~ *Проект 1, модуль №3* ~

Предисловие

В рамках этого проекта вы будете работать с реальными данными - данными игр «Cultist Simulator» и «Book Of Hours», разработанных «Weather Factory». Мы получили официальное разрешение на использование их интеллектуальной собственности в образовательных целях. Надеемся, что вам понравится этот проект.

Также в рамках психологической разгрузки и адаптации к стрессовой ситуации в тексте содержатся мемы и отсылки на массовую и не очень культуру. Кто найдет наибольшее количество отсылок, получит небольшой презент.

ВАЖНО: есть страница Q&A по этому проекту. Вопросы можно задавать в форму. Если ваш вопрос уже присутствует в Q&A, то вы не получите на него отдельного ответа.

Общие слова

Что делаем?

Проект предполагает самостоятельную домашнюю работу по разработке консольного приложения. Вам потребуется:

1. Изучить предложенные теоретические материалы самостоятельно.
2. Самостоятельно поработать с документацией по языку C#, в т.ч. осуществить информационный поиск.
3. Разработать программы, определённые основной задачей и индивидуальным вариантом.
4. Сдать в SmartLMS заархивированную папку с решением (Solution) Visual Studio, где содержится проект(ы) с решением вашей задачи.

Когда сдаем?

Определяется датами, назначенными в SmartLMS.

Что сдаём?

На проверку предоставляется заархивированная папка с решением Visual Studio, в которое включен(ы) проект(ы), через который(е) реализована программа. В названии решения и архива обязательно указать фамилию автора.

Изучить самостоятельно

В этом разделе собраны ссылки, с которыми необходимо ознакомится при выполнения работы. Обратите внимание, что отдельные дополнительные материалы размещены в разделах с требованиями к библиотеке классов и консольному приложению. Самостоятельно ознакомиться с текстовым форматом данных JSON и материалами о потоковом вводе и выводе данных в файлы языка разработки. Для написания проекта потребуется использовать перенаправление стандартного потока ввода-вывода в текстовые файлы.

- Файловый и потоковый ввод-вывод
- Класс `Stream`
- Класс `FileStream`
- Перенаправление стандартного потока вывода консоли в файл
- Получение стандартного потока

Общие требования к работе

В индивидуальном варианте вы получите путь к JSON-файлам, изучив которые спроектируете нестатические классы, подходящие для представления данных в своей программе. Каждый класс следует размещать в отдельном файле с исходным кодом. Стоит отметить, что JSON файлы не отформатированы. Ваша программа **должна** уметь работать с форматированными и неформатированными JSON'ами. JSON'ы которые не считаются валидными по стандарту RFC 8259 читаться **не должны**. Если файл некорректен, то нужно сообщить об этом пользователю и запросить файл повторно.

Ваша программа представляет собой справочную систему, которая за счёт использования экранного меню и диалогов с пользователем, предоставляет возможность работы с данными, представленными в JSON-формате. Программа должна обеспечивать выполнение действий, определённых общими требованиями и индивидуальным вариантом работы.

Пример перенаправления стандартных потоков в файл

```
using System;
using System.IO;
class Program
{
    static void Main()
    {
        using StreamWriter log = new StreamWriter(@"system_log.txt");
        Console.SetOut(log);

        DateTime dt = DateTime.Now;

        Console.WriteLine("Начало системного журнала.");
        Console.WriteLine($"{dt}; {dt.Millisecond} {Milliseconds}");

        for (int k = 0; k < 100000; k++)
            if (k % 10000 == 0)
                Console.WriteLine(k);

        Console.WriteLine("Конец системного журнала.");
        dt = DateTime.Now;
        Console.WriteLine($"{dt}; {dt.Millisecond}{ Milliseconds}");
        log.Dispose(); // вызов Dispose не обязателен при наличии using выше
    }
}
```

При работе с данными:

1. Программа должна сохранять работоспособность при вводе некорректных адресов и имён файлов, с учётом различных платформ запуска файлов. При получении на вход некорректного адреса либо имени файла пользователю выводится сообщение об ошибке и, затем, экранное меню.
2. Некорректно структурированный файл вашей программой не обрабатывается, пользователю выводится сообщение об ошибке и, затем, экранное меню.
3. Программа обязательно должна корректно открывать созданные ею файлы и позволять выполнять над ними все операции в соответствии с пунктами меню.
4. Программа автоматически определяет кодировку файла и выводит данные на экран и в файлы в человекочитаемом виде.
5. Если у объектов будут отсутствовать некоторые из ожидаемых ключей, то **не** прерывайте работу программы и **не** выводите предупреждение. Заполните эти поля значениями по-умолчанию (выберите самостоятельно).

Критерии качества исходного кода:

1. В этой работе мы опираемся на создание нестатических классов и принципы объектно-ориентированного программирования (ООП).
2. Разделите программный код по отдельным файлам, т.е. каждый тип (класс, структура и т.п.) должен быть реализован в отдельном файле с исходным кодом.

