МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р. техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2023 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ЗАСТОСУНОК ДЛЯ РЕКОМЕНДАЦІЙ ВІДЕОІГОР НА БАЗІ ПЛАТФОРМИ .NET

Спеціальність 122 «Комп'ютерні наук»

122 – БКР – 401.21910103

Виконав студент 4-го курсу, групи 401
_____ Є. В. Бардовський
«___» червня 2023 р.

Керівник: д-р. техн. наук, доцент
_____О. В. Козлов
«___» червня 2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р. техн. наук, проф.
_____ Ю. П. Кондратенко

«____» _____2022 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Бардовському Євгенію Вячеславовичу.

1. Тема кваліфікаційної роботи «Застосунок для рекомендацій відеоігор на базі платформи .NET».

Керівник роботи Козлов Олексій Валерійович, д-р. техн. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» _____ 20___ р. № ____

2. Строк представлення кваліфікаційної роботи студентом «___» _____ 2023 р.

3. Вхідні (початкові) дані до роботи: діяльність ринку відеоігор та рекомендаційних систем відеоігор.

Очікуваний результат: застосунок з рекомендаційною системою відеоігор.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз індустрії відеоігор та існуючих рекомендаційних систем;
- огляд існуючих методів та алгоритмів для розробки рекомендаційних систем;

– проєктування рекомендаційної системи відеоігор;
– розробка та здійснення програмної реалізації застосунку для рекомендації відеоігор.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Оцінка ризиків пов'язаних з роботою на ПК та заходи їх зниження».

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	канд. техн. наук, доцент А. О. Алексєєва	

Керівник роботи __д-р. техн. наук, доц. Козлов О. В._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Бардовський Є. В._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «__» _____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

Виконання бакалаврської кваліфікаційної роботи

Тема: «Застосунок для рекомендацій відеоігор на базі платформи .NET»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівника БКР	26.10.2022	30.10.2022	Виконано
2	Отримання завдання на виконання БКР	25.11.2022	25.11.2022	Виконано
3	Складання календарного плану роботи на весь період виконання БКР	26.112021	10.12.2021	Виконано
4	Аналіз предметної області, існуючих аналогів рекомендаційних систем відеоігор	11.12.2021	31.12.2021	Виконано
5	Створення дизайну, проєктування та програмна реалізація застосунку	01.04.2022	01.05.2022	Виконано
6	Проходження переддипломної практики	01.05.2022	14.05.2022	Виконано
7	Збір та аналіз матеріалів, оформлення розділів БКР	01.05.2022	14.05.2022	Виконано
8	Робота над розділами пояснювальної записки БКР	14.05.2022	28.05.2022	Виконано
9	Розробка спеціальної частини з охорони праці	17.05.2022	28.05.2022	Виконано
10	Попередній захист БКР	29.05.2022	29.05.2022	Виконано
11	Корегування роботи за результатами попереднього захисту	02.06.2023	06.06.2023	Виконано
12	Остаточне оформлення пояснювальної записки та слайдів доповіді до захисту	07.06.2023	11.06.2023	Виконано
13	Подання рецензенту та рецензування БКР	15.06.2023	19.06.2023	Виконано
14	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.06.2023	22.06.2023	Виконано
15	Захист БКР перед ЕК	27.06.2023	27.06.2023	Виконано

Розробив студент Бардовський Є. В. _____

(прізвище та ініціали) *(підпис)*

Керівник роботи д-р. техн. наук, доцент Козлов О.В. _____

(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

« » 2023 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи

студента групи 401 ЧНУ ім. Петра Могили

Бардовського Євгенія Вячеславовича

Тема: «Застосунок для рекомендацій відеоігор на базі платформи .NET»

Актуальність створення застосунку для рекомендацій відеоігор обумовлена інтенсивним розвитком галузі відеоігор. Проблема вибору відповідних відеоігор для конкретного користувача залишається актуальною і вимагає розробки інтелектуальних рекомендаційних систем відеоігор.

Об’єкт роботи – процес рекомендації відеоігор на основі переваг користувача та характеристик відеоігор.

Предмет роботи – методи, технології та програмні засоби, що використовуються для аналізу вподобань користувача та формування персоналізованих рекомендацій відеоігор.

Метою бакалаврської кваліфікаційної роботи є підвищення ефективності процесу підбору відеоігор для користувача за рахунок впровадження інтелектуальних алгоритмів рекомендацій.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та додатків. У першому розділі здійснено аналіз існуючих аналогів систем рекомендацій відеоігор. У другому розділі розглядаються методи та алгоритми для розробки рекомендаційних систем. У третьому розділі описано моделювання та проектування рекомендаційної системи відеоігор. У четвертому розділі розглянуто та описано програмну реалізацію застосунку рекомендаційної системи відеоігор.

Бакалаврська кваліфікаційна робота містить 76 сторінок (без додатків), 21 рисунків, 30 джерел та 10 додатків.

Ключові слова: рекомендаційні системи, методи штучного інтелекту, аналіз даних, збір даних, відеоігри.

ABSTRACT

for bachelor's qualification work
of a student of group 401 of Petro Mohyla Black Sea National University

Yevhenii Bardovskyi

Theme: «Application for video games recommendations based on platform .NET»

The relevance of creating an application for video game recommendations is conditioned by the intensive development of the video game industry. The problem of choosing suitable video games for a specific user remains topical and requires the development of intelligent video game recommendation systems.

The object of research is the process of recommending video games based on user preferences and video game characteristics.

The subject of research is the methods, technologies, and software used to analyze user preferences and form personalized video game recommendations.

The goal of the bachelor's qualification work is to increase the efficiency of the video game selection process for the user by implementing intelligent recommendation algorithms.

The explanatory note consists of an introduction, four chapters, conclusions, and appendices. The first chapter analyzes existing analogs of video game recommendation systems. The second chapter examines methods and algorithms for developing recommendation systems. The third chapter describes the modeling and design of the video game recommendation system. The fourth chapter reviews and describes the software implementation of the video game recommendation system application.

The bachelor's qualification work contains 76 pages (without appendices), 21 figures, 30 sources, and 10 appendices.

Keywords: recommendation systems, artificial intelligence methods, data analysis, data collection, video games.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БКР – бакалаврська кваліфікаційна робота

ПК – персональний комп’ютер

SVD – singular value decomposition

TF – term frequency

IDF – inverse document frequency

CSV – comma separated values

JSON – JavaScript object notation

UI – user interface

WPF – Windows Presentation Foundation

MVVM – Model-View-ViewModel

URL – Uniform Resource Locator

API – application programming interface

LINQ – Language Integrated Query

ВСТУП

Актуальність створення застосунку для рекомендацій відеоігор є досить важливою в сучасних умовах. Цифрова ера, в якій ми живемо, привела до безпрецедентного росту ігрової індустрії. За оцінками, щорічний обсяг глобального ринку відеоігор досягає сотень мільярдів доларів, а число гравців по всьому світу сягає мільярдів. Та разом із зростанням різноманітності та кількості відеоігор, користувачі стикаються з проблемою вибору. Яку гру спробувати наступною? Чи є гра, яка б відповідала конкретним уподобанням користувача? Як знайти нові ігри, що відповідають інтересам гравця? Ці питання стають все більш актуальними. Відповідно, розробка застосунку для рекомендацій відеоігор може стати важливим внеском у розвиток ігрової індустрії, покращуючи досвід користувачів і допомагаючи їм знайти ігри, що найбільше відповідають їхнім інтересам.

Мета дослідження полягає у підвищенні ефективності процесу підбору відеоігор для користувача за рахунок впровадження інтелектуальних алгоритмів рекомендацій.

Відповідно до поставленої мети було сформульовано завдання дослідження:

1) Проаналізувати існуючі системи рекомендацій відеоігор, визначити їхні переваги та недоліки.

2) Розробити концепцію рекомендаційного застосунку, що включає визначення основних функцій та вимог до нього.

3) Розробити архітектуру застосунку на основі платформи .NET.

4) Реалізувати основні функції застосунку, включаючи механізм рекомендацій.

Об’єкт дослідження – процес рекомендації відеоігор на основі переваг користувача та характеристик відеоігор.

Предметом дослідження є методи, технології та програмні засоби, що використовуються для аналізу вподобань користувача та формування персоналізованих рекомендацій відеоігор.

Методологічною основою дослідження є комп’ютерні, аналітичні та загальнонаукові методи, які дозволили вивчити предмет та об’єкт дослідження, дослідити процеси та методи систем рекомендацій відеоігор.

Практичне значення даного дослідження полягає в розробці застосунку, який допоможе гравцям знаходити відеоігри, що найкраще відповідають їхнім інтересам і вподобанням. Це може покращити ігровий досвід користувачів, зробити вибір ігор більш персоналізованим та ефективним.

1 АНАЛІЗ ВІДЕОІГОР ТА РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1 Опис сфери відеоігор та рекомендаційних систем

Відеоігри є значною частиною сучасної культури та розваг. З часом, вони перетворилися з простого розважального контенту до складних інтерактивних творінь, що охоплюють широкий спектр жанрів, стилів та тематики. Відеоігри включають у себе елементи мистецтва, історії, музики, спорту, соціальної взаємодії, і навіть освіти.

Ринок відеоігор продовжує рости з кожним роком, з великими компаніями, такими як Sony, Microsoft, і Nintendo, що домінують в галузі консольних ігор, тоді як компанії, такі як EA, Ubisoft, і Activision Blizzard, займають провідні позиції в розробці ігор. Мобільні ігри також стали важливим сегментом ринку, з компаніями, такими як Supercell, King і Tencent, що лідирують в цій області.

Поява та розвиток сучасних персональних комп’ютерів, консолей та мобільних платформ призвели до еволюції продукції ігрових студій. Також з’явились нові способи розповсюдження ігор, а саме через онлайн платформи такі, як: Steam, GOG, Epic Games тощо. Через це ринок почав зростати швидше. За даними на жовтень 2020 року його об'єм зріс до 196,9 мільярдів доларів США (рис. 1.1), що на 19,6% більше, ніж у попередньому періоді. Пандемія Covid-19 спричинила збільшення частки цифрових покупок [1].

З ростом ринку відеоігор збільшувалась багатократно загальна кількість відеоігор. Сьогодні існує понад 3 мільйона різних відеоігор загалом (включаючи мобільні відеоігри) [2]. У такому випадку нерідко виникає ситуація, коли людина не знає в яку відеогру йому пограти. Тому виникають різні системи рекомендацій відеоігор, які допомагають майбутньому гравцю підібрати відеоігри, які йому би потенційно сподобались.

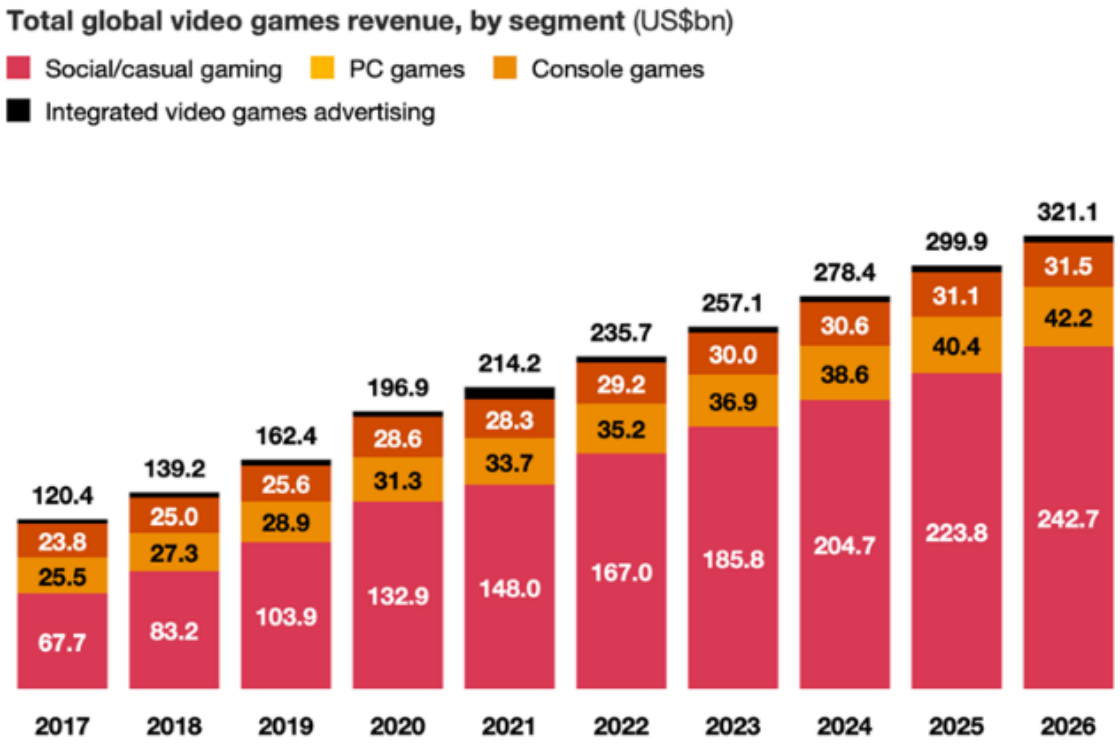

Рисунок 1.1 – Загальний дохід ігрової індустрії у світі та прогноз на 2023-2026 рр.

Рекомендаційні системи — це важливий інструмент в електронній комерції та інтернет-сервісах, від онлайн-магазинів до відеоігрових платформ. Вони допомагають користувачам знайти продукти або сервіси, які їм найімовірніше сподобаються, на основі інформації про їхні попередні взаємодії, відгуки або поведінку. Сьогодні існують 4 основних типи рекомендаційних систем [3]:

– системи, засновані на популярності контенту (Popular Based);
– системи, засновані на фільтрації за вмістом (Content Based);
– системи, засновані на колаборативній фільтрації (Collaborative Filtering);
– гібридні системи.

Системи, засновані на популярності контенту (Popular Based) - це найпростіший вид рекомендаційної системи. Цей тип системи рекомендує найбільш популярні елементи всім користувачам [4]. Цей підхід не враховує індивідуальні вподобання користувача, тому всі користувачі отримують однакові рекомендації. Це може бути корисно для нових користувачів, які ще не мають

історії взаємодії з системою, що відомо як проблема "холодного старту" в рекомендаційних системах. Однак, цей підхід має свої обмеження. Оскільки він не враховує індивідуальні вподобання, він може не бути дуже ефективним для користувачів з унікальними вподобаннями.

Системи, засновані на фільтрації за вмістом (Content Based) – це такі системи, які працюють, аналізуючи інформацію про предмети, які користувач вже оцінив позитивно [3]. Ця система рекомендації знаходить подібні предмети за допомогою порівняння їхніх характеристик. Цей тип системи може працювати добре для користувачів з дуже специфічними вподобаннями, оскільки він забезпечує персоналізовані рекомендації на основі їхніх індивідуальних вподобань. Однак, він може страждати від "проблеми персоналізації", коли користувачам рекомендуються лише дуже схожі елементи, і вони можуть пропустити багато інших цікавих елементів, які не підходять під їхній строгий профіль вподобань.

Системи, засновані на колаборативній фільтрації (Collaborative Filtering) – це системи, які враховують поведінку та вподобання широкої аудиторії користувачів, щоб здійснити прогнози для конкретного користувача [5]. Ці системи використовують матрицю оцінок користувача для різних елементів і потім використовують різні техніки для визначення схожості між користувачами або елементами та прогнозування оцінок для непереглянутих елементів [3]. Одним з основних переваг колаборативної фільтрації є те, що вона не вимагає додаткової інформації про користувачів або елементи, і може рекомендувати користувачам нові і різноманітні елементи. Однак, вона може страждати від "холодного старту" для нових користувачів або елементів, для яких відсутній історичний контекст оцінювання, а також від "проблеми масштабування", коли база користувачів або кількість елементів стає дуже великою [3].

Гібридні системи – це системи, які поєднують методи фільтрації за вмістом та колаборативної фільтрації з метою використання сильних сторін кожного з них та уникнення слабкостей, щоб отримати більш точні і персоналізовані рекомендації [3].

Діаграму з основними типами рекомендаційних систем наведено на рис. 1.2.

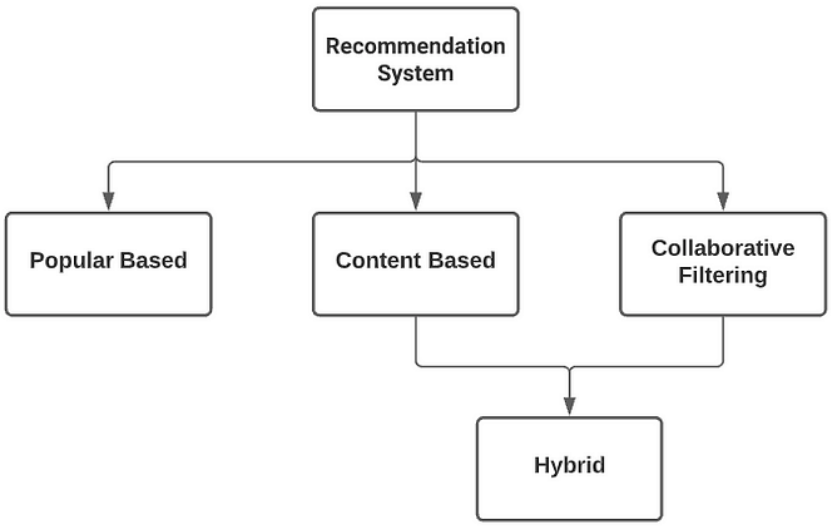

Рисунок 1.2 – Типи рекомендаційних систем

Системи рекомендацій відеоігор, як і будь-які системи рекомендацій, використовують різні алгоритми та стратегії для визначення найкращих рекомендацій. Вони можуть враховувати інтереси користувача, історію гри, відгуки, жанри та інші фактори. Важливою частиною таких систем є здатність адаптуватись до змін у вподобаннях та інтересах користувача, що дозволяє їм надавати актуальні та особисті рекомендації.

Як приклад можна взяти дуже складну систему рекомендацій веб-застосунку для перегляду відео YouTube. Ця рекомендаційна система використовує такі основні аспекти [6]:

– *персоналізація*: YouTube використовує дані про перегляди та взаємодію користувача з платформою (такі як лайки, дизлайки, додавання відео в закладки, кількість переглядів та частота активності) для персоналізації рекомендацій;

– *ранжування відео*: YouTube використовує алгоритми машинного навчання для ранжування відео в результатах пошуку та рекомендаційних списках;

– *контекстуальна рекомендація*: YouTube також враховує контекст користувача при наданні рекомендацій. Наприклад, якщо користувач часто переглядає відео про готування вранці, YouTube може рекомендувати подібні відео в цей час дня;

– *оптимізація для тривалого перегляду*: однією з ключових метрик для YouTube є тривалість перегляду. Тому алгоритми рекомендацій використовуються для максимізації загального часу, який користувач проводить, дивлячись відео на платформі;

– *врахування відгуків користувачів*: YouTube намагається врахувати відгуки користувачів на свої рекомендації. Наприклад, користувачі можуть вказати, що вони не хочуть бачити рекомендації від певного каналу, або що певне відео було нерелевантним;

– *використання нейронних мереж*: для обробки та аналізу великих об'ємів даних, що генеруються користувачами, YouTube використовує нейронні мережі та глибоке навчання, що дозволяє системі навчатися на основі великих об'ємів даних та вдосконалювати свої рекомендації з часом.

Але точні методи та алгоритми, які використовує компанія Google для системи рекомендацій відео на платформі YouTube, є комерційною таємницею.

Незважаючи на важливість та корисність, системи рекомендацій відеоігор все ще стикаються з рядом викликів. Деякі з них включають здатність розуміти комплексність і динаміку ігор, враховувати особистісні характеристики гравців, а також надавати рекомендації, які б є цікавими та новими, але при цьому не надто далекими від вподобань користувача.

Використання сучасних технік аналізу даних та машинного навчання може допомогти створити систему, яка здатна розуміти складність ігор та вподобань гравців, що дозволить надавати персоналізовані рекомендації відеоігор. Найбільш ефективні та розумні системи рекомендацій відеоігор є такі системи, у яких є велика база користувачів, яка може швидко доповнюватися та оновлюватися. Це дозволяє системі вчитися з поведінки користувачів, аналізувати їхні вибори і

враховувати їх в майбутніх рекомендаціях. Такі рекомендаційні системи вже вбудовані майже в усіх онлайн-сервісах, які надають послуги цифрової дистрибуції, наприклад Steam та Epic Games.

1.2 Огляд та аналіз наявних систем рекомендацій відеоігор

Сьогодні існує багато онлайн-сервісів, що займаються продажами та дистрибуцією відеоігор. Майже всі сервіси використовують вбудовані системи рекомендацій відеоігор. Найпопулярніші сервіси, що займаються продажами та дистрибуцією відеоігор [7]:

– *Steam* – це одна з найбільших платформ для цифрової дистрибуції відеоігор від великої компанії по розробці відеоігор Valve. Платформи, для яких продаються відеоігри: ПК (Windows, MacOS, Linux), Steam Deck

– *PlayStation Store* – магазин відеоігор для користувачів PlayStation від японської великої корпорації Sony. Платформи, для яких продаються відеоігри: консолі Play Station

– *Xbox Store* – цифровий магазин відеоігор від Microsoft. Платформи, для яких продаються відеоігри: консолі Xbox, ПК (Windows).

– *Epic Games Store* – це ще одна велика платформа для цифрового розповсюдження відеоігор, власництвом якої є компанія Epic Games, розробник популярної відеогри Fortnite. Платформи, для яких продаються відеоігри: ПК (Windows, MacOS, Linux).

– *GOG.com* – це цифрова платформа розповсюдження відеоігор, відома своєю політикою проти DRM (захист від копіювання). Платформи, для яких продаються відеоігри: ПК (Windows, MacOS, Linux).

– *Nintendo eShop* – офіційний магазин для купівлі цифрових ігор на консолях Nintendo.

