Федеральное государственное бюджетное образовательное учреждение высшего образования
Национальный исследовательский университет «Московский энергетический институт»
Кафедра ВМСС

Лабораторная работа №8

«Объектно-ориентированное программирование на языке C#»

Курс: Технологии разработки программного обеспечения

Группа: А-07м-23

Выполнил: Кретов Н.В.

Проверила: Раскатова М.В.

Москва 2024

СОДЕРЖАНИЕ

1. Цель

Главной задачей данной лабораторной работы является закрепление навыков объектно-ориентированного программирования и разработки многослойной архитектуры проекта, а также ознакомление с механизмом делегатов и их практическое применение.

2. Задание

Реализовать на языке C# приложение с графическим интерфейсом и многоуровневой архитектурой ("глубина" иерархии - не менее 4, "ширина" - не менее 3).

При разработке программы необходимо показать применение делегатов и механизма интерфейсов. Один из классов должен реализовывать как минимум два интерфейса.

Реализовать обработку одного из системных исключений: StackOverflowException, ArrayTypeMismatchException, DivideByZeroException, IndexOutOfRangeException, InvalidCastException, OutOfMemoryException, OverflowException.

Показать пример использования полиморфизма.

3. Описание предметной области

В качестве предметной области была выбрана имитация боевой системы для RPG-игры.

Пользователю должен быть доступен следующий функционал:

- Атака одним из доступных персонажей по выбранному противнику;
- Применение одним из доступных персонажей особого навыка;
- Выбор одного из доступных персонажей для отдыха.

Урон, наносимый персонажем при атаке и применении уникальной способности, а также их стоимость должны зависеть от класса персонажа (волшебник при использовании уникальной способности лечит всех персонажей игрока).

Также должно быть реализовано чтение данных из файла при запуске программы.

Логирования всех происходящих действий в программе (например, создание объектов классов, их удаление, вызов методов и пр.) должно происходить в текстовый файл и в специальную текстовую панель, доступную на главной форме приложения.

4. Разработка

Для удобства весь разработки код был разделен на области (#region) - Enums_and_Structures (куда вошли структура CharacterClass и перечисление CharacterData), Interfaces (здесь находятся три необходимых нам интерфейса – IAbilitable, IAttackable и IRestable), Hierarchy (сюда вошли абстрактные классы Entity, Monster и Human, наследники Human – Archer, Knight и Wizard, а также наследники Monster – Wither и Goblin), Singleton (в это регионе с помощью одноименного паттерна был реализован логгер), а также пространство внутри MainForm также было разделено на области – Variables, Constructors, Handler (вспомогательные функции – CreateCharacter, ReadDataFromCSV, View и MainForm_FormClosed), XxxxButton_Click (обработчики нажатия кнопок Attack, Ability и Rest).

Отдельно был объявлен делегат EnemyDeadDelegate без аргументов и не возвращающий данных, на основе которого было объявлено событие (event) _enemyDeadEvent, вызываемое из главной формы при смерти одного из противников, провоцируя перерисовку пользовательского интерфейса.

Свойство полиморфизма проявляется в классах Archer, Knight и Wizard, а также Wither и Goblin. Данные классы будут помещены в списки, экземпляры класса Entity (_playersTeam и _enemyTeam). Одинаково названные методы Attack, Ability и Reat, переопределенные в этих классах будут вызываться при нажатии соответствующих классов, путем поиска в playersTeam и _enemyTeam атакующего персонажа и цели для атаки соответственно (upcast).

5. Описание классов

Ниже представлено описание полей и методов классов MainForm, который наследуется от стандартного класса Form (пространство имен System.Windows.Forms), Entity, реализующего три интерфейса (IAttackable, IAbilitable, IRestable), Monster, Human, которые наследуются от абстрактного класса Entity, Wither, Goblin, которые наследуются от абстрактного класса Monster, Archer, Knight и Wizard, которые наследуются от абстрактного класса Human.

MainForm : Form

Поля:

- private static string _pathForRead – доступное лишь в MainForm статическое поле, хранящее строку-путь до файла, предназначенного для чтения при запуске игры;
- public static string _pathForWrite – общедоступное статическое поле, хранящее строку-путь до файла, предназначенного для записи логов;
- private static uint _curentID – доступное лишь в MainForm статическое поле, хранящее текущее незанятое ID;
- public static List<Entity> _playersTeam = new List<Entity>() – общедоступное статическое поле, хранящее список екземпляров класса Entity, являющихся управляемыми персонажами;
- public static List<Entity> _enemyTeam = new List<Entity>() – общедоступное статическое поле, хранящее список екземпляров класса Entity, являющихся персонажами-противниками;
- public static Logger _logger – общедоступное статическое поле, хранящее экземпляр класса Logger;
- private static DataGridView _playersTable – доступное лишь в MainForm статическое поле, хранящее список управляемых персонажей;
- private static DataGridView _enemyTable – доступное лишь в MainForm статическое поле, хранящее список персонажей-противников;
- public static RichTextBox _richTextBox – общедоступное статическое поле, хранящее экземпляр класса RichTextBox;
- public static CharacterData characterData – общедоступное статическое поле, хранящее экземпляр перечисленя CharacterData;
- private event EnemyDeadDelegate _enemyDeadEvent – доступное лишь в MainForm поле, хранящее экземпляр события EnemyDeadDelegate.

Методы:

- public MainForm() – общедоступный конструктор класса MainForm;

- private static Entity CreateCharacter(uint id) – доступный лишь в классе MainForm метод, отвечающий за создание персонажей и противников в соответствии со считанным файлом при запуске игры;
- private static void ReadDataFromCSV() – доступный лишь в классе MainForm метод, отвечающий за считывание данных из файла при запуске игры;
- private static void View() – доступный лишь в классе MainForm метод, отвечающий за перерисовку пользовательского интерфейса при возникновении изменении в имеющихся данных о персонажах и противниках;
- private void MainForm_FormClosed(object sender, FormClosedEventArgs e) – метод, вызываемый при закрытии MainForm;
- private void AttackButton_Click(object sender, EventArgs e) – метод, являющийся обработчиков нажатися пользователем кнопки «Attack»;
- private void AbilityButton_Click(object sender, EventArgs e) – метод, являющийся обработчиков нажатися пользователем кнопки «Ability»;
- private void RestButton_Click(object sender, EventArgs e) – метод, являющийся обработчиков нажатися пользователем кнопки «Rest».

Entity : IAttackable, IAbilitable, IRestable

Свойства:

- public uint _id { get; private set; } – общедоступное свойство с приватным сеттером, хранящее уникальный номер персонажа;
- public int _health { get; set; } – общедоступное свойство, хранящее здоровье персонажа;
- public int _stamina { get; protected set; } – общедоступное свойство с защищенным сеттером, хранящее текущий запас сил персонажа;
- public int _specialData { get; protected set; } – общедоступное свойство с защищенным сеттером, хранящее уникальные для персонажа данные (для лучника – количество стрел, для волшебника – манна и т.д.);
- public CharacterClass _characterClass { get; protected set; } – общедоступное свойство с защищенным сеттером, хранящее класс персонажа.

Поля:

- protected int _attackStaminaCost – защищенное поле, хранящее стоимость атаки в единицах силы для персонажа;
- protected int _attackSpecialDataCost – защищенное поле, хранящее стоимость атаки в единицах уникальных для персонажа данных (для лучника – количество стрел, для волшебника – манна и т.д.);
- protected int _abilityStaminaCost – защищенное поле, хранящее стоимость уникальной способности в единицах силы для персонажа;
- protected int _abilitySpecialDataCost – защищенное поле, хранящее стоимость уникальной способности в единицах уникальных для персонажа данных (для лучника – количество стрел, для волшебника – манна и т.д.);
- protected int _attackdamage – защищенное поле, хранящее наносимый персонажем урон при атаке;
- protected int _abilitydamage – защищенное поле, хранящее наносимый персонажем урон при использовании уникальной способности (для волшебника отвечает за количество восстанавливаемого у персонажей здоровья).

