классы для различных элементов интерфейса, таких как кнопки, поля ввода, выпадающие списки и т.д. Каждый класс обеспечивает методы для взаимодействия с соответствующим элементом.

— `utils.py` - модуль, содержащий вспомогательные функции и классы, которые могут быть использованы в разных частях программы. Например, это может включать функции для валидации данных, форматирования текста.

Несколько примеров классов для элементов:

— `Form` (родительский класс) - обеспечивает базовое расположение элементов на форме и может содержать методы для управления формой в целом.

— `Button` (класс элемента) - представляет кнопку на форме. Может иметь методы для изменения текста кнопки, установки обработчиков событий.

— `TextField` (класс элемента) - представляет поле ввода текста на форме. Может иметь методы для установки/изменения текста в поле, получения введенного пользователем текста.

— `ComboBox` (класс элемента) - представляет выпадающий список на форме. Может иметь методы для добавления/удаления элементов списка, получения выбранного элемента.

1.4. Средства обмена данными

В Windows Forms при разработке приложений на C# существует несколько средств обмена данными, которые могут быть использованы для ввода, вывода и внутреннего обмена данными в приложении.

Элементы управления (Controls) - Windows Forms предоставляет различные элементы управления, такие как текстовые поля, метки, кнопки, списки, таблицы и т.д., которые могут использоваться для ввода и вывода данных пользователем.

События (Events) - события позволяют реагировать на действия пользователя, такие как нажатие кнопки, изменение значения в текстовом поле и

т.д. Можно привязать обработчики событий к элементам управления для обработки пользовательского ввода.

Связывание данных (Data Binding) - это механизм, который позволяет автоматически синхронизировать данные между источником данных и элементами управления. Можно привязать список объектов к элементу управления DataGridView, чтобы автоматически отображать данные в таблице.

ADO.NET - это технология, которая предоставляет доступ к базам данных из приложений .NET. Возможно использование ADO.NET для чтения и записи данных из различных источников данных, таких как SQL Server, MS Access, XML и т.д.

Сериализация (Serialization) - сериализация позволяет преобразовать объекты в поток байтов, которые могут быть сохранены на диск или переданы по сети, а затем восстановлены обратно в объекты. Это может использоваться для сохранения и восстановления состояния приложения или передачи данных между приложениями.

XML или JSON для представления структурированных данных в текстовом формате. Windows Forms предоставляет средства для работы с XML и JSON, такие как классы XmlDocument и JsonSerializer.

Локальное хранилище (Local Storage) - использование локального хранилища, такое как файлы или базы данных SQLite, для хранения данных на компьютере пользователя. Это может быть полезно для сохранения настроек приложения или кэширования данных.

1.5. Встроенные элементы

С# поддерживает ключевые концепции объектно-ориентированного программирования:

Классы и объекты: В С# все данные и функции обычно объединяются в классы. Объекты создаются на основе этих классов. Класс определяет структуру

объекта, включая его поля (переменные) и методы (функции). Экземпляры классов создаются с использованием ключевого слова new.

Функции в C# определяются внутри классов и используются для выполнения определенных операций. Они могут принимать параметры и возвращать значения.

Поля представляют данные, хранящиеся в объекте, в то время как свойства обеспечивают доступ к этим данным с помощью геттеров и сеттеров. Свойства обычно используются для обеспечения контроля над доступом к данным и проверки их корректности.

Полиморфизм позволяет объектам различных классов реагировать на одинаковые сообщения. Это может быть достигнуто через перегрузку методов (статический полиморфизм) или через наследование и виртуальные методы (динамический полиморфизм).

Наследование позволяет создавать новый класс на основе существующего класса. Подкласс (или производный класс) наследует свойства и методы базового класса и может добавлять новые функции или изменять существующие.

Интерфейсы определяют набор методов и свойств, которые класс должен реализовать. Класс может реализовать один или несколько интерфейсов, обеспечивая гибкость в организации кода и обеспечивая совместимость между различными типами объектов.

Инкапсуляция означает сокрытие деталей реализации и предоставление интерфейса для взаимодействия с объектом. Это позволяет защитить данные от непосредственного доступа и обеспечить их безопасность и целостность.

Эти основные концепции объектно-ориентированного программирования (ООП) обеспечивают гибкость, модульность и повторное использование кода в C#, делая его мощным языком для разработки приложений.

					РК ТУРАН 1305000 КР ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

ГЛАВА II. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Техническое задание.

1. Введение.

Настоящее техническое задание распространяется на разработку десктоп-приложения на основе игры “Ханойские башни”, предназначенного для создания развития алгоритмических навыков пользователя.

2. Основание для разработки.

Данное приложение разрабатывается в рамках курсового проекта.

3. Назначение.

Приложение предназначено для интеграции в сферу развлечений и образования, то есть должно приносить пользователю положительные эмоции и обучать. Со стороны разработчика, целью является вовлечённость пользователя в игровой процесс.

4. Требования к программному продукту.

А. Требования к функциональным характеристикам.

— Реализация игровой механики "Ханойские башни", включая механику переноса дисков на стержни и автоматического решения головоломки.

— Визуализация игровой формы и всех игровых компонентов.

— Предоставление игрокам информации о текущем состоянии игры - количестве сделанных ходов.

В. Исходные данные.

— Правила и механика игры "Ханойские башни".

— Графические ресурсы для создания визуального оформления игры.

— Microsoft Windows Forms для разработки приложения. Язык программирования C# для разработки игровой логики.

С. Выходные данные.

— Готовое игровое приложение, готовое к запуску на персональных компьютерах и ноутбуках.

					ПК ТУРАН 1305000 КР ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

— Правила игры для пользователя.

5. Требования к надёжности.

— Приложение должно быть стабильным и не выходить из строя при обычном использовании.

— Приложение не должно приводить к сбоям операционной системы или другим критическим ошибкам на платформах.

— Приложение должно быть защищено от стороннего вмешательства с целью взлома для получения внутриигровой выгоды.

6. Требования к составу и параметрам технических средств.

Система должна работать на IBM-совместимых персональных компьютерах.

Минимальная конфигурация:

— Тип процессора - Intel.

— Объем ОЗУ - 256 Мб и выше.

— Видеопамять – 128 Мб и выше.

— Память – 512 Мб.

Требования к информационной и программной совместимости.

Игровое приложение должно быть совместимо с Windows. Игровое приложение должно поддерживать обмен информацией и игровыми данными между двумя игроками через локальную сеть или глобальную сеть.

Игровое приложение должно быть разработано с использованием языка программирования C# в среде Microsoft Windows Forms.

Приложение должно быть совместимо с распространенными видеокартами, обеспечивая корректное воспроизведение видео.

Требования к программной документации.