Ограничения

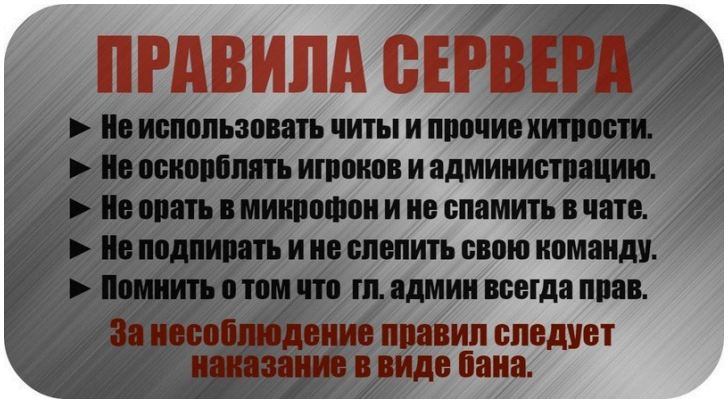

Рис. 1. Мем про правила

В основной и дополнительной задачах **требуется** (да, это блокирующие критерии):

- весь программный код написан на языке программирования C# с учётом использования .net 8.0;
- представленная к проверке библиотека классов решает все поставленные задачи и успешно компилируется.
- в приложении реализован цикл повторения решения, позволяющий повторить работу на других данных без завершения сеанса работы;
- имена классов соответствуют соглашениям об именовании;
- иерархии типов не нарушают принципа подстановки Лисков (Liskov Substitution Principle) и быть спроектированы, исходя из соблюдения принципа инверсии зависимостей (Dependency Inversion Principle);
- спроектированные классы соответствуют принципу единственной ответственности (Single Responsibility Principle) и не нарушают инкапсуляцию;
- поскольку мы активно используем интерфейсы, то важно соблюдение принципа разделения интерфейсов (Interface Segregation Principle);
- каждый нестатический класс (при наличии) содержит, в числе прочего, конструктор без параметров, либо эквивалентные описания, допускающие его явный или неявный вызов;
- не использованы:
  - NuGet-пакеты, использование которых не разрешено условием;
  - `System.Text.Json`;
  - прочий публично доступный код / библиотеки, не относящиеся к стандартным.

В основной и дополнительной задачах **требуется**:

- при перемещении папки проекта библиотеки (копировании / переносе на другое устройство) файлы должны открываться программой также успешно, как и на компьютере создателя (т.е. вашем), т.е. по относительному пути;
- текстовые данные, включая данные на русском языке, успешно декодируются при представлении пользователю и человекочитаемы;
- неуправляемые ресурсы, выделяемые при работе с файлами, обязательно освобождаются программой;
- все созданные при сохранении программой JSON-файлы имеют такую же структуру, как и файл с примером и должны без проблем читаться в качестве входных данных;
- программа не допускает пользователя до решения задач, пока с клавиатуры не будут введены корректные данные;
- консольное приложение обрабатывает исключительные ситуации, связанные:
  - со вводом и преобразованием / приведением данных, как с клавиатуры, так и из файла;
  - с созданием, инициализацией, обращением к элементам массивов и строк;
  - с вызовом методов библиотеки;
- исходный код должен содержать комментарии, объясняющие не очевидные фрагменты и решения, резюме кода, описание целей кода.

Описание задачи

Требования к библиотеке классов

Интерфейс `IJSONObject`

Реализации этого интерфейса должны предоставлять следующие методы:

- `IEnumerable<string> GetAllFields`: возвращает коллекцию строк, представляющую имена всех полей объекта JSON
- `string GetField(string fieldName)`: возвращает значение поля с указанным именем `fieldName` в формате строки. В случае отсутствия поля с заданным именем, возвращает `null`.
- `void SetField(string fieldName, string value)`: присваивает полю с именем fieldName значение value. Если поле с таким именем отсутствует, то должно генерироваться исключение `KeyNotFoundException`.

Классы представляющие объекты

Классы представляют объекты, описанные в JSON файле индивидуального варианта. Вложенные и связанные объекты описываются отдельными классами. В индивидуальном варианте приведено описание типов отношений между связанными объектами. Поля каждого класса должны быть доступны для чтения, но закрыты для записи. Выберите самостоятельно идентификатор класса, который будет логично описывать объект и удовлетворять правилам именования классов. Все такие классы должны реализовывать интерфейс `IJSONObject`.

Класс `JsonParser`

Статический класс `JsonParser` содержит два статических метода: `WriteJson` и `ReadJson`. Эти методы обязательно используют потоки данных, определённые в `System.Console` для чтения и записи данных. Данные подаются в формате JSON, для их парсинга **запрещено** использовать любые специализированные библиотеки. Вы можете воспользоваться подходом на основе:

- конечных автоматов
- регулярных выражений
  - Пространство имён `System.Text.RegularExpressins`
  - Класс `Regex`
  - Краткий справочник элементов языка регулярных выражений

Решение задачи не предполагает использования атрибутов типов, рефлексии и контрактов данных.

Важное примечание

Мы рекомендуем вам реализовывать работу `JsonParser` так, чтобы ему было все равно какой класс или структура «прячется» под интерфейсом. Используйте только методы, описанные в интерфейсе. Такая реализация будет проще и правильнее.

Пример реализации конечного автомата

В примере решим с помощью конечного автомата задачу подсчета количества строк и комментариев в исходном коде, переданном нам на вход. Для упрощения задачи будем считать, что комментарий начинается с символа ‘/’ и заканчивается концом строки, а строка начинается и заканчивается символом ‘«’. В таком случае наш автомат будет иметь следующие три состояния: (1) Комментарий, (2) Строка, (3) Остальной код программы (см. рисунок 2).

```
public class Program
{

    public enum State
    {
        String,
        Comment,
        Program
    }

    public static void Main()
    {
        Console.WriteLine("Введите код программы:");
        string? code = Console.ReadLine();

        State state = State.Program;
        int commentCount = 0;
        int stringCount = 0;

        foreach (var symbol in code ?? "")
        {
            switch (state)
            {
                case State.Program when symbol == '/':
                    commentCount++;
                    state = State.Comment;
                    break;
                case State.Program when symbol == '"':
                    stringCount++;
                    state = State.String;
                    break;
                case State.Comment when symbol == '\n':
                    state = State.Program;
                    break;
                case State.String when symbol == '"':
                    state = State.Program;
                    break;
            }
        }

        Console.WriteLine($"Количество строк: {stringCount}");
        Console.WriteLine($"Количество комментариев: {commentCount}");
    }
}
```

