

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

(назва теми)

ПОШУКУ НАСТІЛЬНИХ ІГОР. RESEARCH AND SOFTWARE

IMPLEMENTATION OF METHODS TO FIND BOARD GAMES

Виконав(ла): студент(ка) 2 курсу, групи КНТ-122м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СЕРДЮК О.В.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ ФКНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ магістр _____
Спеціальність _____ 121 Інженерія програмного забезпечення _____
(код і найменування)
Освітня програма (спеціалізація) _____ Інженерія програмного забезпечення _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри _____ ПЗ, д.т.н, проф. _____
Сергій СУББОТІН
« _____ » _____ 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СЕРДЮКА Олександра Валерійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів пошуку настільних ігор. Research and Software Implementation of Methods to Find Board Games

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович _____,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 14 » листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року
3. Вихідні дані до проєкту (роботи) _____ рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Реалізація програми. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 04 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Дослідження проблеми	3-4 тижні	Розділ 2
4	Розробка архітектури програми	5 тиждень	Розділ 3
5	Розробка програми	6-7 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	8-9 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї	10 тиждень	Додатки
8	Нормоконтроль та рецензування	11 тиждень	
9	Захист роботи	12 тиждень	

Студент(ка)

_____ Олександр СЕРДЮК
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
112 с., 24 рис., 10 табл., 3 дод., 11 джерел.

НАСТІЛЬНІ ІГРИ, ПОШУК ДАНИХ, КЛАСТЕРИЗАЦІЯ, МЕТОД К-
СЕРЕДНІХ, ВЕБЗАСТОСУНОК.

Об'єкт роботи – процес пошуку настільних ігор. Предмет роботи – методи пошуку настільних ігор. Мета роботи – розробити програмне забезпечення пошуку настільних ігор для підвищення зручності користувачів через визначення пріоритетних для них настільних ігор.

Матеріали, методи та технічні засоби: мова Python, фреймворк розробки вебзастосунків Django, система MySQL.

Результати. Результати роботи включають програму, призначену для надання інформації про настільні ігри, забезпечення можливості замовити настільні ігри в магазинах мережі, пошуку нових настільних ігор, в тому числі через надання пріоритетних настільних ігор щодо вибору, реєстрації власних настільних ігор у програмі, надання відгуків про власний досвід, ознайомлення з інформацією щодо досвіду інших користувачів, результати експериментального дослідження кластеризації даних настільних ігор.

Висновки. Наукова новизна роботи полягає в запропонованому методі пошуку настільних ігор, в якому визначається попередній набір даних, встановлюються персональні переваги користувача на основі відгуків про настільні ігри, виконується кластеризація даних на основі методу k-середніх, відбувається визначення пріоритетних настільних ігор для користувача та виконується відбір серед них за параметрами пошуку, встановленими користувачем.

Галузь використання. Продаж настільних ігор.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 112 pages, 24 figures, 10 tables, 3 appendixes, 11 sources.

BOARD GAMES, DATA SEARCH, CLUSTERIZATION, K-MEANS METHOD, WEB INTERFACE.

The object of the work is the process of searching for board games. The subject of the work is the methods of searching for board games. The goal of the work is to develop board game search software to improve user experience by identifying board games that are prioritized for them.

Materials, methods and technical means: Python language, Django web application development framework, MySQL system.

The results. The results of the work include a program designed to provide information about board games, providing the ability to order board games in network stores, searching for new board games, including the provision of priority board games for selection, registering own board games in the program, providing feedback on own experience, familiarization with information about the experience of other users, and the results of an experimental study of clustering of board game data.

Conclusions. The scientific novelty of the work consists in the proposed method of searching for board games, in which a preliminary set of data is determined, personal preferences of the user are established based on reviews of board games, data clustering is performed based on the k-means method, priority board games for the user are determined, and selection is performed among them according to the search parameters set by the user.