Методы:

- public Entity(uint id, int health, int stamina) – общедоступный конструктор класса Entity;
- public virtual void Attack(int indexEnemy) – общедоступный виртуальны метод, по умолчанию отвечающий нанесение персонажем урона по выбранному противнику, индекс которого передается в качестве параметра;
- public virtual void Ability() – общедоступный виртуальны метод, по умолчанию отвечающий нанесение персонажем урона по всем противникам;
- public virtual void Rest() – общедоступный виртуальны метод, по умолчанию отвечающий за восстановление запаса сил у персонажа.

Human : Entity

Свойства:

- public string _name { get; private set; } – общедоступное свойство с приватным сеттером, хранящее уникальные для персонажей-людей имена.

Методы:

- public Human(uint id, int health, int stamina, string name) : base(id, health, stamina) – общедоступный конструктор класса Human.

Monster : Entity

Методы:

- public Monster(uint id, int health, int stamina) : base(id, health, stamina) – общедоступный конструктор класса Monster.

Archer : Human

Методы:

- public Archer(uint id, int health, int stamina, string name, int arrowCount) : base(id, health, stamina, name) – общедоступный конструктор класса Archer;
- public override void Rest() – общедоступный переопределенный метод Rest.

Knight : Human

Методы:

- public Knight(uint id, int health, int stamina, string name, int specialData) : base(id, health, stamina, name) – общедоступный конструктор класса Knight.

Wizard : Human

Методы:

- public Wizard(uint id, int health, int stamina, string name, int mannaCount) : base(id, health, stamina, name) – общедоступный конструктор класса Wizard;
- public override void Ability() – общедоступный переопределенный метод Ability;
- public override void Rest() – общедоступный переопределенный метод Rest.

Witcher : Monster

Методы:

- public Witcher(uint id, int health, int stamina, int mannaCount) : base(id, health, stamina) – общедоступный конструктор класса Witcher;
- public override void Rest() – общедоступный переопределенный метод Rest.

Goblin : Monster

Методы:

- public Goblin(uint id, int health, int stamina, int specialData) : base(id, health, stamina) – общедоступный конструктор класса Goblin.

Logger

Поля:

- private static Logger _instance – доступное лишь в Logger статическое поле, хранящее единственный возможный экземпляр класса Logger.

Методы:

- private Logger() – доступный лишь в классе Logger конструктор класса Logger;
- public static Logger GetInstance() – общедоступный метод, возвращающий ссылку на единственный возможный экземпляр класса Logger;
- public void WriteToLogFile(string message, string pathForWrite) – общедоступный метод, записывающий принимаемое в качестве параметра сообщение по указанному в качестве второго параметра адресу;
- public void WriteToLogPanel(string message) – общедоступный метод, записывающий принимаемое в качестве параметра сообщение в специальную текстовую панель пользовательского интерфейса.

6. Диаграмма классов

На рис. 1 представлена диаграмма классов.

Рис. 1. Диаграмма классов

Как видно из диаграммы, программа представляет из себя одну форму (главное окно программы), которая наследуются от стандартного класса Form (пространство имен System.Windows.Forms), и многослойной иерархии наследуемых классов.

При этом имеется три интерфейса (IAttackable, IAbilitable, IRestable), которые реализуются абстрактным классом Entity (с помощью виртуальных методов Attack, Ability, Rest), от которого наследуются два также абстрактных класса – Monster и Human (Human имеет присущее только ему и его наследникам свойство _name). От абстрактного класса Monster наследуются 2 класса – Wither, переопределяющий виртуальный метод Rest, и Goblin. От абстрактного класса Human наследуются 3 класса – Archer, переопределяющий виртуальный метод Rest, Knight и Wizard, переопределяющий виртуальные методы Ability и Rest.

За логирование в файл и в текстовую панель отвечает класс Logger (реализован с помощью паттерна Singleton).

Также видно наличие структуры (struct) CharacterData, хранящей данные о персонаже, передаваемые в форму из считываемого файла и перечисления (enums) CharacterClass, для удобной работы с классом персонажа.

В главной форме реализовано событие на основе делегата, также представленного на диаграмме.

7. Результат работы программы

На рис. 2-10 представлены результаты работы программы.