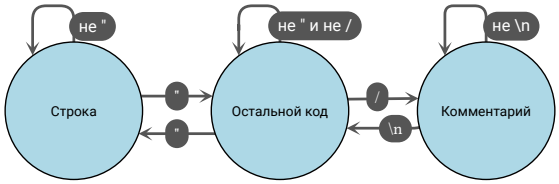

Рис. 2. Состояния конечного автомата и переходы между ними

Требования к консольному приложению

Консольное приложение предоставляет пользователю интерактивный интерфейс для работы с коллекцией объектов, описанных в индивидуальном варианте. Приложение использует библиотеку классов, описанную выше, для работы с JSON.

Взаимодействие с данными

- **Чтение данных:**
  - Приложение должно уметь читать данные из стандартного потока ввода `System.Console` с помощью `Console.ReadLine()`.
  - Приложение должно предоставлять возможность чтения данных из файла, указанного пользователем. Для этого необходимо перенаправить стандартный поток ввода с помощью метода `Console.SetIn()`.
- **Запись данных:**
  - Приложение должно уметь выводить данные в стандартный поток вывода `System.Console` с помощью `Console.WriteLine()`.
  - Приложение должно предоставлять возможность записи данных в файл, указанный пользователем. Для этого необходимо перенаправить стандартный поток вывода с помощью метода `Console.SetOut()`. Пользователь должен иметь возможность перезаписать исходный файл или указать новый путь для сохранения.

Работа с коллекцией объектов

Приложение должно предоставлять пользователю следующие возможности через экранное меню:

- **Ввод данных:** Выбор источника данных (`System.Console` или файл).
- **Фильтрация:**
  - Выбор поля для фильтрации из списка всех доступных полей объекта;
  - Отображение выбранного поля для фильтрации в интерфейсе приложения;
  - Фильтрация объектов коллекции по заданному критерию, связанному с выбранным полем;
  - Пользователь должен вписать массив объектов, которые должны остаться в этом столбце.
- **Сортировка:**
  - Выбор поля для сортировки из списка всех доступных полей объекта и направления сортировки;
  - Отображение выбранного поля для сортировки в интерфейсе приложения;
  - Сортировка объектов коллекции по значениям выбранного поля.
- **Вывод данных:** Выбор места вывода данных (`System.Console` или файл).

Сортировку и фильтрацию нужно реализовывать **только** для полей со строками, целыми и вещественными числами. Сортировка и фильтрация массивов **не** предполагается. Вам будет чем заняться...

Важно

- При фильтрации и сортировке необходимо отображать все поля объекта, а не только поле, используемое для фильтрации/сортировки.
- Для работы с JSON приложение использует статический класс `JsonParser` и интерфейс `IJSONObject`.
- Для возврата к стандартным потокам ввода/вывода после работы с файлами необходимо использовать методы `Console.OpenStandardInput()` и `Console.OpenStandardOutput()`.

Пример структуры меню

```
1. Ввести данные (консоль/файл)
2. Отфильтровать данные
3. Отсортировать данные
4. <основная задача индивидуального варианта>
5. <дополнительная задача индивидуального варианта>
6. Вывести данные (консоль/файл)
7. Выход
```

~ *Индивидуальные варианты* ~

~ *Вариант №1* ~

BoH: Достижения

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с данными о достижениях в игре Book of Hours. Эти данные представлены в формате JSON и хранятся в директориях `./BoH/<locale>/achievements/`

Вам необходимо создать **структуру** «Достижение» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании). Важно учитывать, что некоторые JSON-объекты описывают не отдельные достижения, а группы достижений. Эти объекты можно отличить по наличию поля `"isCategory": true`.

Основная задача

Реализовать TUI для просмотра информации о группах достижений и достижениях в них. Пользователю показывается список всех категорий достижений. В следующем формате:

```
┌──────────────────────────────────┐
│ Цель жизни персонажа выполнена   │
└──────────────────────────────────┘

<...>

┌──────────────────────────────────┐
│ Достижение писателя              │
└──────────────────────────────────┘

<...>

┌──────────────────────────────────┐
│ Победа Общества                  │
└──────────────────────────────────┘
```

Стоит отметить, что «тень» должна выводиться только под той карточкой, которая сейчас выбрана. Перемещение между карточками должна производиться при помощи стрелочек вверх и вниз. Разворачивание/свертывание карточки должно производиться по нажатию клавиши «Enter». Выход из режима просмотра должен производиться при помощи нажатии на «Escape».

Стоит подчеркнуть, что «открытыми» может быть несколько групп. Перемещение по отдельным достижениям реализовывать не подразумевается.

```
┌──────────────────────────────────────────────┐
│ Цель жизни персонажа выполнена               │
└──────────────────────────────────────────────┘

<...>

┌──────────────────────────────────────────────┐
│ Достижение писателя                          │
└──────────────────────────────────────────────┘

<...>

┌──────────────────────────────────────────────┐
│ Достижение приёмов                           │
├──────────────────────────────────────────────┤
│ - Второй завтрак                             │
│ - Я же сказал тебе, что люблю лепёшки        │
│ - «Стресс» и «пирог» — понятия взаимоисключающие │
│ - Еда, шутки и песни                         │
│ - Горячее мороженое на холодный день         │
└──────────────────────────────────────────────┘

<...>

┌──────────────────────────────────────────────┐
│ Победа Общества                              │
└──────────────────────────────────────────────┘
```

В любой момент карточки должны быть одинакового размера. Выводить все карточки на экран необязательно, но при наличии места в терминале, должна вывестись предыдущая и следующая карточки.