Field of use. Sale of board games.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Програмні інструменти для інформування про настільні ігри	11
1.2 Порівняння розроблюваної програми з існуючими	14
1.3 Висновки за розділом 1	15
2 Матеріали і методи.....	17
2.1 Використання методів кластеризації для розв’язання задачі пошуку настільних ігор	17
2.2 Метод пошуку настільних ігор	18
2.3 Висновки за розділом 2	21
3 Розробка архітектури програми.....	23
3.1 Визначення сценаріїв використання програми користувачами.....	23
3.2 Рішення щодо вибору засобів розробки	26
3.3 Визначення структури даних у базі даних	27
3.4 Висновки за розділом 3	41
4 Реалізація програми	42
4.1 Визначення послідовності роботи з програмою	42
4.2 Особливості реалізації програми.....	43
4.3 Висновки за розділом 4	59
5 Експлуатація, тестування та експериментальне дослідження програми	60
5.1 Призначення програми	60
5.2 Виконання програми.....	60
5.3 Експериментальне дослідження	65
5.4 Висновки за розділом 5	74
Висновки	75
Перелік джерел посилання	76
Додаток А Технічне завдання	77

Додаток Б Текст програми	82
Додаток В Слайди презентації.....	105

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ОРВ – об’єктно-реляційне відображення;

ПЗ – програмне забезпечення.

ВСТУП

Ринок настільних ігор переживає прямо зараз свій достатньо бурхливий розвиток, але цей розвиток все ще не спиняється. Згідно з даними, наведеними в статті [1], можна констатувати:

- світовий ринок настільних ігор на даний момент вже оцінюється в межах від 11 до 13,4 мільярдів доларів, а за наявними на цю мить прогнозами протягом 5 років у майбутньому має зрости на приблизно від 7 до 11 %: підтвердження щодо росту можна отримати і розглядаючи минулі дані, так наприклад, за даними дослідницької компанії NPD Group продажі настільних ігор з початку 2022 року до листопада зросли на 28 % порівняно з тим же періодом 2019 року, при цьому карткові ігри зросли на 29 %, а стратегічні карткові ігри (Pokemon, Magic: The Gathering) зросли одразу на 208 %;

- реагуючи на розвиток настільних ігор, багато популярних платформ на даний момент розвиваються за рахунок або відповідного контенту як Youtube, або за рахунок створення нових ігор, як-от платформа краудфандингу Kickstarter, яка спростила випуск ігор невідомими дизайнерами, при цьому онлайн-форум BoardGameGeek надає дані про те, що щороку випускається понад 3000 нових ігор;

- щорічно 24 % гравців купляють 11 і більше нових настільних ігор, 27% – понад 30, 35% – чотири ігри або менше, а 41% – 5-10 нових ігор [1]-[2].

Тож все це вказує з одного боку на те, що все більша кількість учасників з'являється в цій галузі, а відповідно вони потребують самих настільних ігор, їхнього більшого різноманіття, а з іншого боку більше різноманіття є ознакою того, що пошук потрібних варіантів ускладнюється, а тому підбір тих настільних ігор, які можуть бути цікавими користувачу, які йому варто придбати, стає складнішим. Тому для реалізації цього процесу недостатньо тільки наявності відповідних фільтрів пошуку, самого пошуку, а потрібно також мати програмне забезпечення (ПЗ), яке буде надавати певні

рекомендації, надавати загалом інформацію про ігри, надавати можливості ці ігри замовити.

Метою роботи є розробити ПЗ пошуку настільних ігор для підвищення зручності користувачів через визначення пріоритетних для них настільних ігор.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Програмні інструменти для інформування про настільні ігри

Зважаючи на популярність настільних ігор, існує цілий ряд мереж магазинів, а відповідно і вебзастосунків, які дозволяють переглядати інформацію про ігри, замовляти ігри, проте серед всіх вебзастосунків у цій предметній області окремо стоїть BoardGameGeek (рис. 1.1) [3].

