Лабораторная работа № 1.  
Основы совместной разработки.  
Система управления версиями Git

**Цель лабораторной работы**

Закрепление теоретических знаний по основам применения системы контроля версий в разработке программ.

**Постановка задачи**

**Git** – это бесплатная распределенная система контроля версий с открытым исходным кодом, предназначенная для быстрой и эффективной работы со всеми проектами, от небольших до очень крупных.

**Применение:**

* Командная разработка отдельных модулей;
* Возможность отката к любой из предыдущих версий программы. При этом важно знать, что Git хранит только изменения файлов, а не сами версии файлов, что существенно экономит ресурсы ПК.

**Термины:**

**Ветка** представляет собой независимое направление разработки. Ветки выступают в качестве абстрактного представления процессов редактирования, индексирования и выполнения коммитов, которые описаны в первом модуле «Основы Git» этой серии статей. Ветки можно рассматривать как способ запросить новый рабочий каталог, раздел проиндексированных файлов или историю проекта. Новые коммиты записываются в историю текущей ветки, что приводит к образованию форка в истории проекта.

**Репозиторий –** Набор коммитов, а также ветки и теги для идентификации коммитов.

**Тег –** Ссылка, которую обычно используют, чтобы отметить конкретную точку в последовательности коммитов. В отличие от указателя HEAD, тег не обновляется по команде git commit.

**Контроль версий –** Система, которая регистрирует изменения в файле или наборе файлов с течением времени, чтобы впоследствии можно было извлечь определенные версии.

Рассмотрим команды Git:

* git add: Команда git add добавляет содержимое рабочего каталога в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.
* git status: Команда git status показывает состояния файлов в рабочем каталоге и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.
* git commit: Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.
* git reset: Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочем каталоге при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.
* git clean: Команда git clean используется для удаления мусора из рабочего каталога. Это могут быть результаты сборки проекта или файлы конфликтов слияний.
* git branch: Эта команда выступает универсальным инструментом администрирования веток. С ее помощью можно создавать изолированные среды разработки в одном репозитории.
* git checkout: С командой git checkout можно не только получать старые коммиты и прежние версии файлов, но и осуществлять навигацию по существующим веткам. В сочетании с базовыми командами Git она позволяет сосредоточиться на определенном направлении разработки.
* git clone: Создает копию существующего репозитория Git. Клонирование — самый распространенный способ, с помощью которого разработчики могут получить рабочую копию центрального репозитория.
* git fetch: С помощью команды извлечения можно загрузить ветку из другого репозитория вместе со всеми связанными с ней коммитами и файлами, однако при этом изменения не будут интегрированы в локальный репозиторий. Благодаря этому вы сможете проверить изменения перед слиянием с проектом.
* git init: Инициализирует новый репозиторий Git. Если вы хотите использовать в проекте контроль версий, эту команду следует изучить раньше остальных.
* git merge: Эффективный способ интеграции изменений из разошедшихся веток. После разветвления истории проекта командой git branch можно использовать команду git merge, чтобы объединить отдельные ветки.
* git pull: Команда git pull — это автоматизированная версия команды git fetch. Она загружает ветку из удаленного репозитория и сразу же объединяет ее с текущей веткой. Эта команда представляет собой git-эквивалент команды svn update.
* git push: Команда git push противоположна команде извлечения (с некоторыми оговорками). С ее помощью можно перенести локальную ветку в другой репозиторий и без труда опубликовать поступивший код. Эта команда похожа на svn commit с тем исключением, что она отправляет не один набор изменений, а серию коммитов.

**С Git можно работать двумя основными способами:**

* **Через командную строку (терминал)** – это даёт полный контроль над операциями и часто используется в профессиональной разработке.
* **С помощью графического интерфейса (GUI)** – например, в **GitHub Desktop**, **GitKraken** или **Sourcetree**, что упрощает визуализацию изменений и подходит для новичков.

**Работа с Git через терминал**

### Шаг 1: Установка Git

1. Скачайте и установите Git с [официального сайта Git](https://git-scm.com/).
2. Запустите установочный файл и следуйте инструкциям мастера установки.