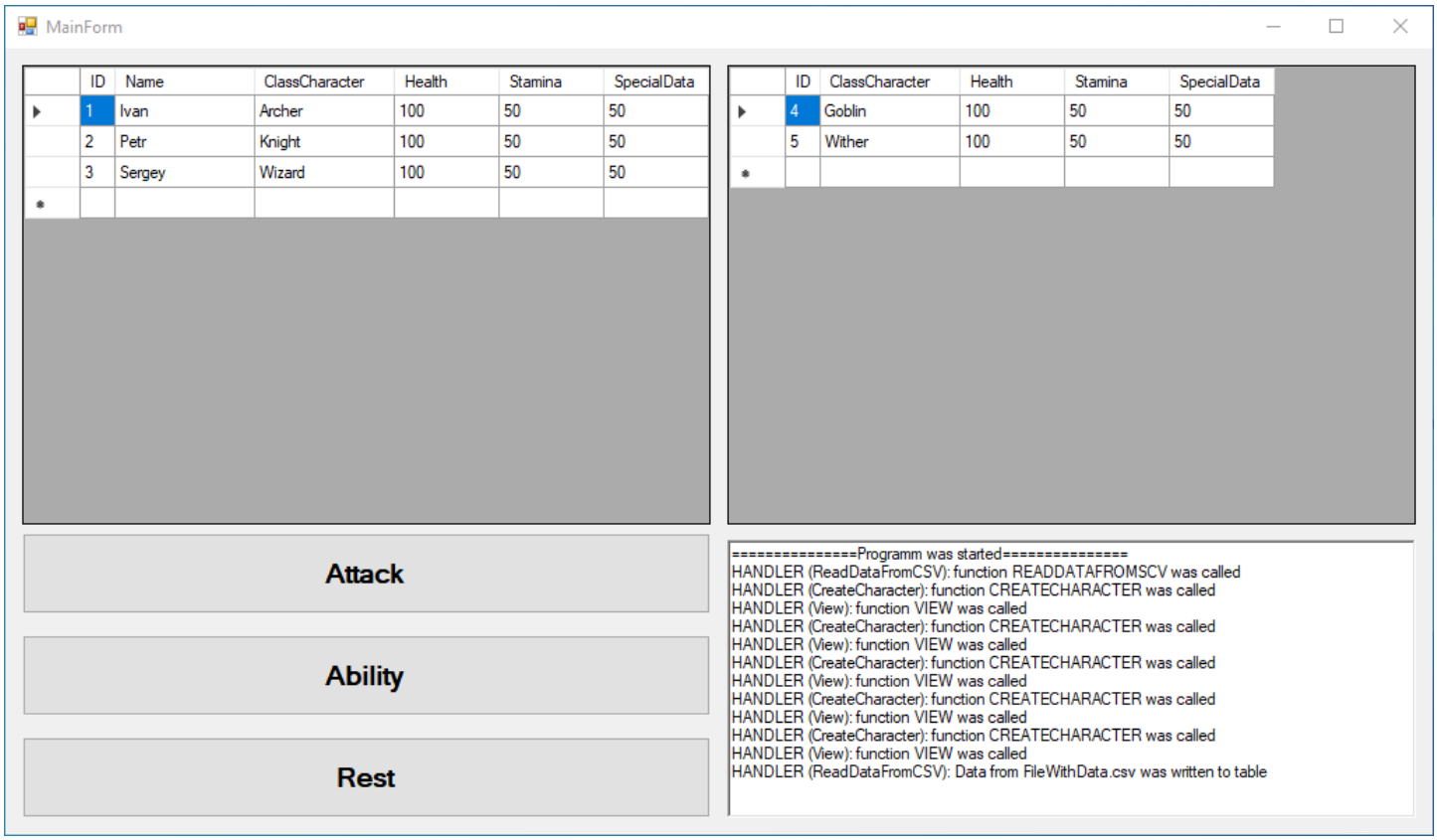

Рис. 2. Запуск программы

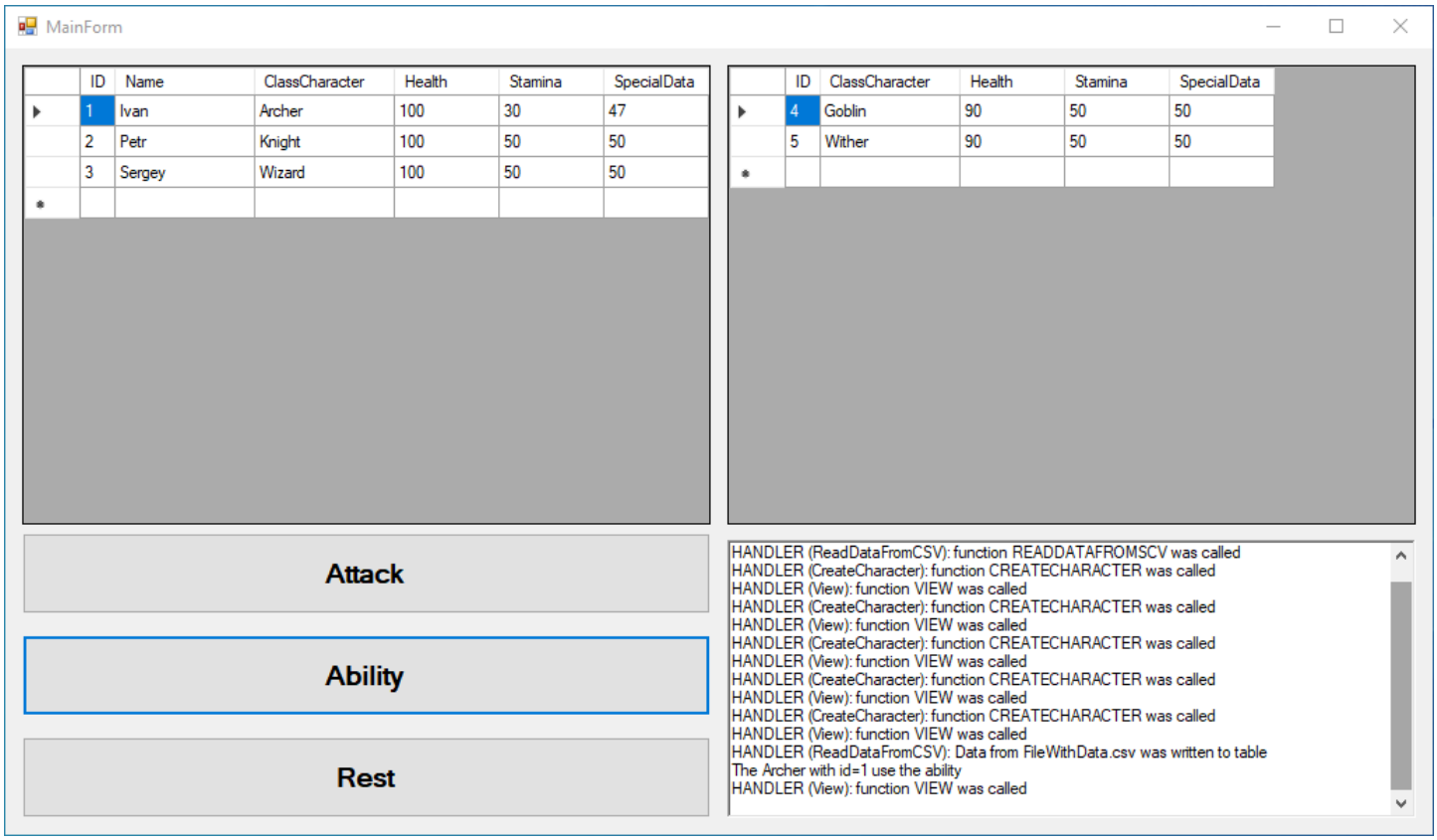

Рис. 3. Результат применения способности лучника Ивана

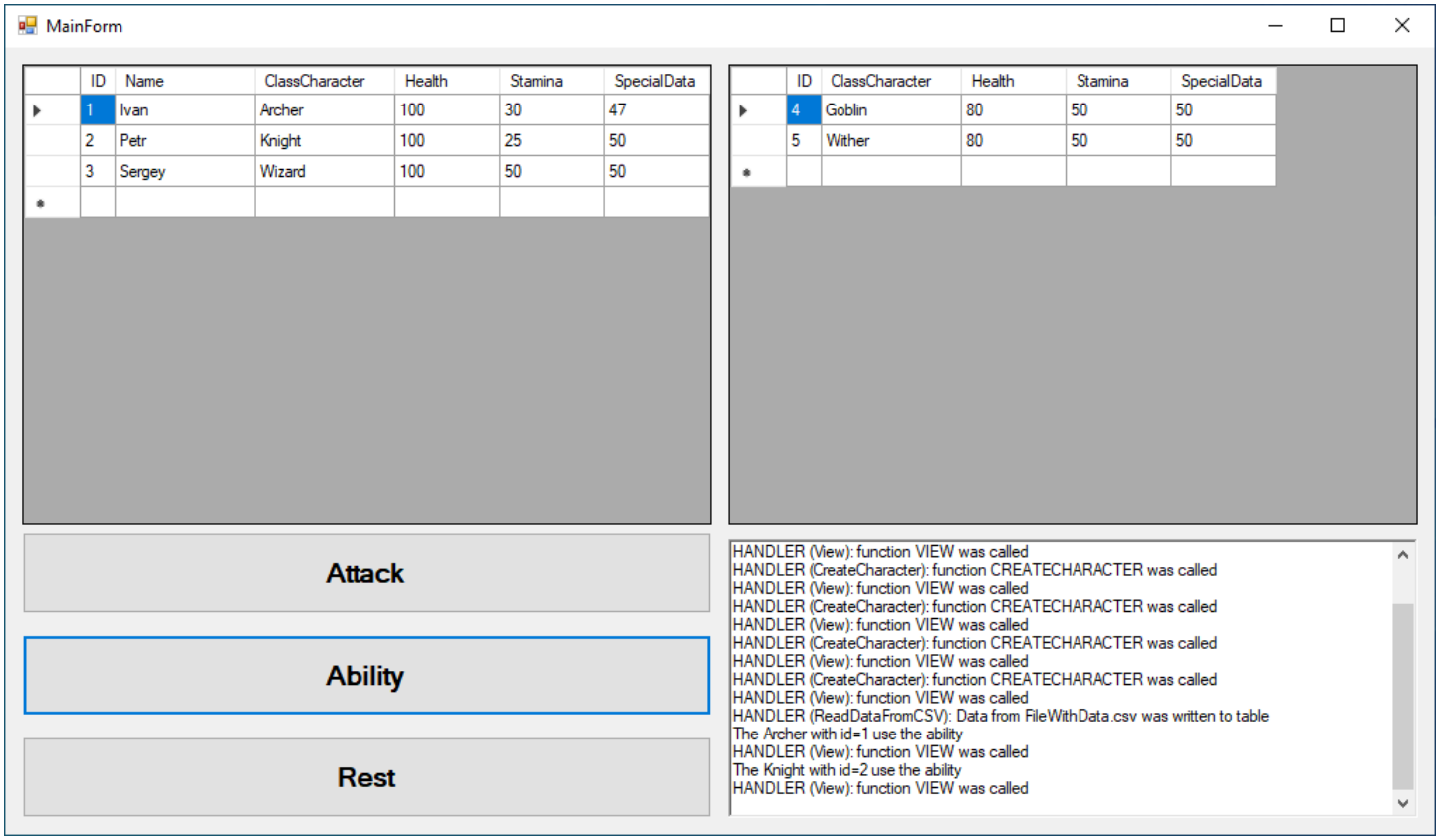

Рис. 4. Результат применения способности рыцаря Петра

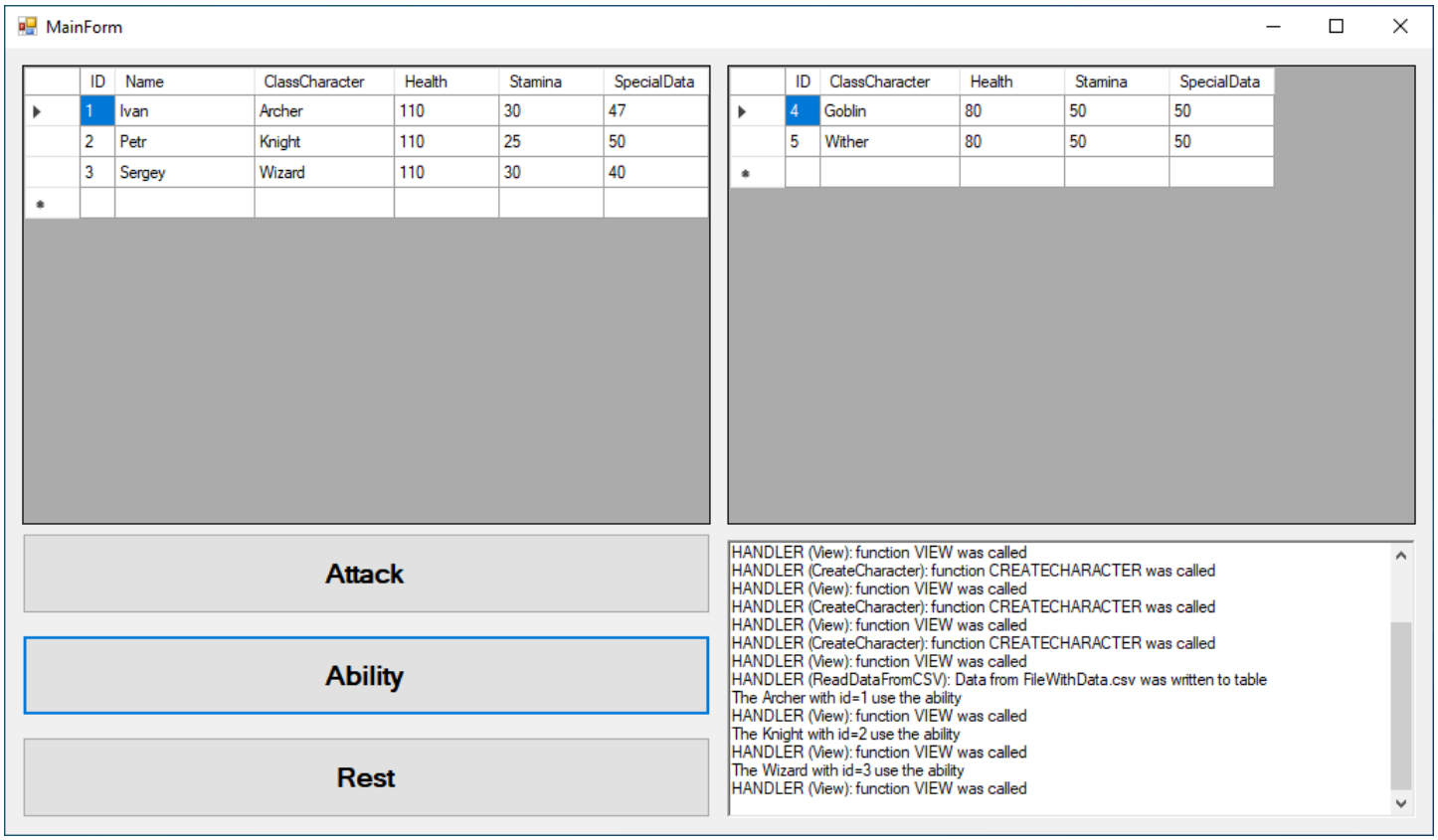

Рис. 5. Результат применения способности волшебника Сергея

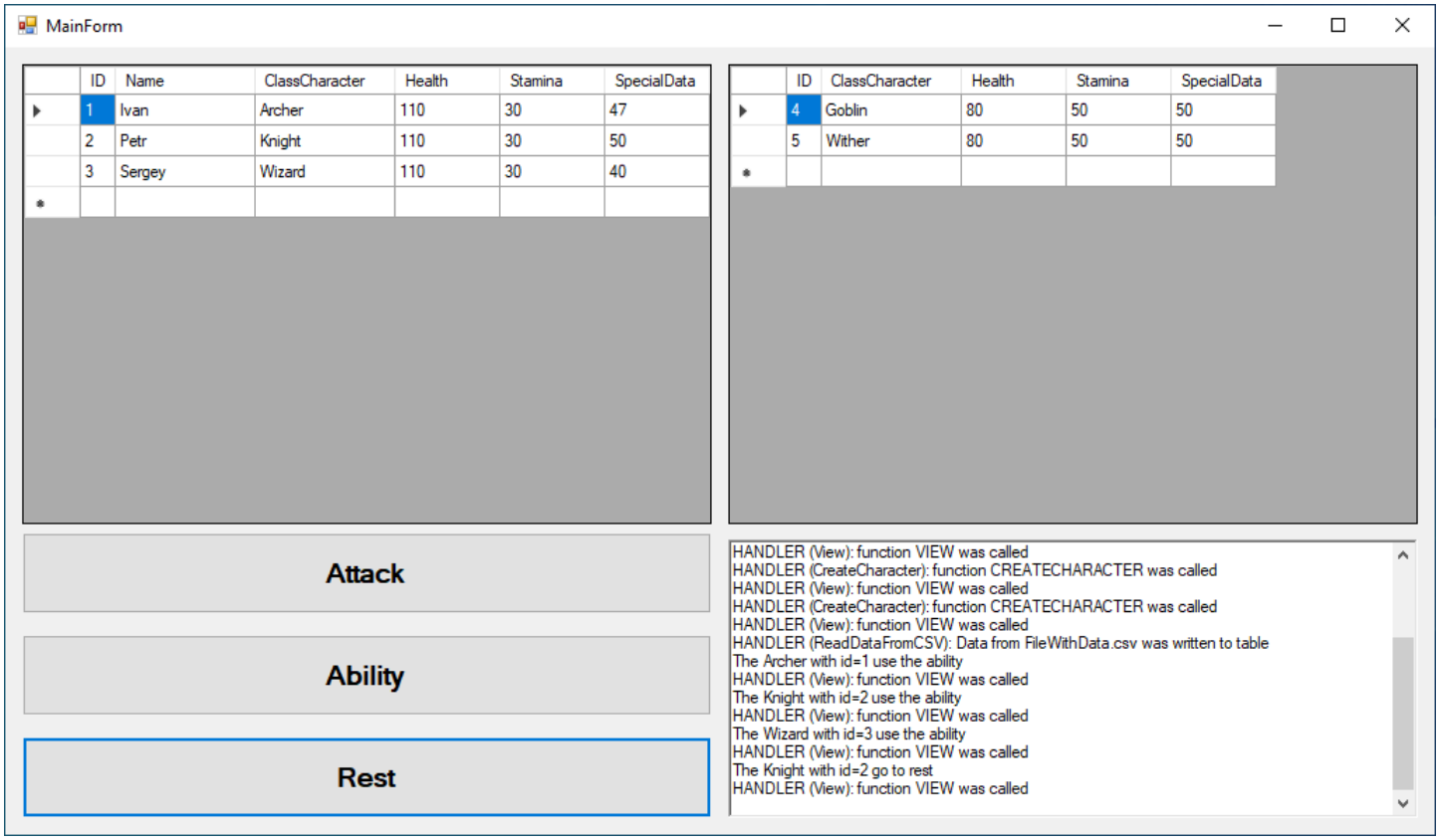

Рис. 6. Результат отдыха рыцаря Петра

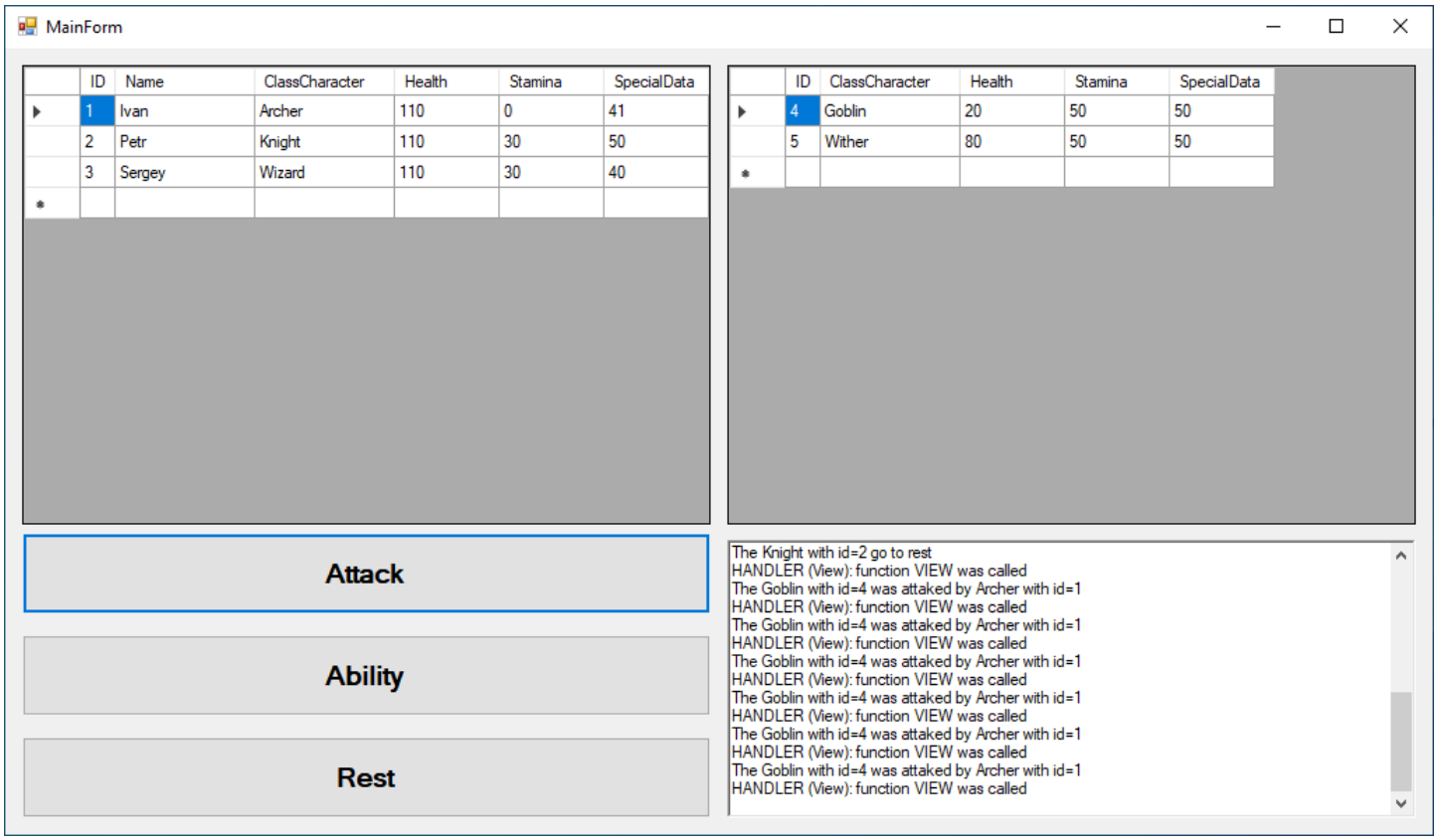

Рис. 7. Результат нескольких атак лучника Ивана

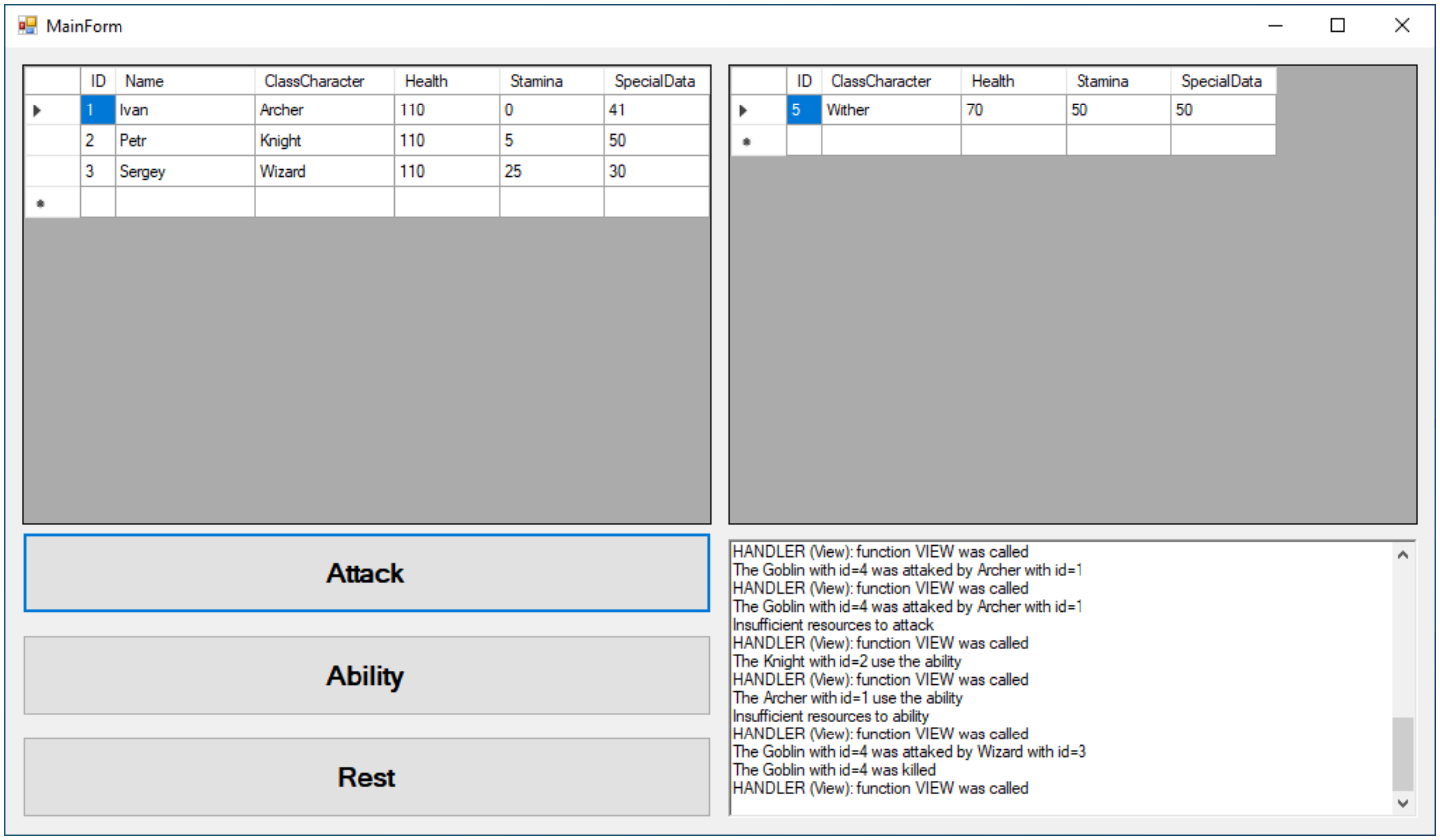

Рис. 8. Смерть противника после нескольких атак