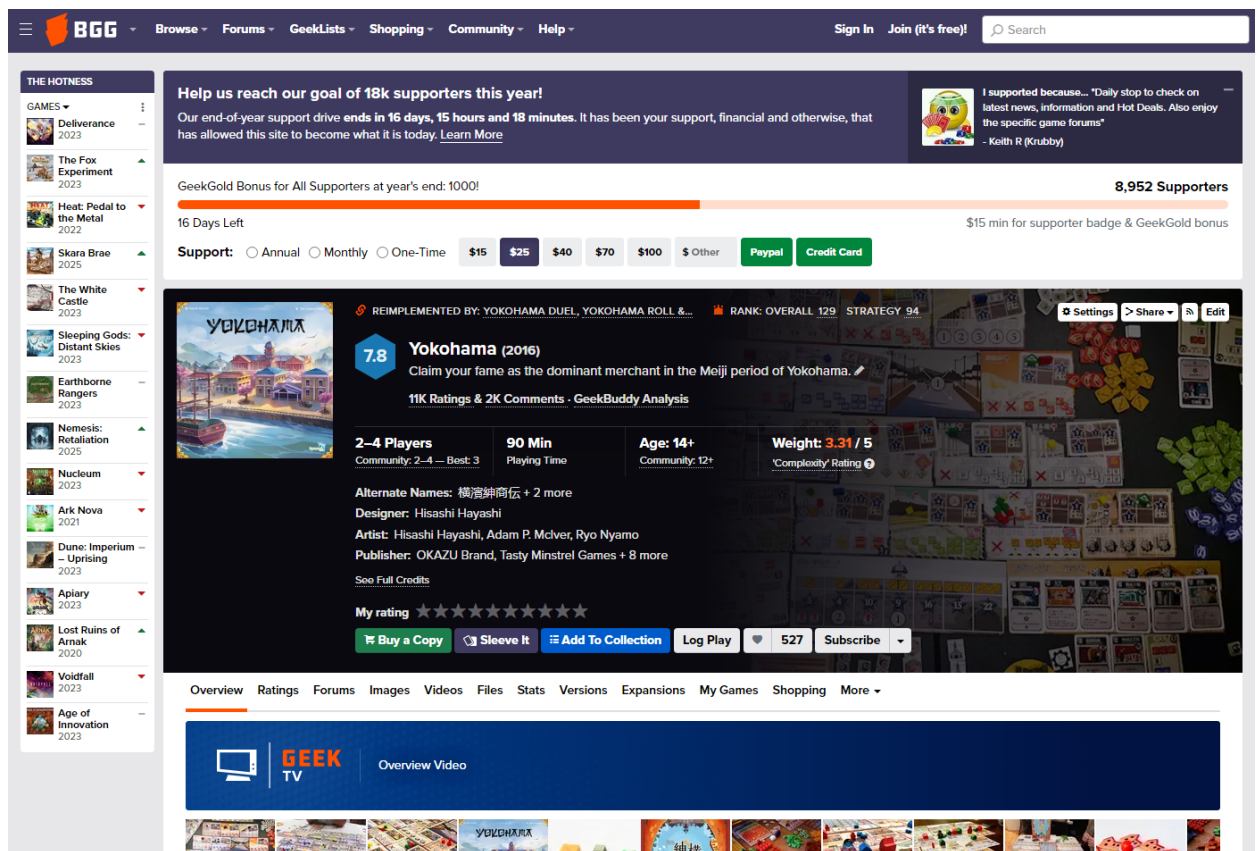


Рисунок 1.1 – Вигляд вебзастосунку BoardGameGeek для користувачів

BoardGameGeek – це онлайн-форум для любителів настільних ігор. Тобто це повністю спеціалізований вебзастосунків, який надає інформацію про настільні ігри, але не надає можливості придбати таку гру, перенаправляючи для цього на інші вебсайти. BoardGameGeek об'єднує фактично світову спільноту любителів настільних ігор, містить багато інформації про кожну гру, її версії, дозволяє ділитися власними враженнями про гру, виконувати пошук настільних ігор, задаючи зокрема назву,

дизайнера, видавця, рік видання, мінімальний вік, оцінку, кількість гравців, час на гру, категорії тощо. Вебсайт також містить окремі розділи, які певним чином всі настільні ігри впорядковують і таким чином дозволяють знаходити потрібні варіанти: найновіші ігри, ігри за категоріями, видавцями, дизайнерами тощо. Тобто поза межами придбання ігор всю іншу інформацію про ігри вебзастосунок достатньо широко представляє і створює цілу спільноту гравців. Однак для пошуку нових ігор потрібно або використовувати традиційний пошук, або переглядати відгуки інших гравців, таким чином ґрунтуючись на їх оцінках як на рекомендаціях, адже певної системи створення рекомендацій під користувача програма не має.

Другою програмою, що було окремо виділено для огляду і аналізу є Zatu Games (рис. 1.2) [4].