— Разработать подробное руководство пользователя, объясняющее правила и игры “Ханойские башни”. Руководство должно включать описание инструкции по началу игры и условиях конца игры.

					ПК ТУРАН 1305000 КР ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

- Создать техническую документацию, описывающую архитектуру приложения и основные компоненты.
- Поддерживать актуальность документации, внося изменения при разработке новых версий приложения.

2.2. Структура информационного обеспечения

Основные поля, используемые в непосредственном решении алгоритма Ханойских башен. (Рисунок 1)

```
private Point initialPosition;
private int startColumn = 0;

private int NumDiscs = 1;
private const int RodWidth = 10;
private const int RodHeight = 300;
private const int RodSpacing = 265;
private const int DiscWidthMin = 50;
private const int DiscWidthStep = 20;
private int stepsCount = 1;

private Panel[] rods;
private PictureBox[] discs;
private int[,] rodStates;

private bool isDragging = false;
private PictureBox selectedDisk = null;
private Point offset;
private bool solvingStopped = false;
```

Рисунок 1. Основные поля.

`private Point initialPosition;` Объект класса `Point`, который определяет стартовую точку для отсчета расположения стержней.

`private int startColumn = 0;` Целочисленная переменная, определяющая стартовый стержень, с которого будут перекладываться диски.

`private int NumDiscs = 1;` Целочисленная переменная, определяющая количество дисков, которое будет создано на стартовом стержне.

`private const int RodWidth = 10;` Константа, определяющая ширину стержней.

private const int RodHeight = 300;; Константа, определяющая высоту стержней.

private const int RodSpacing = 265;; Константа, определяющая дистанцию между стержнями.

private const int DiscWidthMin = 50;; Константа, определяющая минимальную ширину дисков.

private const int DiscWidthStep = 20;; Константа, определяющая шаг между ширинами дисков.

private int stepsCount = 1;; Целочисленная переменная, определяющая количество шагов в алгоритме Ханойских башен.

private Panel[] rods;; Массив панелей, представляющих стержни.

private PictureBox[] discs;; Массив изображений дисков.

private int[,] rodStates;; Двумерный массив, представляющий состояние каждого стержня и положение дисков на них.

private bool isDragging = false;; Флаг, указывающий, перетаскивается ли в данный момент диск.

private PictureBox selectedDisk = null;; Ссылка на выбранный диск для перетаскивания.

private Point offset;; Смещение при перетаскивании для корректного отображения. private bool solvingStopped = false;; Флаг, указывающий, остановлен ли процесс решения задачи.

2.3. Структура программного обеспечения

1. При запуске программы, на экране появляется окно «приветствие», которое другими словами можно назвать «титульным листом», на котором указана тема курсовой работы и имя автора. После нажатия кнопки «Начать» открывается основное окно программы. На этом окне присутствует 5 кнопок, поле для вывода количества колец и башня с изначальным количеством колец.

2. Пользователем выбирается количество колец от 1 до 9.

3. Нажимается кнопка «Нарисовать», вследствие чего появятся кольца.
4. Пользователь может вручную попытаться переложить кольца или нажать кнопку «Решить», что бы это сделал компьютер.
5. После того как кольца будут перемещены, можно сравнить оптимальное количество перемещений с полученным.
6. Действия, выполняемые при нажатии кнопок:
7. «Нарисовать» — рисует башенку с указанным количеством колец.
8. «Решить» — Самым оптимальным способом перекладывает кольца башни.
9. «Стоп» — Останавливает процесс перекладывания колец, после чего пользователь может продолжить ручную.
10. Программа должна быть оптимизирована и быть простой для отладки.

Изначальным окном является титульное, где идёт приветствие (рисунок 2). После, при нажатии на кнопку “Начать”, идёт переход к настройкам игры (Рисунок 3), затем можно нажать на кнопку правила, где содержатся пояснения оп правилам игры (Рисунок 4), либо настроить игровое поле под себя выбрав количество стержней и стартовый стержень (Рисунок 5).

Нажав на кнопку нарисовать (Рисунок 6), на стержнях появится заданное ранее пользователем количество дисков на выбранном стартовом стержне (Рисунок 7). Станет доступна кнопка решить (Рисунок 8), нажав на которую ханойские башни сами начнут решаться в режиме реального времени, при этом пользователь сможет остановить решение, нажав на кнопку “Стоп” (Рисунок 9), когда будет уверен, что может продолжить дальше сам и возобновить в случае повторной неудачи.

При полном решении задачи Ханойских башен будет отображено победное уведомление в отдельном окне, где будет показано, за сколько ходов была решена задача (Рисунок 10).