Так як всі ці платформи мають базу користувачів, дані про вподобань користувачів, то у загальному системи рекомендацій відеоігор цих платформ є саме системи, основані на колаборативній фільтрації (Collaborative Filtering). Цей тип рекомендаційної системи є основою багатьох сучасних рекомендаційних систем відеоігор на великих платформах. Цей підхід використовує величезні набори даних про поведінку користувачів для прогнозування того, які відеоігри користувач може полюбити в майбутньому. Так як для такої рекомендаційної системи використовуються дуже великі дані про вподобань користувачів, то реалізація такої системи вимагає значних обчислювальних потужностей. Однак, з правильною оптимізацією та використанням сучасних технологій, такі системи можуть надавати дуже точні та персоналізовані рекомендації. Такі гіганти, як Microsoft, Sony, Nintendo, Valve та Epic Games можуть собі позволити обчислювані потужності у великих масштабах. Але малі компанії не можуть позволити собі такого. Ці компанії використовують інші типи рекомендаційних систем. Одна з них є система, основана на фільтрації за змістом (Content-Based Filtering). Цей тип рекомендаційної системи часто використовують системи, які не мають даних про вподобань користувачів, але мають дані про відеоігри та їхні характеристики.

Існують доволі велика кількість рекомендаційних систем відеоігор, основаних на фільтрації за змістом. Найпопулярніші з них:

- Games Finder;
- RAWG;
- Internet Game Database (IGDB);
- Steam Peek;
- Quantic Foundry;
- More Games Like.

Розглянемо детально вебсайт *Games Finder*. Games Finder - це вебсайт, що надає рекомендації відеоігор, що базуються на змісті. Цей сайт використовує алгоритми фільтрації за змістом для підбору ігор, які користувачу можуть

сподобатися на основі тих ігор, які він вже грав і оцінив. Головна сторінка цього вебсайту наведена на рис. 1.3.

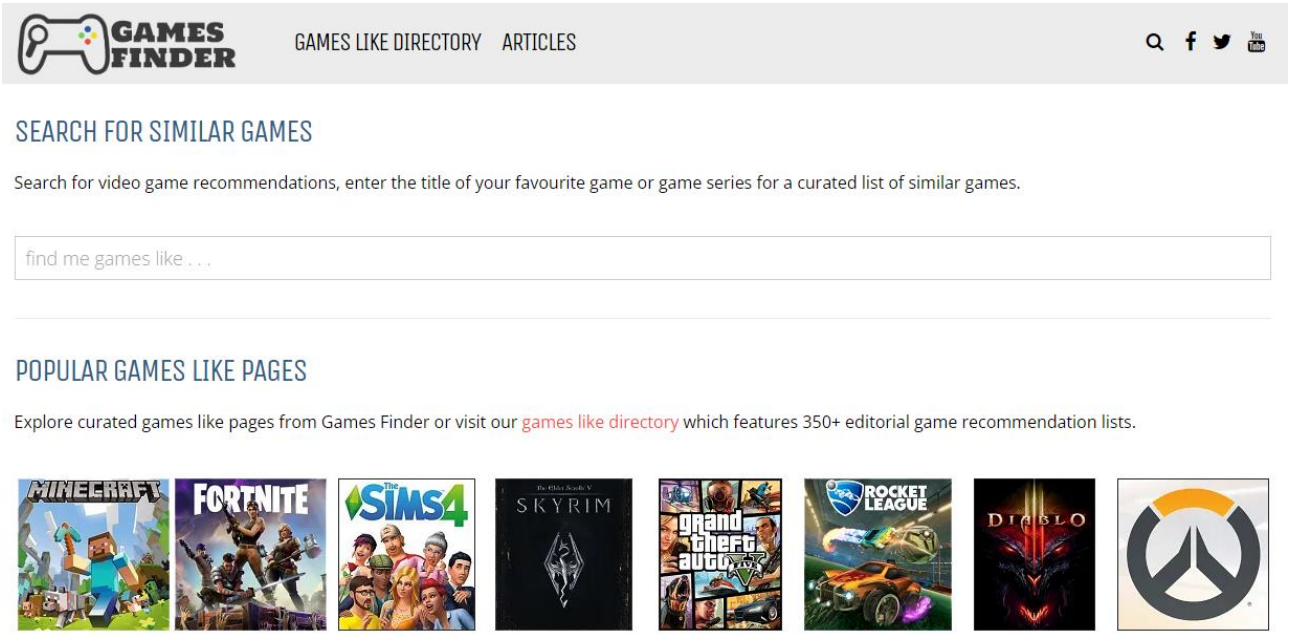

Рисунок 1.3 – Зовнішній вигляд веб-сайту Games Finder

При використані цього сервісу було взято відеогру *The Elder Scrolls V: Skyrim* - відкрита рольова відеогра в жанрі фентезі, сюжет якої обертається навколо героя головного гравця, який є "Дракононародженим" (Dragonborn), і його завдання полягає в тому, щоб знищити злого дракона, який, за пророцтвом, знищить світ. Було знайдемо 39 рекомендацій, які були зарекомендовані згідно з вподобаною відеогрою. Список перших трьох рекомендованих відеоігор:

1) *Gothic* – це рольова відеогра, яка відбувається в середньовічному фентезійному світі, де гравець виступає в ролі безіменного героя, який повинен виконувати різноманітні завдання і місії, борючись з чудовиськами і ворогами, використовуючи магію і зброю;

2) *Dragon Age: Origins* – це рольова відеогра в жанрі фентезі, яка відбувається в вигаданому світі Ferelden. Гра відома своєю глибокою системою

розвитку персонажа, великим кількістю виборів і наслідків, впливом на сюжет, та комплексним боєм, що вимагає стратегічного підходу;

3) *Risen* – це рольова відеогра, яка представляє собою фентезійний світ, що відкривається для дослідження гравцем. Сюжет "Risen" розповідає про безіменного героя, який потрапляє на тропічний острів після корабельної аварії. Острів, відомий як Faranga, покритий руїнами давньої цивілізації і наповнений різноманітними монстрами та іншими небезпеками.

За результатами рекомендацій, можна сказати що система рекомендацій Game Finder рекомендує відеоігри, які дійсно ідейно та геймплейно схожі на відоегру *The Elder Scrolls V: Skyrim*: схожий опис та однаковий жанр відеоігор – ролева гра (RPG). Результат рекомендації наведений на рис. 1.4.

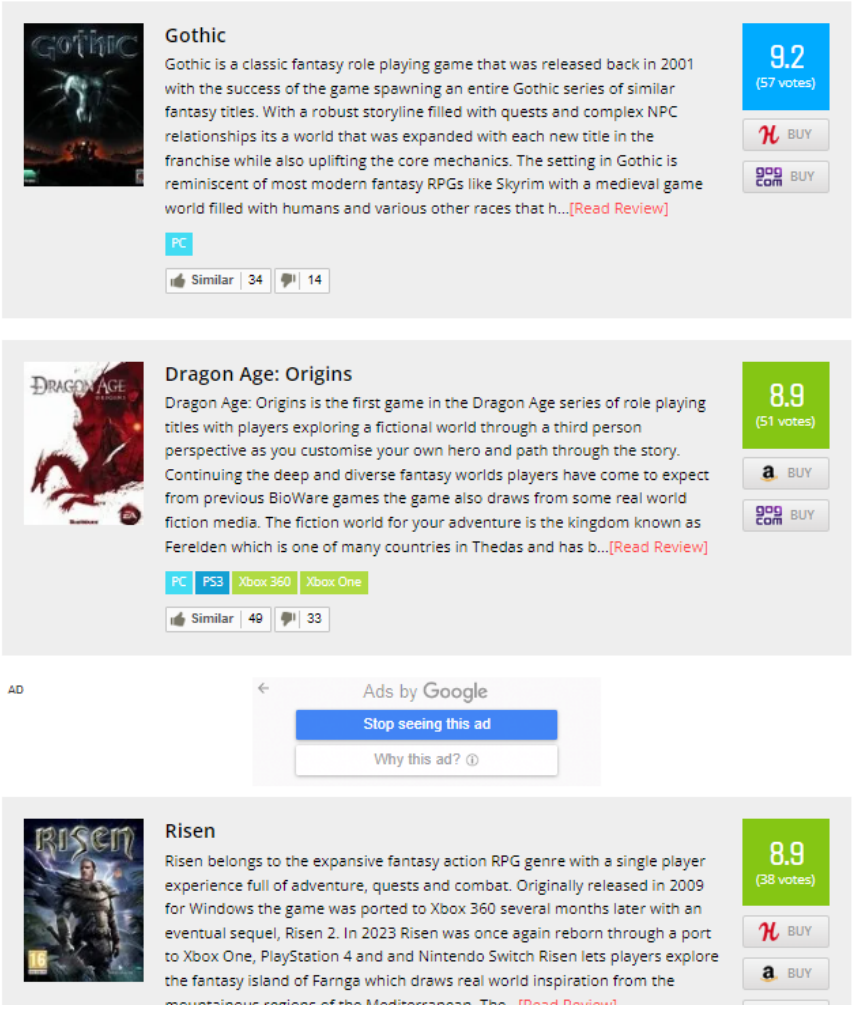

Рисунок 1.4 – Результати рекомендацій веб-сайту Games Finder згідно вподобаної відеогри The Elder Scrolls V: Skyrim

Інші веб-сайти, які реалізують системи рекомендацій відеоігор працюють схожим образом: є список ігор і є стрічка пошуку, після натискання на відеогру або вводу назви гри йде генерація рекомендацій та відправлення їх до користувача сервісу. Загальна їхня відмінність полягає у тому, що у деяких сервісах немає фільтрації ігор (Game Finder), а деякі призначені більше не як рекомендаційні система, а база відеоігор з вбудованою рекомендаційною системою (IGDB, RAWG).

Порівняємо результати рекомендацій вебсайту Games Finder згідно вподобаної відеогри The Elder Scrolls V: Skyrim з веб-сайтом RAWG. Результати рекомендацій наведено на рис. 1.5.

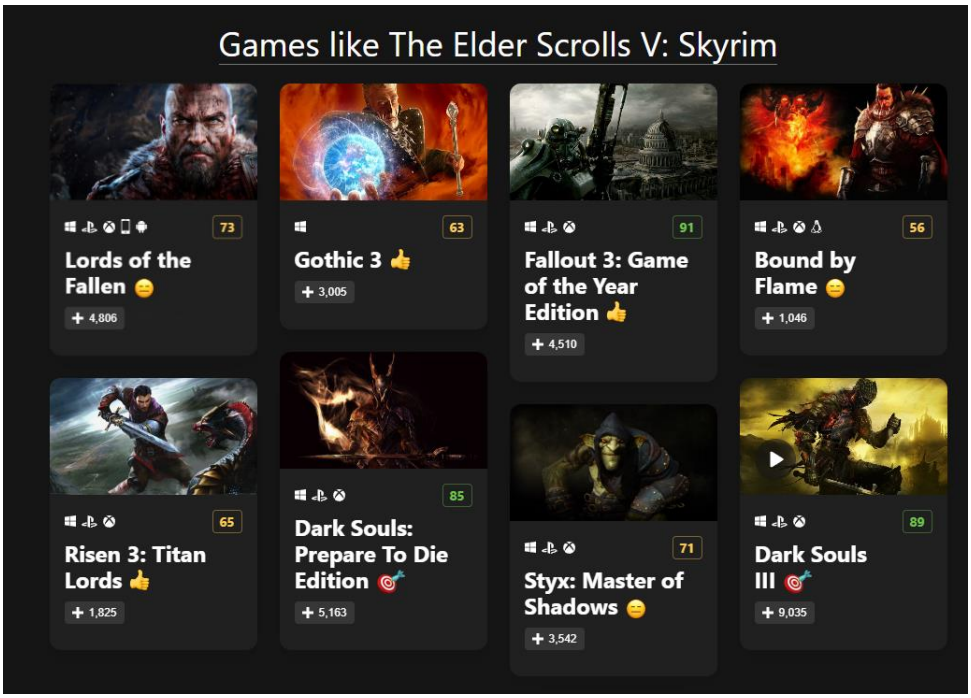

Рисунок 1.5 - Результати рекомендацій веб-сайту RAWG згідно вподобаної відеогри The Elder Scrolls V: Skyrim

Як можна побачити, результат відрізняється з результатом вебсайта Games Finder, але є деякі схожі моменти: в результатах присутні майже однакові відеоігри (тільки серії відеоігор відрізняються) Gothic 3 та Risen 3: Titan Lords, всі відеоігри мають однакову тематику середньовіччя, та однаковий жанр темного фентезі.

1.3 Постановка задачі

Розробка системи рекомендацій відеоігор є актуальною проблемою для досвідчених гравців у відеоігри. Особливо ця проблема набуває ще більш актуальності при різкому зростанні індустрії відеоігор. Здійснений аналіз ринку систем рекомендацій відеоігор дозволив висунути наступні вимоги до рекомендаційної системи відеоігор:

1) *персоналізація*: рекомендаційна система повинна надавати рекомендації, які враховують індивідуальні вподобання користувача. Це означає, що система повинна бути в змозі адаптуватися до вподобань кожного користувача і пропонувати відповідні ігри;

2) *різноманітність*: незважаючи на важливість персоналізації, важливо, щоб рекомендаційна система також забезпечувала достатню різноманітність рекомендацій. Це може допомогти гравцям відкрити для себе нові жанри або ігри, які вони б не розглянули без рекомендацій;

3) *прозорість*: Рекомендаційна система повинна бути прозорою і обґрунтованою. Гравці повинні мати змогу зрозуміти, чому певна гра була рекомендована, що може підвищити їхню довіру до системи;

4) *гнучкий функціонал пошуку та фільтрації бази відеоігор*: застосунок, який реалізує рекомендаційну систему відеоігор не повинен лише вміти добре рекомендувати ігри, але мати в собі гнучкий та ефективний функціонал пошуку та фільтрації відеоігор для того, щоб користувач мав змогу швидко та ефективно знаходити відеоігри для наступних рекомендацій.

Об’єкт дослідження – процес рекомендації відеоігор на основі переваг користувача та характеристик відеоігор.

Предметом дослідження є методи, технології та програмні засоби, що використовуються для аналізу вподобань користувача та формування персоналізованих рекомендацій відеоігор.

Мета дослідження полягає у підвищенні ефективності процесу підбору

відеоігор для користувача за рахунок впровадження інтелектуальних алгоритмів рекомендацій.

Відповідно до поставленої мети було сформульовано такі завдання:

1) проаналізувати існуючі системи рекомендацій відеоігор, визначити їхні переваги та недоліки;

2) розробити концепцію рекомендаційного застосунку, що включає визначення основних функцій та вимог до нього;

3) розробити архітектуру застосунку на основі платформи .NET;

4) реалізувати основні функції застосунку, включаючи механізм рекомендацій.

Висновок до розділу 1

Здійснений аналіз показав, що сучасний стан ігрової індустрії потребує системи рекомендацій відеоігор. У сучасних умовах постійного зростання кількості відеоігор, існує велика потреба у сервісах, які допомагають гравцям орієнтуватися в широкому асортименті та знаходити відеоігри відповідно до їхніх індивідуальних вподобань.

При аналізі наявних систем рекомендацій відеоігор було виявлено, що, хоча ряд систем вже використовуються, вони не завжди враховують усі потреби користувачів, такі як персоналізація, різноманітність, та прозорість. Також багато існуючих сервісів, які мають систему рекомендацій відеоігор, не мають зручного функціонала пошуку та фільтрації бази відеоігор. Все це обумовлює необхідність розробки нової рекомендаційної системи відеоігор, яка відповідала б повною мірою потребам гравців.

Виявлено, що можна поліпшити якість рекомендацій, враховуючи вподобання гравців, які вже зробили певний вибір. Такий механізм можна реалізувати за допомогою алгоритмів, заснованих на аналізу характеристик відеоігор. Таким чином можна реалізувати систему рекомендацій, яка буде рекомендувати відеоігри за змістом вподобаних відеоігор користувача.

2 МЕТОДИ ТА АЛГОРИТМИ ДЛЯ РОЗРОБКИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Методи для реалізації рекомендаційних систем

Як було описано у п. 1.1, існують чотири типи рекомендаційних систем, які реалізуються за допомогою різних методів та алгоритмів для досягнення більш персональних та ефективних рекомендацій. Опишемо стандартні методи та алгоритми для кожного типу рекомендаційної системи.

2.1.1 Методи та алгоритми для реалізації рекомендаційних систем, заснованих на популярності контенту

Рекомендаційні системи, засновані на популярності контенту є дуже простими у реалізації. Звичайно такі системи рекомендують найбільш популярні елементи, тобто елементи, які мають найбільший рейтинг та/або найбільшу кількість переглядів серед користувачів. В таких системах не існує даних про вподобання користувачів. Такі рекомендаційні системи можуть бути ефективні, коли система не має даних про вподобань користувачів, так як не можна вгадати вподобання користувачів не маючи про них дані, але можна рекомендувати найпопулярніші елементи у надії що користувачу щось з цього вподобається. Такі рекомендаційні системи часто використовуються в парі з рекомендаційними системами іншого типу, утворюючи єдину гібридну рекомендаційну систему.

Для реалізації стандартної рекомендаційної системи, заснованої на популярності контенту потрібна база даних або дата сет елементів з їхнім рейтингом та кількості користувацьких голосів за кожний елемент. Маючи такі дані, можна розрахувати значення *зважених рейтингів*, які використовуються для оцінки рейтингу кожного елементу. Ось формула для розрахунку зваженого рейтингового балу [3]:

$$W_R = \frac{v}{v+m} \cdot R + \frac{m}{v+m} \cdot C \quad (2.1)$$

де R – середній рейтинг елемента;

v – кількість голосів за елемент;

m – мінімальна кількість голосів, необхідна для включення елементу до списку популярних елементів (розраховується як 70-й процентиль всієї кількості голосів);

C – середній рейтинг для всього набору елементів.

Розглянемо приклад такої рекомендаційної системи. Нехай є таблиця елементів рекомендаційної системи (табл. 2.1) з такими атрибутами: номер елементу, середній рейтинг (R), кількість голосів (v).

Таблиця 2.1 – Елементи рекомендаційної системи

№ елементу	*Середній рейтинг (R)*	*Кількість голосів (v)*
1	4,7	3000
2	4,3	2000
3	4,7	4000
4	4,8	1500
5	3,5	1000

Мінімальна кількість голосів, необхідна для включення до списку (m), дорівнює 70-му процентилю всієї кількості голосів, що дорівнює у цьому випадку 2800, а середня оцінка всього набору елементів (C) становить 4,4. Так як значення m = 2800, то тільки елементи під номерами 1 та 3 будуть зарекомендовані. Використовуючи формулу (2.1) для елементів під номерами 1 та 3, отримаємо таку таблицю 2.2 з розрахованими зваженими рейтингами (W_R):

Таблиця 2.2 – Елементи та їхні зважені рейтинги

№ елементу	*Зважений рейтинг*
1	4,56
3	4,58

Як бачимо, за результатами розрахунків, 3 елемент буде перший у списку рекомендованих елементів, хоча 1 та 3 елементи мають однаковий середній рейтинг, але кількість голосів у 3 елемента більше на 1000 голосів. Це показує важливість зваженого рейтингу в рекомендаційних системах, основаних на популярності контенту, який враховує не тільки сам рейтинг, але й популярність елемента, що визначається кількістю голосів.

Варто зазначити, що хоча цей тип рекомендаційної системи не використовує дані вподобань користувача, він використовує рейтинги елементів, які надаються користувачами. Для цього варто мати базу користувачів, які будуть активно оцінювати елементи. Також рейтинг може змінюватись якщо елементи можуть змінюватись (наприклад онлайн відеоігри можуть отримувати оновлення, рейтинг серіалів може змінюватись в ході додавання нових сезонів). Тобто ця система рекомендацій не буде ефективною, якщо вона не є вбудованою у який-небудь онлайн сервіс, який містить базу користувачів.

2.1.2 Методи та алгоритми для реалізації рекомендаційних систем, заснованих на колаборативній фільтрації

Колаборативна фільтрація є одним з найпоширеніших типів рекомендаційних систем. Вона використовується для рекомендації продуктів користувачам на основі їхньої історії переглядів, відгуків та оцінок [8].

Взагалі є два основних підходи для реалізації рекомендаційних систем, заснованих на колаборативній фільтрації [3, 8]:

– memory-based;

– model-based.

Може здаватись, що мало варіацій існує для розробки рекомендаційних систем на основі колаборативній фільтрації, однак кожна варіація має майже безліч реалізацій.

Memory-based реалізація. Рекомендаційні системи, основані на колаборативній фільтрації цього типу використовують весь набір оцінок користувачів для обчислення рекомендацій. Цей підхід має дві реалізації: на основі користувачів (*user-based*) та на основі елементів (*item-based*) [3].

Колаборативна фільтрація на основі користувачів - це підхід, який заснований на визначенні схожості між користувачами. Схожість розраховується на основі оцінок, які користувачі виставили елементу. Система відшукає користувачів, які виставили подібні оцінки, і використовує їхні оцінки для генерації рекомендацій для активного користувача. Щоб розрахувати прогнозовану оцінку активного користувача для певного елемента, використовуються оцінки цього елемента іншими користувачами, взяті з урахуванням ступеня їхньої схожості з активним користувачем.

Однак ця реалізація має свої недоліки. Він не масштабується добре для великих наборів даних, оскільки потрібно зберігати матрицю схожості для всіх пар користувачів. Також якщо немає інформації про користувача, для якого потрібно згенерувати рекомендації, тоді не можна зробити прогноз, так як немає можливості розрахувати схожість оцінок користувачів [3, 8]. Така проблема називається холодним стартом (*cold-start problem*) [8]. Також такий підхід може працювати погано, якщо у користувачів немає спільних оцінок.

Колаборативна фільтрація на основі елементів – це варіант колаборативної фільтрації, що використовує схожість між елементами, а не між користувачами, для роботи прогнозу. Цей підхід був запропонований Amazon.com у 1998 році і добре працює в системах з великою кількістю користувачів [9]. Для кожного елемента, який активний користувач ще не оцінив, система шукає елементи, які цей користувач вже оцінив, і які є схожими на розглянутий елемент [9]. Прогнозована

оцінка розраховується як зважена сума оцінок активного користувача для схожих елементів, де ваги - це ступені схожості між елементами.