### Шаг 2: Конфигурация Git

1. Откройте терминал (командную строку) и выполните следующие команды, чтобы настроить имя пользователя и email:
2. git config --global user.name "Your Name"
3. git config --global user.email "youremail@example.com"
4. Можно также настроить редактор по умолчанию:
5. git config --global core.editor "nano"

### Шаг 3: Создание репозитория

1. Создайте новый репозиторий на GitHub (или другом хостинге Git).
2. Инициализируйте Git в вашем проекте:
3. git init

### Шаг 4: Основные команды Git

1. git status: Показывает текущий статус репозитория.
2. git add <file>: Добавляет файлы для отслеживания.
3. git commit -m "Commit message": Фиксирует изменения в репозитории.
4. git push origin <branch>: Отправляет изменения на удаленный репозиторий.
5. git pull origin <branch>: Получает изменения с удаленного репозитория.
6. git clone <repository-url>: Клонирует удаленный репозиторий на локальную машину.

### Шаг 5: Работа с ветками

1. git branch: Показывает список веток.
2. git checkout -b <branch-name>: Создает новую ветку и переключается на неё.
3. git merge <branch>: Объединяет ветку в текущую ветку.
4. git branch -d <branch-name>: Удаляет ветку.

### Шаг 6: Работа с удаленными репозиториями

1. Добавьте удаленный репозиторий:
2. git remote add origin <repository-url>
3. Просмотрите список удаленных репозиториев:
4. git remote -v

### Шаг 7: Разрешение конфликтов

1. При слиянии веток могут возникнуть конфликты.
2. Решите конфликты в файлах, помеченных <<<<<<<, =======, >>>>>>>.
3. После решения конфликтов сделайте git add и git commit.

### Шаг 8: Использование .gitignore

1. Создайте файл .gitignore в корне проекта.
2. Укажите в нем файлы и папки, которые не должны быть отслеживаемыми Git.

### Шаг 9: Просмотр истории

1. git log: Показывает историю коммитов.
2. git diff <file>: Показывает изменения в файле.

### Шаг 10: Дополнительные ресурсы

1. Для более глубокого понимания Git рекомендуется изучить дополнительные команды и концепции, такие как ветвление, ребейзинг и подмодули.

**Работа с Git через GitHub**

**GitHub** — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub Inc.

Для начала работы необходимо зарегистрироваться на сайте <https://github.com/>

и скачать **GitHub Desktop** – приложение, которое позволяет взаимодействовать с GitHub с помощью графического пользовательского интерфейса. <https://desktop.github.com/>

Создадим репозиторий с помощью **GitHub Desktop** .

Запускаем **GitHub Desktop**, нажимаем File -> New repository, выбираем имя репозитория и местоположение (рисунок 1).

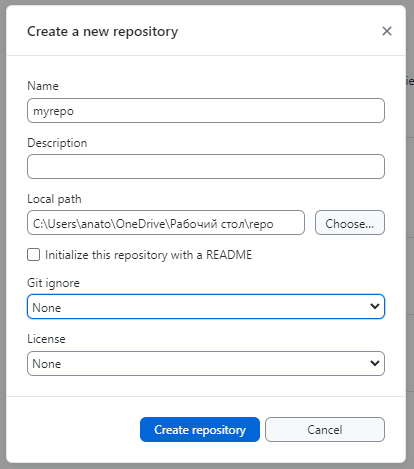


Рисунок 1 – Создание репозитория через GitHub

Не забываем скачать gitignore для C# и положить в репозиторий.

В программе зеленым плюсом отмечено, что было добавлено, видим, что добавился «файл.txt» (рисунок 2).