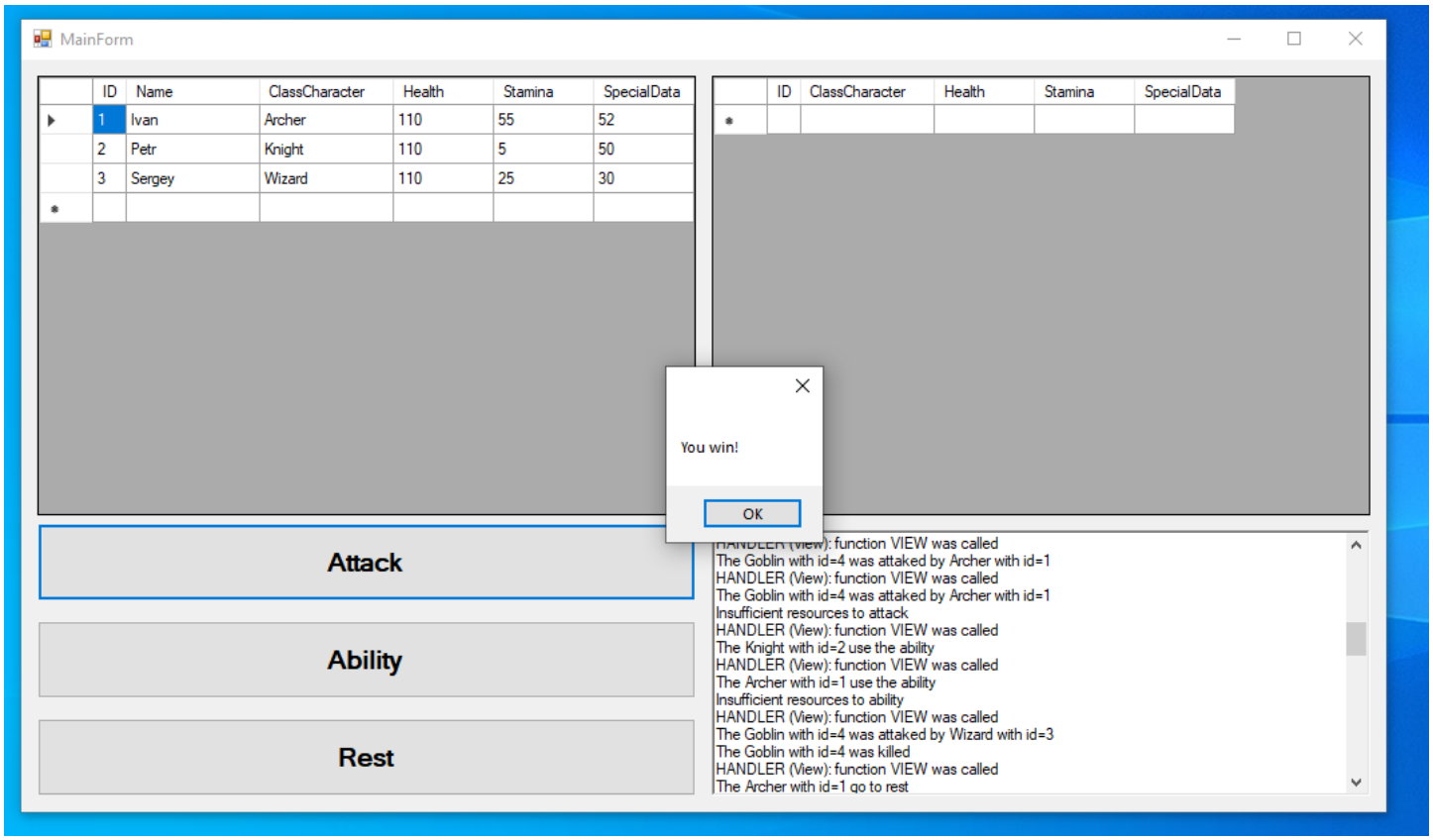

Рис. 9. Победа после смерти всех противников

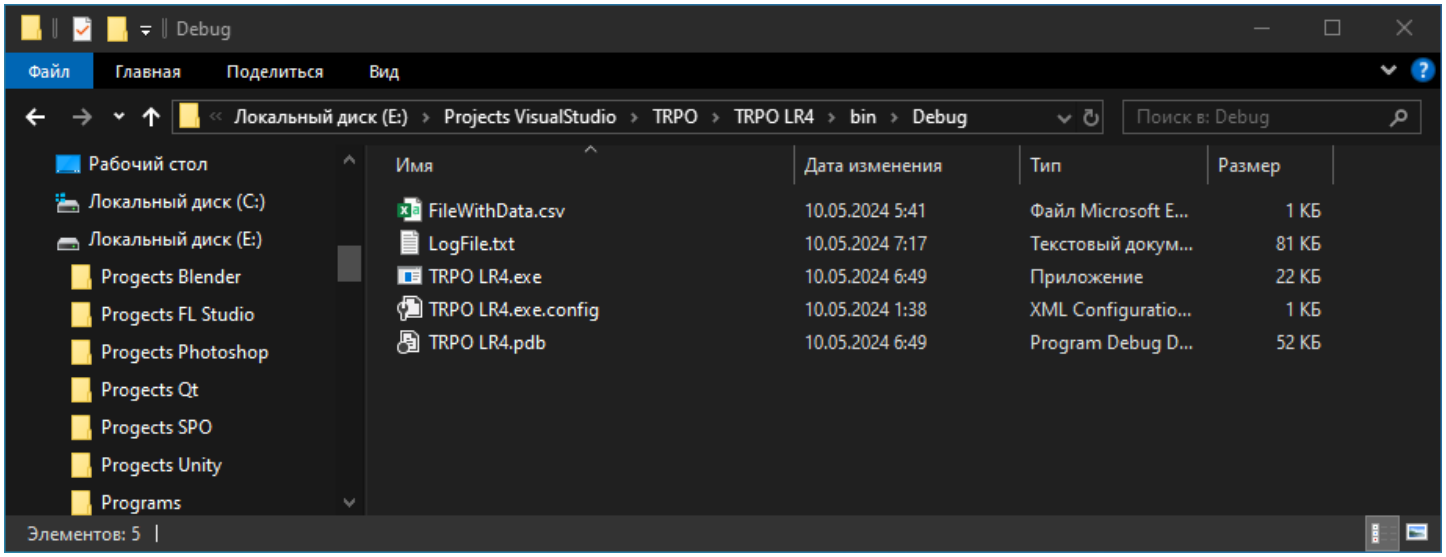

Рис. 10. Репозиторий проекта

Как видно из рис. 10, в репозитории проекта находится файлы FileWithData.csv (файл, содержащий записи, считываемые при запуске программы) и LogFile.txt (текстовый файл с логами).

Пример записи логов в текстовый файл представлен на рис. 11.

```
HANDLER (View): function VIEW was called
The Archer with id=1 use the ability
HANDLER (View): function VIEW was called
The Archer with id=1 use the ability
==============Programm was closed==============
==============Programm was started==============
HANDLER (ReadDataFromCSV): function READDATAFROMSCV was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (ReadDataFromCSV): Data from FileWithData.csv was written to table
==============Programm was started==============
HANDLER (ReadDataFromCSV): function READDATAFROMSCV was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
HANDLER (View): function VIEW was called
HANDLER (CreateCharacter): function CREATECHARACTER was called
```

Рис. 11. Пример логирования

С полным листингом программы можно ознакомиться в приложении А.

ВЫВОДЫ

В результаты выполнения лабораторной работы №8 было разработано приложение, имитирующее боевую систему для RPG-игры, с многоуровневой архитектурой, соответствующей поставленным требованиям ("глубина" иерархии - не менее 4, "ширина" - не менее 3), и пользовательским графическим интерфейсом. Был реализован механизм интерфейсов, при этом абстрактный класс Entity реализует более двух интерфейсов, что соответствует поставленной задаче. Данное приложение может быть в дальнейшем дополнено и/или применено в рамках более сложной системы. При этом в разработке были применен паттерн проектирования Singleton.