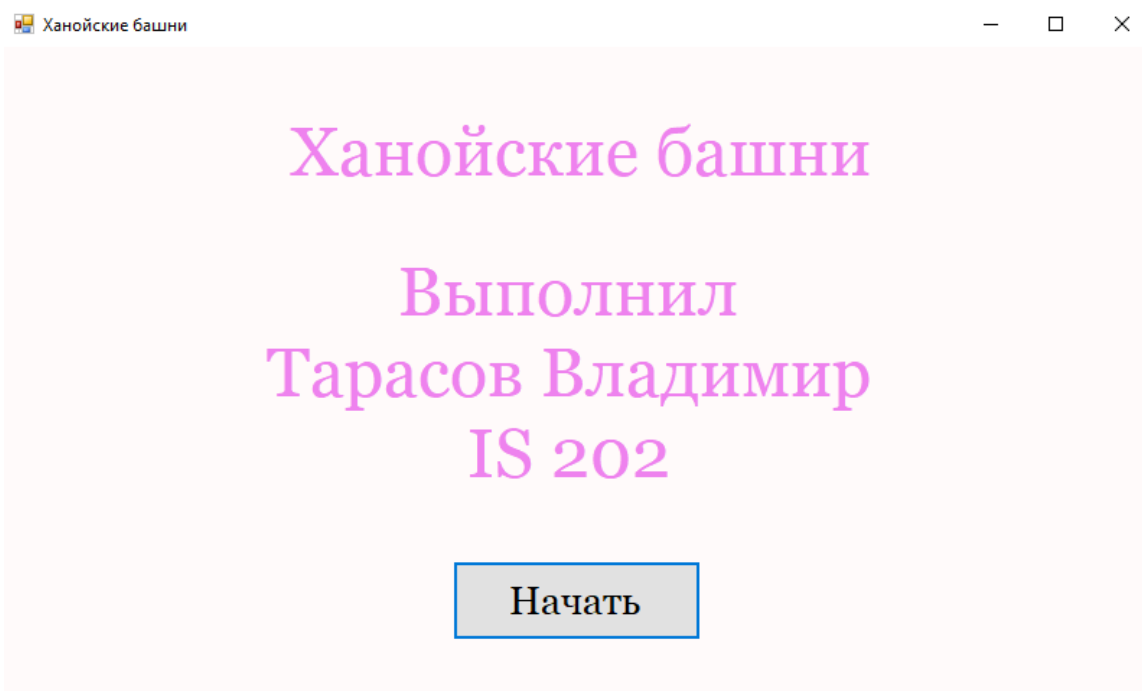


Рисунок 2. Титульный экран

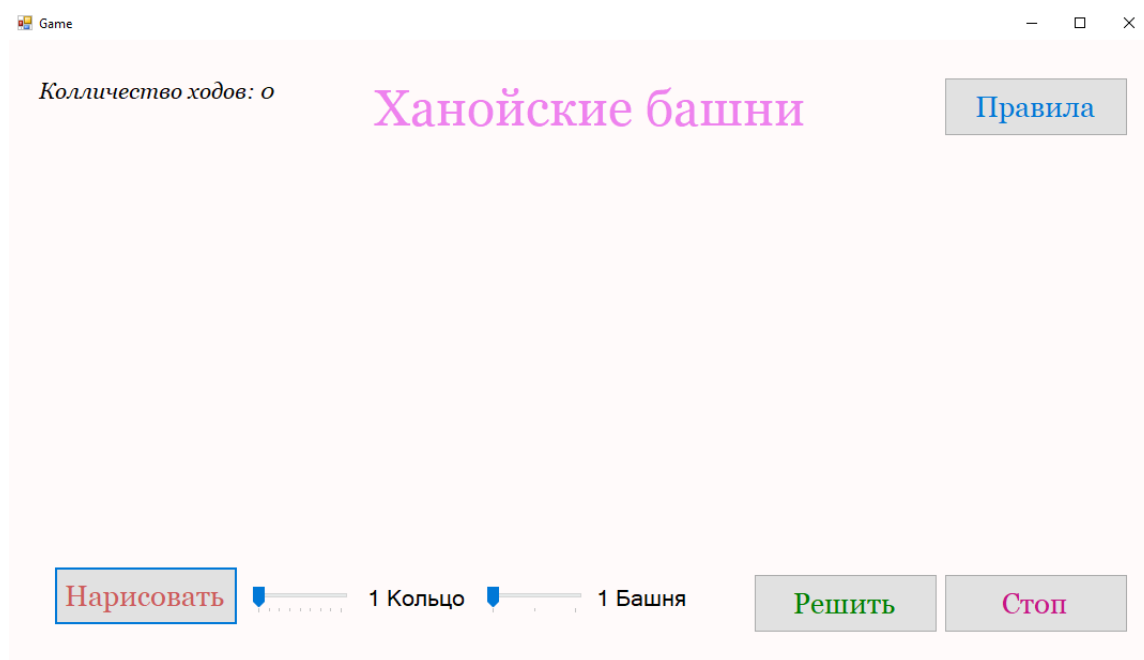


Рисунок 3. Настройки игры

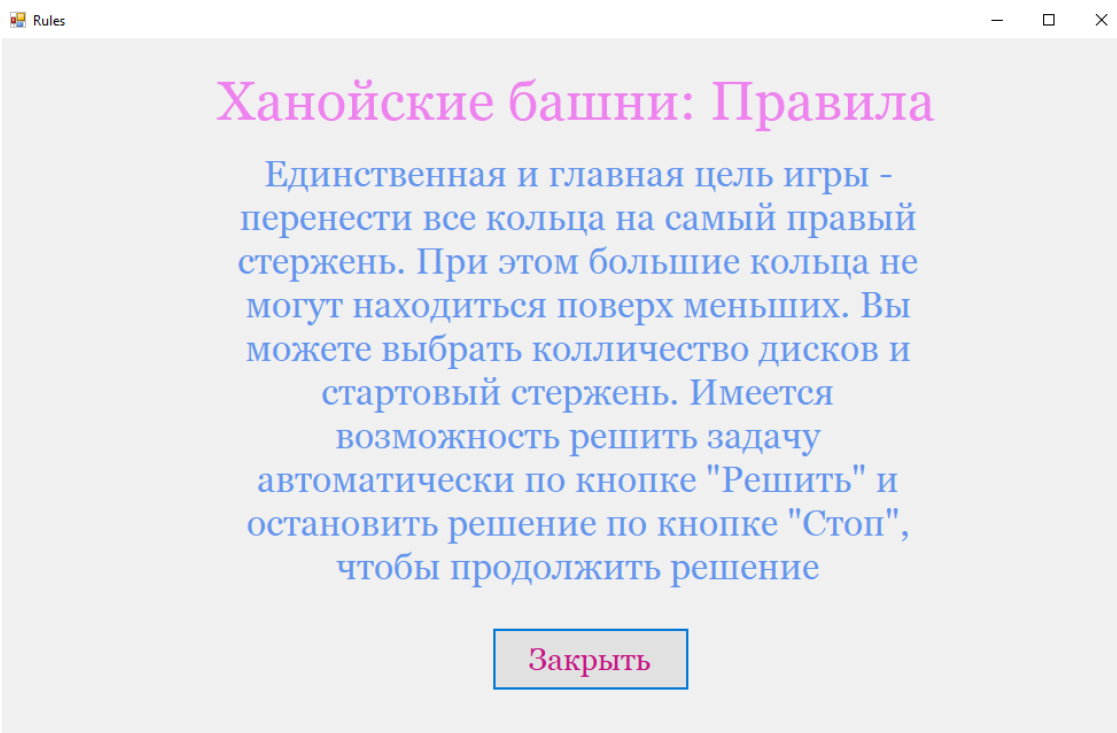


Рисунок 4. Правила игры

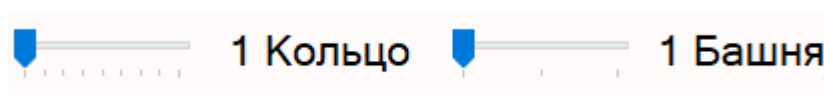


Рисунок 5. Количество дисков и стартовая позиция



Рисунок 6. Кнопка “Нарисовать”