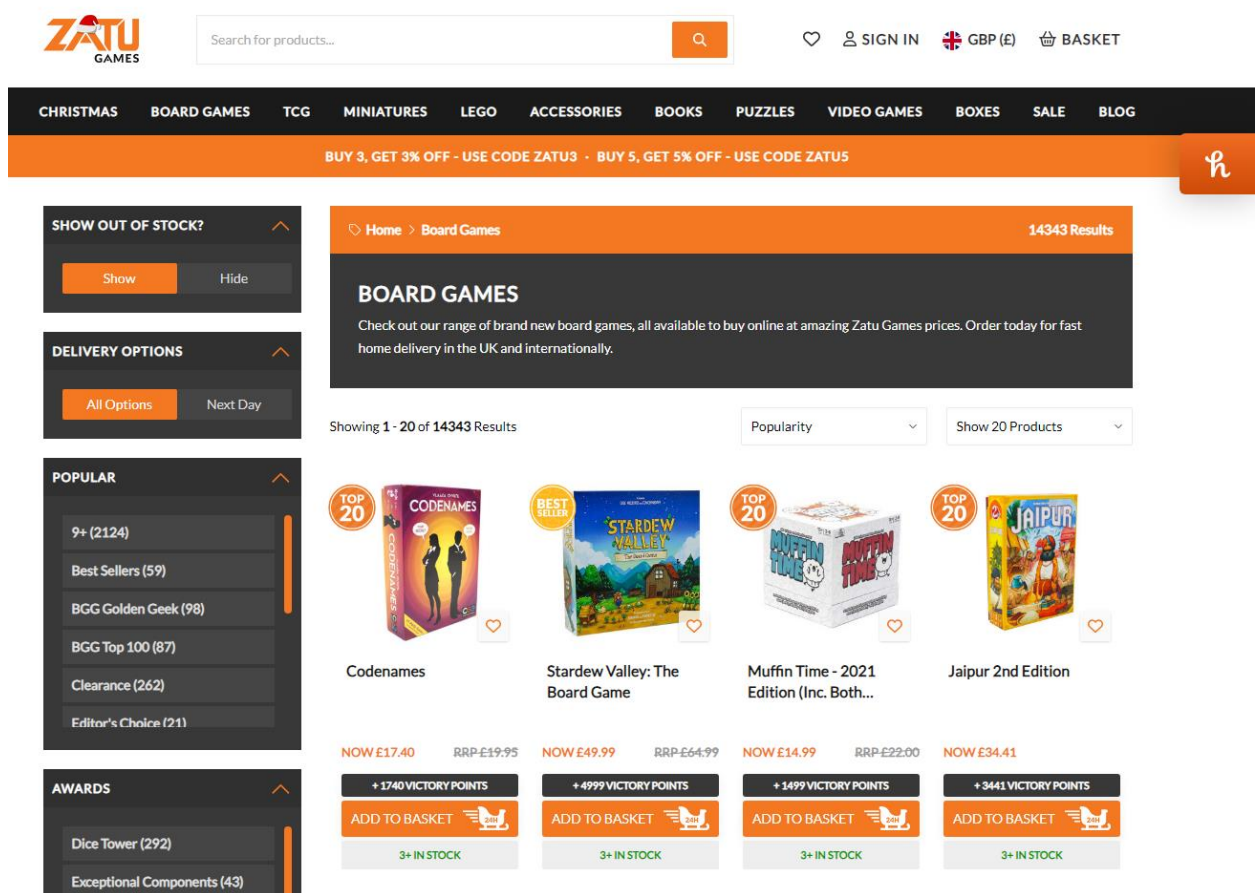


Рисунок 1.2 – Вигляд вебзастосунку Zatu Games для користувачів

Zatu Games, на відміну від BoardGameGeek, направлений на продаж

настільних ігор, надаючи для цього доставку і маючи великий банк таких ігор. Окрім можливості придбати настільну гру вебзастосунок має свою рейтингову систему, яка включає показники візуальної краси, складності, взаємодії з гравцями, якості компонентів, відтворюваності. Ці рейтинги створюються експертами, а звичайні клієнти можуть залишати свої відгуки, виставляючи максимум 5 балів кожній грі та вказуючи свою думку в текстовому вигляді.

Також магазином, але українським є вебсайт Geekach (рис. 1.3) [5].