Цей підхід має декілька переваг. Він може працювати добре, навіть якщо користувачі мають дуже мало спільних оцінок. Крім того, матриця схожості між елементами змінюється повільніше, ніж матриця схожості між користувачами, тому її можна оновлювати менш часто, що у свою чергу призведе до покращення оптимізації системи [4]. Принцип роботи рекомендаційних систем, заснованих на колаборативній фільтрації на основі елементів наведено на рис. 2.1.

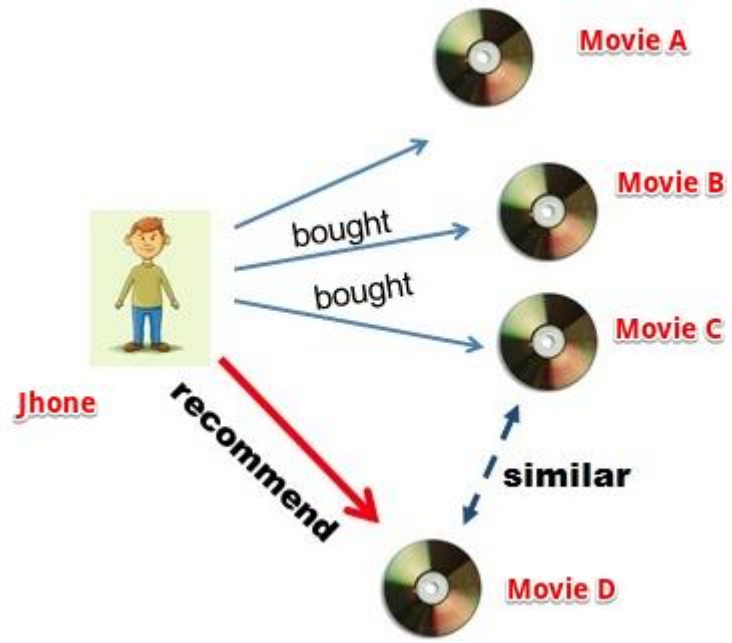

Рисунок 2.1 – Принцип роботи рекомендаційних систем, заснованих на колаборативній фільтрації на основі елементів

Не дивлячись на свої переваги, цей підхід також має спільну проблему з колаборативною фільтрацією на основі користувачів – холодний старт [10]. Для нових елементів, які ще не були оцінені жодним користувачем, система не може знайти схожі елементи та зробити прогноз.

Наведемо приклад використання стандартної системи рекомендації, основаної на колаборативної фільтрації на основі користувачів. Нехай є 5

користувачів і вони оцінювали 5 різних елементів. Отримаємо матрицю оцінок користувачів. Порожні клітинки означають, що користувач ще не оцінив цей елемент.

Таблиця 2.3 – Матриця оцінок користувачів

Користувач	*Елемент 1*	*Елемент 2*	*Елемент 3*	*Елемент 4*	*Елемент 5*
User 1	4	3		5	
User 2	5		3		4
User 3		4	5		3
User 4	3		4	5	
User 5		5		3	4

Наступний крок – це обчислення схожості користувчів згідно їхній оцінок. Існують багато способів визначення схожості елементів, але у даному прикладі буде розглянуто метод схожість косинуса. Для застосування цього методу заповнюємо порожні клітинки нулями. Розрахуємо схожість між User 1 та User 2, застосовуючи схожість косинуса. Для цього спочатку обчислимо скалярний добуток векторів оцінок користувачів [11]:

$$U_1 \cdot U_2 = 4 \cdot 5 + 3 \cdot 0 + 0 \cdot 3 + 5 \cdot 0 + 0 \cdot 4 = 20 \quad (2.2)$$

де U_1 та U_2 – вектори оцінок User 1 та User 2 відповідно.

Потім обчислюємо норму кожного вектора. Це можна зробити, взявши квадратний корінь із суми квадратів всіх елементів вектора [11]:

$$||U_1|| = \sqrt{4^2 + 3^2 + 0^2 + 5^2 + 0^2} = \sqrt{50} \quad (2.3)$$

$$||U_2|| = \sqrt{5^2 + 0^2 + 3^2 + 0^2 + 4^2} = \sqrt{50} \quad (2.4)$$

Подібність косинуса обчислюється як скалярний добуток двох векторів, поділений на добуток їх норм [11]:

$$\cos(U_1, U_2) = \frac{U_1 \cdot U_2}{||U_1|| \cdot ||U_2||} = \frac{20}{\sqrt{50} \cdot \sqrt{50}} = \frac{2}{5} = 0{,}4 \qquad (2.5)$$

Отже, схожість між User 1 та User 2 за схожістю косинуса дорівнює 0,4.

Повторивши попередні кроки розрахунку коефіцієнта схожості для всіх пар користувачів, отримаємо таку матрицю схожості:

Таблиця 2.4 – Матриця схожості користувачів

Користувач	*User 1*	*User 2*	*User 3*	*User 4*	*User 5*
User 1	1	0,4	0,24	0,74	0,6
User 2	0,4	1	0,54	0,54	0,32
User 3	0,24	0,54	1	0,4	0,64
User 4	0,74	0,54	0,4	1	0,3
User 5	0,6	0,32	0,64	0,3	1

Маючи цю матрицю можна спрогнозувати рекомендації для всіх користувачів матриці, наприклад для користувача User 1 рекомендації будуть згенеровані згідно вподобань користувача User 4, так як максимальним коефіцієнтом схожості для User 1, згідно матриці схожості є 0,74 (не враховуючи значення 1). Тобто для користувача User 1 будуть рекомендовані такі елементи у такому порядку: Елемент 4, Елемент 3, Елемент 1.

Model-based реалізація. Рекомендаційні системи, основані на колаборативній фільтрації можуть бути реалізовані багатьма способами [3]. Системи цього типу використовують різні алгоритми машинного навчання, статистичні аналізи і моделі даних для створення прогнозу. Ці методи зазвичай використовують певну модель для навчання на вхідних даних і потім використовують цю модель для прогнозування або визначення відсутніх значень [3]. Популярні методи та алгоритми model-based реалізації:

1. *Матрична факторизація*. Є одним з найпопулярніших і найефективніших методів model-based колаборативної фільтрації. Основна ідея цього методу полягає в розкладанні матриці користувацьких рейтингів на дві або більше матриць таким чином, що їхній добуток наближається до початкової матриці [12]. Одним з найвідоміших прикладів матричної факторизації є алгоритм *Singular Value Decomposition* (SVD). Він розкладає матрицю на три інші матриці: U, Σ і V^T, де U і V — ортогональні матриці, а Σ — діагональна матриця [13]. Цей метод часто використовується в системах рекомендацій, особливо після того, як він був використаний в переможній моделі Netflix Prize 2009 [14]. Ось кілька основних причин використання SVD:

– пониження розмірності даних, зберігаючи при цьому найбільш важливу інформацію;

– оптимізація обчислень за рахунок зменшення матриці;

– вирішення проблеми масштабування, змінюючи пропущені значення на розумні прогнози.

Матрична факторизація, хоча і є потужним інструментом для роботи з великими матрицями, має декілька недоліків:

– вимагає достатньої кількості даних для ефективної роботи;

– проблема холодного старту;

– процес факторизації вимагає значних обчислювальних ресурсів;

– матрична факторизація, особливо коли використовуються складні моделі, може призвести до перенавчання;

– моделі, засновані на матричній факторизації, часто важко інтерпретувати.

На рис. 2.2 можна побачити як працює матрична факторизація. З цього рисунку можна побачити, що матрицю User-item було розділено на 2 матриці: матриця користувачів та матриця елементів. Добуток цих 2 матриць є результатом головної матриці User-item.

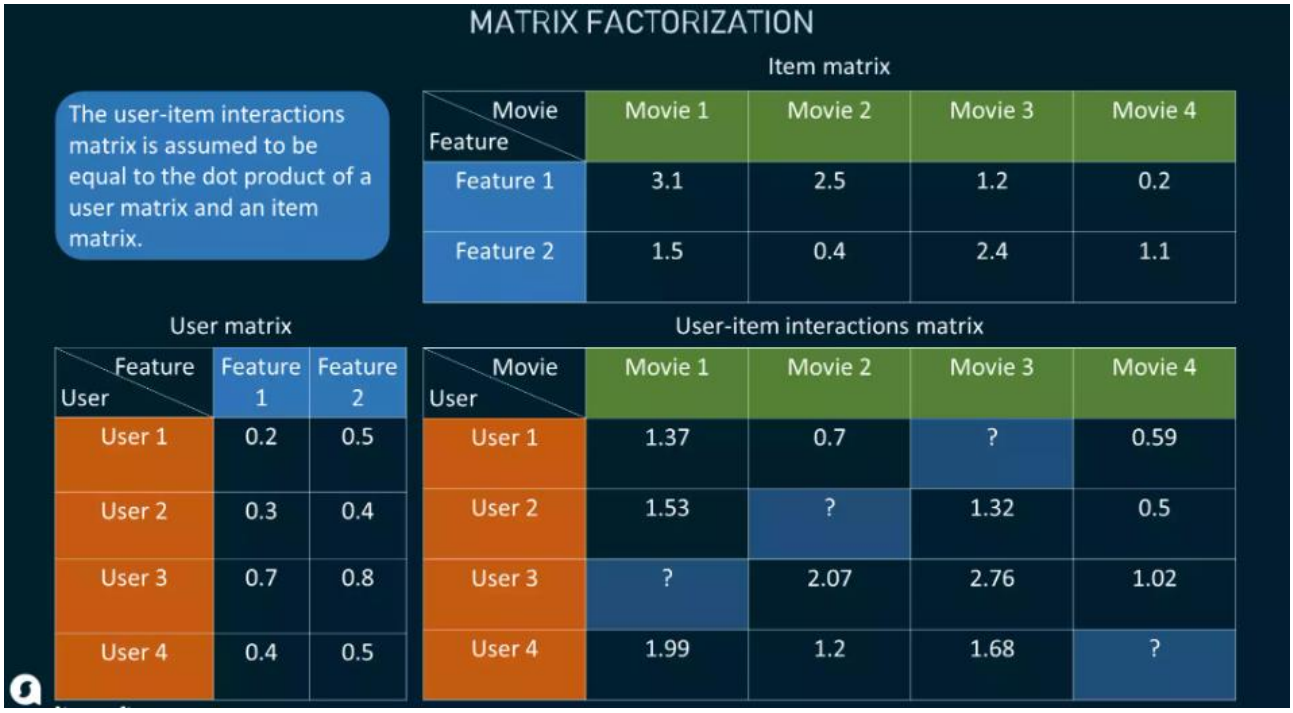

Рисунок 2.2 – Демонстрація роботи матричної факторизації

2. *Нейронні мережі*. Нейронні мережі також можуть бути використані для створення моделі в методах model-based колаборативної фільтрації [3]. Нейронні мережі є найдосконалішим підходом, який можна використати для усунення обмежень матричної факторизації [15]. Це дозволяє моделювати нелінійні взаємодії в даних і знаходити приховані закономірності в даних, які неможливо виявити інакше.

Подібно до того, як працює нейронна система людини, глибокі нейронні мережі складаються з нейронів (вузлів), розташованих у різних взаємопов’язаних шарах (вхідний рівень, ряд прихованих шарів і вихідний рівень) [15]. Нейронні мережі можна навчити отримувати функції безпосередньо з вмісту (відео, тексту, аудіо чи зображення) та/або давати рекомендації.

Наприклад, YouTube, використовує *Google Brain* — систему штучного інтелекту для надання відповідних рекомендацій щодо своїх відео [15]. Ця система рекомендацій складається з двох нейронних мереж: одна для створення кандидатів, яка приймає історію переглядів користувачів як вхідні дані та використовує

колаборативну фільтрацію для вибору відео, а друга для ранжування сотень відео в порядку.

Важливо зазначити, що всі ці методи мають свої переваги та недоліки. Наприклад, вони можуть враховувати складні взаємозв'язки між користувачами та предметами, але при цьому вони можуть бути витратними з точки зору обчислювальних ресурсів та часу на навчання моделі.

2.1.3 Методи та алгоритми для реалізації рекомендаційних систем, заснованих на фільтрації за змістом

Рекомендаційні системи, засновані на фільтрації за змістом використовують детальну інформацію про елементи для генерації рекомендацій. Вони аналізують атрибути елементів, які користувач оцінив позитивно в минулому, та рекомендують елементи, які мають подібні атрибути. Рекомендаційні системи цього типу також як й інші рекомендаційні системи можуть використовувати різні методи та алгоритми для реалізації.

Метод аналіз тексту. Більшість елементів має їхній опис, який може містити різні слова, які більш детально описують елемент. Ці слова можуть бути проаналізовані і використані для розробки профілю користувача. Методи, такі як TF-IDF, мішок слів (bag of words) та ембендинг слів (наприклад Word2Vec, GloVe) використовуються для перетворення тексту в числові вектори, які можна аналізувати [16].

Варто окремо розглянути метод TF-IDF (від англ. TF – term frequency, IDF – inverse document frequency). TF-IDF – статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів [17]. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших документах колекції. Показник TF-IDF використовується в задачах аналізу текстів та інформаційного пошуку. Його можна застосовувати як один з критеріїв

релевантності документа до пошукового запиту, а також при розрахунку міри спорідненості документів при кластеризації.

TF (*term frequency* – частота слова) – відношення числа входжень обраного слова до загальної кількості слів документа. Таким чином, оцінюється важливість слова в межах обраного документа.

$$TF = \frac{n_i}{\sum_k n_k} \tag{2.6}$$

де n_i є число входжень слова в документ, а в знаменнику – загальна кількість слів в документі.

IDF (*inverse document frequency* – обернена частота документа) – інверсія частоти, з якою слово зустрічається в документах колекції. Використання IDF зменшує вагу широковживаних слів.

$$IDF = log \frac{|D|}{|d_i \supset t_i|} \tag{2.7}$$

де $|D|$ – кількість документів колекції;

$|d_i \supset t_i|$ – кількість документів, в яких зустрічається слово t_i (коли $n_i \neq 0$).

$$TF\text{-}IDF = TF \cdot IDF \tag{2.8}$$

Більшу вагу TF-IDF отримають слова з високою частотою появи в межах документа та низькою частотою вживання в інших документах колекції.

Далі для знаходження схожості елементів за матрицею TF-IDF використовуються різні метрики схожості, але найбільш використаним є метрика схожості синуса.

Метод аналіз тексту, як й інші методи реалізації рекомендаційних систем, заснованих на фільтрації за змістом, має як і переваги, так і недоліки. Переваги метода аналіз тексту:

– *розширена можливість аналізу*. В порівнянні з підходами, які використовують лише метадані, аналіз тексту може виявити додаткову інформацію, яка міститься в описах елементів, і, отже, може забезпечити більш глибокий аналіз;

– *не залежить від структурованості даних***.** Він може ефективно працювати з неструктурованими даними, тоді як багато інших підходів вимагають структурованих метаданих;

– *різноманітність рекомендацій*. Цей метод може ефективно рекомендувати нові або менш популярні елементи, що відрізняється від колаборативної фільтрації, яка залежить від оцінок інших користувачів.

Недоліки метода аналіз тексту:

– *труднощі в обробці тексту*. Обробка тексту значно складніша, ніж обробка структурованих метаданих, і вимагає більш високих витрат ресурсів.

– *потенційно менш точні*. Цей метод може бути менш точним у порівнянні з тими, що використовують структуровані метадані, особливо якщо текстові описи не є добре написаними або не містять корисної інформації.

– *відсутність певних типів інформації*. На відміну від методів, що використовують графові структури, аналіз тексту не може використовувати інформацію про взаємозв'язки між елементами.

Метод використання ознак. Цей підхід до фільтрації за змістом залежить від використання структурованих даних або "ознак" (features), які можуть бути відомі про кожен елемент [18]. Наприклад, в контексті відеоігор, ознаки можуть включати жанр, розробника, рейтинг, тип відеоігри (одиночна, мультиплеєр), і так далі. В основному, використання ознак включає такі кроки: екстракція ознак, обчислення схожості та формування рекомендацій.

1. *Екстракція ознак*. Першим кроком є екстракція або вибір ознак, які можна використовувати для розрізнення елементів. Це може включати вибір відомих ознак з бази даних або використання методів машинного навчання для вибірки ознак з неструктурованих даних, таких як текстові описи.

2. *Обчислення схожості*. На основі цих ознак обчислюється схожість між елементами. Зазвичай використовуються метрики подібності, такі як Евклідова відстань або подібність косинуса, але можуть використовуватися і інші метрики в залежності від типу ознак.

3. *Формування рекомендацій*. Нарешті, використовуючи ці схожості, формуються рекомендації для користувача. Це може бути зроблено шляхом вибору елементів, які найбільше схожі на ті, які користувач вже оцінив високо, або шляхом використання більш складних алгоритмів, які враховують схожість між багатьма елементами одночасно.

Метод використання ознак є найпопулярнішим методом для рекомендаційних систем, основаних на фільтрації за змістом, оскільки цей метод дозволяє використовувати великі обсяги структурованих даних, що доступні про елементи, і може бути легко адаптовано для різних контекстів і типів елементів. Однак, метод використання ознак також вимагає, щоб відповідні ознаки були доступні, і щоб ці ознаки дійсно були корисними для прогнозування інтересів користувача.

Методи машинного навчання. Ці методи можуть бути використані в рекомендаційних системах, заснованих на фільтрації за змістом, для кращого розуміння змісту елементів і визначення, які з них найбільш підходять для конкретних користувачів [19]. Популярні методи машинного навчання для рекомендаційних систем, основаних на фільтрації за змістом наведено нижче.

1. *Класифікація*. Одним з основних підходів до машинного навчання, що використовується в цьому контексті, є класифікація. Тут ідея полягає в тому, щоб використовувати алгоритми класифікації, такі як дерева рішень, наївний Баєсовський класифікатор, машини опорних векторів, для навчання моделі, яка

може передбачити, чи сподобається користувачу конкретний елемент на основі його ознак.

2. *Регресія*. Іншим підходом є регресія, яка використовується для прогнозування дійсного числового значення, наприклад, оцінки користувача, на основі ознак елемента.

3. *Кластеризація*. Кластеризація може бути використана для групування схожих елементів разом. Користувачеві потім можуть бути рекомендовані елементи з тих груп (або кластерів), до яких належать елементи, які він уже оцінив високо.

4. *Глибоке навчання*. Цей підхід використовує нейронні мережі з багатьма шарами (відомі як глибокі нейронні мережі) для визначення складних зв'язків між ознаками елементів і інтересами користувачів. Глибоке навчання може бути особливо корисним, коли є багато ознак, або коли є складні зв'язки між ознаками.

Ці методи машинного навчання можуть бути використані як самостійно, так і в комбінації, залежно від конкретної ситуації. У будь-якому випадку, вони мають потенціал збільшити ефективність рекомендаційних систем, заснованих на фільтрації за змістом, допомагаючи краще зрозуміти зміст елементів і як він відповідає інтересам користувачів. Ці методи мають як і переваги, так і недоліки. Переваги методів машинного навчання:

– вивчення складних шаблонів даних, що не можуть бути легко зрозумілі людьми;
– автоматизація;
– адаптація до нових даних;
– персоналізація даних користувачів.

Недоліки методів машинного навчання:

– перенавчання моделі, якщо навчання було на основі обмеженого набору даних;
– необхідність великого обсягу даних;

– можуть вимагати значного часу та обчислювальних ресурсів, особливо для великих наборів даних;

– можуть бути складними та непрозорими.

2.2 Технології розробки рекомендаційної системи

Існує багато технологій, які використовуються для розробки рекомендаційних систем. Найпопулярнішим стеком технологій для розробки рекомендаційних систем є мова програмування Python та її бібліотеки: Scikit-learn, Surprise, TensorFlow, LightFM, Keras. Це обумовлено тим, що існує багато таких бібліотек на мові програмування Python, так як це найпопулярніша мова програмування у 2023 році та є найпопулярнішою мовою програмування для таких професій, як Data Scientist та Machine Learning Engineer [20]. Такі інженери як раз розробляють рекомендаційні системи. Але у рамках роботи БКР було вирішено розробляти застосунок для рекомендацій відеоігор на базі платформи .NET, так як ця платформа ідеально підходе для розробки десктопних застосунків та також має свої бібліотеки для розробки рекомендаційних систем. Почнемо з самої платформи .NET та її визначення.

.NET є відкритою платформою для розробки різноманітних типів застосунків, створеною компанією Microsoft. Вона включає в себе ряд мов програмування (наприклад, C#, F#, VB.NET), єдиної системи типів (Common Type System), виконавчого середовища (Common Language Runtime), бібліотеки класів, інтерфейси та API для розробки веб, мобільних, настільних, геймінгу та IoT застосунків, хмарних сервісів, а також машинного навчання [21]. Платформа .NET розроблена з метою полегшення процесу розробки програмного забезпечення, забезпечуючи високий рівень міжопераційності, продуктивності та безпеки. Вона включає в себе широкий набір бібліотек і інструментів, які дозволяють розробникам писати менше коду, автоматизувати багато процесів і зменшувати кількість помилок. Важливо відмітити, що є дві основні версії платформи: .NET

Framework і .NET Core. .NET Framework — це оригінальна версія платформи, яка була представлена в 2002 році і підтримується тільки на Windows. .NET Core — це більш нова, кросплатформена версія, що підтримується на Windows, Linux і macOS. У 2020 році Microsoft об'єднала ці дві версії в одну платформу під назвою .NET 5.0, що поєднує в собі переваги обох платформ. Сьогодні існує вже 7.0 версія платформи, реліз якої був 11 квітня, 2023 року, тобто ця платформа активно та швидко розвивається та оновлюється.