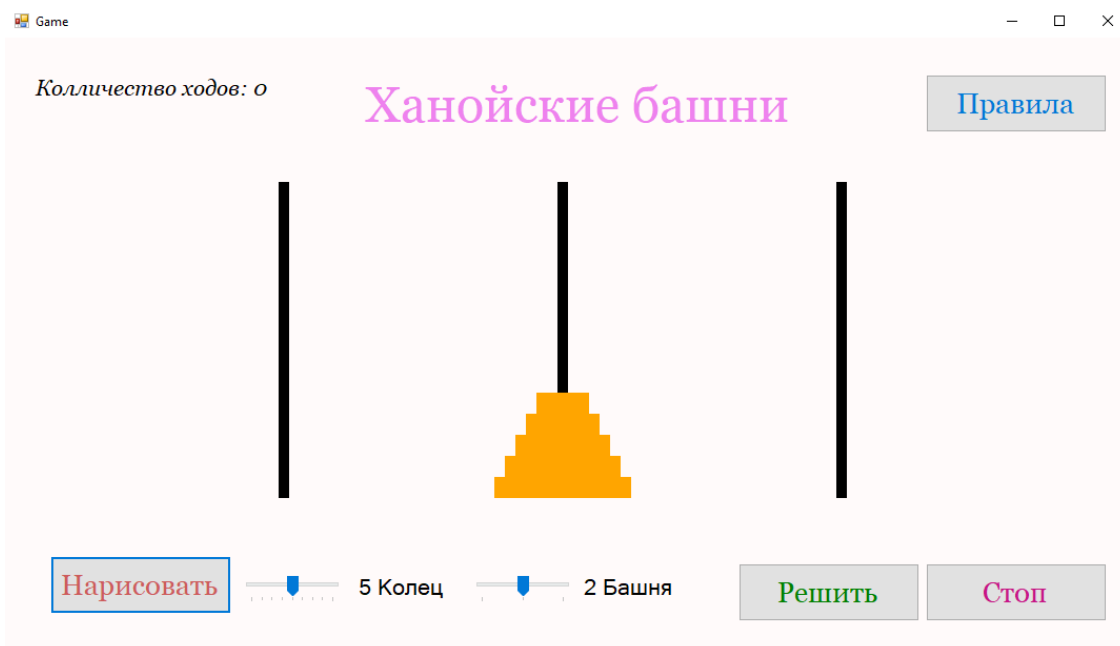


Рисунок 7. Стартовый расклад



Рисунок 8. Кнопка “Решить”



Рисунок 9. Кнопка “Стоп”

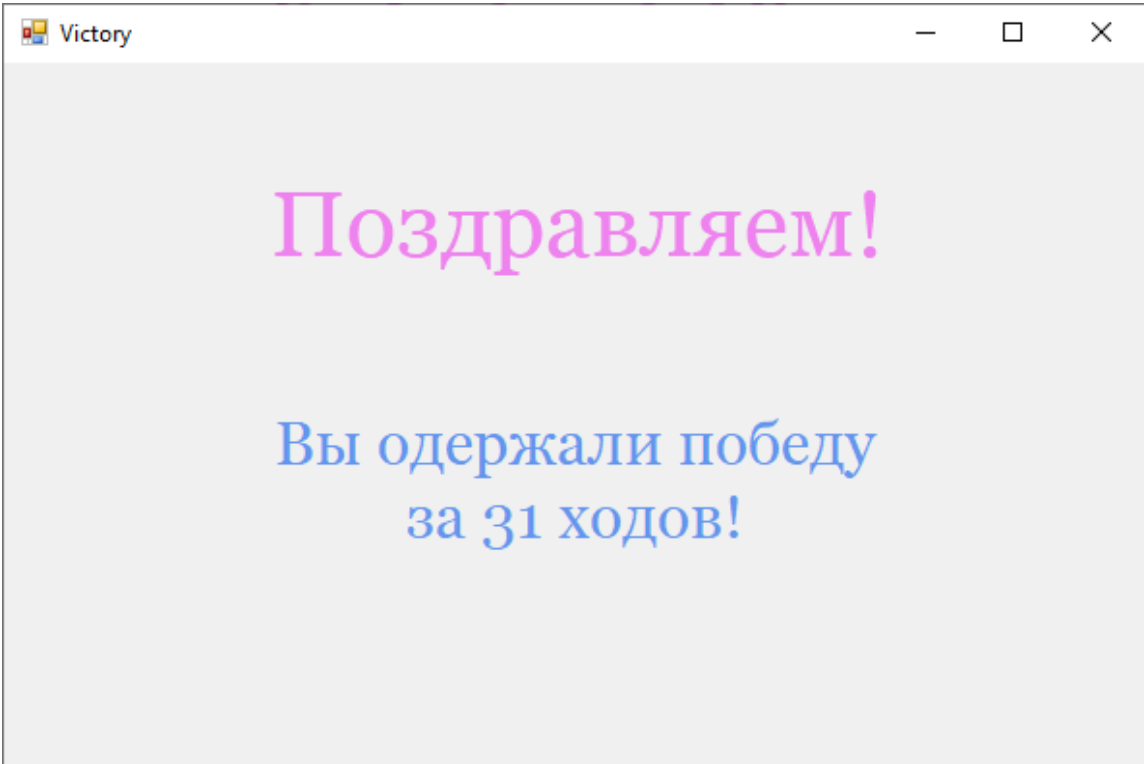


Рисунок 10. Победный экран

2.4. Алгоритмы по созданию функций

Функции отвечающие за регулировку количества дисков и стартового стержня с помощью событий Windows Forms. (Рисунок 11)

```

Ссылка: 1
private void circlesCount_Scroll(object sender, EventArgs e)
{
    NumDiscs = circlesCount.Value;
    label1.Text = circlesCount.Value.ToString() + " Колец";
}

Ссылка: 1
private void startTower_Scroll(object sender, EventArgs e)
{
    startColumn = startTower.Value-1;
    label2.Text = startTower.Value.ToString() + " Башня";
}

Ссылка: 1
private void rules_Click(object sender, EventArgs e)
{
    Rules rules = new Rules();
    rules.Show();
}

```

Рисунок 11. События регулирования

Метод `circlesCount_Scroll`: обработчик события изменения значения ползунка `circlesCount`. Обновляет переменную `NumDiscs` значением текущего положения ползунка. Обновляет текст метки `label1`, отображая количество колец. Изменение количества колец в задаче Ханойских башен при перемещении ползунка.

Метод `startTower_Scroll`: обработчик события изменения значения ползунка `startTower`. Обновляет переменную `startColumn` значением текущего положения ползунка, уменьшая на 1 (возможно, для соответствия индексам в коде). Обновляет текст метки `label2`, отображая номер выбранной башни. Изменение стартовой башни в задаче Ханойских башен при перемещении ползунка.

Метод `rules_Click`: обработчик события нажатия на кнопку `rules`. Создает новый экземпляр формы `Rules` и отображает ее с помощью метода `Show()`. Отображение окна с правилами игры или задачи.