Дополнительная задача

Реализовать взаимодействие с Steam Web API для получения процента людей, получивших каждое конкретное достижение.

При реализации требуется:
- использование стандартной библиотеки `System.Net.Http`
- обращаться можно только к официальному Steam Web API, размещенного по адресу `api.steampowered.com`
- все запросы должны выполняться **без** API-ключей.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.

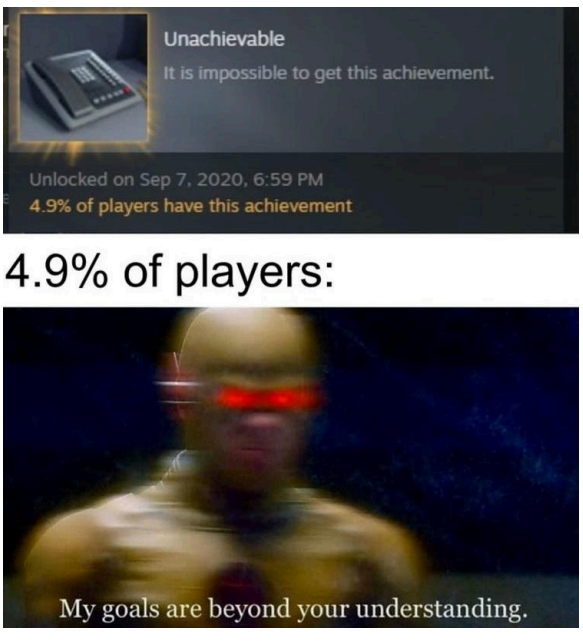

Рис. 3. Невозможное возможно

~ *Вариант №2* ~

BoH: Наборы

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с наборами объектов в Book of Hours. Эти данные представлены в формате JSON и хранятся в директориях `./BoH/<locale>/decks/`

Вам необходимо создать **<u>класс</u>** «Набор» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Пользователю показывается список всех наборов, он выбирает один ИЛИ несколько наборов. Программа должна вывести таблицу, где в первом столбце будут находится объекты, содержащиеся в одном из наборов, в последующих столбцах - кол-во таких объектов в этом наборе. Формат таблицы должен быть следующим:

Объекты	d.chat.generic	d.chat.scholar
precursor.mem.hindsight	1	4

Дополнительная задача

Реализовать генерацию информационного изображения об объектах в наборе. На изображении должно быть:
- системное название выбранного набора
- название набора
- все изображения объектов набора

В реализации требуется использование NuGet библиотеки `SkiaSharp`.

<u>Требования к реализации</u>. <u>Требования к реализации поиска изображений (механизм Б)</u>.

~ *Вариант №3* ~

BoH: Аспекты

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Book of Hours. Эти данные представлены в формате JSON и хранятся в директориях:

- `./BoH/<locale>/elements/_aspects_salons.json`
- `./BoH/<locale>/elements/_aspects.json`

Вам необходимо создать **структуру** «Аспект» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основные задачи

Реализуйте следующие функции:

- **Дозагрузка данных**
  Функция должна обеспечивать добавление новых данных к существующим без удаления старых. При обработке новых данных, необходимо проверять наличие аспекта с таким же id в коллекции. Если аспект с таким id уже существует, старый объект заменяется новым. Если аспект с таким id отсутствует, новый объект добавляется в коллекцию.
- **Редактирование данных**
  Функция должна позволять пользователю редактировать существующий аспект по его id. Если аспект с указанным id не найдено, должен быть создан новый объект с этим id. Способ реализации редактирования не регламентируется.
- **Удаление данных**
  Пользователь вводит `id` объекта, который он хочет удалить. Этот объект удаляется.
- **Слияние данных**
  Функция должна объединять данные из двух файлов, выбранных пользователем. Если аспект присутствует только в одном из файлов, оно добавляется в итоговый файл. Если аспект присутствует в обоих файлах, приложение должно запросить у пользователя, какую версию аспекта (из какого файла) следует использовать в итоговом файле.

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность приложения должна быть расширена:

- Меню выбора файла:
  - Отображение всех файлов в текущей директории.
  - Возможность навигации по директориям (переход в дочернюю/родительскую директорию).
- Индикатор выполнения:
  - Отображение панели загрузки во время выполнения длительных операций (чтение/запись файлов, слияние данных).
- Инструмент слияния (Merge tool):
  - Разделение экрана на две части для отображения информации о «спорных» аспектах из каждого файла.

Требования к реализации.

~ *Вариант №4* ~

BoH: Возможности

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с возможностями в Book of Hours. Эти данные представлены в формате JSON `./BoH/<locale>/elements/abilities*.json`

Вам необходимо создать **<u>класс</u>** «Возможности» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Напишите обозреватель `xtriggers`. В виде ключа хранится `id` триггера, а в значении - объект ИЛИ массив объектов, которые получаются в результате работы триггера.