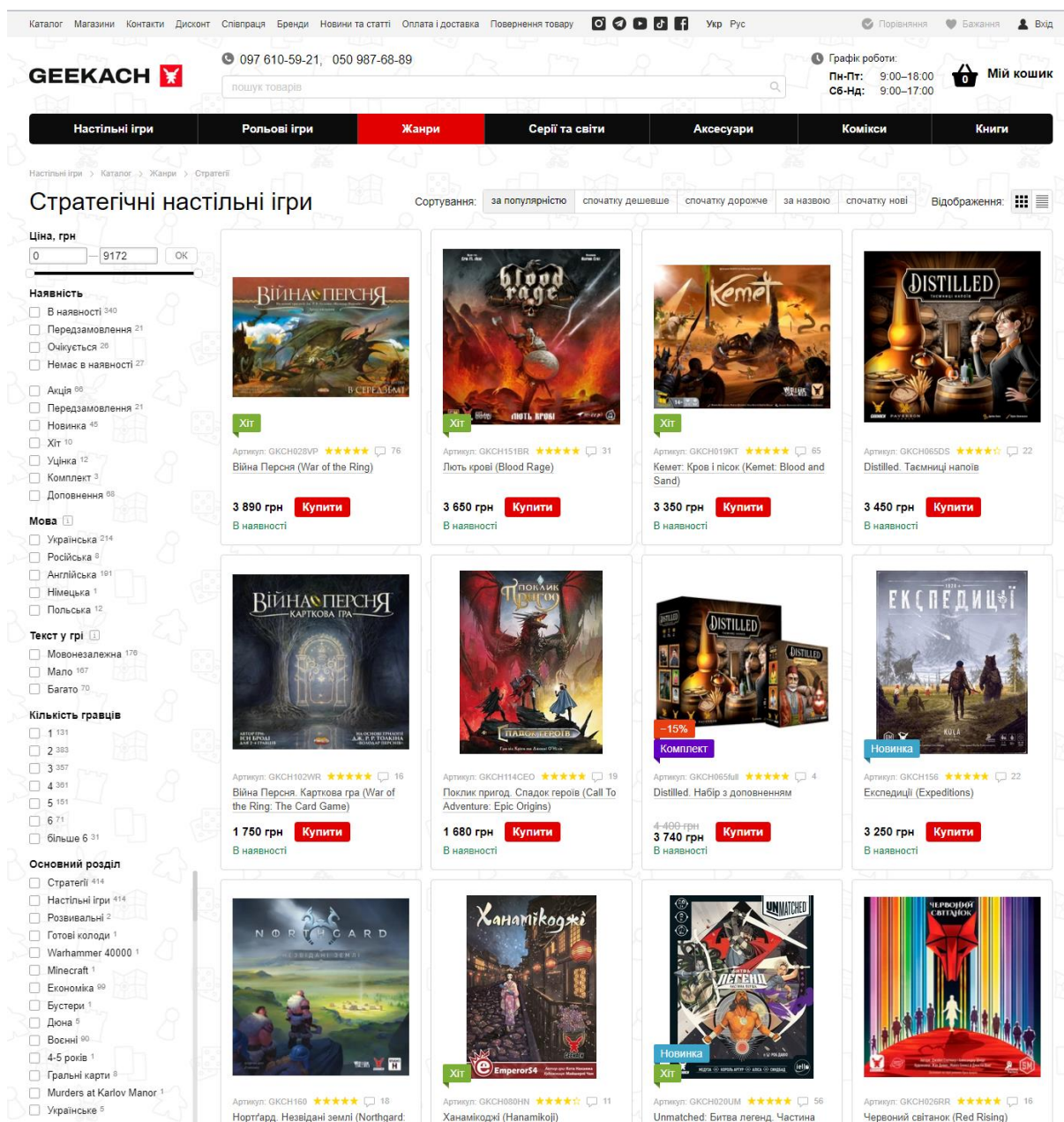


Рисунок 1.3 – Вигляд вебзастосунку Geekach для користувачів

Zatu Games має пошук тільки за назвою гри, відповідно знайти настільні ігри можна за категоріями, рейтингами, а Geekach надає велику кількість фільтрів, які підвищують ймовірність того, що клієнт знайде ту настільну гру, яка задовольняє його інтереси порівняно з Zatu Games. При цьому обидва вебзастосунки не мають системи рекомендацій, що було характерно і для BoardGameGeek. Для Geekach пошук настільних ігор представлений окрім перегляду настільних ігор за жанрами, серіями. Користувачі можуть залишати відгуки за іграми, при цьому вебсайт містить достатньо велику кількість відгуків за різними іграми. За кожною грою надається інформація про її вартість та умови доставки.