Далее идёт инициализация игры, метод `InitializeGame` вызывается при нажатии кнопки “Нарисовать”. (Рисунок 12, 13)

```
private void InitializeGame()
{
    stepsCount = 1;
    // Очистим предыдущие диски перед созданием новых
    if (discs != null)
    {
        foreach (var disc in discs)
        {
            if (disc != null)
            {
                this.Controls.Remove(disc);
                disc.Dispose();
            }
        }
    }

    rods = new Panel[3];
    for (int i = 0; i < 3; i++)
    {
        rods[i] = new Panel
        {
            Width = RodWidth,
            Height = RodHeight,
            BackColor = Color.Black,
            Left = (i + 1) * RodSpacing - RodWidth / 2,
            Top = ClientSize.Height - RodHeight - 150,
            Parent = this
        };
    }
}
```

Рисунок 12. Очистка старых дисков и инициализация новых стержней

```
discs = new PictureBox[NumDiscs];
for (int i = 0; i < NumDiscs; i++)
{
    discs[i] = new PictureBox();
    discs[i].Width = DiscWidthMin + (NumDiscs - i - 1) * DiscWidthStep; //Правило уменьшения ширины дисков, с больших до малых
    discs[i].Height = 20;
    discs[i].BackColor = Color.Orange;
    discs[i].Left = rods[startColumn].Left + (RodWidth - discs[i].Width) / 2; // Координата по ширине
    discs[i].Top = rods[startColumn].Top + RodHeight - (i + 1) * discs[i].Height; // Координата по высоте
    discs[i].Parent = this;
    discs[i].Tag = startColumn; // Изначально все диски на стартовом стержне
    discs[i].BringToFront();
    discs[i].MouseDown += new MouseEventHandler(Disc_MouseDown);
    discs[i].MouseMove += new MouseEventHandler(Disc_MouseMove);
    discs[i].MouseUp += new MouseEventHandler(Disc_MouseUp);
}

rodStates = new int[3, NumDiscs]; //Какие диски есть на стержнях
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < NumDiscs; j++)
    {
        if (i == startColumn)
        {
            rodStates[i, j] = Array.IndexOf(discs, discs[j]); //Записываем в таблицу стержней индекс диска - его вес
        }
        else
            rodStates[i, j] = NumDiscs; //Наименьший диск = NumDiscs-1, поэтому 0 = NumDiscs
    }
}
```

Рисунок 13. Создание массива дисков и обновление статуса стержней

Ссылка 1

```
private void begin_Click(object sender, EventArgs e)
{
    InitializeGame();
}
```

Рисунок 14. Обработчик события начала игры

Для реализации переноса дисков с одного стержня на другой используются события мыши, такие как MouseDown, MouseEventArgs, MouseUp. (Рисунок 15, 16, 17)

```

Ссылка: 1
private void Disc_MouseDown(object sender, MouseEventArgs e)
{
    PictureBox clickedDisk = sender as PictureBox;

    int currentRod = (int)clickedDisk.Tag;
    int diskWeight = Array.IndexOf(disks, clickedDisk);

    int topDiskOnRod = GetTopDiskWeight(currentRod);

    // Проверяем, является ли данный диск верхним на стержне
    if (diskWeight == topDiskOnRod)
    {
        isDragging = true;
        selectedDisk = clickedDisk;
        offset = e.Location;
        selectedDisk.BringToFront();

        initialPosition = selectedDisk.Location;
    }
}

```

Рисунок 15. Взятие диска со стержня

```

Ссылка: 1
private void Disc_MouseMove(object sender, MouseEventArgs e)
{
    if (isDragging && selectedDisk != null) //Если выбран диск, он перемещается с мышкой
    {
        selectedDisk.Left = e.X + selectedDisk.Left - offset.X;
        selectedDisk.Top = e.Y + selectedDisk.Top - offset.Y;
    }
}

```

Рисунок 16. Передвижение диска с курсором мыши

```

Ссылка: 1
private void Disc_MouseUp(object sender, MouseEventArgs e)
{
    if (isDragging && selectedDisk != null)
    {
        isDragging = false;

        int targetRod = -1;
        bool droppedOnRod = false;

        for (int i = 0; i < 3; i++)
        {
            Rectangle expandedBounds = rods[i].Bounds;
            expandedBounds.Inflate(30, 30); // Увеличение хитбокса стержня

            if (expandedBounds.Contains(this.PointToClient(MousePosition)))
            {
                targetRod = i;
                droppedOnRod = true;
                break;
            }
        }

        if (droppedOnRod) //Если отпущен на стержень
        {
            if (CanMove(selectedDisk, targetRod))
            {
                MoveDisk(selectedDisk, targetRod);
            }
            else
            {
                selectedDisk.Location = initialPosition; // Возвращаем диск на исходную позицию
            }
        }
        else
        {
            selectedDisk.Location = initialPosition;
        }
    }
}

```

Рисунок 17. Отпуск диска на стержень

Метод Disc_MouseDown: обработчик события нажатия кнопки мыши на диске. Получает объект диска, на который было нажатие. Определяет текущий стержень (currentRod) и вес диска (diskWeight). Получает вес верхнего диска на текущем стержне (topDiskOnRod). Проверяет, является ли данный диск верхним на стержне. Если диск верхний, устанавливает флаг перетаскивания (isDragging) в true, запоминает выбранный диск (selectedDisk), запоминает смещение относительно клика (offset) и текущую позицию диска (initialPosition).

Метод Disc_MouseMove: обработчик события перемещения мыши при удерживании кнопки мыши на диске. Если флаг перетаскивания установлен в true и выбран диск (selectedDisk), изменяет его координаты в соответствии с текущим положением мыши.

Метод Disc_MouseUp: обработчик события отпускания кнопки мыши после перемещения диска. Если флаг перетаскивания установлен в true и выбран

диск (selectedDisk): Определяет, на какой стержень был отпущен диск. Увеличивает границы стержня для более удобного попадания диска. Проверяет, был ли диск отпущен на стержень (droppedOnRod). Если диск был отпущен на стержень: Проверяет, можно ли переместить диск на выбранный стержень согласно правилам игры. Если можно, перемещает диск на выбранный стержень; в противном случае, возвращает диск на исходную позицию. Если диск был отпущен вне стержней, возвращает диск на исходную позицию.