Мова програмування для розробки застосунку на платформі .NET була обрана найпопулярніша мова платформи – *C#*. C# є сучасною, об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft в рамках їх платформи .NET. Мова програмування C# є однією з основних мов, використовуваних у розробці програмного забезпечення на платформі .NET [22]. C# наслідує багато характеристик та синтаксису від мови C++ та Java, але має багато своїх власних інновацій. C# є статично типізованою мовою, що допомагає забезпечити безпеку типів та виявлення помилок на етапі компіляції. Вона також підтримує ряд основних парадигм програмування, включаючи об'єктно-орієнтоване, процедурне, та функціональне програмування. За допомогою C# розробляються різні типи програм, включаючи веб-додатки, мобільні додатки, настільні додатки, відеоігри (особливо з використанням движка Unity), та багато іншого. Сьогодні існує 11 версія мови програмування, реліз якої був 8 листопада, 2022 [22]. Це також говорить, що мова, як і платформа, розвивається та оновлюється активно.

Для завантаження додаткових пакетів (бібліотек) на платформі .NET використовувався *NuGet* - пакетним менеджером для платформи .NET. Він дозволяє розробникам поділитися та використовувати перевірені бібліотеки та пакети, що містять корисний код, який можна використати в своїх проектах. За допомогою цього пакетного менеджера, було завантажено інші бібліотеки, які використовувались для розробки рекомендаційної системи застосунку.

Що до бібліотек для розробки рекомендаційних систем на платформі .NET, то існує декілька бібліотек, які вирішують цю задачу.

ML.NET. Це відкритий проект від Microsoft, який надає розробникам .NET можливість використовувати машинне навчання в їх застосунках без необхідності мати глибокі знання в області машинного навчання [23]. Бібліотека надає можливість розробляти моделі прогнозування, бінарної класифікації, багатокласової класифікації, регресії та рекомендаційних систем. У ML.NET для реалізації рекомендаційних систем є такі класи:

– *TextFeaturizing*: використовується для векторизації тексту для розробки content-based рекомендаційних систем;

– *MatrixFactorizationTrainer*: використовується для матричної факторизації для розробки collaborative-filtering рекомендаційних систем;

– *TextLoader*: використовується для завантаження даних;

– *NormalizeMinMax*: використовується для мін-макс нормалізації даних;

– *OneHotEncoding*: використовується для обробки категорійних даних.

Хоча ML.NET є потужним інструментом для розробки інтелектуальних систем, таких як рекомендаційні системи, але він має власні значні недоліки перед іншими інструментами:

– *обмежений набір алгоритмів*. Порівняно з іншими фреймворками, такими як scikit-learn, TensorFlow, або PyTorch, ML.NET має менший набір алгоритмів, особливо для деяких задач машинного навчання.

– *відсутність підтримки деяких функцій*. Можуть виявитися випадки, коли ML.NET не підтримує деякі функції, доступні в інших фреймворках. Наприклад, підтримка роботи з глибоким навчанням обмежена.

– *документація та спільнота*. Хоча ML.NET постійно розвивається, його документація та спільнота ще не на рівні з такими мовами як Python. Може бути важко знайти відповіді на специфічні питання або отримати допомогу.

Accord.NET Framework. Це фреймворк для .NET, який надає широкий спектр наукових рутин для роботи з машинним навчанням, математикою, науковим

обчисленням і багато іншого [24]. Його ціль - надати стандартний фреймворк для наукових досліджень на мові C#, і він включає бібліотеки для обробки зображень, наукових обчислень, математики та багато іншого. Accord.NET включає більш 40 різних алгоритмів машинного навчання, включаючи методи для регресії, класифікації, кластеризації, а також деякі алгоритми для рекомендаційних систем. Загалом цей фреймворк є низькорівневим у порівнянні з ML.NET, так як ML.NET має більше абстракцій, а Accord.NET надає більш гнучкий доступ до алгоритмів машинного навчання і наукових обчислень, але при цьому може вимагати більше кодування і глибокого розуміння математики та алгоритмів машинного навчання. Через це Accord.NET є менш зручним для швидкої розробки прототипів рекомендаційних систем у порівняні з ML.NET. Однак, для досвідчених розробників, які шукають більше гнучкості та контролю над своїми алгоритмами, Accord.NET може бути дуже потужним інструментом. Так як цей фреймворк надає більше контролю над алгоритмами, одним з ключових переваг цього фреймворку також може бути оптимізація, що є досить вагомим фактором вибору фреймворку.

Accord.NET має також самі недоліки, як і ML.NET. Але одним з головних недоліків фреймворку є те, що сьогодні цей проєкт більше не підтримується спільнотою. Репозиторій на платформі GitHub є заархівованим 19 листопада, 2020 року, тобто його вже ніхто не може розробляти. Це обумовлено тим, що головний розробник, після майже 15 років розробки вирішив, що проєкт досяг своїх головних цілей.

Крім бібліотек, які надають алгоритми та методи для розробки рекомендаційних систем, також треба інструмент для зчитування вхідних даних системи. За звичай вхідні дані завжди представлені у вигляді файлу у форматі CSV (comma separated values). Існує дуже ефективна бібліотека для зчитування файлів у форматі CSV на платформі .NET – CsvHelper. Ця бібліотека може швидко та оптимізовано зчитувати дані, відразу ж створювати дата сет з об’єктів заздалегідь зазначеного класу [25].

Висновок до розділу 2

В рамках написання даного розділу було проведено аналіз різних алгоритмів і методів, які використовуються в рекомендаційних системах.

Спершу були розглянуті рекомендаційні системи, засновані на колаборативній фільтрації. Вони використовують оцінки і відгуки користувачів для створення рекомендацій, але можуть стикнутися з проблемами холодного старту, що означає, що новим користувачам або новим предметам може бути важко отримати рекомендації через відсутність історичних даних.

Далі були описані рекомендаційні системи, засновані на фільтрації за змістом, які використовують описові характеристики предметів для створення рекомендацій. Цей підхід добре працює для нових предметів або предметів з унікальними характеристиками, але він може не враховувати персональні вподобання користувача.

Також були розглянуті різні технології для розробки рекомендаційних систем, включаючи мову програмування Python та її бібліотеки, а також платформу .NET та відповідні фреймворки для розробки рекомендаційних систем, такі як ML.NET та Accord.NET.

На основі проведеного аналізу, стало зрозуміло, що немає універсального підходу до розробки рекомендаційних систем. Вибір конкретного методу або алгоритму залежить від специфіки задачі, доступних даних та обмежень ресурсів. Загалом, існує велика кількість технологій і методів, які можуть бути застосовані для розробки ефективної рекомендаційної системи.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ВІДЕОІГОР

Проаналізувавши всі типи рекомендаційних систем та актуальні методи та алгоритми для їх реалізації було вирішено розробити *рекомендаційну систему відеоігор*, основаної на *фільтрації за змістом*, використовуючи *метод аналізу текстів* описів відеоігор. Це обумовлено тим, що:

− немає доступу до великої бази користувачів, яка оновлюється у реальному часі;

− немає можливості обчислювати великі матриці оцінок користувачів;

− цей тип системи може рекомендувати непопулярні відеоігри;

− метод аналіз тексту може доволі точно знаходити схожі відеоігри за їхнім описом.

Таким чином, було визначено, що для цієї конкретної задачі рекомендаційна система, заснована на фільтрації за змістом з використанням аналізу тексту, буде найефективнішим рішенням. У такій системі аналіз тексту відіграє ключову роль у виявленні схожості між різними відеоіграми. Це здійснюється шляхом вивчення опису відеоігри, який зазвичай містить важливу інформацію про її зміст.

Для успішної реалізації такої рекомендаційної системи потрібно зробити:

1) знайти або розробити дані (датасет) відеоігор з їхнім описом;
2) зробити очищення даних;
3) спроєктувати рекомендаційну систему: вхідні дані алгоритму, кроки виконання алгоритму, результат виконання алгоритму;
4) тестування та можливу оптимізацію рекомендаційної системи.

3.1 Підготовка та опис вхідних даних рекомендаційної системи відеоігор

Як вхідні дані рекомендаційної системи було знайдено великий дата сет відеоігор платформи Steam використовуючи платформу Kaggle. Основні вимоги до дата сету були такі:

− містив назву відеоігор та їхній опис;

− містив такі дані, як жанр, розробник, загальні характеристики відеоігор для фільтрації та зручного пошуку користувачем у застосунку;

− містив багато унікальних відеоігор (більше 20 000);

− містив не тільки популярні відеоігри;

− повинен бути у форматі CSV.

Знайдений датасет на платформі Kaggle відповідав усім вимогам, які були вище зазначені.

Для маніпуляцією над датасетом було використано мову програмування Python та її бібліотеку Pandas. Сьогодні це найпопулярніший стек технологій для роботи з датасетами. Pandas – бібліотека Python, яка надає великі можливості для аналізу та маніпуляції з даними, працює швидко та легка у використанні. За допомогою цього стеку було оброблено датасет відеоігор.

Для аналізу датасету треба створити об’єкт класу DataFrame (так називається датасети у бібліотеки Pandas). Щоб створити датафрейм, треба зчитати файл датасету. Для цього існує метод *read_csv*, який приймає аргументом ім’я файлу. Після зчитування файлу можна робити маніпуляцію над датасетом. Спершу переглянемо перших та останніх 5 рядків. Перші 5 рядків датасету наведено на рис. 3.1. Останні 5 рядків датасету наведено на рис. 3.2.

url	types	name	desc_snippet	recent_reviews	all_reviews	release_date	developer	publisher	popular_tags
!0/DOOM/	app	DOOM	Now includes all three premium DLC packs (Unto...	Very Positive,(554),- 89% of the 554 user revi...	Very Positive,(42,550),- 92% of the 42,550 use...	May 12, 2016	id Software	Bethesda Softworks,Bethesda Softworks	FPS,Gore,Action,Demons,Shooter,First-Person,Gr...
)80/PLAY...	app	PLAYERUNKNOWN'S BATTLEGROUNDS	PLAYERUNKNOWN'S BATTLEGROUNDS is a battle roya...	Mixed,(6,214),- 49% of the 6,214 user reviews ...	Mixed,(836,608),- 49% of the 836,608 user revi...	Dec 21, 2017	PUBG Corporation	PUBG Corporation,PUBG Corporation	Survival,Shooter,Multiplayer,Battle Royale,PvP...
)90/BATT...	app	BATTLETECH	Take command of your own mercenary outfit of '...	Mixed,(166),- 54% of the 166 user reviews in t...	Mostly Positive,(7,030),- 71% of the 7,030 use...	Apr 24, 2018	Harebrained Schemes	Paradox Interactive,Paradox Interactive	Mechs,Strategy,Turn-Based,Turn-Based Tactics,S...
100/DayZ/	app	DayZ	The post-soviet country of Chernarus is struck...	Mixed,(932),- 57% of the 932 user reviews in t...	Mixed,(167,115),- 61% of the 167,115 user revi...	Dec 13, 2018	Bohemia Interactive	Bohemia Interactive,Bohemia Interactive	Survival,Zombies,Open World,Multiplayer,PvP,Ma...
)/EVE_On...	app	EVE Online	EVE Online is a community-driven spaceship MMO...	Mixed,(287),- 54% of the 287 user reviews in t...	Mostly Positive,(11,481),- 74% of the 11,481 u...	May 6, 2003	CCP	CCP,CCP	Space,Massively Multiplayer,Sci-fi,Sandbox,MMO...

Рисунок 3.1 – Перші 5 рядків датасету відеоігор

url	types	name	desc_snippet	recent_reviews	all_reviews	release_date	developer	publisher	popular_tags
5/Rock...	app	Rocksmith® 2014 Edition – Remastered – Sabaton...	NaN	NaN	NaN	Feb 12, 2019	Ubisoft - San Francisco	NaN	Casual,Simulation
2/Rock...	app	Rocksmith® 2014 Edition – Remastered – Stone T...	NaN	NaN	NaN	Feb 5, 2019	Ubisoft - San Francisco	NaN	Casual,Simulation
0/Fant...	app	Fantasy Grounds - Quests of Doom 4: A Midnight...	NaN	NaN	NaN	Jul 31, 2018	SmiteWorks USA, LLC	NaN	RPG,Indie,Strategy,Software,Turn-Based,Fantasy...
/Mega...	app	Mega Man X5 Sound Collection	NaN	NaN	NaN	Jul 24, 2018	CAPCOM CO., LTD	CAPCOM CO., LTD,CAPCOM CO., LTD	Action
10/Stor...	app	Stories In Stone	An RPG about a tribe exploring an ancient worl...	NaN	NaN	Aug 8, 2018	16 Bit Psych,Kyle B	Self-Publish,Self-Publish	RPG,Adventure

Рисунок 3.2 – Останні 5 рядків датасету відеоігор

Переглянемо загальну характеристику датасету. За допомогою метода *info* можна вивести на екран загальну характеристику датасету. Загальна характеристика датасету наведено на рис. 3.3.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40833 entries, 0 to 40832
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   url                     40833 non-null  object
 1   types                   40831 non-null  object
 2   name                    40817 non-null  object
 3   desc_snippet            27612 non-null  object
 4   recent_reviews          2706 non-null   object
 5   all_reviews             28470 non-null  object
 6   release_date            37654 non-null  object
 7   developer               40490 non-null  object
 8   publisher               35733 non-null  object
 9   popular_tags            37888 non-null  object
 10  game_details            40313 non-null  object
 11  languages               40797 non-null  object
 12  achievements            12194 non-null  float64
 13  genre                   40395 non-null  object
 14  game_description        37920 non-null  object
 15  mature_content          2897 non-null   object
 16  minimum_requirements    21069 non-null  object
 17  recommended_requirements  21075 non-null  object
 18  original_price          35522 non-null  object
 19  discount_price          14543 non-null  object
dtypes: float64(1), object(19)
memory usage: 6.2+ MB
```

Рисунок 3.3 – Загальна характеристика датасету

Як бачимо з характеристики з рисунків, усього є 40 833 записів та 20 колонок у датасеті. Розмір всього датасету становить 6.2 МБ (мегабайтів). Також на рис. 3.3 можна побачити, що деякі значення у комірках не заповнені. Це можна побачити на рис. 3.2, де у деяких комірках є значення NaN – порожнє значення. Для коректної роботи системи рекомендацій, треба видалити рядки з порожніми значеннями у колонці *desc_snipet*. Ця колонка містить короткий опис кожного елементу датасету. По цій колонці буде проводитись аналіз тексту для генерації рекомендацій відеоігор. Тому рядки з порожне значення у цій колонці є недопустимим. Колонка *types* містить тип елементу датасету. Для перегляду всіх можливих значень, які знаходяться у колонці *types*, було переглянуто унікальні значення у колонці за допомогою Pandas. Унікальні значення колонки *types*: app, bundle, sub, NaN. Було з’ясовано, що значення app вказує на те, що рядок – це відеоігра. Тому треба видалити всі рядки, у яких значення у колонці *types* не дорівнює значенню app. Також було видалено рядки, у яких значення у колонці *popular_tags* було порожнім. Ця колонка містить ключові слова елементів (для відеоігор це жанр, тип відеоігри,

основні вороги і т. д.). За цими ключовими словами у застосунку користувач буде фільтрувати список відеоігор, для того щоб користувач зміг ефективно шукати відеоігри.

Також для знаходження картинки для відеоігор потрібно знати ідентифікатор відеоігри для платформи Steam. Значення ідентифікатору можна отримати зі значення колонки *url*, яке містить посилання на сторінку магазину Steam. Для цього було написано окремо функцію *get_digit_string*, яка приймає параметром рядок, з якого треба знайти та повернути ідентифікатор відеоігри у Steam. Цей індетифікатор записується у нову колонку *app_id*.

Для того щоб значно скоротити розмірність датасету, було видалено непотрібні колонки датасету та залишити тільки потрібні дані для рекомендаційної системи відеоігор та застосунку. Колонки, які залишились: app_id, url, name, desc_snipet, release_date, developer, publisher, popular_tags, genre, original_price. Опис колонок очищеного датасету:

- app_id – ідентифікатор відеоігри в платформі Steam;
- ulr – посилання на сторінку магазину платформи Steam;
- name – назва відеоігри;
- desc_snipet – короткий опис відеоігри;
- release_date – дата виходу відеоігри;
- developer – розробник відеоігри;
- publisher – видавець відеоігри;
- popular_tags – ключові слова відеоігри;
- genre – жанр відеоігри;
- original_price – звичайна ціна відеоігри в магазині Steam.

Після очищення даних отримуємо датасет у розмірі 1.9 МБ з 24 774 рядками, які містять 10 колонок. Після очищення даних було збережено датасет у форматі CSV. Цей датасет у форматі CSV буде використовуватися застосунком для рекомендацій відеоігор.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24774 entries, 0 to 24773
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   app_id          24774 non-null  object
 1   url             24774 non-null  object
 2   name            24774 non-null  object
 3   desc_snippet    24774 non-null  object
 4   release_date    24594 non-null  object
 5   developer       24617 non-null  object
 6   publisher       24377 non-null  object
 7   popular_tags    24774 non-null  object
 8   genre           24507 non-null  object
 9   original_price  22265 non-null  object
dtypes: object(10)
memory usage: 1.9+ MB
```

Рисунок 3.4 – Характеристика датасету після очищення даних

Повний код очищення та підготовки даних наведено у додатку А.

3.2 Проєктування рекомендаційної системи відеоігор

Після опису та очищення вхідних даних для нашої системи у розділі 3.1, наступним кроком є проєктування самої рекомендаційної системи. Цей процес включає в себе кілька ключових елементів:

– опис архітектури рекомендаційної системи відеоігор;
– підготовка даних для рекомендаційної системи відеоігор;
– модель рекомендаційної системи відеоігор.

У цьому розділі буде детально розглянуто кожен з цих аспектів з метою створення ефективної та надійної рекомендаційної системи відеоігор.

Опис архітектури рекомендаційної системи відеоігор. Для розробки архітектури рекомендаційної системи було спроєктовано ключові модулі, які взаємодіють між собою для надання найвищої якості рекомендацій.

Модуль зчитування даних. Головна задача цього модуля – правильно та ефективно зчитувати дані. У нашому випадку цей модуль зчитує дані з CSV файлу,

тобто з датасету, який був остаточно сформований у п. 3.1. Але також варто зазначити, що було б краще реалізувати цей модуль так, щоб його можна розширити, щоб можна було б зчитувати дані не тільки з CSV файлу, а з інших джерел даних, наприклад з бази даних, з файлу формату JSON (JavaScript Object Notation). Таким чином модуль зчитування даних стає масштабованим, що надає можливість реалізувати зчитування даних по іншому, якщо джерело даних інше. Ця масштабованість модуля надає можливість у майбутньому розширити модуль зчитування даних для того, щоб легко пристосуватися до зміни джерела даних. Можна ще зазначити, що так проєктуються інформаційні системи за принципом *SOLID*. SOLID – це акронім для п’яти основних принципів об’єктного проєктування [26]. Кожна літера означає окремий принцип об’єктного проєктування. Ці принципи вперше були названі Робертом Мартіном – автор таких популярних книжок у сфері програмування, як «Чистий код. Створення, аналіз та рефакторінг» та «Чиста архітектура. Мистецтво розроблення програмного забезпечення». Другий принцип SOLID (англійська літера O) – принцип відкритості/закритості (*Open/closed principle*). Цей принцип зазначає, що програмні сутності повинні бути відкритими для розширення, але закритими для змін [26]. Тобто, має бути спосіб змінювати поведінку програмних модулів без потреби змінювати їхній вихідний код. Саме розділення модуля зчитування даних на підмодулі є гарним прикладом використання принципу відкритості/закритості так, як це дозволяє змінювати поведінку зчитування файлу в залежності від джерела даних.

Модуль обробки даних. Хоча у п. 3.1 було проведено очищення даних та підготовки їх до рекомендаційної системи відеоігор, є потрібність у обробки даних для певної рекомендаційної системи. Головна задача цього модуля – обробляти вхідні дані згідно з моделлю рекомендаційної системи. Також цей модуль краще реалізувати масштабованим, так як для різних рекомендаційних систем треба по різному обробляти дані. Взагалі всі модулі краще реалізувати масштабованими, так як у майбутньому може змінитись джерело даних та взагалі модель

рекомендаційної системи відеоігор. У випадку рекомендаційної системи відеоігор БКР модуль обробки даних обробляє дані у 2 етапи:

– токенізація тексту по словам;
– видалення стоп-слів.

Ці етапи обробляють дані згідно з обраною моделлю рекомендаційної системи відеоігор – рекомендаційна система, основана на фільтрації за змістом, використовуючи метод аналізу тексту.

Модуль генерації рекомендацій відеоігор. Це ключовий компонент рекомендаційної системи відеоігор. Модуль генерації рекомендацій використовує вхідні дані та результати обробки даних, щоб створювати рекомендації для користувачів. Він використовує методи та алгоритми аналізу тексту, а саме метод *TF-IDF* для векторизації описів відеоігор та метрику *подібності косинуса* для виявлення схожості між вхідними відеоіграми, які обрав користувач, та між всіма відеоіграми у датасеті. Точкою входу цього модуля є обрані користувачем відеоігри, які сподобались користувачу. На основі цих відеоігор цей модуль генерує рекомендації користувачу. Також модуль генерації рекомендацій відеоігор відповідає за те, у якому вигляді будуть виводитись результати рекомендацій відеоігор. У випадку поточної рекомендаційної системи відеоігор, цей модуль буде повертати відсортований список рекомендованих відеоігор за їхніми рекомендаційними оцінками. Цей модуль також буде надавати інформацію користувачу про релевантність рекомендації, тобто буде виводити оцінки рекомендацій (значення подібності рекомендованих відеоігор). Це надає коритсувачу прозорість системи, щоб користувач розумів пріоритетність рекомендованих відеоігор. Також цей модуль повинен бути гарно оптимізований, так як кількість даних є великою і можуть займати багато оперативної пам’яті.

Підготовка даних для рекомендаційної системи відеоігор. Як було зазначено зверху, за підготовку даних для рекомендаційної системи відповідає *модуль обробки даних*. Також було зазначено, що підготовка даних відбувається в 2 етапи,

а саме токенізація тексту по словам та видалення стоп-слів. Варто детально описати ці процеси.