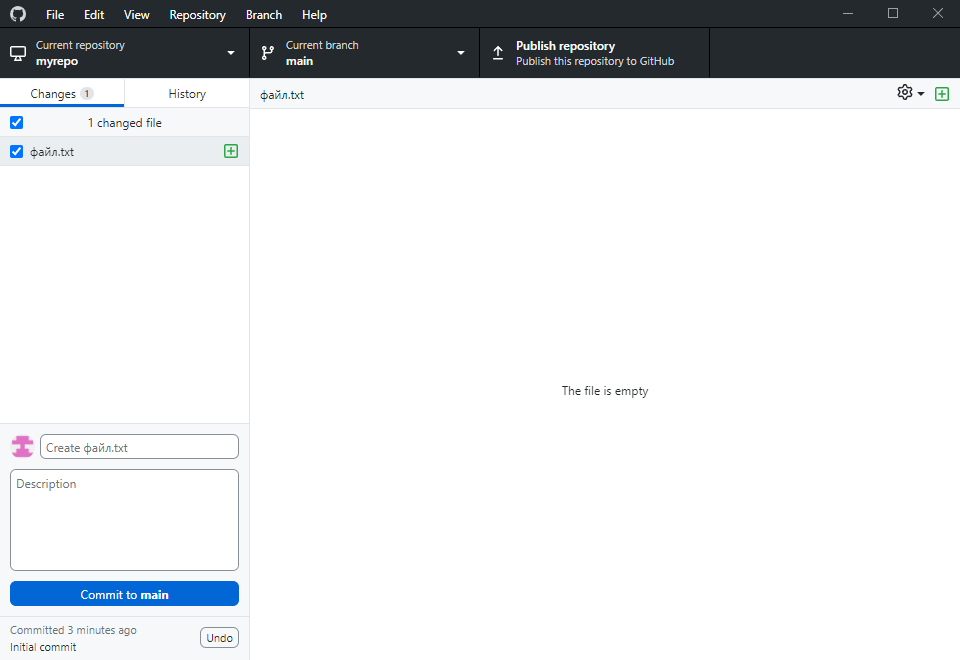


Рисунок 2 – Создание репозитория через GitHub

Зафиксируем изменения с помощью коммита, введем название комита и нажмем **Commit to main** (рисунок 3).

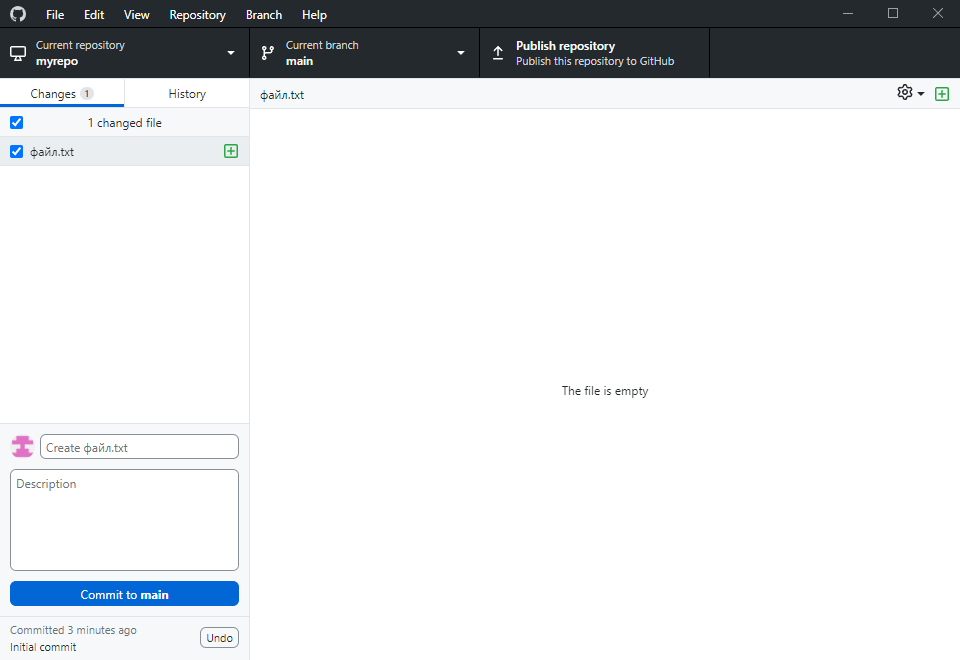


Рисунок 3 – Фиксация изменений

Пока что репозиторий локальный (находится на нашем компьютере), чтобы к нашему репозиторию был доступ из интернета необходимо опубликовать его. Для этого нужно нажать publish repository (рисунок 4).