Вводит `id` возможности. Формат вывода должен быть следующим:

```
┌──────────────────────────────┐
│ ID: xcho3 │ Label: ++Хор     │
├──────────────────────────────┤
│ Description:                 │
│ Полифония голосов [Одна из   │
│ девяти частей души человека.]│
├──────────────────────────────┤
│ Triggers:                    │
└──────────────────────────────┘
..............................
┌──────────────────────────────┐
│ contamination.bloodlines     │
│ ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ │
│ georgia                      │
└──────────────────────────────┘
..............................
```

Если что, ширина таблицы задается первой «строкой». Если на последующих строках текст не помещается - нужно сделать перенос строки по ближайшему пробелу. Если такого нет, то осуществить перенос «разрывом» слова, добавив в него дефис. Перенос может осуществляться не по правилам русского языка (например, может быть внутри слога).

Дополнительная задача

Запросите у пользователя:
- путь до JSON-файла с аспектами (`./BoH/<locale>/elements/_aspects.json`)
- путь до директорий с изображениями (от 0 до бесконечности)

Используя NuGet библиотеку `SkiaSharp` повторите элемент интерфейса из игры:

Необходимо реализовать:
- внешнюю рамку со скруглениями
- вывод изображения способности
- заголовок способности
- разделительную черту между заголовком и описанием
- описание способности
- внутреннюю рамку со скруглениями
- иконки аспектов (если не найдена - вопросительный знак в квадрате) с указанием количества
- между аспектами должен быть разделитель (можете использовать ♦)

Файл с аспектами нужно использовать для получения значения поля `noartneeded`, полная реализация структуры не ожидается.

<u>Требования к реализации</u>. <u>Требования к реализации поиска изображений (механизм А)</u>.

Рис. 5. Студент, приступивший к проекту за 3 часа до дедлайна.

~ *Вариант №5* ~

BoH: Умения

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с умениями в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/skills.json`

Вам необходимо создать **структуру** «Умения» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Реализуйте раздел, который по `id` умения и текущему уровню умения будет выдавать необходимые карточки для дальнейшего улучшения. Формат вывода:

```
ID:          s.bells.brazieries
LABEL:       Колокола и жаровни
DESCRIPTION: Тот гулкий звон, что порождает молот.
LEVEL:       1

LEVEL UP:
```

```
ID: x1
LABEL: Извлечённый урок
```

```
ID: x2
LABEL: Воспоминание
```

Важно: ширина всех карточек должна быть одинаковой.

Дополнительная задача

Напишите функцию конвертации загруженных из JSON данных в Excel-таблицу (*.xlsx) и обратно.

Требования к реализации.

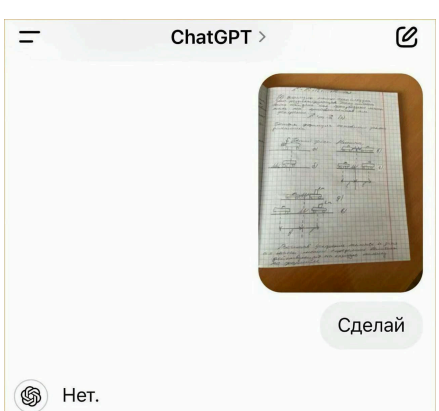

Рис. 6. Напоминание, что использование генеративных моделей без указания созданных фрагментов запрещено

~ *Вариант №6* ~

BoH: Вещи

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с вещами в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/aspecteditems.json`

Вам необходимо создать **структуру** «Вещь» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Игрок может взять несколько объектов и совершить вместе с ними какие-то действия. Например, изучая артефакт, читать книгу, или переплавить металлические вещи в кузнице. Для того, чтобы действие прошло успешно, нужно собрать определенный перечень аспектов всех объектов, которые участвуют в действии.

Пользователь вбивает в программу аспекты необходимые ему для успешного совершения действия, указывает необходимые объекты. Программа должна подобрать объекты, вместе с которыми все пройдет успешно. Выведите до 10 наименьших комбинаций или сообщение о том, что такие комбинации невозможны. Формат вывода:

```
┌ Необходимые предметы:
├─┐ ID:    oil.umbrous
│ │ LABEL: Как у Дзюндзи Ито
│ └──────────────────────
│
├─┐ ID:    stargall.ink
│ │ LABEL: Звёздная тушь
│ └──────────────────
│
└─┐ ID:    mirror.storm
  │ LABEL: Пережившее шторм зеркало
  └──────────────────────────────
```

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность должна быть расширена:

- Меню выбора файла:
  - Отображение всех файлов в текущей директории.
  - Возможность навигации по директориям (переход в дочернюю/родительскую директорию).
- Индикатор выполнения:
  - Отображение панели загрузки во время выполнения длительных операций (чтение/запись файлов, слияние данных).

Требования к реализации.

~ *Вариант №7* ~

BoH: Книги

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с книгами в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/tomes.json`

Вам необходимо создать **структуру** «Книга» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Разработать приложение, которое позволяет пользователю ввести ID триггера (например, `mastering.lantern`). Приложение должно вывести информацию о всех объектах, которые добавляются этим триггером. Формат вывода должен быть следующим:

ID книги	ID объекта	Уровень
t.thesunsdesign	lemon.pink	2
	x.watchmansparadoxes	1

Дополнительная задача

Реализовать возможность добавления, удаления и редактирования триггеров и их эффектов при помощи TUI интерфейса, реализованного при помощи `Terminal.GUI`.

Требования к реализации.

Рис. 7. Перечитайте еще раз условие. Может быть, вы откроете его для себя с другой стороны

~ *Вариант №8* ~

BoH: Посетители

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с посетителями в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/visitors*.json`

Вам необходимо создать **структуру** «Посетитель» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Разработать приложение, которое анализирует `xexts` посетителей и определяет, какие посетители связаны друг с другом (например, дружат, планируют встретиться, недолюбливают друг друга). Приложение должно вывести список связей между посетителями. Формат вывода:

Статус	ID посетителя	Имя посетителя
Дружба	agdistis	Мистер Питер Агдистис
	chaima	Лалла Хаима

Дополнительная задача

Визуализировать связи между посетителями с помощью графа, используя библиотеку `SkiaSharp`. Если два посетителя могут поладить, то связь должна быть зеленой, если могут только поссориться - красной. В ином случае - связь не отображать.