Токенізація тексту по словам – стандартний метод NLP (Natural Language Processing), який поділяє текст на слова-компоненти або «токени» [27]. В контексті природної мови, «токен» часто відповідає слову. Наприклад, речення «Відеогра пригодницького жанру» було б токенізовано до списку {«Відеогра», «пригодницького», «жанру»}. В контексті рекомендаційної системи відеоігор, токенізація тексту по словам допомагає визначити ключові елементи в описі відеоігри, що може включати сюжет, тип головних ворогів, розробника, жанр та інші важливі деталі. Ці токени потім можна використовувати для визначення схожості між різними відеоіграми і для рекомендації відеоігор, що відповідають інтересам користувача. Також варто зазначити, що у поточній системі рекомендацій відеоігор токенізовані слова завжди будуть з маленької літери, так як значення токенів «Відеогра» та «відеогра» мають однакове смислове значення, але для системи це 2 різних токени.

Видалення стоп-слів. Стоп-слова, або шумові слова – це слова, які не несуть смислового навантаження, в результаті чого після обробки тексту ці слова видаляються для більшої ефективності аналізу тексту [27]. Звичайно, під стоп-словами мають на увазі артиклі, вигуки, сполучники і так далі. Вот приклад стоп слів англійської мови: about, me, ago, and, at, better, before, cannot, could, do і так далі. Цей процес спрощує аналіз тексту і підвищує ефективність моделі, оскільки система не витрачає ресурси на обробку нерелевантних слів. В результаті, рекомендаційна система може більш точно визначати, які відеоігри схожі між собою на основі їх описів, що призводить до більш точних рекомендацій для користувачів. Для поточної рекомендаційної системи в Інтернеті було знайдено файл-список, який містить 450 стоп слів, тому модуль обробки даних буде ефективно обробляти дані для більш точної та оптимізованої роботи рекомандаційної системи.

Модель рекомендаційної системи відеоігор. Проектування моделі рекомендаційної системи відеоігор - це ключовий етап створення рекомендаційної системи, який вимагає вибору відповідних методів та алгоритмів, які можуть ефективно працювати з даними відеоігор. Вибір моделі рекомендаційної системи є валживом кроком, так як існують багато моделей, які використовують багато різних методів та алгоритмів. Основні моделі рекомендаційних систем були описані в п. 2. В моделі рекомендаційної системи відеоігор БКР було вирішено використовувати модель, яка використовую аналіз тексту, зокрема аналізу описів відеоігор. Для цього треба було спроєктувати таку рекомендаційну систему, яка аналізує описи відеоігор, використовуючи методи NLP та вичислює схожість між відеоіграми за їх описом після обробки тексту. Основні етапи роботи моделі рекомендаційної системи відеоігор:

- навчання моделі рекомендаційної системи;
- векторизація описів відеоігор;
- обчислення подібності векторизованих описів відеоігор;
- генерування результатів рекомендацій.

Переглянемо детально ці процеси.

Навчання моделі рекомендаційної системи. Після формування даних для рекомендаційної системи відеоігор, модель повинна бути створена з цими вхідними даними. Там чином модель вже буде знати о даних і буде готова сформувати векторизовані описи відеоігор.

Векторизація описів відеоігор. Це ключовий процес у моделі рекомендаційної системи відеоігор БКР. Векторизація описів проводиться для формування числових векторів з описів відеоігор, щоб можна було визначити схожість описів відеоігор, в результаті якого буде згенеровано рекомендовані відеоігри. Існує багато методів векторизації тексту, але для даної рекомендаційної системи було обрано метод TF-IDF. Цей метод було описано детально у п. 2.1.3. Цей метод показує не тільки частоту використання токенів (слів) у документі, а й важливість токену у документі. Якщо в описів відеоігор певний токен зустрічається

багато разів, то його важливість не є значущою. З цього маємо, що певні токени, які можуть характеризувати відеогру є більш важливими ніж токени, які зустрічаються багато разів у документі. Таким чином покращується процес рекомендації відеоігор. Також варто зазначити, що формули для TF та IDF будуть іншими, ніж формули у п. 2.1.3.

Формула для TF:

$$TF(t,d) = 1 + \log\left(f(t,d)\right) \tag{3.1}$$

де t – термін (токен);

d – документ;

$f(t,d)$ – кількість разів, коли термін t з'являється в документі d.

Логарифмічна нормалізація використовується для вирівнювання впливу різних складових на кінцевий результат. У контексті TF, логарифмічна нормалізація допомагає зменшити вплив термінів, які з'являються дуже часто.

Формула для IDF:

$$IDF(t) = \log\left(\frac{N}{1+n_t}\right) \tag{3.2}$$

де t – термін (токен),

N – загальна кількість документів,

n_t – кількість документів, що містять термін t.

IDF Smooth - це варіант IDF, який використовується для згладжування результату за допомогою додавання одиниці до знаменника, щоб уникнути ділення на нуль.

Варто зазначити, що метод TF-IDF вже реалізований у фреймворку Accord.NET. Це дає змогу не реалізовувати цей метод з нуля, а використати його через фреймворк. Також цей метод у бібліотеці має різні імплементації у

залежності від обраних формул. Формули 3.1 та 3.2 вже реалізовані в методі TF-IDF фреймворку Accord.NET.

Обчислення подібності векторизованих описів відеоігор. Після векторизації описів відеоігор з'являється можливість обчислення схожості описів відеоігор з використання різних метрик. У даній рекомендаційної системи відеоігор використовується метрика схожість косинуса, так як підготовлені дані були векторизовані, а метрика схожість косинуса обчислює косинус кута між векторами. Для двох заданих векторів токенів (слів), A та B, косинус подібності, $\cos(\theta)$, представляється за допомогою скалярного добутку та довжини, як

$$\text{Подібність} = \cos(\theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum A_i B_i}{\sqrt{\sum A_i^2}\sqrt{\sum B_i^2}} \qquad (3.3)$$

де A_i та B_i – координати вектору A та B відповідно.

Таким чином рекомендаційна системи відеоігор БКР розраховує подібність відеоігор використовуючи векторизацію описів відеоігор та визначення їхньої подібності.

Метрика подібності косинуса також вже реалізована у фреймворку Accord.NET, що надає можливості легко вбудувати цю метрику у розрахунках подібності між описами відеоіграми.

Генерування результатів рекомендацій. На цьому етапі формуються остаточні результати рекомендаційного процесу. Сформована матриця подібності відеоігор сортується за значенням подібності відеоігор у порядку спадання. Це означає, що першими у списку будуть більш подібні відеоігри. Також з цього списку треба видаляти перший елемент, так як це буде та сама вподобана відеогра користувача. Якщо користувачу вподобалось більше, ніж однієї відеогри, тоді треба інакше формувати результати рекомендації. Для цього треба зробити такі кроки:

– сформулювати вектори зі значень подібності векторизованих описів відеоігор для всіх відеоігор, які вподобались користувачу;

– об’єднати сформульовані вектори подібності для кожної вподобаної відеоігри користувача, сформулювавши єдиний середній вектор подібності, використовуючи середнє арифметичне.

Таким чином формується результати рекомендаційної системи відеоігор для користувача, щоб він зміг передивитись рекомендовані відеоігри та подивитись на значення подібності відеоігри з відеогрою або відеоіграми, які вподобались користувачу.

Висновок до розділу 3

У третьому розділі БКР було проведено детальний аналіз та проектування рекомендаційної системи. Спочатку була виконана робота по знаходженню даних для рекомендаційної системи відеоігор. Після знаходження датасету відеоігор було виконано значну роботу по очищенню та підготовки даних. Очищення та підготовка даних відбувались з використанням мови програмування Python таких її бібліотек, як Pandas та NumPy. У результаті отримати готовий датасет відеоігор розмірністю 24 774 відеоігор.

Далі було виконано роботу з проєктування рекомендаційної системи відеоігор. Була спроєктована модель рекомендаційної системи відеоігор. Було детально описано структуру системи, включаючи всі 3 головні модуля рекомендаційної системи, їх взаємодію та принципи функціонування. Було вибрано тип рекомендаційної системи, методи та процеси обробки даних, методи обчислення схожості відеоігор та методи генерації результатів рекомендацій, необхідних для ефективного функціонування системи.

Загалом, можна зазначити, що розроблена модель рекомендаційної системи відеоігор дозволяє ефективно генерувати релевантні рекомендації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ РЕКОМЕНДАЦІЇ ВІДЕОІГОР

4.1 Архітектура проєкту

Для ефективної розробки застосунків слід приділити більше увазі проєктуванню архітектури. Правильно спроектована архітектура є основою успішного та масштабованого застосунку. Для розробки застосунку для рекомендацій відеоігор було використано середовище для розробки *Visual Studio 2022*. Visual Studio – це інтегроване середовище розробки, є творчим стартовим майданчиком, який можна використовувати для редагування, налагодження та збірки коду, а також для публікації програми. На додаток до стандартного редактора та відладчика, що надаються більшістю інтегрованих середовищ розробки, Visual Studio включає компілятори, засоби завершення коду, графічні конструктори та багато інших функцій для покращення процесу розробки програмного забезпечення [28]. Visual Studio є стандартним та ідеальним варіантом для розробки застосунків на базі платформи .NET. Тому було вирішено використовувати Visual Studio для розробки застосунку.

Сама архітектура застосунку для рекомендацій відеоігор складається з одного загального проєкту та трьох проєктів. Загальний проєкт у Visual Studio ще називають *рішенням*. Перелік підпроєктів загального рішення застосунку:

– *GameRecommendations.Shared* – проєкт, який містить загальні класи, які будуть використовуватись у багатьох інших проєктах;

– *GameRecommendations.RecommendationSystem* – проєкт, який містить реалізацію рекомендаційної системи відеоігор. Проєкт містить реалізацію всіх 3-х модулів рекомендаційної системи, що наведені у п. 3.2;

– *GameRecommendations* – проєкт, який містить реалізацію інтерфейсу користувача.

Вигляд списку проєктів у рішенні Visual Studio можна подивитись на рис. 4.1.

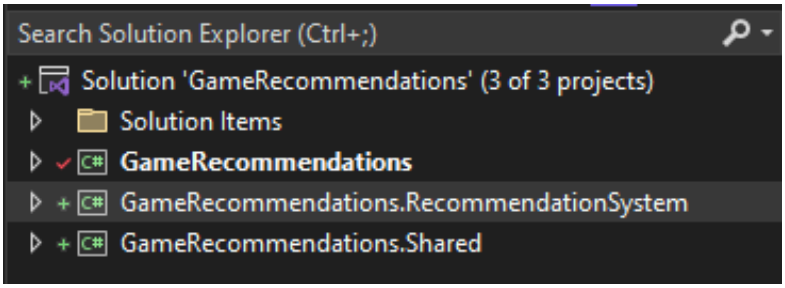

Рисунок 4.1 – Список проєктів у рішенні Visual Studio

4.2 Програмна реалізація рекомендаційної системи відеоігор

Розробка застосунку для рекомендацій відеоігор почалася саме з проєктів GameRecommendations.Shared та GameRecommendations.RecommendationSystem. Ці проєкти містять основну складову бізнес логіки застосунку, а саме реалізацію рекомендаційної системи відеоігор. Варто зазначити, що реалізація всього застосунку було здійснено згідно з парадигмою об'єктно-орієнтованого програмування (ООП) та з дотриманням всіх його принципів. Почнемо з GameRecommendations.Shared.

GameRecommendations.Shared. Цей проєкт містить 3 класи:

– *VideoGame* – доменна модель відеоігри, використовується всіма проєктами;

– *VideoGameCsvMap* – допоміжний клас для зчитування датасету відеоігор, який перетворює зчитаний рядок датасету в об'єкт класу VideoGame;

– *TagsConverter* – допоміжний клас для класу VideoGamesCsvMap, який відразу після зчитування рядка датасету перетворює рядок популярних тегів у список популярних тегів та записує їх відповідно до об'єкту класа VideoGame.

Варто детально розглянути модель відеоігри VideoGame:

```
public class VideoGame
{
    public int AppId { get; set; }

    public string Name { get; set; }

    public string Url { get; set; }
```

```
    public string ReleaseDate { get; set; }

    public string Developer { get; set; }

    [TypeConverter(typeof(TagsConverter))]
    public string[] PopularTags { get; set; }

    public string Genre { get; set; }

    public string Description { get; set; }

    public string Price { get; set; }

    public string? ImageUrl { get; set; }

    public bool IsLiked { get; set; }

    public int? RecommendationScore { get; set; }
}
```

Як зазначено у лістингу зверху, модель відеоігри має всі властивості з датасету відеоігор. Опис майже всіх властивостей моделі можна прочитати у п. 3.1. Але також є додаткові властивості, а саме ImageUrl, IsLiked та RecommendationScore. ImageUrl містить посилання на картинку відеоігри, IsLiked зазначає чи користувач визначив цю гру як вподобаною, RecommendationScore показує оцінку рекомендації, а саме оцінка подібності відеоігри між вподобаними відеоіграми користувача. Треба також зазначити, що властивості ImageUrl та RecommendationScore можуть не мати значення, тобто містити значення null. Це зазначається написанням знаку «?» після написання типу у властивості (наприклад int?). Відеогра може не мати своєї картинки, так як її картинка відповідного формату була не знайдена. Відеогра може не мати оцінки рекомендації на етапі вибору вподобаних відеоігор. Також використовується атрибут [TypeConverter(typeof(TagsConverter))] з використанням класу TagsConverter. Саме цей атрибут допомагає класу VideoGamesCsvMap перетворити рядок популярних тегів у список з рядків популярних тегів відеоігри.

Реалізацію класів VideoGamesCsvMap та TagsConverter наведено у додатку Б.

GameRecommendations.RecommendationSystem. Цей проєкт містить інтерфейси та класи, які описують та реалізують рекомендаційну систему відеоігор. Проєкт містить 2 папки, 3 інтерфейси та 3 класи:

1) папка Contracts:
   - інтерфейс IDataLoader;
   - інтерфейс IDataProcessor;
   - інтерфейс IRecommender.
2) папка Services:
   - клас FileDataLoader;
   - клас DataProcessor;
   - клас VideoGamesRecommender.

Інтерфейси були написані для того, щоб описати логіку взаємодії системи між модулями. Інтерфейси задають певний контракт класам, таким чином архітектура система буде залежати тільки від контрактів, від очікуваної поведінки модулів, а не від певної реалізації. Це дає змогу робити архітектуру системи більш масштабованою. Також інтерфейси допомагають у використанні зручної техніки dependency injection (DI) та у написанні модульних тестів, так як якщо клас, який треба протестувати, буде залежати тільки від інтерфейсів, тоді можна замінити реальний клас, який реалізує певний інтерфейс, на mock-об’єкт, який буде реалізовувати інтерфейс згідно з поведінкою модульного тесту.

Клас FileDataLoader реалізує інтерфейс IDataLoader та є реалізацією модуля зчитування даних рекомендаційної системи відеоігор. Цей клас має тільки 3 методи, які реалізують методи інтерфейсу IDataLoader:

– *LoadDataAsync(string source)* – асинхронно за допомогою бібліотеки CsvHelper зчитує весь датасет відеоігор, назва файлу якого вказана в параметрі source, у список відеоігор. Записує та повертає зчитані дані у вигляді списку класу List<VideoGame>;

– *GetLoadedData* – повертає зчитувані дані у вигляді списку. Якщо дані не були зчитані, метод кидає виняток InvalidOperationException, вказуючи на те, що дані не були зчитані;

– *LoadVideoGamesTags(IEnumerable<VideoGame> videoGames)* – формує список популярних тегів відеоігор з відеоігор, які були передані параметром videoGames.

Повна реалізація інтерфейсу IDataLoader та класу FileDataLoader наведена у додатку В.

Клас DataProcessor реалізує інтерфейс IDataProcessor<string[][]> та є реалізацією модуля обробки даних рекомендаційної системи відеоігор. Слід зазначити, що інтерфейс IDataProcessor є узагальненим, тобто при реалізації інтерфейсу треба вказувати конкретний тип узагальненого параметра. Це було зроблено для того, щоб інтерфейс не залежав від типу повертаючого значення у методах. У випадку рекомендаційної системи відеоігор БКР було задано тип *string[][]* як тип узагальненого параметра, так як ця рекомендаційна система буде робити токенізацію текстів описів відеоігор, отже на результат роботи класу DataProcessor повинен бути двовимірний масив з токенів для кожної відеоігри. Цей клас має тільки один єдиний метод, який реалізує метод інтерфейсу IDataProcessor: ProcessDataAsync(IEnumerable<VideoGame> data). Цей метод асинхронно виконує обробку даних та їх підготовку до самого модуля генерації рекомендацій. Спочатку йде завантаження стоп-слів з файлу english stopwords.txt. Цей файл буде знаходитись у папці Data, яка буде знаходитись у UI проєкті застосунку GamesRecommendations. Сам файл було завантажено з GitHub репозиторія. Потім йде процес токенізації текстів описів відеоігор. Після токенізації йде процес видалення стоп-слів з токенізованих текстів описів відеоігор. Слід зазначити, що процес видалення стоп-слів є паралельним, тобто використовується декілька потоків для видалення стоп-слів. Паралелізація досягається за допомогою методу розширення AsParallel. В кінці метод повертає токенізовані тексти описів відеоігор з видаленими стоп-словами.

Повна реалізація інтерфейсу IDataProcessor та класу DataProcessor наведено у додатку Г.

Клас VideoGamesRecommender реалізує інтерфейс IRecommender та його єдиний метод RecommendVideoGamesAsync. Цей клас реалізує модуль генерації рекомендацій відеоігор. Також цей клас має залежність від класів, які реалізують інтерфейси IDataLoader та IDataProcessor<string[][]> у вигляді параметрів конструктора класу VideoGamesRecommender. У випадку рекомендаційної системи БКР це класи FileDataLoader та DataProcessor відповідно. Варто зазначити, що клас VideoGamesRecommender має також 3 приватні методи, які використовуються у методі RecommendVideoGamesAsync. Метод RecommendVideoGamesAsync приймає параметром список відеоігор, які подобались користувачу, асинхронно генерує рекомендації для користувача у вигляді списку всіх відеоігор з їхньою оцінкою рекомендації згідно з вподобаних відеоігор користувача. Перелік приватних методів у порядку їх викликів в методі RecommendVideoGamesAsync:

– *GetOrCalculateTfIdfMatrixAsync* – метод, який асинхронно обчислює або повертає матрицю TF-IDF, якщо вона вже обчислена. Цей метод використовує клас, який реалізує інтерфейс IDataProcessor<string[][]> для асинхронної обробки даних, а саме токенізація текстів описів відеоігор. Після токенізації йде обчислення матриці TF-IDF. Вона обчислюється за допомогою об’єкта класу TFIDF, який реалізується у бібліотеці Accord.NET. Результатом виконання методу є запис TF-IDF матриці та її повернення;

– *CalculateSimilarityScoresVectors* – метод, який обчислює вектори подібностей згідно з вподобаних відеоігор користувача, викликається після методу GetOrCalculateTfIdfMatrixAsync. Метод CalculateSimilarityScoresVectors приймає 2 параметри: матрицю TF-IDF та вподобані відеоігри користувача. На кожну вподобану відеогру користувача обчислюється вектор подібності між всіма відеоіграми та вподобаною відеогрою. Подібність косинуса між відеоіграми обчислюється за допомогою об’єкта класу Cosine та його методу Similarity. У аргументах метода Similarity передаються рядок матриці TF-IDF за індексом вподобаної відеоігри та всі рядки матриці TF-IDF по черзі. Таким чином обчислюються вектори подібності між всіма відеоіграми та вподобаними

відеоіграми. Результатом методу є список з векторів подібності. Кількість векторів подібності у списку дорівнює кількості вподобаних відеоігор. Елементи у векторів подібності є кортежи у вигляді індексу відеоігри та значення подібності;

– *CalculateAverageSimilarityScoresVector* – метод, який усереднює вектори подібності відеоігор та перетворює їх у єдиний вектор подібності. Приймає параметром список векторів подібності відеоігор. Усереднення обчислюється звичайним методом середнього арифметичного. Також усереднене значення округляється та помножається на 100. Так як звичайне значення подібності косинуса дорівнює в межах [0;1], то не дуже зручно користувачу сприймати оцінку рекомендації у такому вигляді. Тому було вирішено використовувати оцінку рекомендації як значення в межах [0;100]. Результатом методу є повернення усередненого вектору оцінок рекомендації відеоігор.

Після виклику методу CalculateAverageSimilarityScoresVector метод RecommendVideoGamesAsync записує оцінки рекомендації до відеоігор та врешті решт повертає відсортований список рекомендованих відеоігор за оцінкою рекомендації за спаданням.

Повна реалізація інтерфейсу IRecommender та класу VideoGamesRecommender наведено у додатку Д.

4.3 Програмна реалізація інтерфейса застосунку для рекомендацій відеоігор

Реалізація інтерфейсу застосунку знаходиться у проєкті GamesRecommendations. Його структуру можна подивитись на рис. 4.2. Сам інтерфейс застосунку реалізовано на платформі Windows Presentation Foundation (WPF) – частина екосистеми платформи .NET та являє собою підсистему для побудови графічних інтерфейсів [29]. Є аналогом WinForms. WPF надає комплексний набір функцій розробки застосунків, які включають мову XAML,

елементи управління, прив'язку до даних, макет, двовимірну і тривимірну графіку, анімацію, стилі, шаблони, документи, мультимедіа, текст і типографічні функції.