1.2 Порівняння розроблюваної програми з існуючими

Усі розглянуті програми мають свої особливості, тому в підсумку порівняння програми OSBoardSoft, яка розробляється, було виконано одразу зі всіма виокремленими аналогами. Розробка повинна дозволяти виконувати пошук настільних ігор за різними фільтрами, на відміну від Zatu Games, але так само як і BoardGameGeek, і Geekach. Програма має об'єднувати в собі засоби отримання інформації про настільні ігри, в тому числі впорядкування таких ігор у вигляді власних переліків, що відсутнє в аналогах, та можливості замовлення ігор в магазині, який керує даним вебзастосунком. Окрім того вирішено надавати можливості реєстрації сеансів ігор, що характерно тільки для BoardGameGeek, таким чином збільшуючи масив даних користувачів у програмі, що має сприяти залученості користувачів до роботи з вебзастосунком. Відповідно більша залученість визначає більшу ймовірність придбання настільних ігор з вебсайту. Також відмінністю розробки (табл. 1.1) є те, що вона повинна дозволяти виконувати пошук за пріоритетними для користувача настільними іграми, таким чином дозволяючи отримати пропозицію щодо настільної гри, не визначаючи параметрів пошуку, якщо це не необхідно.

Таблиця 1.1 – Характеристика розроблюваної та існуючих програм

Показник	BoardGame-Geek	Zatu Games	Geekach	OSBoardSoft
Пошук настільних ігор	Так	Так, тільки за назвою	Так	Так
Замовлення настільних ігор	Ні	Так	Так	Так
Збереження настільних ігор у програмі	Так, у наперед визначені переліки	Так	Так	Так, у власні переліки
Реєстрація сеансів	Так	Ні	Ні	Так
Рейтинг настільних ігор	За відгуками	За оцінками експертів	За відгуками	За відгуками
Пошук за пріоритетними для користувача настільними іграми	Ні	Ні	Ні	Так

1.3 Висновки за розділом 1

Набір завдань для виконання в роботі:

- аналіз існуючих програм, які виконують інформування про настільні ігри;
- дослідження методів і моделей пошуку настільних ігор;
- проектування ПЗ пошуку настільних ігор;
- реалізація програми пошуку настільних ігор.

Аналіз існуючих програм, які виконують інформування про настільні ігри, є результатом виконання розділу 1. Аналіз включав онлайн-форум для любителів настільних ігор BoardGameGeek, програми для підтримки продажу настільних ігор Zatu Games, Geekach, для яких визначено функціональні можливості, з яких основні виділені для реалізації у програму пошуку настільних ігор, що створюється в роботі, а також визначено її особливості у вигляді пошуку за пріоритетними для користувача настільними іграми, створення переліків настільних ігор та збереження ігор в цих переліках.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Використання методів кластеризації для розв'язання задачі пошуку настільних ігор

Враховуючи те, що кількість настільних ігор весь час збільшується, ефективний пошук для користувача все більше ускладнюється. Отримання переліку всіх настільних ігор, які відповідають заданим критеріям пошуку вже недостатньо, адже тоді єдиним інструментом орієнтування в результатах, які формально відповідають запиту користувача, є сортування цих даних, однак це не зменшує кількість результатів пошуку. Важливим кроком щодо розв'язання цієї проблеми є відділення тих настільних ігор, які потенційно можуть цікавити користувача. Тоді якщо його просто цікавлять настільні ігри, в тому числі ті, що викликають у нього інтерес, і ті, що ні, тобто потрібно знайти інформацію про специфічну гру, то він може використовувати звичайний пошук, а якщо ж він шукає наступну настільну гру для себе, то такий пошук можна виконувати вже на основі пріоритетних для нього настільних ігор. Такі пріоритетні ігри також можна вважати рекомендаціями, але кількість рекомендацій в такому випадку може бути значною, тобто на отриманих результатах вже можна виконувати звичайний пошук у підсумку. Тому логічним є розподілення даних про настільні ігри на певні кластери, а для кожного користувача визначення кластера, настільні ігри з яких йому найбільше подобаються. Тоді таким чином визначений кластер настільних ігор як раз і дає виділити набір пріоритетних для користувача настільних ігор. Таким чином, задача пошуку настільних ігор є задачею кластеризації.