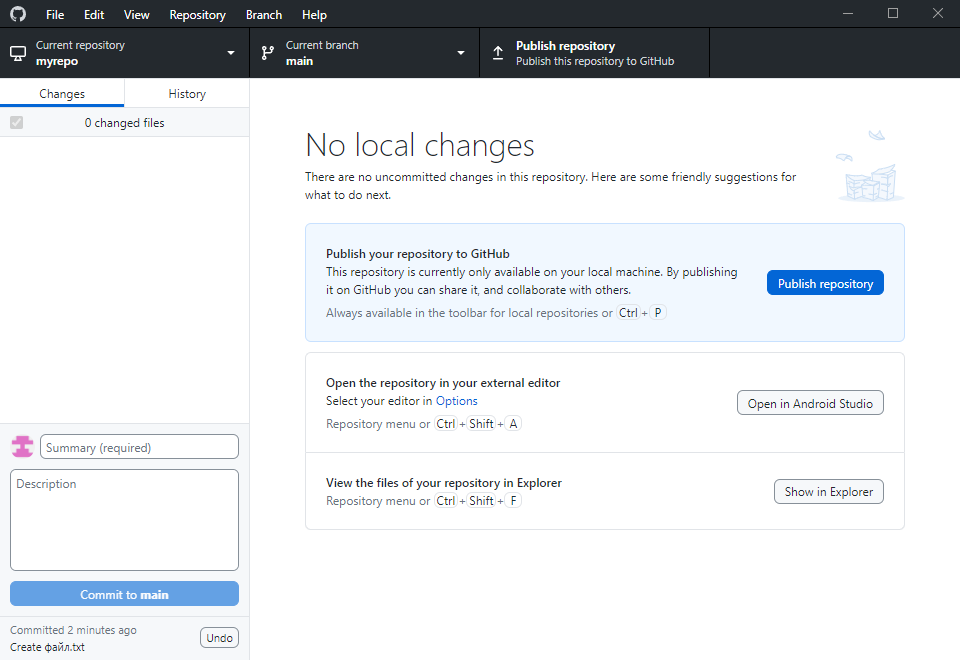


Рисунок 4 – Публикация репозитория

После опубликования наш репозиторий появится в аккаунте GitHub на сайте.

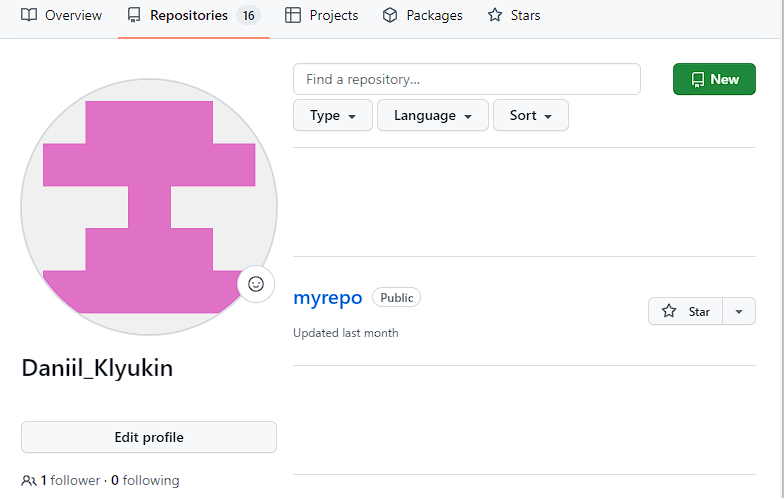


Рисунок 5 – Страница аккаунта в GitHub

Перейдем в репозиторий (рисунок 6). Здесь отображаются файлы, находящиеся в репозитории, текущая ветка, а также кнопки для скачивания и работы с репозиторием. Чтобы добавить справку о программе в виде файла README необходимо нажать Add a README.

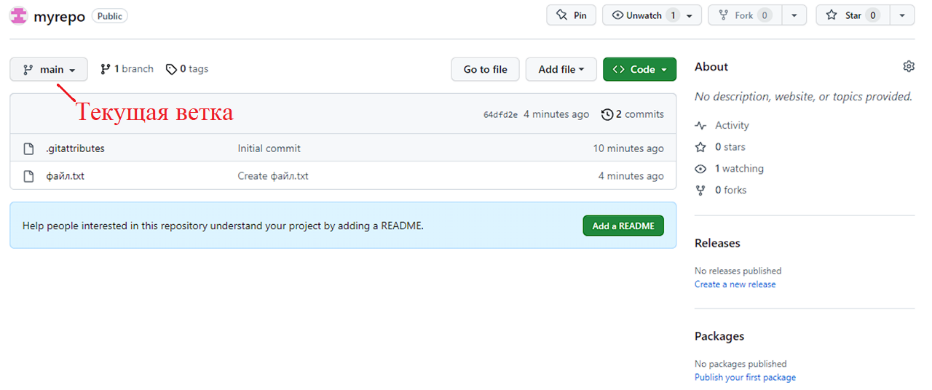


Рисунок 6 – Страница аккаунта в GitHub

Для создания новой ветки необходимо зайти **Branch** -> **New branch**, после чего у вас добавится новая ветка (рисунок 7).

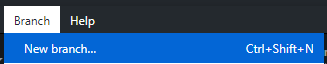


Рисунок 7 – Создание ветки в GitHub Desktop

Далее необходимо заполнить информацию о ветке, выбрать на основе какой ветки будет создана данная (рисунок 8).