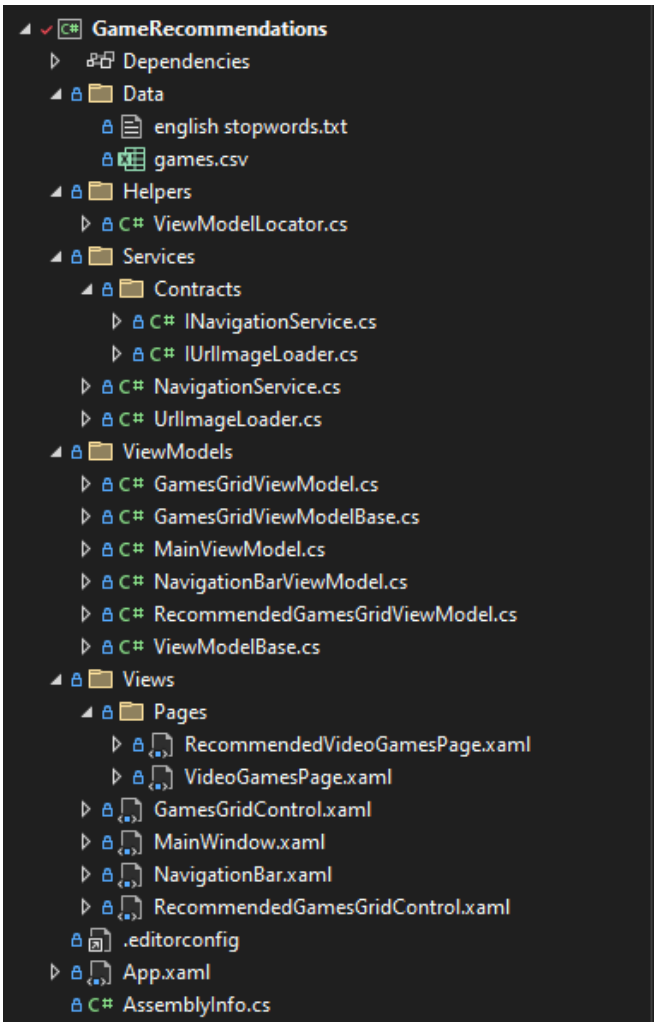

Рисунок 4.2 – Структура проєкта GamesRecommendations

При розробці графічного інтерфейсу користувача було використано бібліотеку MaterialDesignThemes – пакет, який додає та доповнює компоненти та стилі до WPF у популярному дизайні Material Design, який використовується в ОС Android. Це допомогло створити графічний інтерфейс красивим та лаконічним.

Також при розробці інтерфейсу користувача використовувався популярний патерн проєктування Model-View-ViewModel (MVVM). Цей патерн розділяється на три частини [30]:

– Model (укр. модель) – доменна модель, яка є частиною бізнес логіки;

– View (укр. представлення) – графічний інтерфейс (вікна, списки, кнопки і т.д.). Частіше за всього «слухає» зміни властивостей ViewModel та оновлює інтерфейс згідно цих змін;

– ViewModel (укр. модель представлення) – модель, яка тісно пов’язана з View. Після кожної зміни, які відбулись у моделі представлення, ця модель повинна оповістити представлення про зміни.

Схема роботи патерну MVVM представлена на рис. 4.3.

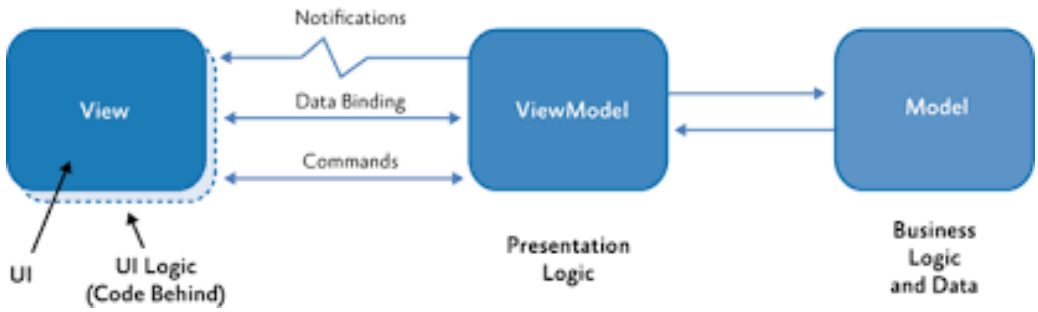

Рисунок 4.3 – Схема роботи патерну MVVM

Використовуючи WPF та MVVM можна ефективно розробляти UI застосунки, які будуть мати зрозумілу архітектуру. Також такі технології позволяють проєктувати графічний інтерфейс на окремі графічні частини.

Застосунок для рекомендацій відеоігор має такі графічні складові інтерфейсу:

1) навігаційна панель;
2) сторінка з відеоіграми:
   – панель пошуку та фільтрації відеоігор;
   – список відеоігор у вигляді карт з картинкою відеоігри та з можливістю зробити відеогру вподобаною;
   – панель зміни сторінок;
3) сторінка з рекомендованими відеоіграми:
   – список рекомендованих відеоігор у вигляді карт з картинкою відеоігри та значенням оцінки рекомендації;

– панель зміни сторінок.

Перед тим, як описувати графічні складові інтерфейсу застосунку, опишемо головний клас програми App.

Клас App наслідує клас Application та є точкою входу програми. У класу App є конструктор, який реєструє DI контейнер за допомогою бібліотеки Microsoft.Extensions.DependencyInjection та Microsoft.Extensions.Hosting. У DI контейнері реєструється головне представлення MainWindow, всі моделі представлення та сервіси як Singleton, тобто всі класи, які були зареєстровані у DI контейнері будуть мати лише один об’єкт. Так як застосунок водночас може лише один користувач використовувати, реєстрація залежностей як Singleton сервісів допоможе уникнути випадків надмірного перенавантаження системи. Також у конструкторі ініціалізується ViewModelLocator – це допоміжний статичний клас, який може повертати моделі представлення, які були зареєстровані в DI контейнері. Це розробилось для зручного задання моделі представлення до представлення за допомогою прив’язки даних у представлені атрибутом DataContext. Також у класу App є 2 перевизначених методи: OnStartup та OnExit, які визначають логіку запуску та вимкнення застосунку відповідно.

У файлі App.xaml підключається та задаються кольори теми Material Design. Також у цьому файлі визначається ViewModelLocator для того, щоб його можна було задати у DataContext.

Представлення MainWindow.xaml є головним представленням застосунку. Головне представлення містить такі контроли (компоненти):

– навігаційна панель;
– контрол Frame.

Контрол Frame використовується для відображення різних сторінок. У випадку застосунку БКР він використовується для відображення сторінки з відеоіграми та сторінки з рекомендованими відеоіграми.

Моделлю представлення MainWindow.xaml є MainViewModel, яка наслідує від базової моделі представлення ViewModelBase, яка реалізує логіку повідомлення

про зміни властивостей за допомогою реалізації інтерфейсу INotifyPropertyChanged та методу SetProperty. Повна реалізація ViewModelBase наведена у додатку Е. У моделі представлення MainViewModel є залежність від навігаційного сервісу, який реалізує інтерфейс INavigationService. Цей інтерфейс має подію на зміну сторінки OnPageChanged та метод зміни сторінки ChangePage(Page page), який приймає параметром сторінку. Повна реалізація інтерфейсу INavigationService та класу NavigationService наведено у додатку Ж. У конструкторі MainViewModel відбувається підпис на подію зміни сторінки та переключення на сторінку з відеоіграми (VideoGamesPage). При зміні сторінки змінюється властивість CurrentPage у моделі представлення MainViewModel. Таким чином реалізується переключення між сторінками застосунку у головному представлені, так як контрол Frame містить значення властивості CurrentPage. Повна реалізація MainViewModel наведена у додатку Е.

Перейдемо до реалізації графічних компонентів. Почнемо з навігаційної панелі.

Навігаційна панель. Вона реалізується представленням NavigationBar.xaml та моделлю представлення NavigationBarViewModel. У представлені головним компонентом виступає TabControl з двома елементами, які представляють елементи навігації по сторінкам з відеоіграми та з рекомендованими відеоіграми. Дизайн навігаційної панелі зображено на рис. 4.4. Модель представлення NavigationBarViewModel має залежність від сервісу навігації. Також ця модель представлення має властивість SelectedPageIndex яка вказує на індекс поточної сторінки. Коли задається значення цієї властивості, проводиться переключення сторінок за допомогою метода ChangePage відповідно з їхнім індексом. Властивість SelectedPageIndex дорівнює значенню властивості SelectedIndex контрола TabControl. Повна реалізація NavigationBarViewModel наведено у додатку Ж.

Рисунок 4.4 – Навігаційна панель

Сторінка з відеоіграми. Ця сторінка реалізується представленням VideoGamesPage.xaml, контролом GamesGridControl та його моделлю представлення GamesGridViewModel. VideoGamesPage.xaml – це не звичайне представлення, а саме сторінка (Page), для того щоб можливо було переключатись на неї. Ця сторінка містить один єдиний контрол GamesGridControl.

GamesGridControl є найскладнішим графічним компонентом застосунку для рекомендації відеоігор. Він реалізується за допомогою моделі представлення GamesGridViewModel, яка в свою чергу наслідує абстрактної моделі представлення GamesGridViewModelBase. Це розділення на базовий абстрактний клас та конкретний клас моделі представлення зробилось з метою перевикористання базової логіки відображення відеоігор та панелі переключення сторінок у моделях представлення GamesGridViewModel та RecommendedGamesGridViewModel.

GamesGridViewModelBase залежить від класу, який реалізує інтерфейс IUrlImageLoader, а саме від класу UrlImageLoader. Цей клас для кожної сторінки перегляду відеоігор записує URL картинки до відеоігри. Запис URL картинки відбувається за допомогою бібліотеки SteamGridDb.Net, яка в свою чергу використовує API SteamGridDb. SteamGridDb – це велика база різних картинок до відеоігор електронного магазину Steam. У класу UrlImageLoader є один єдиний метод, який реалізує метод інтерфейсу IUrlImageLoader – *GetImageUrlAsync(int appId)*. Цей метод по параметру appId, який вказує на ідентифікатор відеоігри у бази даних Steam, намагається повернути URL картинки відеоігри. Якщо такої картинки або відеоігри не було знайдено, тоді повертається URL картинки, яка вказує на те, що картинка до відеоігри відсутня.

Клас GamesGridViewModelBase має такі властивості та методи:

– *videoGamesToView* – поле, яке містить відеоігри, які будуть розділятися по сторінках;

– *PagedVideoGames* – властивість, яка містить відеоігри, які будуть відображатись на сторінці. Слід зазначити, що ця властивість має тип ObservableCollection<VideoGame>. ObservableCollection – це тип колекції, який повідомляє про зміни всередені колекції. Таким чином є можливість динамічного оновленя інтерфейсу сторінок відеоігор;

– *CurrentPage* – властивість, яка містить значення поточної сторінки відеоігор;

– *TotalPages* – властивість, яка вказує на загальну кількість сторінок відеоігор;

– *NextPageCommand* – властивість, яка вказує на команду, яка переключає на наступну сторінку відеоігор. Ініціалізується у конструкторі;

– *PreviousPageCommand* – властивість, яка вказує на команду, яка переключає на попереднью сторінку відеоігор;

– *CalculateTotalPages* – метод, який обчислює загальну кількість сторінок відеоігор;

– *LoadVideoGamesPageAsync* – метод, який асинхронно завантажує сторінку відеоігор за переданим параметром номеру сторінки. Цей метод використовує клас UrlImageLoader для завантаження URL картинок для відеоігор, які будуть показуватись на сторінці. Також в цьому методі викликається метод CalculateTotalPages для обчислення загального обсягу сторінок та задається властивість PagedVideoGames для відображення сторінки відеоігор;

– *NextPageAsync* – метод, який асинхронно переключає на наступну сторінку відеоігор. Викликається командою NextPageCommand;

– *PreviousPageAsync* – метод, який асинхронно переключає на попереднью сторінку відеоігор. Викликається командою PreviousPageCommand.

Клас GamesGridViewModel є моделью представлення контролу GamesGridControl та наслідується від базової моделі представлення

GamesGridViewModelBase. Має залежності від класів, які реалізують інтерфейс IDataLoader, IUrlImageLoader та INavigationService. У цьому класі клас FileDataLoader завантажує всі відеоігри з датасету коли в перший раз було завантажено сторінку з всіма відеоіграми (VideoGamesPage), клас NavigationService дозволяє визначити коли завантажувати відеоігри з датасету.

Повна реалізація класу GamesGridViewModelBase наведено у додатку И.

Клас GamesGridViewModel, окрім властивостей та методів, які реалізовані в базовому класі GamesGrdiViewModel, має такі властивості та методи:

– *AllTags* – властивість, яка містить в собі всі популярні теги відеоігор з датасету;

– *SelectedTags* – властивість, яка вказує на вибрані теги для фільтрації користувачем;

– *SearchQuery* – властивість, яка вказує на текст у полі пошуку;

– *FilterQuery* – властивість, яка вказує на текст тегу у полі вибору тегу для фільтрації;

– *ViewLikedVideoGames* – властивість, яка вказує чи показувати тільки вподобані відеоігри користувача;

– *SearchVideoGamesCommand* – властивість, яка вказує на команду, яка робить пошук відеоігор за текстом;

– *AddFilterCommand* – властивість, яка вказує на команду, яка додає тег для фільтрації відеоігор;

– *RemoveFilterCommand* – властивість, яка вказує на команду, яка прибирає тег для фільтрації відеоігор;

– *ToggleViewLikedVideoGamesCommand* – властивість, яка вказує на команду, яка переключає відображення тільки вподобаних відеоігор користувача;

– *LoadGamesAsync* – метод, який асинхронно завантажує відеоігри з датасету при першому переключенні на сторінку всіх відеоігор;

– *TrySearchAndApplyFiltersAsync* – метод, який асинхронно намагається робити пошук та фільтрувати відеоігри. Спочатку перевіряє на наявність тексту

пошуку, якщо присутній то проводиться пошук відеоігор за допомогою технології платформи .NET *LINQ* (Language Integrated Query). За допомогою LINQ виконуються майже всі маніпуляції над колекціями (проєкція, пошук, фільтрація і т.д.). Потім після пошуку проводиться фільтрація за допомогою метода ApplyFilters. Якщо текст пошуку відсутній, тоді проводиться тільки фільтрація. В кінці методу TrySearchAndApplyFiltersAsync виконується завантаження першої сторінки знайдених та відфільтрованих відеоігор. Метод викликається в команді SearchVideoGamesCommand;

– *ApplyFilters* – метод, який робить фільтрацію відеоігор. Фільтрація відбувається за вподобаними відеоіграми користувача та вибраних тегів для фільтрації відеоігор. Також якщо відсутні будь-які фільтри, тоді повертає список всіх відеоігор;

– *ClearSearchAsync* – метод, який асинхронно накладає фільтри до всіх відеоігор та завантажує першу сторінку відфільтрованих відеоігор. Таким чином пошук відеоігор за текстом не відбувається. Викликається в команді SearchVideoGamesCommand;

– *AddFilterAsync* – метод, який асинхронно додає тег, який передається як параметр, до вибраних тегів для фільтрації відеоігор. Цей також прибирає тег, який вибрав користувач зі загального списку тегів. В кінці метод фільтрує відеоігри за вибраними тегами та завантажує першу сторінку відфільтрованих відеоігор;

– *RemoveFilterAsync* – метод, який асинхронно прибирає вибраний тег для фільтрації відеоігор. Метод також додає прибраний тег до загального списку тегів. В кінці методу йде виклик методу TrySearchAndApplyFiltersAsync;

– *ToggleViewLikedVideoGamesAsync* – метод, який асинхронно переключає фільтрацію відеоігор, які вподобались користувачу.

Повна реалізація класу GamesGridViewModel наведено у додатку К.

Представлення GamesGridControl складається з панелі пошуку та фільтрації відеоігор, списку відеоігор у вигляді карт з можливістю зробити відеогру вподобаною та з панелі керування сторінками списку відеоігор.

Панель пошуку та фільтрації складається з таких компонентів:

– поле пошуку відеоігор за текстом;
– поле вибору тегів для фільтрації відеоігор;
– кнопка додання вибраного тегу для фільтрації відеоігор;
– горизонтальний список вибраних тегів для фільтрації відеоігор;
– перемикач для переключення відображення списку вподобаних відеоігор.

Кожний елемент списку відеоігор у вигляді карт з можливістю зробити відеогру вподобаною складається з таких компонентів:

– картинка до відеоігри;
– назва відеоігри;
– кнопка, яка робить відеогру вподобаною.

Панель керування сторінками списку відеоігор складається з таких компонентів:

– кнопка, яка переключає на попередню сторінку списку відеоігор;
– кнопка, яка переключає на наступну сторінку списку відеоігор;
– текст, який показує поточну сторінку та загальну кількість сторінок списку відеоігор.

Дизайн сторінки з всіма відеоіграми та її графічними компонентами наведено на рис. 4.5. Процес пошуку відеоігор за текстом наведено на рис. 4.6. Процес фільтрації відеоігор наведено на рис. 4.7. Процес фільтрації відеогра за вподобаними відеоіграми користувача наведено на рис. 4.8