Одним з основних методів розв'язання задачі пошуку настільних ігор є метод k -середніх: після визначення кількості груп даних k , які і називаються кластерами, що є однорідними та компактними, відбувається випадкове визначення 3 центроїдів і пов'язування їх із k мітками, тоді для кожної точки у вибірці даних обчислюється відстань до цих центроїдів і

пов'язується кожна така точка з найближчим центроїдом і відповідною йому міткою центроїду (номером кластеру), далі цей процес повторюється, доки нові центроїди не перестануть змінюватися від попередніх [6]-[7].

2.2 Метод пошуку настільних ігор

У підсумку для розв'язання задачі пошуку настільних ігор було створено метод, який базується на попередньому обробленні наявних настільних ігор, виділенні з них тих, які можуть відноситися до пріоритетних в принципі, потім кластеризації всіх даних, які залишились у вибірці, визначенні персонального вибору користувача на основі минулого досвіду, а тоді співвіднесення і вибір підходящого кластеру настільних даних до цього досвіду, після чого всі настільні ігри, які відносяться до того ж кластера, вважаються пріоритетними, а користувач може задавати додаткові фільтри та відбирати додатково з них ті, які йому підходять.

Таким чином, обмеженням даного методу пошуку настільних ігор є те, що він потребує попереднього визначення досвіду користувача, тобто наявності у користувача відгуків про настільні ігри. В тому чи іншому вигляді, тобто у вигляді розширеного відгуку або у вигляді лише оцінки користувач повинен висловити своє відношення до існуючих ігор. При чому серед всіх оцінок або відгуків повинен бути щонайменше один позитивний. Позитивним вирішено вважати оцінки від 8 за шкалою у 10 балів та від 4 за шкалою у 5 балів. Тобто такі оцінки вказують на те, що користувачу точно більше сподобалась ця гра. Такі умови з одного боку визначають, що користувач повинен мати певний досвід оцінювання настільних ігор, але з іншого боку це визначає, що користувач не обов'язково повинен мати накопичений одразу досвід, а тим більше у вигляді відгуків. Для отримання даних користувачу може бути просто запропоновано пройти опитування, де вказати свої відношення до декількох настільних ігор. З цього опитування

вже можна буде виділити інформацію про те, яким настільним іграм віддає перевагу користувач.

Окрім даного етапу має бути виконано етап, на якому має бути виконано попереднє оброблення наявних настільних ігор, виділені з них ті, які можуть відноситися до пріоритетних в принципі. Це питання може бути врегульовано особами, які приймають рішення, наприклад, у певному магазині з продажу настільних ігор. Наприклад, до такого переліку можуть бути віднесені тільки ті настільні ігри, які магазин рекламує. Тобто в підсумку рекомендації будуть відповідати персональним перевагам користувача, але обиратися з набору, який сформований за попередніми вимогами. З іншого боку настільними іграми, які рекламуються, може бути лише частина ігор, а інші представлені вже додатковими без окремої реклами (наприклад, половина таких, половина таких). З іншого боку до даного набору настільних ігор можуть бути віднесені ті, які спільнота вважає вартими уваги, тобто ті, які мають позитивну оцінку.

Кожна настільна гра повинна включати значення за наступним набором характеристик:

- рік видання настільної гри;
- мінімальна кількість гравців, які можуть грати в цю настільну гру;
- максимальна кількість гравців, які можуть грати у цю настільну гру;
- середній ігровий час;
- мінімальний вік, з якого можна грати в цю настільну гру;
- рейтинг (оцінка), виставлений користувачами;
- рівень складності гри (може використовуватися окрема оцінка за цим критерієм);
- кількість користувачів, які володіють цією настільною грою, що фактично говорить про популярність настільної гри;
- основна категорія настільної гри;
- дві основні ігрові механіки (кидання ігрових костей та інші такі дії), що використовуються в цій грі.

Три останні значення (категорія і дві ігрові механіки) є категоріальними значеннями, тобто відповідні категорії та механіки мають бути виділені, а кожній з них встановлено у відповідність фактично номер категорії для представлення у вибірці.

Після того, як виділено початковий набір даних та персональні характеристики користувача (клієнта), має бути виконано кластеризацію даних (рис. 2.1).



Рисунок 2.1 – Метод пошуку настільних ігор

Кластеризацією має бути виконано на основі використання методу k-середніх, для якого має бути встановлено необхідну кількість кластерів. Кількість кластерів необхідно задавати, враховуючи, що ця кількість визначає кількість настільних ігор, які будуть вважатися для користувач пріоритетними. Тобто попередньо відібрана кількість настільних ігор, поділена на кількість кластерів, визначає кількість результатів.