CanMove - Функция, которая проверяет возможность передвижения диска, делает это с помощью сравнения индекса целевого диска с верхним. Метод CanMove предназначен для проверки возможности перемещения диска на указанный стержень согласно правилам игры в Ханойских башнях. (Рисунок 18)

```

Ссылка 2
private bool CanMove(PictureBox disk, int targetRod)
{
    int upperDisk = GetTopDiskWeight(targetRod); // Вес верхнего диска

    // Если верхний диск целевого стержня больше перемещаемого диска или целевой стержень пустой
    if (upperDisk == NumDiscs || Array.IndexOf(discs, disk) > upperDisk)
    {
        return true;
    }
    return false;
}

```

Рисунок 18. Проверка на возможность движения диска

Метод GetTopDiskWeight - вспомогательный, он выполняет поиск и возвращает вес верхнего диска на указанном стержне. (Рисунок 19)

```

Ссылка 2
private int GetTopDiskWeight(int targetRod)
{
    for (int j = NumDiscs - 1; j >= 0; j--)
    {
        if (rodStates[targetRod, j] != NumDiscs)
        {
            return rodStates[targetRod, j];
        }
    }
    return NumDiscs;
}

```

Рисунок 19. Получение верхнего диска

Далее идёт метод MoveDisk, самая важная функция, которая реализует саму логику перемещения диска на стержень. (Рисунок 20)

```

Ссылка: 2
private void MoveDisk(PictureBox disk, int targetRod)
{
    int currentDiskIndex = 0;
    int destinationDiskIndex = 0;
    int currentRod = (int)disk.Tag; //Тег - номер стержня, на котором находится диск
    int DiskWeight = Array.IndexOf(discs, disk); //Где 0 - самый тяжёлый
    for (int j = 0; j < NumDiscs; j++)
    {
        if (rodStates[currentRod, j] == DiskWeight)
        {
            currentDiskIndex = j; //Позиция диска на старом стержне, где 0 - самый низ
        }
    }
    for (int j = 0; j < NumDiscs; j++)
    {
        if (rodStates[targetRod, j] == NumDiscs)
        {
            destinationDiskIndex = j; //Позиция диска на стержне назначения, где 0 - самый низ
            break;
        }
    }
    // Обновить состояния для старого и целевого стержня
    rodStates[currentRod, currentDiskIndex] = NumDiscs;
    rodStates[targetRod, destinationDiskIndex] = DiskWeight;

    // Позиция диска на целевом стержне
    disk.Left = rods[targetRod].Left + (RodWidth - disk.Width) / 2;
    disk.Top = rods[targetRod].Top + RodHeight - (destinationDiskIndex + 1) * disk.Height;
    // Обновить номер стержня, на котором находится диск
    disk.Tag = targetRod;

    steps.Text = "Количество ходов: " + stepsCount++;
    CheckGameCompletion();
}

```

Рисунок 20. Логика перемещения диска

Этот метод выполняет следующие действия:

1. Определяет текущий индекс диска на текущем стержне и индекс позиции для диска на целевом стержне.
2. Обновляет состояния для старого и целевого стержней.
3. Устанавливает новую позицию для диска на целевом стержне.
4. Обновляет номер стержня, на котором находится диск.
5. Увеличивает количество ходов.
6. Проверяет завершение игры.

Метод CheckGameCompletion проверяет, завершена ли игра в Ханойских башнях. В данном случае, он определяет, завершена ли игра при условии, что на третьем стержне в наивысшей позиции находится самый маленький диск. (Рисунок 21)

```

Ссылка: 1
private void CheckGameCompletion()
{
    if (rodStates[2, NumDiscs - 1] == NumDiscs - 1) //Если на 3-м стержне в наивысшей позиции находится наименьший диск
    {
        stepsCount--;
        Victory victory = new Victory();
        victory.Show();
        victory.label1.Text = "Вы одержали победу за " + stepsCount + " ходов!";
        solvingStopped = true; // Установить флаг solvingStopped в true при победе
    }
}

```

Рисунок 21. Проверка на конец игры

Метод solution_Click представляет собой обработчик события нажатия кнопки "Решение" на форме. (Рисунок 22)

```

Ссылка: 1
private void solution_Click(object sender, EventArgs e)
{
    solvingStopped = false;
    SolveHanoi(NumDiscs, startColumn, 2, 1-startColumn);
}

```

Рисунок 22. Обработчик кнопки "Решить"

Метод SolveHanoi рекурсивно решает головоломку Ханойских башен. В данном случае, он используется для автоматического решения башен с использованием рекурсии. (Рисунок 23)

```

Ссылка: 3
private void SolveHanoi(int n, int sourceRod, int destinationRod, int auxiliaryRod)
{
    if (n > 0 && !solvingStopped)
    {
        SolveHanoi(n - 1, sourceRod, auxiliaryRod, destinationRod);

        // Добавляем проверку кнопки стоп внутри цикла
        Application.DoEvents();
        if (solvingStopped)
            return; // Выйти из метода, если кнопка стоп была нажата
        if (CanMove(discs[NumDiscs - n], destinationRod))
        {
            MoveDisk(discs[NumDiscs - n], destinationRod);
        }
        // Проверяем кнопку стоп после перемещения
        Application.DoEvents();
        if (solvingStopped)
            return; // Выйти из метода, если кнопка стоп была нажата

        SolveHanoi(n - 1, auxiliaryRod, destinationRod, sourceRod);
        System.Threading.Thread.Sleep(5000 / (NumDiscs*NumDiscs));
    }
}

```

Рисунок 24. Алгоритм автоматического решения

Метод stopSolution_Click представляет собой обработчик события нажатия кнопки "Стоп" во время автоматического решения Ханойских башен. (Рисунок 25)

```

Ссылка 1
private void stopSolution_Click(object sender, EventArgs e)
{
    solvingStopped = true;
}

```

Рисунок 25. Остановка автоматического решения

Метод Game_FormClosed представляет собой обработчик события закрытия формы. В данном случае, он используется для завершения текущего процесса при закрытии формы. (Рисунок 26)

```

Ссылка 1
private void Game_FormClosed(object sender, FormClosedEventArgs e)
{
    System.Diagnostics.Process.GetCurrentProcess().Kill();
}

```

Рисунок 26. Обработчик закрытия формы

2.5. Отладка программы и тестирование модулей

Для простых отладочных задач использовался метод Console.WriteLine() для вывода информации в консоль. Это позволяет отслеживать ход выполнения программы и значения переменных. Данный метод помещался в функции, отвечающие за перестановку дисков и выводил информацию о их текущем положении для отслеживания в режиме реального времени.

Исключения не использовались в силу линейной архитектуры программного обеспечения и невозможности их возникновения в данном контексте.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы был разработан программный комплекс на языке программирования C# с использованием технологии Windows Forms для визуализации и решения головоломки "Ханойские башни".

Программа позволяет пользователю интерактивно управлять ходом игры, перемещая диски между стержнями, а также предоставляет функционал автоматического решения задачи. В ходе разработки программы были использованы принципы объектно-ориентированного программирования, включая создание классов, работу с событиями и обработку пользовательского ввода.

Язык программирования C# и технология Windows Forms обеспечили удобство в разработке графического интерфейса, а также предоставили богатые возможности для взаимодействия пользователя с приложением. Теоретическая часть курсовой включала в себя описание используемых технологий, таких как C# и Windows Forms, а также обоснование выбора данных технологий для решения поставленной задачи.