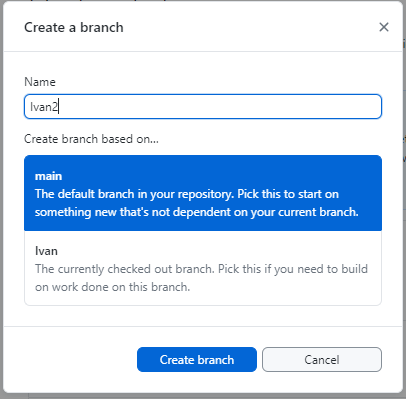


Рисунок 8 – Заполнение данных о ветке

**Обратите внимание**, что когда вы меняете ветку, то файлы в вашей папке с проектом также автоматически обновляются!

Чтобы ветка появилась в удаленном репозитории её также нужно опубликовать.

Чтобы забрать изменения с удаленного репозитория необходимо нажать **Fetch**, например, когда кто-то изменил файлы и вы хотите обновить свой проект в соответствии с изменениями удаленного репозитория (рисунок 9).

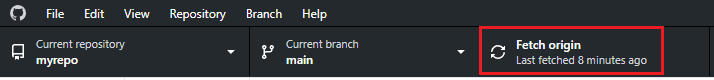


Рисунок 9 – Обновление информации о ветке

Данную команду желательно выполнять каждый раз перед началом работы с репозиторием.

**Задание на лабораторную работу**

1. Зарегистрироваться на GitHub (<https://github.com/>), установить Git (<https://git-scm.com/downloads>), установить GitHub Desktop (<https://desktop.github.com/>)
2. Разбиться на **группы по 2 человека**.
3. Создать и инициализировать репозиторий, добавить gitignore.
4. Каждый студент в группе разрабатывает модули проекта (в истории разработки должна быть отражена работа обоих студентов).
5. Расчетная часть независима от другого кода (является API), взаимодействие с ней происходит только через вызов её методов.
6. Графический интерфейс обращается к API и отображает текущее состояние игры.
7. Разработать файл документации Readme.md на GitHub странице репозитория.
8. Защита работы включает демонстрацию работы программы и историю разработки в системе Git.
9. Для успешной защиты:

* программа должна быть задокументирована с помощью комментариев,
* код программы не должен содержать не используемые блоки, должен быть «чистым» и стилистически верно оформлен,
* интерфейс программы должен быть гибким и удобным.

**Варианты заданий на лабораторную работу**

Можно выбрать произвольный вариант (без повторений), либо придумать свою игру, игра не должна быть пошаговой.

1. Разработать игру "Змейка" с возможностью управления змейкой и сбором еды для увеличения длины.
2. Разработать игру "Пинг-понг" с управлением ракеткой (платформой) и отскоком мяча.
3. Разработать игру "Арканоид" с управлением платформы для разрушения блоков.
4. Разработать игру "Космические защитники" с управлением космическим кораблем и стрельбой по врагам.
5. Разработать игру "Тетрис" с падающими фигурами для составления линий.
6. Разработать игру "Битва за выживание" с управлением персонажем и сражениями с противниками.
7. Разработать игру "Бегущий человек" с управлением персонажем и избеганием препятствий.
8. Разработать игру "Лабиринт" с перемещением персонажа к выходу, избегая ловушек.

**Пример разработки приложения**

**Обзор проекта**

Создадим простую игру "Угадай число", где компьютер загадывает число от 1 до 100, а игрок пытается его угадать. Игра будет включать:

* Логику игры в отдельном классе (расчетная часть)
* Графический интерфейс на Windows Forms
* Ведение статистики попыток
* Подсказки ("больше/меньше")

**Шаги по настройке репозитория**

**Шаг 1: Установка и настройка GitHub Desktop**

1. **Скачайте и установите** GitHub Desktop с [официального сайта](https://desktop.github.com/)
2. **Авторизуйтесь** с вашим GitHub аккаунтом
3. **Настройте Git** (имя и email) в File → Options → Git

**Шаг 2: Создание репозитория (выполняет один из участников)**

1. Нажмите "Create a New Repository"
2. Заполните:
   * Name: GuessNumberGame
   * Local path: выберите папку для проекта
   * Initialize with README: ✅
   * Git ignore: VisualStudio
   * License: MIT
3. Нажмите "Create repository"

**Шаг 3: Приглашение соавтора**

1. Перейдите на GitHub.com в созданный репозиторий
2. Settings → Collaborators → Add people
3. Введите GitHub-логин партнера и подтвердите

**Шаг 4: Клонирование репозитория (для второго участника)**

1. В GitHub Desktop: File → Clone repository
2. Выберите репозиторий из списка доступных
3. Укажите локальную папку и нажмите "Clone"