Требования к реализации. Требования к реализации поиска изображений (механизм A).

~ *Вариант №9* ~

BoH: Рабочие места

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с рабочими местами в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/verbs/workstations_*.json`

Вам необходимо создать **структуру** «Рабочее место» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Разработать приложение, которое принимает ID рабочего стола и выводит информацию о требованиях (required) и запретах (forbidden) для каждого слота (slots). Учитывать поля essential.

```
Рабочее место: library.rowenarium.spencer.down

Описание: Rowena the Ligeian, protectress of the House, holds the Mare's Key of worm-stone, which
opens the door to Nowhere. Now and then, she has lent it unwisely. But only once has she lent it
to Ys. Spencer Hobson's visit has extended the Rowenarium 'almost-down'. [The Rowenarium now
admits Knock intentions. It also provides Winter Aspect.]

Подсказки: ["rose","winter","grail","moth","knock"]

Звук: Eerie

Слоты:
  ┌Слот: Soul (a)
  │ ├─Необходимо: ability: 1
  │ ├─Требуется: rose: 1 winter: 1 grail: 1 moth: 1 knock: 1
  │ └─Запрещено:
  └─
  ┌Слот: Skill (s)
  │ ├─Необходимо: skill: 1
  │ ├─Требуется: rose: 1 winter: 1 grail: 1 moth: 1 knock: 1
  │ └─Запрещено:
  └─
  ┌Слот: Memory (m)
  │ ├─Необходимо: memory: 1
  │ ├─Требуется: rose: 1 winter: 1 grail: 1 moth: 1 knock: 1
  │ └─Запрещено:
  └─
  ┌Слот: + (i1)
  │ ├─Необходимо:
  │ ├─Требуется: ability: 1 remains: 1 cooperative: 1 omen: 1
  │ └─Запрещено: fatigued: 1
  └─
  ┌Слот: + (i2)
  │ ├─Необходимо:
  │ ├─Требуется: memory: 1 pigment: 1 key: 1 remains: 1
  │ └─Запрещено: fatigued: 1
  └─

Аспекты:
  difficulty.prentice: 5
  difficulty.scholar: 10
  difficulty.keeper: 15
  e.skolekosophy: 1
  winter: 2
  workstationhasaspect: 1

Категория: Workstation

Является ли областью: true
```

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность должна быть расширена возможностью сравнения требований слотов разных рабочих столов.

Требования к реализации.

Рис. 8.

~ *Вариант №10* ~

CS: Достижения

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с данными о достижениях в игре Cultist Simulator. Эти данные представлены в формате JSON и хранятся в директориях `./CS/<locale>/achievements/`

Вам необходимо создать **структуру** «Достижение» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании). Важно учитывать, что некоторые JSON-объекты описывают не отдельные достижения, а группы достижений. Эти объекты можно отличить по наличию поля `"isCategory": true`.

Основная задача

Реализовать TUI для просмотра информации о группах достижений и достижениях в них. Пользователю показывается список всех категорий достижений в следующем формате:

```
┌──────────────────────────────────┐
│ Цель жизни персонажа выполнена   │
└──────────────────────────────────┘

<...>

┌──────────────────────────────────┐
│ Достижение писателя              │▒
└──────────────────────────────────┘▒
 ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

<...>

┌──────────────────────────────────┐
│ Победа Общества                  │
└──────────────────────────────────┘
```

Стоит отметить, что «тень» должна выводиться только под той карточкой, которая сейчас выбрана. Перемещение между карточками должно осуществляться при помощи стрелочек вверх и вниз. Разворачивание/сворачивание карточки должно производиться по нажатию клавиши «Enter». Выход из режима просмотра должен производиться по нажатию клавиши «Escape».

Стоит подчеркнуть, что «открытыми» может быть несколько групп. Перемещение по отдельным достижениям реализовывать не нужно.

```
┌──────────────────────────────────────────────┐
│ Цель жизни персонажа выполнена               │
└──────────────────────────────────────────────┘

<...>

┌──────────────────────────────────────────────┐
│ Достижение писателя                          │▒
└──────────────────────────────────────────────┘▒
 ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

<...>

┌──────────────────────────────────────────────┐
│ Достижение приёмов                           │
├──────────────────────────────────────────────┤
│ - Второй завтрак                             │
│ - Я же сказал тебе, что люблю лепёшки        │
│ - «Стресс» и «пирог» — понятия взаимоисключающие
│ - Еда, шутки и песни                         │
│ - Горячее мороженое на холодный день         │
└──────────────────────────────────────────────┘

<...>

┌──────────────────────────────────────────────┐
│ Победа Общества                              │
└──────────────────────────────────────────────┘
```

В любой момент карточки должны быть одинакового размера. Выводить все карточки на экран необязательно, но при наличии места в терминале, должны вывестись предыдущая и следующая карточки.

Дополнительная задача

Реализовать взаимодействие с Steam Web API для получения процента персонажей, получивших каждое конкретное достижение.

При реализации требуется:
- использование стандартной библиотеки `System.Net.Http`
- обращаться можно только к официальному Steam Web API, размещенного по адресу `api.steampowered.com`
- все запросы должны выполняться **без** API-ключей.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.