Рассмотрены основные элементы языка программирования C#, способы структурирования программ и средства обмена данными. В процессе работы были реализованы ключевые компоненты программы, включая отрисовку стержней и дисков, пользовательский ввод для перемещения дисков, автоматическое решение задачи методом Ханойских башен с возможностью прерывания процесса.

Визуализация и интерактивность приложения позволяют пользователям на практике понять и изучить принципы решения головоломки. Таким образом, разработанный программный комплекс предоставляет удобное и интересное средство для изучения и практического применения алгоритмов решения задачи Ханойских башен, а также демонстрирует применение основных принципов разработки программ на языке C# с использованием Windows Forms.

					ПК ТУРАН 1305000 КР ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

В ходе выполнения данной курсовой работы были приобретены навыки, важные для развития в области программирования и разработки программного обеспечения.

Были углублены знания в языке программирования C#. Освоены основные концепции языка, такие как работа с классами, методами, событиями, использование массивов и работа с библиотеками. Разработка пользовательского интерфейса с использованием Windows Forms: Приобретён навык по созданию графического интерфейса пользователя (GUI) с использованием технологии Windows Forms.

В процессе разработки приложения, особенно при автоматическом решении головоломки, можно было столкнуться с необходимостью управления многозадачностью и обработкой асинхронных событий, чтобы обеспечить плавное и отзывчивое взаимодействие с приложением.

Разработка автоматического решения Ханойских башен включала в себя использование рекурсии. Стало возможным понимание и применение принципов рекурсивного программирования, что способствует развитию алгоритмического мышления.

Были решены задачи управления памятью и ресурсами при создании и уничтожении графических элементов, таких как диски и стержни, что помогло лучше понять принципы управления ресурсами в приложениях.

В процессе создания графического интерфейса освоены принципы работы с графическими библиотеками, настройку и манипуляции с элементами управления, а также принципы построения пользовательских интерфейсов для лучшего взаимодействия с конечным пользователем.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Дейтел, П., & Дейтел, Х. (2013). "С# 2012. Комплект курсов." Москва: Издательский дом "Вильямс".
2. Троелсен, А. (2012). "Язык программирования С# 5.0 и платформа .NET 4.5." Санкт-Петербург: Питер.
3. Гриффитс, Э. (2007). "Программирование в С# 2005 для профессионалов." Москва: Издательский дом "Вильямс".
4. Шилдт, Г. (2010). "С# 4.0. Полное руководство." Москва: Издательский дом "Вильямс".
5. Гамма, Э., Хелм, Р., Джонсон, Р., & Влиссидес, Д. (2007). "Приемы объектно-ориентированного проектирования. Паттерны проектирования." Москва: Питер.
6. Макконнелл, С. (2004). "Совершенный код. Мастер-класс." Санкт-Петербург: Питер.
7. Бейли, М. (2006). "Algorithms in C#." Addison-Wesley.
8. Liberty, J., & MacDonald, B. (2003). "Learning C#." O'Reilly Media.
9. Официальная документация Microsoft для Windows Forms.
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-8.0>

```
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;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
namespace HanoiTowers
{
    public partial class HelloScreen : Form
    {
        public HelloScreen()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            title.Visible = false;
            author.Visible = false;
            begin.Visible = false;
            Game game = new Game();
            game.Show();
            this.Hide();
        }
    }
}
```



```

        private void HelloScreen_FormClosing(object sender,
FormClosingEventArgs e)
    {
        System.Diagnostics.Process.GetCurrentProcess().Kill();
    }
}

public partial class Rules : Form
{
    public Rules()
    {
        InitializeComponent();
    }

    private void begin_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}

public partial class Victory : Form
{
    public Victory()
    {
        InitializeComponent();
    }
}

public partial class Game : Form
{
    private Point initialPosition;

```

```
private int startColumn = 0;
```

```
private int NumDiscs = 1;
```

```
private const int RodWidth = 10;
```

```
private const int RodHeight = 300;
```

```
private const int RodSpacing = 265;
```

```
private const int DiscWidthMin = 50;
```

```
private const int DiscWidthStep = 20;
```

```
private int stepsCount = 1;
```

```
private Panel[] rods;
```

```
private PictureBox[] discs;
```

```
private int[,] rodStates;
```

```
private bool isDragging = false;
```

```
private PictureBox selectedDisk = null;
```

```
private Point offset;
```

```
private bool solvingStopped = false;
```

```
public Game()
```

```
{
```

```
    InitializeComponent();
```

```
}
```

```
private void circlesCount_Scroll(object sender, EventArgs e)
```

```
{
```

```
    NumDiscs = circlesCount.Value;
```

```
    label1.Text = circlesCount.Value.ToString() + " Колец";
```

```
}
```

```
private void startTower_Scroll(object sender, EventArgs e)
{
    startColumn = startTower.Value-1;
    label2.Text = startTower.Value.ToString() + " Башня";
}
```

```
private void rules_Click(object sender, EventArgs e)
{
    Rules rules = new Rules();
    rules.Show();
}
```

```
private void InitializeGame()
{
    stepsCount = 1;
    // Очистим предыдущие диски перед созданием новых
    if (discs != null)
    {
        foreach (var disc in discs)
        {
            if (disc != null)
            {
                this.Controls.Remove(disc);
                disc.Dispose();
            }
        }
    }
}
```

```
rods = new Panel[3];
for (int i = 0; i < 3; i++)
```

```

{
    rods[i] = new Panel
    {
        Width = RodWidth,
        Height = RodHeight,
        BackColor = Color.Black,
        Left = (i + 1) * RodSpacing - RodWidth / 2,
        Top = ClientSize.Height - RodHeight - 150,
        Parent = this
    };
}

discs = new PictureBox[NumDiscs];
for (int i = 0; i < NumDiscs; i++)
{
    discs[i] = new PictureBox();
    discs[i].Width = DiscWidthMin + (NumDiscs - i - 1) * DiscWidthStep;
    //Правило уменьшения ширины дисков, с больших до малых
    discs[i].Height = 20;
    discs[i].BackColor = Color.Orange;
    discs[i].Left = rods[startColumn].Left + (RodWidth - discs[i].Width) / 2;
    // Координата по ширине
    discs[i].Top = rods[startColumn].Top + RodHeight - (i + 1) *
discs[i].Height; // Координата по высоте
    discs[i].Parent = this;
    discs[i].Tag = startColumn; // Изначально все диски на стартовом
стержне
    discs[i].BringToFront();
    discs[i].MouseDown += new MouseEventHandler(Disc_MouseDown);