**Шаг 5: Начало работы (общие шаги)**

1. Создайте новую ветку для задачи:
   * Current branch → New
   * Имя ветки: feature/game-logic (для логики) или feature/gui (для интерфейса)
2. Откройте проект в Visual Studio через File → Open → Project/Solution

**Шаг 6: Рабочий процесс в команде**

1. **Перед началом работы**:
   * Pull последние изменения (Fetch origin → Pull)
   * Создайте новую ветку для своей задачи
2. **Регулярные коммиты**:

* - Вносите изменения в код
* - В GitHub Desktop:
  + Проверьте изменения в Changes
  + Напишите осмысленное описание коммита
* Нажмите "Commit to [branch-name]"

1. **Синхронизация изменений**:
   * Push ваших коммитов: Push origin
   * Pull изменений партнера: Fetch origin → Pull

**Шаг 7: Решение конфликтов (если возникнут)**

1. При конфликте слияния:
   * Откройте файл с конфликтами в Visual Studio
   * Выберите нужные изменения (или объедините вручную)
   * Сделайте коммит с разрешением конфликта

**Шаг 8: Создание Pull Request для слияния**

1. Переключитесь на ветку main
2. Сделайте Pull последних изменений
3. Branch → Create Pull Request
4. Укажите:
   * Base: main
   * Compare: ваша ветка
5. Добавьте описание изменений
6. Попросите партнера провести code review

**Шаг 9: Code Review и слияние**

1. Партнер проверяет изменения в:
   * Files changed
   * Оставляет комментарии при необходимости
2. После одобрения:
   * Нажмите "Merge pull request"
   * Подтвердите слияние
   * Удалите ветку (если больше не нужна)

**Шаг 10: Обновление локальной версии**

1. После слияния:
   * Переключитесь на main
   * Сделайте Pull чтобы получить изменения
   * Удалите локальную ветку (Branch → Delete)

**Рекомендации по организации работы:**

1. **Разделение задач**:
   * Один студент: GameLogic класс
   * Второй студент: MainForm интерфейс
2. **Частота синхронизации**:
   * Делайте Pull перед началом работы
   * Делайте Push после каждой завершенной задачи
3. **Соглашение о коммитах**:

* feat: добавлен класс GameLogic
* fix: исправление проверки ввода
* refactor: изменение структуры кода
* docs: обновление README

1. **Работа с Issues**:
   * Создавайте Issues для новых задач
   * Назначайте исполнителей
   * Закрывайте после реализации

**Шаги по разработке проекта:**

**Шаг 1: Создание проекта**

1. Откройте Visual Studio
2. Создайте новый проект: **Windows Forms App (.NET Framework)**
3. Назовите проект "GuessNumberGame"

**Шаг 2: Реализация игровой логики**

Класс GameLogic.cs

Создадим класс, отвечающий за всю игровую логику:

namespace GuessNumberGame;

public partial class GameLogic

{

private int \_secretNumber;

private int \_attempts;

private readonly int \_minValue;

private readonly int \_maxValue;

private readonly List<int> \_guessHistory;

public int Attempts => \_attempts;

public IReadOnlyList<int> GuessHistory => \_guessHistory.AsReadOnly();

public int MinValue => \_minValue;

public int MaxValue => \_maxValue;

public GameLogic(int min = 1, int max = 100)

{

\_minValue = min;

\_maxValue = max;

\_guessHistory = new List<int>();

ResetGame();

}

public void ResetGame()

{

var random = new Random();

\_secretNumber = random.Next(\_minValue, \_maxValue + 1);

\_attempts = 0;

\_guessHistory.Clear();

}

public GuessResult MakeGuess(int number)

{

if (number < \_minValue || number > \_maxValue)

throw new ArgumentOutOfRangeException(nameof(number),

$"Число должно быть между {\_minValue} и {\_maxValue}");

\_attempts++;

\_guessHistory.Add(number);

if (number == \_secretNumber)

return GuessResult.Correct;

return number < \_secretNumber ? GuessResult.TooLow : GuessResult.TooHigh;

}

}

namespace GuessNumberGame;

public enum GuessResult

{

TooLow,

Correct,

TooHigh

}

**Шаг 3: Создание пользовательского интерфейса**

Форма MainForm.cs

Разработаем основной интерфейс игры:

namespace GuessNumberGame

{

public partial class MainForm : Form

{

private readonly GameLogic \_game;

private readonly Color \_defaultBackColor;

public MainForm()