~ *Вариант №11* ~

CS: Наборы

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с наборами объектов в Cultist Simulator. Эти данные представлены в формате JSON и хранятся в директориях `./CS/<locale>/decks/`

Вам необходимо создать **класс** «Набор» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Пользователю показывается список всех наборов, он выбирает один ИЛИ несколько наборов. Программа должна вывести таблицу, где в первом столбце будут находится объекты, содержащиеся в одной из групп, в последующих столбцах - кол-во таких объектов в этом наборе. Формат таблицы должен быть следующим:

Объекты	d.chat.generic	d.chat.scholar
precursor.mem.hindsight	1	4

Дополнительная задача

Реализовать генерацию информационного изображения об объектах в наборе. На изображении должно быть:
- системное название выбранного набора
- название набора
- все изображения объектов набора

При реализации требуется использование NuGet библиотеки `SkiaSharp`.

Требования к реализации. Требования к реализации поиска изображений (механизм Б).

Рис. 9. У нас не так. Есть вопрос? Задайте его в форму!

~ *Вариант №12* ~

CS: Аспекты

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Cultist Simulator. Эти данные представлены в формате JSON и хранятся в директориях:

- `./CS/<locale>/elements/_aspects.json`

Вам необходимо создать **структуру** «Аспект» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основные задачи

Реализуйте следующие функции:

- **Дозагрузка данных**
  Функция должна обеспечивать добавление новых данных к существующим (без удаления старых). При обработке новых данных необходимо проверять наличие достижения с таким же id в коллекции. Если достижение с таким id уже существует, старый объект заменяется новым. Если достижение с таким id отсутствует, новый объект добавляется в коллекцию.
- **Редактирование данных**
  Функция должна позволять пользователю редактировать существующее достижение по его id. Если достижение с указанным id не найдено, должен быть создан новый объект с этим id. Способ реализации редактирования не регламентируется.
- **Удаление данных**
  Пользователь вводит `id` объекта, который он хочет удалить. Этот объект удаляется.
- **Слияние данных**
  Функция должна объединять данные из двух файлов, выбранных пользователем. Если достижение присутствует только в одном из файлов, оно добавляется в итоговый файл. Если достижение присутствует в обоих файлах, приложение должно запросить у пользователя, какую версию достижения (из какого файла) следует использовать в итоговом файле.

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность приложения должна быть аналогична консольной версии, но с дополнительными элементами интерфейса:

- Меню выбора файла:
  - Отображение всех файлов в текущей директории.
  - Возможность навигации по директориям (переход в дочернюю/родительскую директорию).
- Индикатор выполнения:
  - Отображение панели загрузки во время выполнения длительных операций (чтение/запись файлов, слияние данных).
- Инструмент слияния (Merge tool):
  - Разделение экрана на две части для отображения информации о «спорных» достижениях из каждого файла.
- Кнопку «О программе» в `MenuBar`

~ *Вариант №13* ~

CS: Книги

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Cultist Simulator. Эти данные представлены в формате JSON: `./CS/<locale>/elements/books_*.json`

Вам необходимо создать **структуру** «Книга» для представления этих данных в приложении.

Основная задача

Разработать приложение, которое позволяет пользователю ввести один или несколько аспектов (например, `auctionable`) и список столбцов для итогового вывода. Приложение должно вывести список всех книг, которые соответствуют указанным аспектам. Должны вывестись только столбцы, указанные пользователем. Форма вывода:

ID	DESCRIPTION
tantraworms	The worm has eaten a hole in me and now it is a part of me.

Все значения в ячейках должны быть отцентрованы.

Дополнительная задача

Напишите конвертацию загруженных данных в Excel-таблицу (*.xlsx) и обратно.

Требования к реализации.

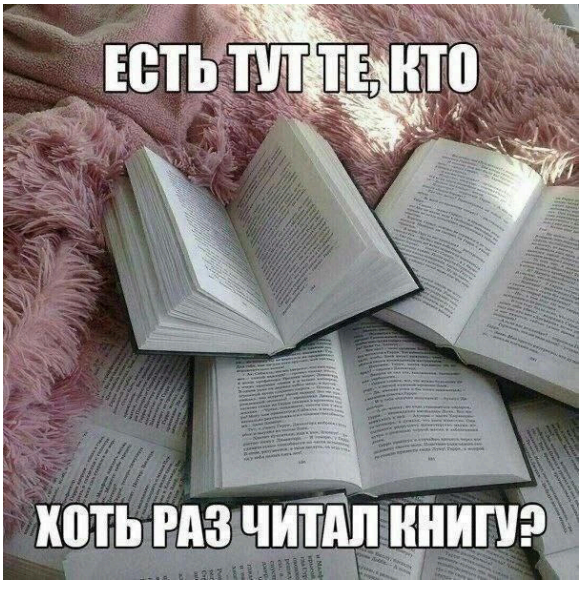

Рис. 10.

~ *Вариант №14* ~

CS: Фрагменты знаний

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Cultist Simulator. Эти данные представлены в формате JSON: `./CS/<locale>/elements/fragments.json`

Вам необходимо создать **структуру** «Фрагмент» для представления этих данных в приложении.

Основная задача

Разработать приложение, которое позволяет пользователю ввести `id` фрагмента знания и отображает информацию о его аспектах (`aspects`) и возможных сочетаниях (`slots`). Для каждого сочетания необходимо вывести описание, а также перечень `required` аспектов для осуществления сочетания.

```
Фрагмент знания: Тайна Ножа (fragmentedge)

Аспекты:
  edge: 2
  lore: 1
  subvertable_lore: 1
  challenge.practical: 1

Сочетания:
  Сочетание: Изучите с фрагментом знаний Грани того же уровня, чтобы повысить уровень. Изучите со
знаниями Зимы того же уровня, чтобы обратить и получить знания Зимы.
    Требуемые аспекты:
      fragmentedge: 1
      fragmentwinter: 1
```

Дополнительная задача

Создать функцию, генерирующую PDF-отчет о выбранном фрагменте знания. Используйте библиотеку `PdfSharpCore`. Отчет должен включать название фрагмента, описание, список аспектов с их уровнями и список возможных сочетаний.