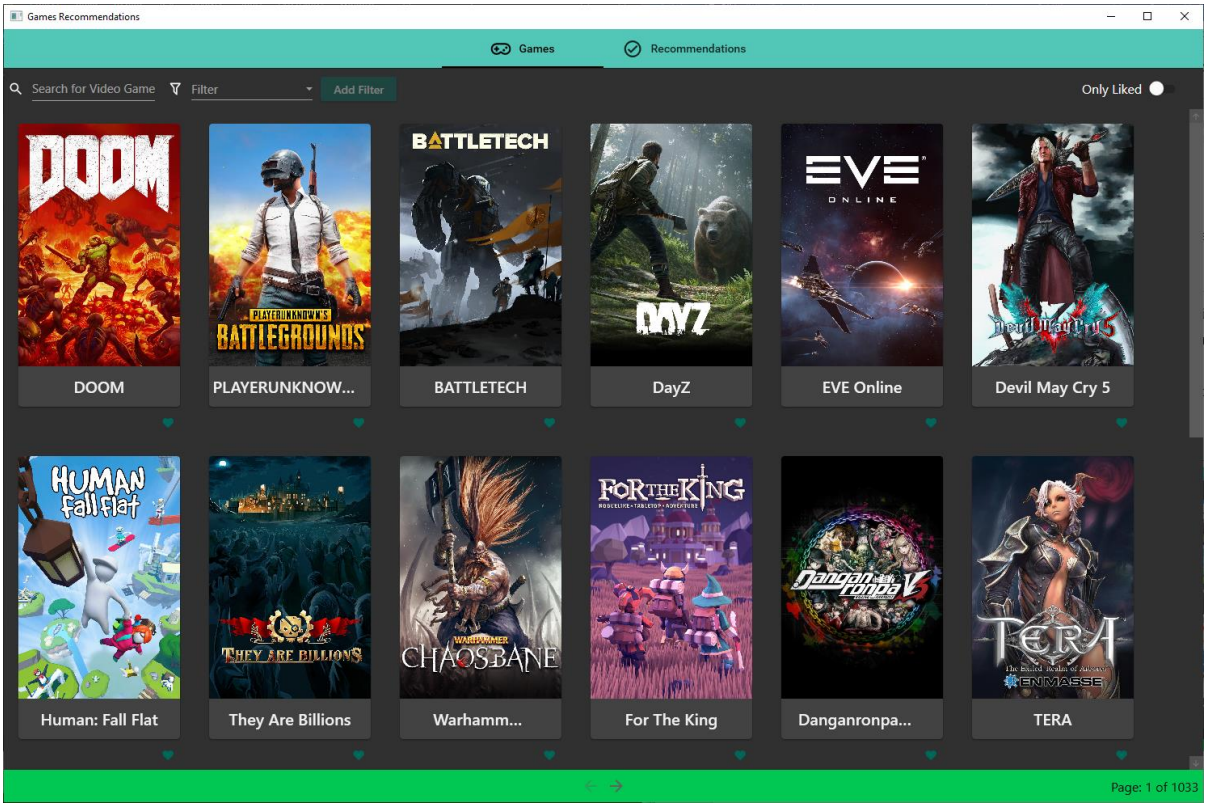

Рисунок 4.5 – Сторінка зі всіма відеоіграми

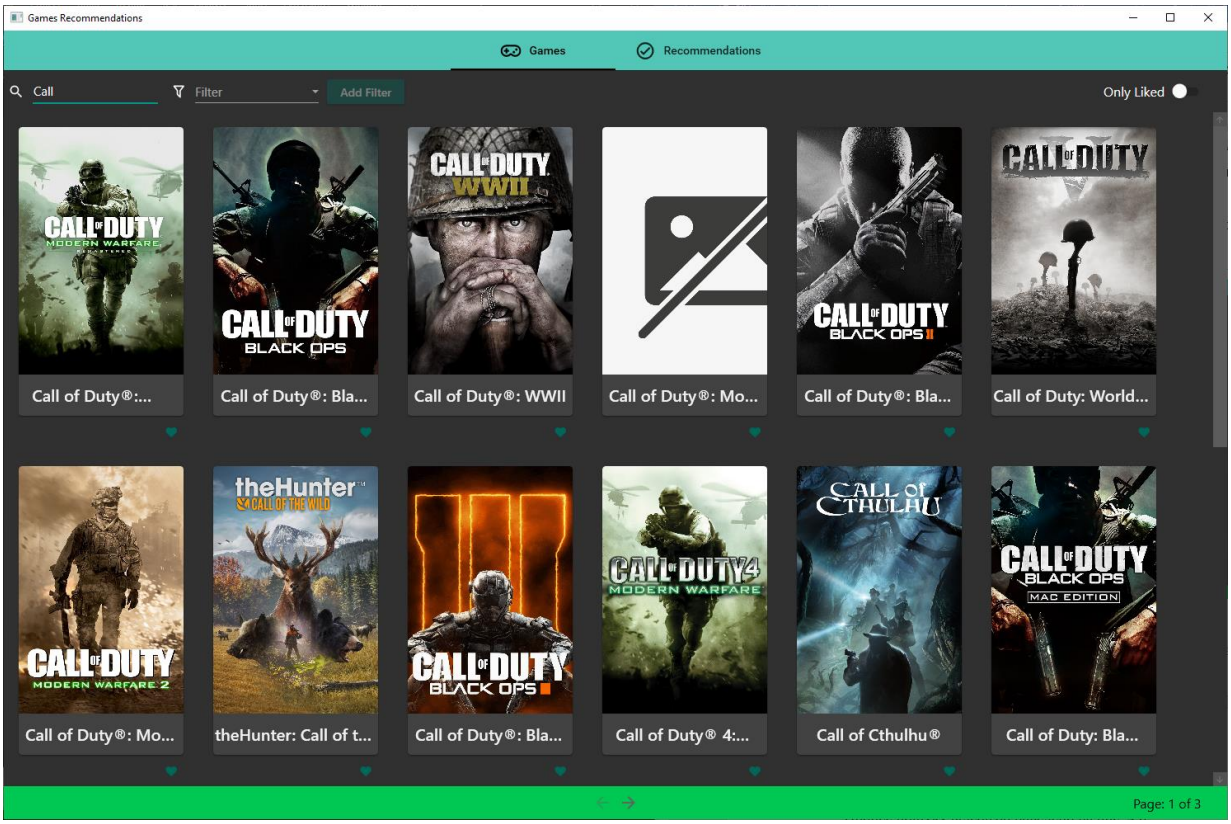

Рисунок 4.6 – Результат процесу пошуку відеоігор

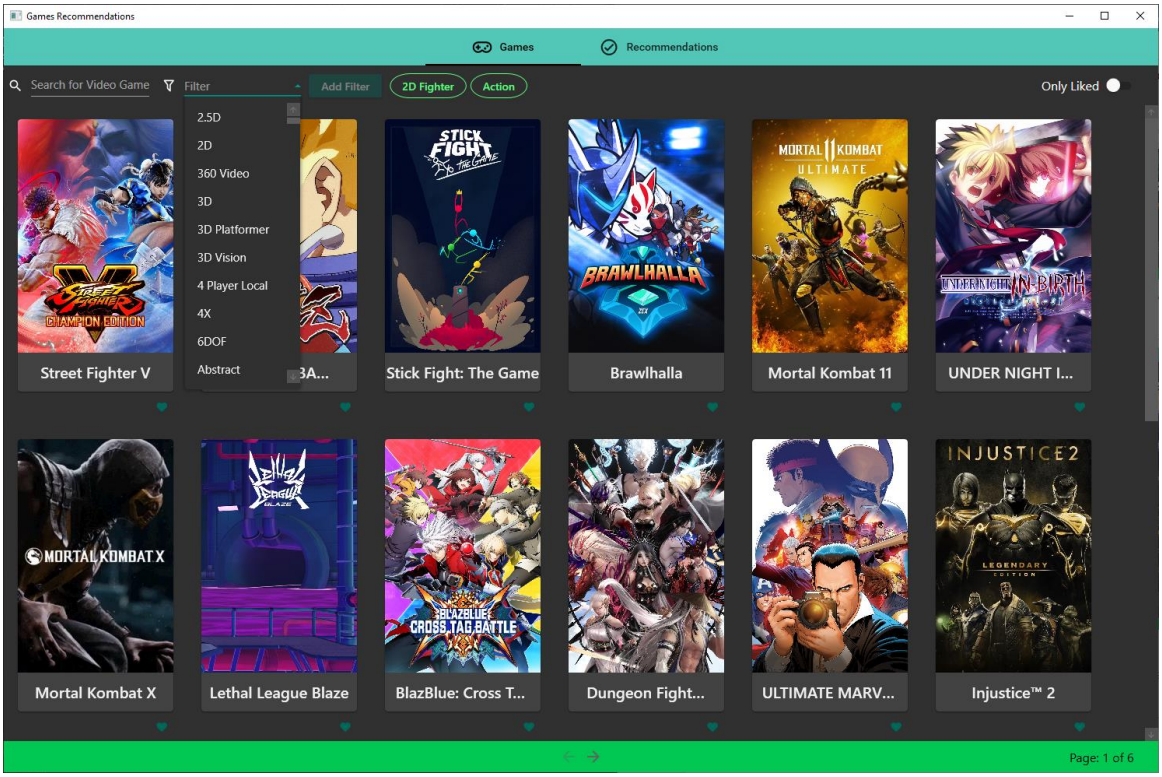

Рисунок 4.7 – Результат фільтрації відеоігор

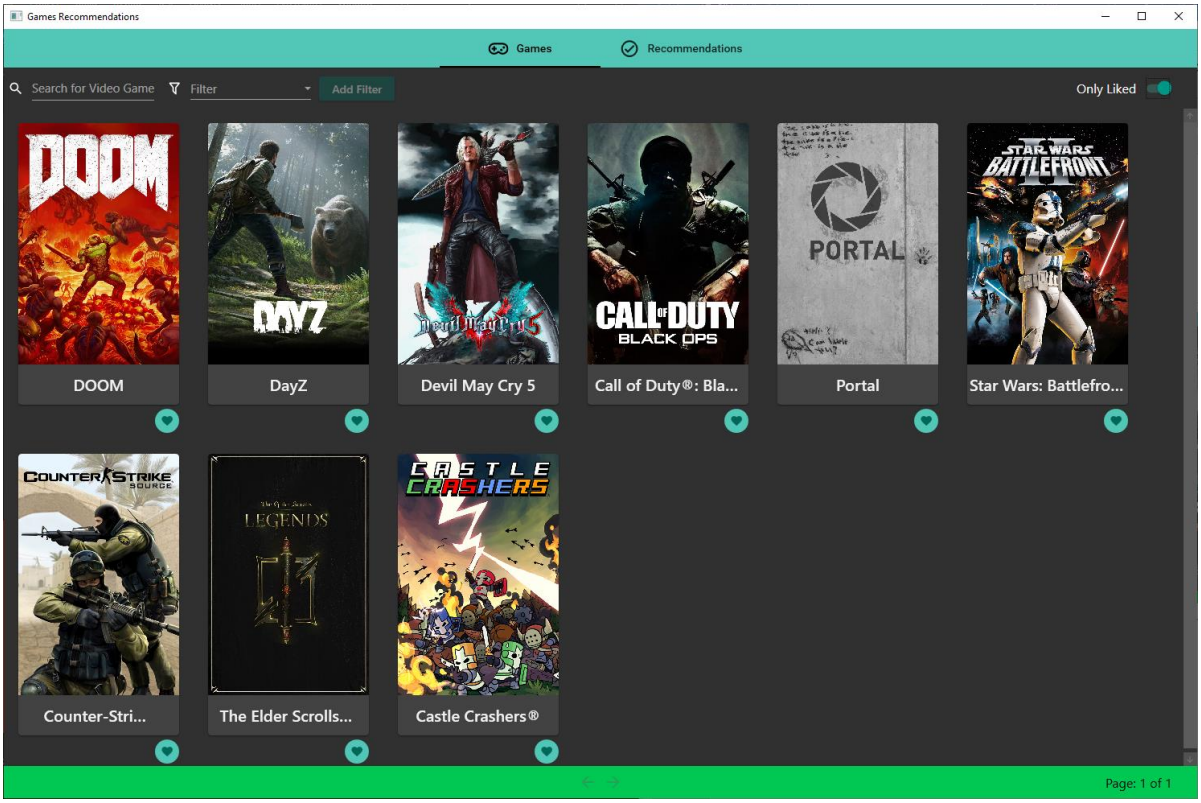

Рисунок 4.8 – Результат фільтрації відеоігор за вподобаними відеоіграми користувача

Сторінка з рекомендованими відеоіграми. Ця сторінка за дизайном дуже схожа зі сторінкою зі всіма відеоіграми, але для спрощення розробки та підвищення лаконічності було вирішено зробити окрему сторінку для рекомендованих відеоігор. Сторінка з рекомендованими відеоіграми реалізована у представлені RecommendedVideoGamesPage, яке містить контрол RecommendedGamesGridControl. Цей контрол є основною реалізацією сторінки з рекомендованими відеоіграми. Моделлю представлення цього контрола є клас RecommendedGamesGridViewModel.

Клас RecommendedGamesGridViewModel наслідує абстрактний клас GamesGridViewModelBase та майже нічим не відрізняється від базового класу, так як на цій сторінці з рекомендованими відеоіграми не буде панелі пошуку та фільтрації. Цей клас має залежності від класів, які реалізують інтерфейси IUrlImageLoader, IDataLoader та IRecommender. У конструкторі цього класу за допомогою метода *OnPageChanged* інтерфейса INavigation задається логіка, коли за відбувається процес рекомендації, а саме при переключенні на сторінку з рекомендованими відеоіграми. Процес рекомендації відбувається, коли викликається метод інтерфейсу IRecommender *RecommendVideoGamesAsync*. Аргументом цього інтерфейсу є вподобані відеоігри користувача. Після закінчення процесу рекомендації відеоігор, йде завантаження першої сторінки з рекомендованими відеоіграми за допомогою метода *LoadVideoGamesPageAsync*.

Повна реалізація класу RecommendedGamesGridViewModel наведено у додатку Л.

Представлення контролу RecommendedGamesGridControl має дуже схожу реалізацію з представленням контролу GamesGridControl. Представлення контролу RecommendedGamesGridControl не має панелі пошуку та фільтрації відеоігор та не має можливості зробити відеогру вподобаною у списку відеоігор. Але для кожної відеоігри у списку відеоігор є надпис з оцінкою рекомендації відеоігри.

Результат рекомендації за одною вподобаною відеогрою наведено на рис 4.9. Результат рекомендації за двома вподобаними відеоіграми наведено на рис 4.10.

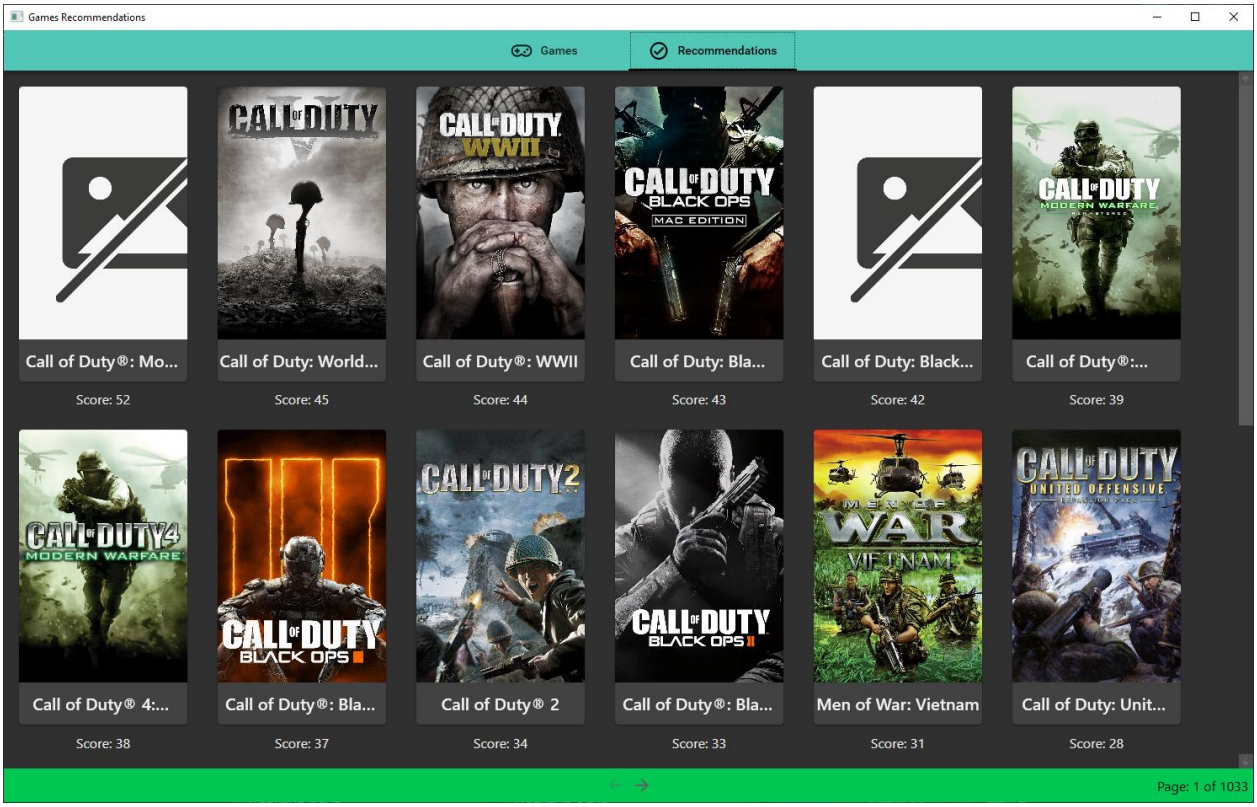

Рисунок 4.9 – Рекомендовані відеоігри згідно з відеогрою Call of Duty: Black Ops

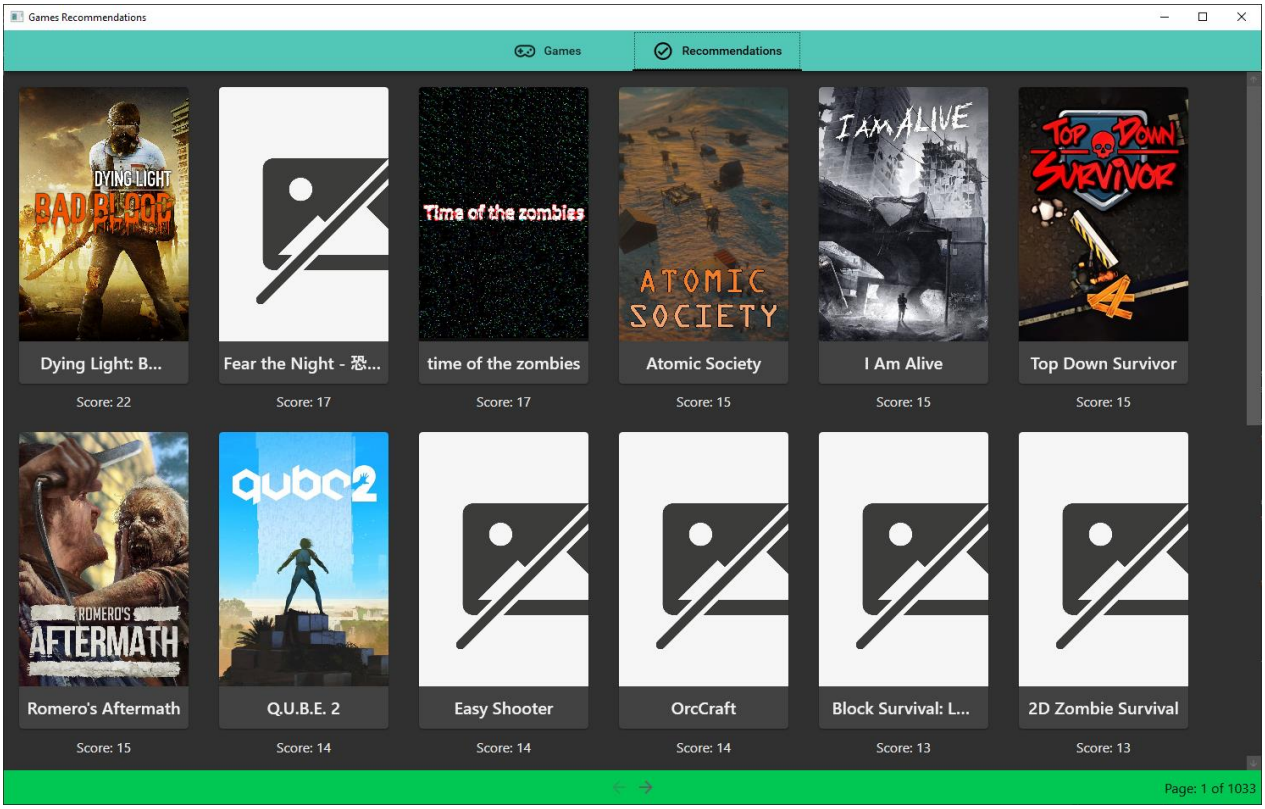

Рисунок 4.10 – Рекомендовані відеоігри згідно відеоіграми The Forest та Unturned

Висновки до розділу 4

У даному розділі було описано розробку та програмну реалізацію застосунку для рекомендації відеоігор. Було розглянута архітектура застосунку, а також основні компоненти, функціональні вимоги та особливості реалізації.

Під час проєктування архітектури застосунку було приділено особливу увагу вибору підходящих архітектурних шаблонів та структуруванню компонентів. Застосовані шаблони та архітектурні рішення забезпечують ефективну комунікацію між компонентами та розширюваність застосунку.

Програмна реалізація застосунку дозволяє користувачам з легкістю переглядати список відеоігор, застосовувати фільтрацію та здійснювати пошук відповідно до їх вподобань. Це дозволяє користувачам знаходити нові та цікаві відеоігри, що відповідають їхнім інтересам. Також була виконана головна задача – розробити систему, яка рекомендує відеоігри згідно вподобаних відеоігор користувача. У процесі розробки застосунку було дотримано сучасні технології та практики розробки програмного забезпечення.

ВИСНОВКИ

Сфера відеоігор пройшла довгий шлях розвитку, починаючи від простих аркадних ігор до складних мультиплатформенних проектів. Насиченість ринку відеоігор змушує користувача ретельно підбирати в яку відеогру йому пограти. Тому розробка рекомендаційних системи відеоігор стає все більш актуальною.

Впродовж роботи було виявлено, що сьогодні існує багато різних рекомендаційних систем відеоігор. При аналізі наявних систем рекомендацій відеоігор було виявлено, що, хоча ряд систем вже використовуються, вони не завжди враховують усі потреби користувачів, такі як персоналізація, різноманітність, та прозорість. Також багато існуючих сервісів, які мають систему рекомендацій відеоігор, не мають зручного функціонала пошуку та фільтрації бази відеоігор. Все це обумовлює необхідність розробки нової рекомендаційної системи відеоігор, яка відповідала б повною мірою потребам гравців.

Виявлено, що можна поліпшити якість рекомендацій, враховуючи вподобання гравців, які вже зробили певний вибір. Такий механізм можна реалізувати за допомогою алгоритмів, заснованих на аналізу характеристик відеоігор. Таким чином можна реалізувати систему рекомендацій, яка буде рекомендувати відеоігри за змістом вподобаних відеоігор користувача.

Було проведено аналіз різних алгоритмів і методів, які використовуються в рекомендаційних системах. На основі проведеного аналізу, стало зрозуміло, що немає універсального підходу до розробки рекомендаційних систем. Вибір конкретного методу або алгоритму залежить від специфіки задачі, доступних даних та обмежень ресурсів. Загалом, існує велика кількість технологій і методів, які можуть бути застосовані для розробки ефективної рекомендаційної системи.

Було проведено детальний аналіз та проектування рекомендаційної системи. Спочатку була виконана робота по знаходженню даних для рекомендаційної системи відеоігор. Після знаходження датасету відеоігор було виконано значну роботу по очищенню та підготовки даних. Очищення та підготовка даних

відбувались з використанням мови програмування Python таких її бібліотек, як Pandas та NumPy. У результаті отримати готовий датасет відеоігор розмірністю 24 774 відеоігор.

Було виконано роботу з проєктування рекомендаційної системи відеоігор. Була спроєктована модель рекомендаційної системи відеоігор. Було детально описано структуру системи, включаючи всі 3 головні модуля рекомендаційної системи, їх взаємодію та принципи функціонування. Було вибрано тип рекомендаційної системи, методи та процеси обробки даних, методи обчислення схожості відеоігор та методи генерації результатів рекомендацій, необхідних для ефективного функціонування системи.

Під час проєктування архітектури застосунку було приділено особливу увагу вибору підходящих архітектурних шаблонів та структуруванню компонентів. Застосовані шаблони та архітектурні рішення забезпечують ефективну комунікацію між компонентами та розширюваність застосунку.

Програмна реалізація застосунку дозволяє користувачам з легкістю переглядати список відеоігор, застосовувати фільтрацію та здійснювати пошук відповідно до їх вподобань. Це дозволяє користувачам знаходити нові та цікаві відеоігри, що відповідають їхнім інтересам. Також була виконана головноа задача – розробити систему, яка рекомендує відеоігри згідно вподобаних відеоігор користувача. У процесі розробки застосунку було дотримано сучасні технології та практики розробки програмного забезпечення.

Також було описано та проаналізовано застосовані технології. Використання технологій, таких як платформа .NET, мова програмування C#, систему WPF та бібліотеки Accord.Net, дозволило створити потужну рекомендаційну систему. Інструменти, такі як TF-IDF та метрика подібності косинуса, були використані для аналізу тексту і порівняння схожості між відеоіграми по їхнім змістовим описом. Використання цих технологій та методів допомогло у реалізації ефективної рекомендаційної системи для відеоігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. В. Гречко, Н. В. Захаров, М. О. Фалько. Аналіз динаміки розвитку ринку відеоігор, джерел його фінансування та особливостей монетизації продукції в даній сфері. *Ефективна економіка*. 2019. № 975. URL: http://www.economy.nayka.com.ua/pdf/5_2021/4.pdf (дата звернення: 03.05.2023).

2. Gaming Statistics – 2023. *TrueList*: вебсайт. URL: https://truelist.co/blog/gaming-statistics (дата звернення: 04.05.2023).

3. A Complete Guide To Recommender Systems. *Towards Data Science*: вебсайт. URL: https://towardsdatascience.com/a-complete-guide-to-recommender-system-tutorial-with-sklearn-surprise-keras-recommender-5e52e8ceace1 (дата звернення: 04.05.2023).

4. Recommender Systems. *Medium*: вебсайт. URL: https://medium.com/the-owl/recommender-systems-f62ad843f70c (дата звернення: 06.05.2023).

5. Francesco Ricci, Lior Rokach, Bracha Shapira. Recommender Systems Handbook. *Springer*. 2011. P. 1–35. URL: http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf (дата звернення: 06.05.2023).

6. On YouTube’s recommendation system. *Blog.YouTube*: вебсайт. URL: https://blog.youtube/inside-youtube/on-youtubes-recommendation-system (дата звернення: 10.05.2023).