{

InitializeComponent();

\_game = new GameLogic();

\_defaultBackColor = guessTextBox.BackColor;

UpdateStats();

}

private void guessButton\_Click(object sender, EventArgs e)

{

if (!int.TryParse(guessTextBox.Text, out int guess))

{

MessageBox.Show("Пожалуйста, введите целое число!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var result = \_game.MakeGuess(guess);

guessTextBox.BackColor = \_defaultBackColor;

switch (result)

{

case GuessResult.Correct:

MessageBox.Show($"Поздравляем! Вы угадали число за {\_game.Attempts} попыток!",

"Победа!", MessageBoxButtons.OK, MessageBoxIcon.Information);

\_game.ResetGame();

break;

case GuessResult.TooLow:

statusLabel.Text = "Загаданное число больше!";

break;

case GuessResult.TooHigh:

statusLabel.Text = "Загаданное число меньше!";

break;

}

UpdateStats();

guessTextBox.SelectAll();

guessTextBox.Focus();

}

catch (ArgumentOutOfRangeException ex)

{

guessTextBox.BackColor = Color.LightPink;

statusLabel.Text = ex.Message;

}

}

private void UpdateStats()

{

attemptsLabel.Text = $"Попыток: {\_game.Attempts}";

rangeLabel.Text = $"Диапазон: {\_game.MinValue}-{\_game.MaxValue}";

historyListBox.DataSource = null;

historyListBox.DataSource = \_game.GuessHistory;

}

private void newGameButton\_Click(object sender, EventArgs e)

{

\_game.ResetGame();

statusLabel.Text = "Введите число и нажмите Угадать";

UpdateStats();

guessTextBox.Text = "";

guessTextBox.Focus();

}

private void guessTextBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == (char)Keys.Enter)

{

guessButton.PerformClick();

e.Handled = true;

}

}

}

}

Дизайн программы представлен на рисунке 9.

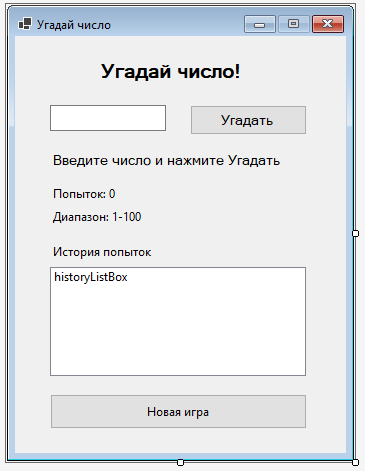


Рисунок 9 – Главное окно программы

**Шаг 4: Запуск и тестирование**

1. Нажмите F5 для запуска приложения
2. Попробуйте угадать число:
   * Вводите числа в текстовое поле
   * Получайте подсказки "больше/меньше"
   * Следите за историей попыток
3. При правильном ответе получите сообщение о победе
4. Нажмите "Новая игра" для сброса

**Архитектурные особенности**

1. **Разделение логики и интерфейса**:
   * GameLogic содержит всю игровую логику
   * MainForm отвечает только за отображение и взаимодействие
2. **Принцип единственной ответственности**:
   * Каждый класс и метод выполняет одну четкую задачу
3. **Инкапсуляция**:
   * Все внутренние поля GameLogic приватные
   * Доступ к данным только через публичные свойства и методы
4. **Обработка ошибок**:
   * Проверка ввода на корректность
   * Обработка исключений для чисел вне диапазона

Код дизайнера окна:

namespace GuessNumberGame

{

partial class MainForm

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

private void InitializeComponent()

{

this.titleLabel = new System.Windows.Forms.Label();

this.guessTextBox = new System.Windows.Forms.TextBox();

this.guessButton = new System.Windows.Forms.Button();

this.statusLabel = new System.Windows.Forms.Label();

this.attemptsLabel = new System.Windows.Forms.Label();

this.rangeLabel = new System.Windows.Forms.Label();

this.historyLabel = new System.Windows.Forms.Label();

this.historyListBox = new System.Windows.Forms.ListBox();

this.newGameButton = new System.Windows.Forms.Button();

this.SuspendLayout();

// titleLabel

this.titleLabel.AutoSize = true;

this.titleLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 14F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.titleLabel.Location = new System.Drawing.Point(70, 20);

this.titleLabel.Name = "titleLabel";

this.titleLabel.Size = new System.Drawing.Size(160, 24);

this.titleLabel.TabIndex = 0;

this.titleLabel.Text = "Угадай число!";