Было реализовано чтение данных из файла формата scv. Решена задача логирования всех происходящих действий в программе в файл формата txt и текстовую панель главного окна программы.

В ходе выполнения лабораторной работы №8 было произведено знакомство с механизмом делегатов и событий.

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace TRPO_LR4
{
    #region Enums_and_Structures
    public enum CharacterClass
    {
        Archer = 1,
        Knight = 2,
        Wizard = 3,
        Goblin = 4,
        Wither = 5
    }

    public struct CharacterData
    {
        public string name;
        public int health;
        public int stamina;
        public int specialData;
        public CharacterClass characterClass;
    }
    #endregion

    public partial class MainForm : Form
    {
        #region Variables
        private static string _pathForRead;
        public static string _pathForWrite;
        private static uint _curentID;

        public static List<Entity> _playersTeam = new List<Entity>();
        public static List<Entity> _enemyTeam = new List<Entity>();

        public static Logger _logger;
        private static DataGridView _playersTable;
        private static DataGridView _enemyTable;
        public static RichTextBox _richTextBox;

        public static CharacterData characterData;
        #endregion

        #region Constructors
        public MainForm()
        {
            InitializeComponent();

            _curentID = 1;
            _pathForRead = "FileWithData.csv";
            _pathForWrite = "LogFile.txt";

            _richTextBox = richTextBoxLog;

            _logger = Logger.GetInstance();
            _logger.WriteToLogFile("==============Programm was started==============", _pathForWrite);
            _logger.WriteToLogPanel("==============Programm was started==============");

            characterData = new CharacterData();
            _playersTable = PlayerTable;
            _enemyTable = EnemyTable;

            ReadDataFromCSV();
```

```
        }
        #endregion

        #region Handler
        private static Entity CreateCharacter(uint id)
        {
            _logger.WriteToLogFile("HANDLER (CreateCharacter): function CREATECHARACTER was called",
_pathForWrite);
            _logger.WriteToLogPanel("HANDLER (CreateCharacter): function CREATECHARACTER was called");
            switch (characterData.characterClass)
            {
                case CharacterClass.Archer:
                    return new Archer(id, characterData.health, characterData.stamina, characterData.name,
characterData.specialData);
                case CharacterClass.Knight:
                    return new Knight(id, characterData.health, characterData.stamina, characterData.name,
characterData.specialData);
                case CharacterClass.Wizard:
                    return new Wizard(id, characterData.health, characterData.stamina, characterData.name,
characterData.specialData);
                case CharacterClass.Goblin:
                    return new Goblin(id, characterData.health, characterData.stamina,
characterData.specialData);
                case CharacterClass.Wither:
                    return new Witcher(id, characterData.health, characterData.stamina,
characterData.specialData);
                default:
                    _curentID--;
                    _logger.WriteToLogFile("HANDLER (CreateCharacter): character was not added",
_pathForWrite);
                    _logger.WriteToLogPanel("HANDLER(CreateCharacter): character was not added");
                    return null;
            }
        }

        private static void ReadDataFromCSV()
        {
            _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): function READDATAFROMSCV was called",
_pathForWrite);
            _logger.WriteToLogPanel("HANDLER (ReadDataFromCSV): function READDATAFROMSCV was called");
            try
            {
                using (StreamReader reader = new StreamReader(_pathForRead, Encoding.UTF8))
                {
                    string record;

                    while (!reader.EndOfStream)
                    {
                        record = reader.ReadLine();

                        string[] data = record.Split(new char[] { ',', '.' },
StringSplitOptions.RemoveEmptyEntries);

                        if (!int.TryParse(data[1], out int characterClass) || !int.TryParse(data[2], out int
health) || !int.TryParse(data[3], out int stamina) || !int.TryParse(data[3], out int specialData))
                        {
                            continue;
                        }

                        characterData.name = data[0];
                        characterData.characterClass = (CharacterClass)characterClass;
                        characterData.health = health;
                        characterData.stamina = stamina;
                        characterData.specialData = specialData;

                        if (characterData.characterClass == CharacterClass.Archer ||
characterData.characterClass == CharacterClass.Knight || characterData.characterClass ==
CharacterClass.Wizard)
                        {
```

```
                        _playersTeam.Add(CreateCharacter(_curentID));
                        _curentID++;
                    }
                    else if (characterData.characterClass == CharacterClass.Goblin ||
characterData.characterClass == CharacterClass.Wither)
                    {
                        _enemyTeam.Add(CreateCharacter(_curentID));
                        _curentID++;
                    }
                    View();
                }
                _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): Data from FileWithData.csv was
written to table", _pathForWrite);
                _logger.WriteToLogPanel("HANDLER (ReadDataFromCSV): Data from FileWithData.csv was
written to table");
            }
        }
        catch
        {
            _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): FileWithData.csv was not read or was
empty", _pathForWrite);
            _logger.WriteToLogPanel("HANDLER (ReadDataFromCSV): FileWithData.csv was not read or was
empty");
            return;
        }
    }

    private static void View()
    {
        _logger.WriteToLogFile("HANDLER (View): function VIEW was called", _pathForWrite);
        _logger.WriteToLogPanel("HANDLER (View): function VIEW was called");

        _playersTable.Rows.Clear();

        foreach (Human human in _playersTeam)
        {
            _playersTable.Rows.Add(human._id, human._name, human._characterClass, human._health,
human._stamina, human._specialData);

        }

        _enemyTable.Rows.Clear();

        foreach (Monster monster in _enemyTeam)
        {
            _enemyTable.Rows.Add(monster._id, monster._characterClass, monster._health,
monster._stamina, monster._specialData);
        }
    }
    private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
    {
        _logger.WriteToLogFile("==============Programm was closed==============", _pathForWrite);
        _logger.WriteToLogPanel("==============Programm was closed==============");
    }
    #endregion
    #region XxxxButton_Click
    private void AttackButton_Click(object sender, EventArgs e)
    {
        int indexCharacter;
        int indexEnemy;

        try
        {
            if (_playersTable.CurrentCell == null)
            {
                _logger.WriteToLogFile("There is no selected character in the table to attack enemy",
_pathForWrite);
                _logger.WriteToLogPanel("There is no selected character in the table to attack enemy");
                throw new Exception("There is no selected character in the table to attack enemy");
```

```
                }
                else if (_enemyTable.CurrentCell == null)
                {
                    _logger.WriteToLogFile("There is no selected enemy in the table to attack him",
_pathForWrite);
                    _logger.WriteToLogPanel("There is no selected enemy in the table to attack him");
                    throw new Exception("There is no selected enemy in the table to attack him");
                }
                else
                {
                    indexCharacter = _playersTable.CurrentCell.RowIndex;
                    indexEnemy = _enemyTable.CurrentCell.RowIndex;
                }

                if (indexCharacter != -1 && _playersTeam.Count > indexCharacter)
                {
                    _logger.WriteToLogFile($"The {_enemyTeam[indexEnemy]._characterClass.ToString()} with
id={_enemyTeam[indexEnemy]._id} was attaked by {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id}", _pathForWrite);
                    _logger.WriteToLogPanel($"The {_enemyTeam[indexEnemy]._characterClass.ToString()} with
id={_enemyTeam[indexEnemy]._id} was attaked by {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id}");
                    _playersTeam[indexCharacter].Attack(indexEnemy);

                    if (_enemyTeam[indexEnemy]._health <= 0)
                    {
                        _logger.WriteToLogFile($"The {_enemyTeam[indexEnemy]._characterClass.ToString()}
with id={_enemyTeam[indexEnemy]._id} was killed", _pathForWrite);
                        _logger.WriteToLogPanel($"The {_enemyTeam[indexEnemy]._characterClass.ToString()}
with id={_enemyTeam[indexEnemy]._id} was killed");
                        _enemyTeam.RemoveAt(indexEnemy);
                    }