7. Top 10 Best Online Video Game Stores. *Top10*: вебсайт. URL: https://www.top10.com/online-video-game-stores (дата звернення: 12.05.2023).

8. Xiaoyuan Su, Taghi M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*. 2009. Vol. 2009, P. 19. DOI: 10.1155/2009/421425.

9. Comprehensive Guide on Item Based Collaborative Filtering. *Towards Data Science*: вебсайт. URL: https://towardsdatascience.com/comprehensive-guide-on-item-based-recommendation-systems-d67e40e2b75d (дата звернення: 15.05.2023).

10. Collaborative Filtering Advantages & Disadvantages. *Developers.Google*: вебсайт. URL: https://developers.google.com/machine-learning/recommendation/collaborative/summary (дата звернення: 15.05.2023).

11. Sidorov Grigori, Gelbukh Alexander, Gómez-Adorno Helena, Pinto David. Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model. *Computación y Sistemas*. 2014. Vol. 18, No. 3. P. 491-504. DOI: 10.13053/CyS-18-3-2043.

12. Koren Yehuda, Bell Robert, Volinsky Chris. Matrix Factorization Techniques for Recommender Systems. *Computer*. 2009. Vol. 42. P. 30–37. DOI: 10.1109/MC.2009.263.

13. Matrix Factorization — Singular Value Decomposition (SVD) Explained. *Towards Data Science*: вебсайт. URL: https://towardsdatascience.com/recommendation-system-matrix-factorization-svd-explained-c9a50d93e488 (дата звернення: 20.05.2023).

14. Netflix Update: Try This at Home. *Sifter*: вебсайт. URL: https://sifter.org/~simon/journal/20061211.html (дата звернення: 21.05.2023).

15. Recommender Systems: Behind the Scenes of Machine Learning-Based Personalization. *Altexsoft*: вебсайт. URL: https://www.altexsoft.com/blog/recommender-system-personalization (дата звернення: 23.05.2023).

16. Text vectorization algorithms in NLP. *Medium*: вебсайт. URL: https://medium.com/data-science-in-your-pocket/text-vectorization-algorithms-in-nlp-109d728b2b63 (дата звернення: 25.05.2023).

17. Manning C.D., Raghavan P., Schutze H. Introduction to Information Retrieval. *Cambridge University Press*. 2009. Vol. 6. P. 100. DOI: 10.1017/CBO9780511809071.007. ISBN 978-0-511-80907-1.

18. A Complete Guide on Feature Extraction Techniques. *Analytics Vidhya*: вебсайт. URL: https://www.analyticsvidhya.com/blog/2022/05/a-complete-guide-on-feature-extraction-techniques (дата звернення: 30.05.2023).

19. Recommendation systems and machine learning: driving personalization. *Itransition*: вебсайт. URL: https://www.itransition.com/machine-learning/recommendation-systems (дата звернення: 31.05.2023).

20. TIOBE Index for June 2023. *TIOBE*: вебсайт. URL: https://www.tiobe.com/tiobe-index (дата звернення: 02.06.2023).

21. .NET. *dotnet.microsoft*: вебсайт. URL: https://dotnet.microsoft.com (дата звернення: 02.06.2023).

22. A tour of the C# language. *learn.microsoft*: вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp (дата звернення: 02.06.2023).

23. What is ML.NET and how does it work. *learn.microsoft*: вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work (дата звернення: 02.06.2023).

24. Accord.NET Framework. *accord-framework*: вебсайт. URL: http://accord-framework.net/index.html (дата звернення: 05.06.2023).

25. CsvHelper. *Joshclose*: вебсайт. URL: https://joshclose.github.io/CsvHelper (дата звернення: 06.06.2023).

26. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. *Prentice Hall*. 2018. P. 58.

27. All you need to know about text preprocessing for NLP and Machine Learning. *KDnuggets*: вебсайт. URL: https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html (дата звернення: 07.06.2023).

28. Visual Studio 2022. *visualstudio.microsoft*: вебсайт. URL: https://visualstudio.microsoft.com (дата звернення: 09.06.2023).

29. Desktop Guide (WPF .NET). *learn.microsoft*: вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview (дата звернення: 11.06.2023).

30. Model-View-ViewModel. *learn.microsoft*: вебсайт. URL: https://learn.microsoft.com/dotnet/architecture/mvvm (дата звернення: 11.06.2023).

ДОДАТОК А

Підготовка та очищення даних датасету

```
import numpy as np
import pandas as pd

df = pd.read_csv("./steam_games.csv")
df = df[df['types'] == 'app']
df = df[df['name'].isna() == False]
df = df[df['desc_snippet'].isna() == False]
df = df[df['popular_tags'].isna() == False]
df = df.reset_index(drop=True)

def get_digit_string(strings):
    return list(filter(lambda s: s.isdigit(), strings))[0]

app_ids = list(map(lambda url: get_digit_string(url.split('/')), df['url'].values))
df['app_id'] = app_ids

df = df[["app_id", "url", "name", "desc_snippet", "release_date", "developer",
"publisher", "popular_tags","genre","original_price"]]
df = df.reset_index(drop=True)

df.to_csv('games.csv')
```

ДОДАТОК Б

Реалізація класів VideoGamesCsvMap та TagsConverter

```
public class VideoGameCsvMap : ClassMap<VideoGame>
{
    public VideoGameCsvMap()
    {
        Map(g => g.AppId).Name("app_id");

        Map(g => g.Name).Name("name");

        Map(g => g.Url).Name("url");

        Map(g => g.ReleaseDate).Name("release_date");

        Map(g => g.Developer).Name("developer");

        Map(g => g.PopularTags).Name("popular_tags").TypeConverter<TagsConverter>();

        Map(g => g.Genre).Name("genre");

        Map(g => g.Description).Name("desc_snippet");

        Map(g => g.Price).Name("original_price");
    }
      }


public class TagsConverter : DefaultTypeConverter
{
    public override object? ConvertFromString(string? text, IReaderRow row, MemberMapData
memberMapData)
    {
        return text?.Split(',');
    }
}
```

ДОДАТОК В

Реалізація інтерфейсу IDataLoader та класу FileDataLoader

```
public interface IDataLoader
{
    Task<List<VideoGame>> LoadDataAsync(string source);

    List<string> LoadVideoGamesTags(IEnumerable<VideoGame> videoGames);

    List<VideoGame> GetLoadedData();
}



public class FileDataLoader : IDataLoader
{
    private List<VideoGame>? _data;

    public async Task<List<VideoGame>> LoadDataAsync(string source)
    {
        using var streamReader = new StreamReader(source);
        using var csvReader = new CsvReader(streamReader, CultureInfo.InvariantCulture);

        csvReader.Context.TypeConverterCache.AddConverter<string[]>(new TagsConverter());
        csvReader.Context.RegisterClassMap<VideoGameCsvMap>();

        _data = await csvReader.GetRecordsAsync<VideoGame>().ToListAsync();

        return _data;
    }

    public List<VideoGame> GetLoadedData()
    {
        return _data ?? throw new InvalidOperationException("Data has not been loaded.");
    }

    public List<string> LoadVideoGamesTags(IEnumerable<VideoGame> videoGames)
    {
        return videoGames.SelectMany(g => g.PopularTags).Distinct().OrderBy(x =>
x).ToList();
    }
}
```

ДОДАТОК Г

Реалізація інтерфейсу IDataProcessor та класу DataProcessor

```
public interface IDataProcessor<T>
{
      Task<T> ProcessDataAsync(IEnumerable<VideoGame> data);
}


public class DataProcessor : IDataProcessor<string[][]>
{
    public async Task<string[][]> ProcessDataAsync(IEnumerable<VideoGame> data)
    {
        var stopWords = new HashSet<string>(await File.ReadAllLinesAsync("Data/english
stopwords.txt"));
        var tokenizedDescriptionsList = data.Select(x =>
x.Description.Tokenize().ToList()).ToList();

        var processedData = tokenizedDescriptionsList.AsParallel().Select(description =>
            description.Where(word => !stopWords.Contains(word)).ToArray()
        ).ToArray();

        return processedData;
    }
}
```

ДОДАТОК Д

Реалізація інтерфейсу IRecommender та класу VideoGamesRecommender

```
public interface IRecommender
{
    Task<List<VideoGame>> RecommendVideoGamesAsync(IEnumerable<VideoGame> videoGames);
}


public class VideoGamesRecommender : IRecommender
{
    private readonly IDataLoader _dataLoader;
    private readonly IDataProcessor<string[][]> _dataProcessor;
    private List<VideoGame> _videoGames;
    private double[][]? _tfIdfMatrix;

    public VideoGamesRecommender(IDataLoader dataLoader, IDataProcessor<string[][]>
dataProcessor)
    {
        _dataLoader = dataLoader;
        _dataProcessor = dataProcessor;
    }

    public async Task<List<VideoGame>> RecommendVideoGamesAsync(IEnumerable<VideoGame>
videoGames)
    {
        _videoGames = _dataLoader.GetLoadedData();

        var tfIdfMatrix = await GetOrCalculateTfIdfMatrixAsync();

        var similarityScoresVectors = CalculateSimilarityScoresVectors(tfIdfMatrix,
videoGames).ToList();
        var averagedSimilarityScores =
CalculateAverageSimilarityScoresVector(similarityScoresVectors);

        foreach (var (gameIndex, similarityScore) in averagedSimilarityScores)
        {
            _videoGames[gameIndex].RecommendationScore = (int)similarityScore;
        }

        return _videoGames.OrderByDescending(g => g.RecommendationScore).ToList();
    }

    private async Task<double[][]> GetOrCalculateTfIdfMatrixAsync()
    {
        if (_tfIdfMatrix != null)
        {
            return _tfIdfMatrix;
        }

        var processedData = await _dataProcessor.ProcessDataAsync(_videoGames);

        var tfidf = new TFIDF
        {
            Tf = TermFrequency.Log,
            Idf = InverseDocumentFrequency.Smooth,
        };

        tfidf.Learn(processedData);
```

```
        _tfIdfMatrix = tfidf.Transform(processedData);

        return _tfIdfMatrix;
    }

    private IEnumerable<List<(int Index, double RecommendationScore)>>
CalculateSimilarityScoresVectors(double[][] tfIdfMatrix, IEnumerable<VideoGame>
likedVideoGames)
    {
        var likedVideoGamesNames = likedVideoGames.Select(g => g.Name);
        var likedVideoGamesIndexes = likedVideoGamesNames.Select(name =>
_videoGames.FindIndex(g => g.Name == name));

        var cosine = new Cosine();

        foreach (var (likedVideGame, likedVideGameIndex) in
likedVideoGames.Zip(likedVideoGamesIndexes, Tuple.Create))
        {
            var gameTfIdfMatrixRow = tfIdfMatrix[likedVideGameIndex];
            var gameSimilarityScores = new ConcurrentBag<(int, double)>();

            for (int i = 0; i < tfIdfMatrix.GetLength(0); i++)
            {
                var tfIdfRow = tfIdfMatrix[i];
                gameSimilarityScores.Add((i, cosine.Similarity(gameTfIdfMatrixRow,
tfIdfRow)));
            }

            yield return gameSimilarityScores.ToList();
        }
    }

    private IEnumerable<(int Index, double RecommendationScore)>
CalculateAverageSimilarityScoresVector(List<List<(int Index, double
RecommendationScore)>> similatiryVectors)
    {
        for (var i = 0; i < similatiryVectors[0].Count; i++)
        {
            var gameIndex = similatiryVectors[0][i].Index;

            var averageScore = 0.0;

            for (var j = 0; j < similatiryVectors.Count; j++)
            {
                averageScore += similatiryVectors[j][i].RecommendationScore;
            }

            averageScore /= similatiryVectors.Count;

            averageScore = Math.Round(averageScore * 100);

            yield return (gameIndex, averageScore);
        }
    }
}
```

ДОДАТОК Е

Реалізація класів ViewModelBase та MainViewModel

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    protected virtual bool SetProperty<T>(ref T storage, T value, [CallerMemberName]
string propertyName = "")
    {
        if (EqualityComparer<T>.Default.Equals(storage, value))
        {
            return false;
        }

        storage = value;
        this.OnPropertyChanged(propertyName);

        return true;
    }
}


public class MainViewModel : ViewModelBase
{
    private Page _currentPage;

    public MainViewModel(INavigationService navigationService)
    {
        navigationService.OnPageChanged += (page) => CurrentPage = page;

        navigationService.ChangePage(new VideoGamesPage());
    }

    public Page CurrentPage
    {
        get => _currentPage;
        set => SetProperty(ref _currentPage, value);
    }
}
```

ДОДАТОК Ж

Реалізація класу NavigationBarViewModel

```
public class NavigationBarViewModel : ViewModelBase
{
    private readonly INavigationService _navigationService;
    private int _selectedPageIndex;

    public NavigationBarViewModel(INavigationService navigationService)
    {
        _navigationService = navigationService;
    }

    public int SelectedPageIndex
    {
        get => _selectedPageIndex;
        set
        {
            SetProperty(ref _selectedPageIndex, value);

            switch (value)
            {
                case 0:
                    _navigationService.ChangePage(new VideoGamesPage());
                    break;
                case 1:
                    _navigationService.ChangePage(new RecommendedVideoGamesPage());
                    break;
                default:
                    break;
            }
        }
    }
}


public interface INavigationService
{
    event Action<Page> OnPageChanged;

    void ChangePage(Page page);
}


public class NavigationService : INavigationService
{
    public event Action<Page>? OnPageChanged;

    public void ChangePage(Page page) => OnPageChanged?.Invoke(page);
}
```

ДОДАТОК И

Реалізація класу GamesGridViewModelBase

```
public abstract class GamesGridViewModelBase : ViewModelBase
{
    private readonly IUrlImageLoader _urlImageLoader;
    private readonly int _itemsPerPage = 24;
    private int _totalPages;
    private int _currentPage;

    protected List<VideoGame> _videoGamesToView;
    protected ObservableCollection<VideoGame> _pagedVideoGames;

    protected GamesGridViewModelBase(IUrlImageLoader urlImageLoader)
    {
        _urlImageLoader = urlImageLoader;

        NextPageCommand = new RelayCommand(async () => await NextPageAsync(), () =>
CurrentPage < TotalPages);
        PreviousPageCommand = new RelayCommand(async () => await PreviousPageAsync(), ()
=> CurrentPage > 1);
    }

    public ObservableCollection<VideoGame> PagedVideoGames
    {
        get => _pagedVideoGames;
        set => SetProperty(ref _pagedVideoGames, value);
    }

    public int CurrentPage
    {
        get => _currentPage;
        set => SetProperty(ref _currentPage, value);
    }

    public int TotalPages
    {
        get => _totalPages;
        set
        {
            SetProperty(ref _totalPages, value);

            ((RelayCommand)NextPageCommand).NotifyCanExecuteChanged();
            ((RelayCommand)PreviousPageCommand).NotifyCanExecuteChanged();
        }
    }

    public ICommand NextPageCommand { get; }

    public ICommand PreviousPageCommand { get; }

    protected int CalculateTotalPages(int itemsCount) => (int)Math.Ceiling(itemsCount /
(double)_itemsPerPage);

    protected async Task LoadVideoGamesPageAsync(int page)
    {
        CurrentPage = page;

        var toSkip = (_currentPage - 1) * _itemsPerPage;
```

```
        var toTake = _videoGamesToView.Count - toSkip < _itemsPerPage ?
_videoGamesToView.Count - toSkip : _itemsPerPage;

        var pagedVideoGames = _videoGamesToView.GetRange(toSkip, toTake);

        var loadImagesTasks = pagedVideoGames.Select(async game =>
        {
            if (string.IsNullOrEmpty(game.ImageUrl))
            {
                game.ImageUrl = await _urlImageLoader.GetImageUrlAsync(game.AppId);
            }
        });

        await Task.WhenAll(loadImagesTasks);

        TotalPages = CalculateTotalPages(_videoGamesToView.Count);
        PagedVideoGames = new ObservableCollection<VideoGame>(pagedVideoGames);
    }

    private async Task NextPageAsync()
    {
        await LoadVideoGamesPageAsync(CurrentPage + 1);
    }

    private async Task PreviousPageAsync()
    {
        if (CurrentPage > 1)
        {
            await LoadVideoGamesPageAsync(CurrentPage - 1);
        }
    }
}
```

ДОДАТОК К

Реалізація класу GamesGridViewModel

```
public class GamesGridViewModel : GamesGridViewModelBase
{
    private readonly IDataLoader _dataLoader;
    private string _searchQuery;
    private string _filterQuery;
    private bool _viewLiked = false;
    private List<VideoGame> _allVideoGames;
    private ObservableCollection<string> _allTags;
    private ObservableCollection<string> _selectedTags = new();

    public GamesGridViewModel(IDataLoader dataLoader, IUrlImageLoader urlImageLoader,
INavigationService navigationService)
        : base(urlImageLoader)
    {
        _dataLoader = dataLoader;

        AddFilterCommand = new RelayCommand(async () => await
AddFilterAsync(FilterQuery), () => !string.IsNullOrEmpty(FilterQuery));
        RemoveFilterCommand = new RelayCommand<string>(async (filter) => await
RemoveFilterAsync(filter));
        ToggleViewLikedVideoGamesCommand = new RelayCommand(async () => await
ToggleViewLikedVideoGamesAsync());
        SearchVideoGamesCommand = new RelayCommand(async () =>
        {
            if (string.IsNullOrEmpty(SearchQuery))
            {
                await ClearSearchAsync();
            }
            else
            {
                await TrySearchAndApplyFiltersAsync();
            }
        });


        navigationService.OnPageChanged += async (page) =>
        {
            if (page is VideoGamesPage)
            {
                if (_allVideoGames == null)
                {
                    await LoadGamesAsync();

                    _videoGamesToView = _allVideoGames;

                    await LoadVideoGamesPageAsync(1);

                    AllTags = new
ObservableCollection<string>(dataLoader.LoadVideoGamesTags(_allVideoGames));
                }
            }
        };
    }

    public ObservableCollection<string> AllTags
    {
```

```
        get => _allTags;
        set => SetProperty(ref _allTags, value);
    }

    public ObservableCollection<string> SelectedTags
    {
        get => _selectedTags;
        set => SetProperty(ref _selectedTags, value);
    }

    public string SearchQuery
    {
        get => _searchQuery;
        set => SetProperty(ref _searchQuery, value);
    }

    public string FilterQuery
    {
        get => _filterQuery;
        set
        {
            SetProperty(ref _filterQuery, value);

            ((RelayCommand)AddFilterCommand).NotifyCanExecuteChanged();
        }
    }

    public bool ViewLikedVideoGames
    {
        get => _viewLiked;
        set => SetProperty(ref _viewLiked, value);
    }

    public ICommand SearchVideoGamesCommand { get; }

    public ICommand AddFilterCommand { get; }

    public ICommand RemoveFilterCommand { get; }

    public ICommand ToggleViewLikedVideoGamesCommand { get; }

    private async Task LoadGamesAsync()
    {
        var sourceFile = "./Data/games.csv";

        _allVideoGames = await _dataLoader.LoadDataAsync(sourceFile);

        TotalPages = CalculateTotalPages(_allVideoGames.Count);
    }

    private async Task TrySearchAndApplyFiltersAsync()
    {
        if (!string.IsNullOrEmpty(SearchQuery))
        {
            var searchedGames = _allVideoGames.Where(g => g.Name.Contains(SearchQuery,
StringComparison.InvariantCultureIgnoreCase));

            _videoGamesToView = ApplyFilters(searchedGames).ToList();
        }
        else
        {
            _videoGamesToView = ApplyFilters(_allVideoGames).ToList();
```

```
        }

        await LoadVideoGamesPageAsync(1);
    }

    private IEnumerable<VideoGame> ApplyFilters(IEnumerable<VideoGame> videoGames)
    {
        if (ViewLikedVideoGames)
        {
            videoGames = videoGames.Where(g => g.IsLiked);
        }

        if (SelectedTags.Count > 0)
        {
            videoGames = videoGames.Where(g => SelectedTags.All(tag =>
g.PopularTags.Contains(tag)));
        }

        return videoGames;
    }

    private async Task ClearSearchAsync()
    {
        _videoGamesToView = ApplyFilters(_allVideoGames).ToList();

        await LoadVideoGamesPageAsync(1);
    }

    private async Task AddFilterAsync(string filter)
    {
        SelectedTags.Add(filter);
        AllTags.Remove(filter);

        _videoGamesToView = _videoGamesToView.Where(g => SelectedTags.All(tag =>
g.PopularTags.Contains(tag))).ToList();

        await LoadVideoGamesPageAsync(1);
    }

    private async Task RemoveFilterAsync(string filter)
    {
        SelectedTags.Remove(filter);
        AllTags.Add(filter);
        AllTags = new ObservableCollection<string>(AllTags.OrderBy(x => x));

        await TrySearchAndApplyFiltersAsync();
    }

    private async Task ToggleViewLikedVideoGamesAsync()
    {
        if (ViewLikedVideoGames)
        {
            _videoGamesToView = _videoGamesToView.Where(g => g.IsLiked).ToList();

            await LoadVideoGamesPageAsync(1);
        }
        else
        {
            await TrySearchAndApplyFiltersAsync();
        }
    }
```

ДОДАТОК Л

Реалізація класу RecommendedGamesGridViewModel

```
public class RecommendedGamesGridViewModel : GamesGridViewModelBase
{
    public RecommendedGamesGridViewModel(IUrlImageLoader urlImageLoader, IDataLoader
dataLoader, IRecommender recommender, INavigationService navigationService)
        : base(urlImageLoader)
    {
        navigationService.OnPageChanged += async (page) =>
        {
            if (page is RecommendedVideoGamesPage)
            {
                var likedVideoGames = dataLoader.GetLoadedData().Where(g => g.IsLiked);

                var recommendedGames = await
recommender.RecommendVideoGamesAsync(likedVideoGames);

                _videoGamesToView = recommendedGames.Where(g => !g.IsLiked).ToList();

                await LoadVideoGamesPageAsync(1);
            }
        };
     }
 }
```