// guessTextBox

this.guessTextBox.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.guessTextBox.Location = new System.Drawing.Point(30, 60);

this.guessTextBox.Name = "guessTextBox";

this.guessTextBox.Size = new System.Drawing.Size(100, 26);

this.guessTextBox.TabIndex = 1;

this.guessTextBox.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.guessTextBox\_KeyPress);

// guessButton

this.guessButton.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.guessButton.Location = new System.Drawing.Point(150, 60);

this.guessButton.Name = "guessButton";

this.guessButton.Size = new System.Drawing.Size(100, 26);

this.guessButton.TabIndex = 2;

this.guessButton.Text = "Угадать";

this.guessButton.UseVisualStyleBackColor = true;

this.guessButton.Click += new System.EventHandler(this.guessButton\_Click);

// statusLabel

this.statusLabel.AutoSize = true;

this.statusLabel.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));

this.statusLabel.Location = new System.Drawing.Point(30, 100);

this.statusLabel.Name = "statusLabel";

this.statusLabel.Size = new System.Drawing.Size(185, 17);

this.statusLabel.TabIndex = 3;

this.statusLabel.Text = "Введите число и нажмите Угадать";

// attemptsLabel

this.attemptsLabel.AutoSize = true;

this.attemptsLabel.Location = new System.Drawing.Point(30, 130);

this.attemptsLabel.Name = "attemptsLabel";

this.attemptsLabel.Size = new System.Drawing.Size(64, 13);

this.attemptsLabel.TabIndex = 4;

this.attemptsLabel.Text = "Попыток: 0";

// rangeLabel

this.rangeLabel.AutoSize = true;

this.rangeLabel.Location = new System.Drawing.Point(30, 150);

this.rangeLabel.Name = "rangeLabel";

this.rangeLabel.Size = new System.Drawing.Size(95, 13);

this.rangeLabel.TabIndex = 5;

this.rangeLabel.Text = "Диапазон: 1-100";

// historyLabel

this.historyLabel.AutoSize = true;

this.historyLabel.Location = new System.Drawing.Point(30, 180);

this.historyLabel.Name = "historyLabel";

this.historyLabel.Size = new System.Drawing.Size(93, 13);

this.historyLabel.TabIndex = 6;

this.historyLabel.Text = "История попыток";

// historyListBox

this.historyListBox.FormattingEnabled = true;

this.historyListBox.Location = new System.Drawing.Point(30, 200);

this.historyListBox.Name = "historyListBox";

this.historyListBox.Size = new System.Drawing.Size(220, 95);

this.historyListBox.TabIndex = 7;

// newGameButton

this.newGameButton.Location = new System.Drawing.Point(30, 310);

this.newGameButton.Name = "newGameButton";

this.newGameButton.Size = new System.Drawing.Size(220, 30);

this.newGameButton.TabIndex = 8;

this.newGameButton.Text = "Новая игра";

this.newGameButton.UseVisualStyleBackColor = true;

this.newGameButton.Click += new System.EventHandler(this.newGameButton\_Click);

// MainForm

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(284, 361);

this.Controls.Add(this.newGameButton);

this.Controls.Add(this.historyListBox);

this.Controls.Add(this.historyLabel);

this.Controls.Add(this.rangeLabel);

this.Controls.Add(this.attemptsLabel);

this.Controls.Add(this.statusLabel);

this.Controls.Add(this.guessButton);

this.Controls.Add(this.guessTextBox);

this.Controls.Add(this.titleLabel);

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;

this.MaximizeBox = false;

this.Name = "MainForm";

this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;

this.Text = "Угадай число";

this.ResumeLayout(false);

this.PerformLayout();

}

private System.Windows.Forms.Label titleLabel;

private System.Windows.Forms.TextBox guessTextBox;

private System.Windows.Forms.Button guessButton;

private System.Windows.Forms.Label statusLabel;

private System.Windows.Forms.Label attemptsLabel;

private System.Windows.Forms.Label rangeLabel;

private System.Windows.Forms.Label historyLabel;

private System.Windows.Forms.ListBox historyListBox;

private System.Windows.Forms.Button newGameButton;

}

}

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»
      9. Введение в Git. <https://ru.hexlet.io/courses/intro_to_git>
      10. Основы Git. <https://githowto.com/ru/setup>
      11. Обучающее приложение по использованию Git <https://learngitbranching.js.org/?locale=ru_RU>
      12. Git и GitHub Курс Для Новичков. <https://www.youtube.com/watch?v=zZBiln_2FhM>

# 