~ *Вариант №15* ~

CS: Хранилища

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Cultist Simulator. Эти данные представлены в формате JSON: `./CS/<locale>/elements/vaults.json`

Вам необходимо создать **структуру** «Хранилище» для представления этих данных в приложении.

Основная задача

Разработать приложение, которое позволяет пользователю просматривать информацию о хранилищах. Приложение должно отображать для выбранного хранилища его label, список aspects с их значениями и информацию о доступных slots для экспедиций. Для каждого слота необходимо выводить его label, description и required аспекты.

Пример вывода для ID «vaultcapital1»:

```
Хранилище: Особняк Стрэткойна (vaultcapital1)

Аспекты:
  vault: 1
  location: 1
  vaultcapital: 1

Слоты для экспедиций:
├─Лидер: Для начала экспедиции мне нужен хотя бы один последователь. Позже я смогу выбрать ещё.
│ └─Требуемые аспекты:
│   └─follower: 1
├─Помощник: Я могу добавить ещё одного последователя, если думаю, что он пригодится в экспедиции.
│ └─Требуемые аспекты:
│   └─follower: 1
├─Средства: Для начала экспедиции мне нужен хотя бы одна карта средств.
│ └─Требуемые аспекты:
│   └─funds: 1
└─Дополнительные средства: Я могу добавить ещё средств, если опасаюсь, что текущих не хватит.
  └─Требуемые аспекты:
    └─funds: 1
```

Дополнительная задача

Напишите конвертацию загруженных данных в Excel-таблицу (*.xlsx) и обратно.

Требования к реализации.

Рис. 11.

~ *Вариант №16* ~

CS: Последователи

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с аспектами в Cultist Simulator. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/followers.json`

Вам необходимо создать **структуру** «Последователь» для представления этих данных в приложении.

Основная задача

Разработать функциональность консольного приложения, которая будет отображать дерево возможных «продвижений» последователя, начиная с указанного пользователем.

Предположим, пользователь ввел ID начального последователя: auclair_a.

```
Последователь: auclair_a (Оклер, знакомая)
├── recruiting --> auclair_b (Оклер, неофит)
│   ├── promotiontodisciple --> auclair_c (Оклер, адепт)
│   │   └── promotionto_d_winter --> auclair_d (Молчаливая измождённая женщина)
│   ├── killmortal --> corpse
│   ├── derangemortal --> lunatic
│   └── capturefollower --> auclair_p
└── rebellion --> auclair_r
```

Дополнительная задача

Расширить функциональность приложения, чтобы визуализировать дерево развития последователя не только в текстовом виде, но и графически, используя библиотеку SkiaSharp.

У каждого «узла» должно быть выведено изображение последователя.

Требования к реализации. Требования к реализации поиска изображений (механизм A).

Рис. 12.

~ *Приложение: Требования к реализации генерации изображений* ~

- Все элементы изображения, требуемые в индивидуальном варианте, не должны пересекаться
- При размещении на Canvas изображений, их размер не должен быть менее 50% от исходного размера
- При размещении на Canvas текста:
  - размер шрифта не должен быть менее 10
  - разрешенные шрифты:
    - `Times New Roman`
    - `HSE Sans`
    - `HSE Slab`
- от НЕ белого пикселя до границы Canvas должно быть не менее 50 белых пикселей

Небольшое послесловие

Рекомендуется реализовать дополнительную библиотеку для работы с объектами, которая, в свою очередь, будет использовать `SkiaSharp`.

Рекомендую прочитать эту и эту статью. В них находится все, что вам нужно.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.

~ *Приложение: Алгоритм поиска изображений* ~

У пользователя должна быть возможность указать несколько директорий для поиска изображений. Ниже будут описаны алгоритмы поиска внутри одной директории. Если в индивидуальном варианте есть работа с изображениями, то там должно быть указание на алгоритм, которому должна следовать программа

Алгоритм А: Объекту сопоставляются все изображения, найденные по маске `<id>.png`, где `<id>` - `ID` объекта
Алгоритм Б: Объекту сопоставляются все изображения, найденные по маскам `<id>.png` и `_<id>.png`, где `<id>` - `ID` объекта

~ *Приложение: Требования к реализации TUI* ~

Данный раздел не распространяется на 1 и 10 вариант. Эти требования распространяются только на дополнительные задания.

В интерфейсе должен быть доступен весь функционал, что и в консольном приложении, то есть:

- импортирование/экспортирование файла с данными
- фильтрация данных
- сортировка данных
- основная задача из индивидуального варианта

Также требуется добавить кнопку «О программе» в `MenuBar`. При нажатии появляется информация о том, кто эту программу делал и номер индивидуального варианта.

Требуется сдать **две** версии программы: с классическим терминальным интерфейсом и с TUI.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.

Рис. 13. да да....

~ *Приложение: Требования к реализации конвертации в Excel документ* ~

- Запрещается хранить массивы в ячейках
- Первая строка каждой страницы таблицы должна иметь акцентный цвет и содержать названия столбцов
- При конвертации в табличный и из табличного вида информация не должна теряться. Порядок объектов должен быть сохранен.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.