                    View();

                    if (_enemyTeam.Count == 0)
                    {
                        MessageBox.Show("You win!");
                    }
                }
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
        }

        private void AbilityButton_Click(object sender, EventArgs e)
        {
            int indexCharacter;

            try
            {
                if (_playersTable.CurrentCell == null)
                {
                    _logger.WriteToLogFile("There is no selected character in the table to ability",
_pathForWrite);
                    _logger.WriteToLogPanel("There is no selected character in the table to ability");
                    throw new Exception("There is no selected character in the table to ability");
                }
                else
                {
                    indexCharacter = _playersTable.CurrentCell.RowIndex;
                }

                if (indexCharacter != -1 && _playersTeam.Count > indexCharacter)
                {
                    _logger.WriteToLogFile($"The {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id} use the ability", _pathForWrite);
```

```
                    _logger.WriteToLogPanel($"The {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id} use the ability");
                    _playersTeam[indexCharacter].Ability();

                    _enemyTeam.RemoveAll(x => x._health <= 0);

                    View();

                    if (_enemyTeam.Count == 0)
                    {
                        MessageBox.Show("You win!");
                    }
                }
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
        }

        private void RestButton_Click(object sender, EventArgs e)
        {
            int indexCharacter;

            try
            {
                if (_playersTable.CurrentCell == null)
                {
                    _logger.WriteToLogFile("There is no selected character in the table to rest",
_pathForWrite);
                    _logger.WriteToLogPanel("There is no selected character in the table to rest");
                    throw new Exception("There is no selected character in the table to rest");
                }
                else
                {
                    indexCharacter = _playersTable.CurrentCell.RowIndex;
                }

                if (indexCharacter != -1 && _playersTeam.Count > indexCharacter)
                {
                    _logger.WriteToLogFile($"The {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id} go to rest", _pathForWrite);
                    _logger.WriteToLogPanel($"The {_playersTeam[indexCharacter]._characterClass.ToString()}
with id={_playersTeam[indexCharacter]._id} go to rest");
                    _playersTeam[indexCharacter].Rest();

                    View();
                }
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
        }
        #endregion
    }

    #region Interfaces
    public interface IAbilitable
    {
        void Ability();
    }

    public interface IAttackable
    {
        void Attack(int indexEnemy);
    }

    public interface IRestable
```

```
    {
        void Rest();
    }
    #endregion

    #region Hierarchy
    public abstract class Entity : IAttackable, IAbilitable, IRestable
    {
        public uint _id { get; private set; }
        public int _health { get; set; }
        public int _stamina { get; protected set; }
        public int _specialData { get; protected set; }
        public CharacterClass _characterClass { get; protected set; }

        protected int _attackStaminaCost;
        protected int _attackSpecialDataCost;
        protected int _abilityStaminaCost;
        protected int _abilitySpecialDataCost;
        protected int _attackdamage;
        protected int _abilitydamage;

        public Entity(uint id, int health, int stamina)
        {
            _id = id;
            _health = health;
            _stamina = stamina;
        }

        public virtual void Attack(int indexEnemy)
        {
            if (_specialData >= _attackSpecialDataCost && _stamina >= _attackStaminaCost)
            {
                _specialData -= _attackSpecialDataCost;
                _stamina -= _attackStaminaCost;
                MainForm._enemyTeam[indexEnemy]._health -= _attackdamage;
            }
            else
            {
                MainForm._logger.WriteToLogFile("Insufficient resources to attack", MainForm._pathForWrite);
                MainForm._logger.WriteToLogPanel("Insufficient resources to attack");
            }
        }

        public virtual void Ability()
        {
            if (_specialData >= _abilitySpecialDataCost && _stamina >= _abilityStaminaCost)
            {
                _specialData -= _abilitySpecialDataCost;
                _stamina -= _abilityStaminaCost;

                foreach (Monster monster in MainForm._enemyTeam)
                {
                    monster._health -= _abilitydamage;
                }
            }
            else
            {
                MainForm._logger.WriteToLogFile("Insufficient resources to ability",
MainForm._pathForWrite);
                MainForm._logger.WriteToLogPanel("Insufficient resources to ability");
            }
        }

        public virtual void Rest()
        {
            _stamina += 5;
        }
    }
```

```
    public abstract class Human : Entity
    {
        public string _name { get; private set; }

        public Human(uint id, int health, int stamina, string name) : base(id, health, stamina)
        {
            _name = name;
        }
    }
public class Archer : Human
 {
     public Archer(uint id, int health, int stamina, string name, int arrowCount) : base(id, health,
stamina, name)
     {
         _attackSpecialDataCost = 1;
         _attackStaminaCost = 5;

         _abilitySpecialDataCost = 3;
         _abilityStaminaCost = 20;

         _attackdamage = 10;
         _abilitydamage = 10;

         _characterClass = CharacterClass.Archer;
         _specialData = arrowCount;
     }

     public override void Rest()
     {
         _specialData++;
         _stamina += 5;
     }
 }

 public class Knight : Human
 {
     public Knight(uint id, int health, int stamina, string name, int specialData) : base(id, health,
stamina, name)
     {
         _attackSpecialDataCost = 0;
         _attackStaminaCost = 10;

         _abilitySpecialDataCost = 0;
         _abilityStaminaCost = 25;

         _attackdamage = 15;
         _abilitydamage = 10;

         _characterClass = CharacterClass.Knight;
         _specialData = specialData;
     }
 }

 public class Wizard : Human
 {
     public Wizard(uint id, int health, int stamina, string name, int mannaCount) : base(id, health,
stamina, name)
     {
         _attackSpecialDataCost = 10;
         _attackStaminaCost = 5;

         _abilitySpecialDataCost = 10;
         _abilityStaminaCost = 20;

         _attackdamage = 10;
         _abilitydamage = 10;

         _characterClass = CharacterClass.Wizard;
         _specialData = mannaCount;
```

```
    }

    public override void Ability()
    {
        if (_specialData > _abilitySpecialDataCost && _stamina > _abilityStaminaCost)
        {
            _specialData -= _abilitySpecialDataCost;
            _stamina -= _abilityStaminaCost;

            foreach (Human human in MainForm._playersTeam)
            {
                human._health += _abilitydamage;
            }
        }
        else
        {
            MainForm._logger.WriteToLogFile("Insufficient resources to ability", MainForm._pathForWrite);
            MainForm._logger.WriteToLogPanel("Insufficient resources to ability");
        }
    }

    public override void Rest()
    {
        _specialData += 5;
        _stamina += 5;
    }
}

public abstract class Monster : Entity
{
    public Monster(uint id, int health, int stamina) : base(id, health, stamina)
    {
    }
}

public class Witcher : Monster
{
    public Witcher(uint id, int health, int stamina, int mannaCount) : base(id, health, stamina)
    {
        _attackSpecialDataCost = 10;
        _attackStaminaCost = 5;

        _attackdamage = 10;
        _abilitydamage = 10;

        _characterClass = CharacterClass.Wither;
        _specialData = mannaCount;
    }

    public override void Rest()
    {
        _specialData += 5;
        _stamina += 5;
    }
}
   public class Goblin : Monster
   {
       public Goblin(uint id, int health, int stamina, int specialData) : base(id, health, stamina)
       {
           _attackStaminaCost = 5;
           _attackSpecialDataCost = 0;

           _abilityStaminaCost = 20;
           _abilitySpecialDataCost = 0;

           _attackdamage = 10;
           _abilitydamage = 10;

           _characterClass = CharacterClass.Goblin;
```

```
            _specialData = specialData;
        }
    }
    #endregion

    #region Singleton
    public sealed class Logger
    {
        private static Logger _instance;

        private Logger() { }

        public static Logger GetInstance()
        {
            if (_instance == null)
            {
                _instance = new Logger();
            }
            return _instance;
        }

        public void WriteToLogFile(string message, string pathForWrite)
        {
            using (StreamWriter writer = new StreamWriter(pathForWrite, true, Encoding.ASCII))
            {
                writer.WriteLine(message);
            }
        }

        public void WriteToLogPanel(string message)
        {
            MainForm._richTextBox.Text += message + "\n";
        }
    }
    #endregion
}
```