```

```

        discs[i].MouseMove += new MouseEventHandler(Disc_MouseMove);
        discs[i].MouseUp += new MouseEventHandler(Disc_MouseUp);
    }

    rodStates = new int[3, NumDiscs]; //Какие диски есть на стержнях
    for (int i = 0; i < 3; i++)
    {

        for (int j = 0; j < NumDiscs; j++)
        {
            if (i == startColumn)
            {
                rodStates[i, j] = Array.IndexOf(discs, discs[j]); //Записываем в
таблицу стержней индекс диска - его вес
            }
            else
            {
                rodStates[i, j] = NumDiscs; //Наименьший диск = NumDiscs-1,
поэтому 0 = NumDiscs
            }
        }
    }

    private void begin_Click(object sender, EventArgs e)
    {
        InitializeGame();
    }

    private void Disc_MouseDown(object sender, MouseEventArgs e)
    {

```

```
PictureBox clickedDisk = sender as PictureBox;
```

```
int currentRod = (int)clickedDisk.Tag;
```

```
int diskWeight = Array.IndexOf(discs, clickedDisk);
```

```
int topDiskOnRod = GetTopDiskWeight(currentRod);
```

```
// Проверяем, является ли данный диск верхним на стержне
```

```
if (diskWeight == topDiskOnRod)
```

```
{
```

```
    isDragging = true;
```

```
    selectedDisk = clickedDisk;
```

```
    offset = e.Location;
```

```
    selectedDisk.BringToFront();
```

```
    initialPosition = selectedDisk.Location;
```

```
}
```

```
}
```

```
private void Disc_MouseMove(object sender, MouseEventArgs e)
```

```
{
```

```
    if (isDragging && selectedDisk != null) //Если выбран диск, он  
    перемещается с мышкой
```

```
{
```

```
    selectedDisk.Left = e.X + selectedDisk.Left - offset.X;
```

```
    selectedDisk.Top = e.Y + selectedDisk.Top - offset.Y;
```

```
}
```

```
}
```

```

private void Disc_MouseUp(object sender, MouseEventArgs e)
{
    if (isDragging && selectedDisk != null)
    {
        isDragging = false;

        int targetRod = -1;
        bool droppedOnRod = false;

        for (int i = 0; i < 3; i++)
        {
            Rectangle expandedBounds = rods[i].Bounds;
            expandedBounds.Inflate(30, 30); // Увеличение хитбокса стержня

            if (expandedBounds.Contains(this.PointToClient(MousePosition)))
            {
                targetRod = i;
                droppedOnRod = true;
                break;
            }
        }

        if (droppedOnRod) //Если отпущен на стержень
        {
            if (CanMove(selectedDisk, targetRod))
            {
                MoveDisk(selectedDisk, targetRod);
            }
            else

```

```

        {
            selectedDisk.Location = initialPosition; // Возвращаем диск на
исходную позицию
        }
    }
    else
        selectedDisk.Location = initialPosition;
    }
}

```

```

private bool CanMove(PictureBox disk, int targetRod)
{
    int upperDisk = GetTopDiskWeight(targetRod); // Вес верхнего диска

    // Если верхний диск целевого стержня больше перемещаемого диска
или целевой стержень пустой
    if (upperDisk == NumDiscs || Array.IndexOf(discs, disk) > upperDisk)
    {
        return true;
    }
    return false;
}

```

```

private int GetTopDiskWeight(int targetRod)
{
    for (int j = NumDiscs - 1; j >= 0; j--)
    {
        if (rodStates[targetRod, j] != NumDiscs)
        {

```



```

        return rodStates[targetRod, j];
    }
}
return NumDiscs;
}

```

```

private void MoveDisk(PictureBox disk, int targetRod)
{
    int currentDiskIndex = 0;
    int destinationDiskIndex = 0;
    int currentRod = (int)disk.Tag; //Тег - номер стержня, на котором
находится диск
    int DiskWeight = Array.IndexOf(discs, disk); //Где 0 - самый тяжёлый
    for (int j = 0; j < NumDiscs; j++)
    {
        if (rodStates[currentRod, j] == DiskWeight)
        {
            currentDiskIndex = j; //Позиция диска на старом стержне, где 0 -
самый низ
        }
    }
    for (int j = 0; j < NumDiscs; j++)
    {
        if (rodStates[targetRod, j] == NumDiscs)
        {
            destinationDiskIndex = j; //Позиция диска на стержне назначения,
где 0 - самый низ
            break;
        }
    }
}

```

```

    }

    // Обновить состояния для старого и целевого стержня
    rodStates[currentRod, currentDiskIndex] = NumDiscs;
    rodStates[targetRod, destinationDiskIndex] = DiskWeight;

    // Позиция диска на целевом стержне
    disk.Left = rods[targetRod].Left + (RodWidth - disk.Width) / 2;
    disk.Top = rods[targetRod].Top + RodHeight - (destinationDiskIndex + 1)
* disk.Height;

    // Обновить номер стержня, на котором находится диск
    disk.Tag = targetRod;

    steps.Text = "Количество ходов: " + stepsCount++;
    CheckGameCompletion();
}

private void CheckGameCompletion()
{
    if (rodStates[2, NumDiscs - 1] == NumDiscs - 1) //Если на 3-м стержне в
наивысшей позиции находится наименьший диск
    {
        stepsCount--;
        Victory victory = new Victory();
        victory.Show();
        victory.label1.Text = "Вы одержали победу за " + stepsCount + "
ходов!";

        solvingStopped = true; // Установить флаг solvingStopped в true при
победе
    }
}

```

```
}
```

```
private void solution_Click(object sender, EventArgs e)
{
    solvingStopped = false;
    SolveHanoi(NumDiscs, startColumn, 2, 1-startColumn);
}
```

```
private void SolveHanoi(int n, int sourceRod, int destinationRod, int
auxiliaryRod)
{
    if (n > 0 && !solvingStopped)
    {
        SolveHanoi(n - 1, sourceRod, auxiliaryRod, destinationRod);

        // Добавляем проверку кнопки стоп внутри цикла
        Application.DoEvents();
        if (solvingStopped)
            return; // Выйти из метода, если кнопка стоп была нажата
        if (CanMove(discs[NumDiscs - n], destinationRod))
        {
            MoveDisk(discs[NumDiscs - n], destinationRod);
        }
        // Проверяем кнопку стоп после перемещения
        Application.DoEvents();
        if (solvingStopped)
            return; // Выйти из метода, если кнопка стоп была нажата

        SolveHanoi(n - 1, auxiliaryRod, destinationRod, sourceRod);
    }
}
```

```
        System.Threading.Thread.Sleep(5000 / (NumDiscs*NumDiscs));  
    }  
}
```

```
private void stopSolution_Click(object sender, EventArgs e)  
{  
    solvingStopped = true;  
}
```

```
private void Game_FormClosed(object sender, FormClosedEventArgs e)  
{  
    System.Diagnostics.Process.GetCurrentProcess().Kill();  
}  
}  
}
```