Після формування кластерів серед даних настільних ігор має бути визначено ті відгуки користувача, які слід вважати позитивними, а тоді співвіднесено настільні ігри, яких ці відгуки стосуються, з певними кластерами, що були сформовані. Тобто фактично виконано ідентифікацію того, до якого кластера кожна така настільна гра належить.

Після цього має бути визначено, який кластер зустрічається найчастіше серед цих варіантів. Саме такий кластер і має бути визначений таким, якому встановлено пріоритет стосовно даного користувача. Всі настільні ігри, що віднесені до цього кластера, можуть бути рекомендовані користувачу.

На останньому етапі користувач визначає параметри власного пошуку, а серед всіх пріоритетних для нього настільних ігор відбираються ті, що цим параметрам пошуку відповідають. Сформований таким чином набір настільних ігор повертається користувачу як результат роботи методу пошуку настільних ігор.

2.3 Висновки за розділом 2

Постановка задачі пошуку настільних ігор, визначення інструментів розв'язання задачі, створення зокрема методу пошуку настільних ігор є результатами виконання розділу 2.

Наукова новизна роботи полягає в запропонованому методі пошуку настільних ігор, в якому визначається попередній набір даних, встановлюються персональні переваги користувача на основі відгуків про

настільні ігри, виконується кластеризація даних на основі методу k-середніх, відбувається визначення пріоритетних настільних ігор для користувача та виконується відбір серед них за параметрами пошуку, встановленими користувачем.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Визначення сценаріїв використання програми користувачами

Програму можуть використовувати 3 типи користувачів:

- гравці, які є користувачами, які можуть отримувати інформацію про настільні ігри, ознайомлюватися з інформацією про інші настільні ігри, фіксувати і ділитися власним досвідом, замовляти собі нові настільні ігри;
- працівники магазинів з продажу настільних ігор, які виконують роботу з внесення даних щодо настільних ігор у конкретному магазині, керування замовленнями настільних ігор, оформленими на доступний їм магазин;
- адміністратори системи, які можуть виконувати дії з підтримки загальної роботи щодо внесення інформації про магазини з продажу настільних ігор, а також будь-які інші дії, доступні в програмі.

Гравці мають доступними для себе наступні сценарії:

- перегляд сторінки з інформацією про настільну гру;
- замовлення настільних ігор для придбання;
- перегляд власних оформлених замовлень настільних ігор;
- скасування оформленого замовлення настільних ігор;
- перегляд рейтингу настільних ігор за параметрами;
- визначення відгуку про настільну гру;
- реєстрація проведеного сеансу настільної гри;
- оновлення даних сеансу настільної гри;
- перегляд власних сеансів настільних ігор;
- перегляд спільних сеансів настільних ігор з іншими гравцями;
- перегляд даних і настільних ігор творця;
- перегляд даних і настільних ігор виробника;
- перегляд усіх виробників настільних ігор;
- перегляд усіх творців, долучених до створення настільних ігор;
- пошук настільних ігор;

- створення персонального переліку настільних ігор;
- додавання настільної гри до персонального переліку;
- вилучення настільної гри з персонального переліку;
- перегляд настільних ігор з персонального переліку;
- перегляд власних персональних переліків настільних ігор;
- перегляд інформації про гравця;
- перегляд відгуків гравця;
- оформлення підписки на гравця;
- оформлення підписки на виробника настільних ігор;
- оформлення підписки на творця;
- скасування підписки на гравця;
- скасування підписки на виробника настільних ігор;
- скасування підписки на творця;
- перегляд творців, на яких оформлено підписку;
- перегляд гравців, на яких оформлено підписку;
- перегляд виробників настільних ігор, на яких оформлено підписку;
- перегляд настільних ігор творців, на яких оформлено підписку;
- перегляд відгуків гравців, на яких оформлено підписку;
- перегляд настільних ігор виробників, на яких оформлено підписку;
- перегляд відгуків про настільну гру;
- оновлення власних даних;
- авторизація;
- визначення власної настільної гри;
- скасування визначення власної настільної гри;
- перегляд настільних ігор у власності користувача;
- пошук за пріоритетними для користувача настільними іграми

(рис. 3.1).

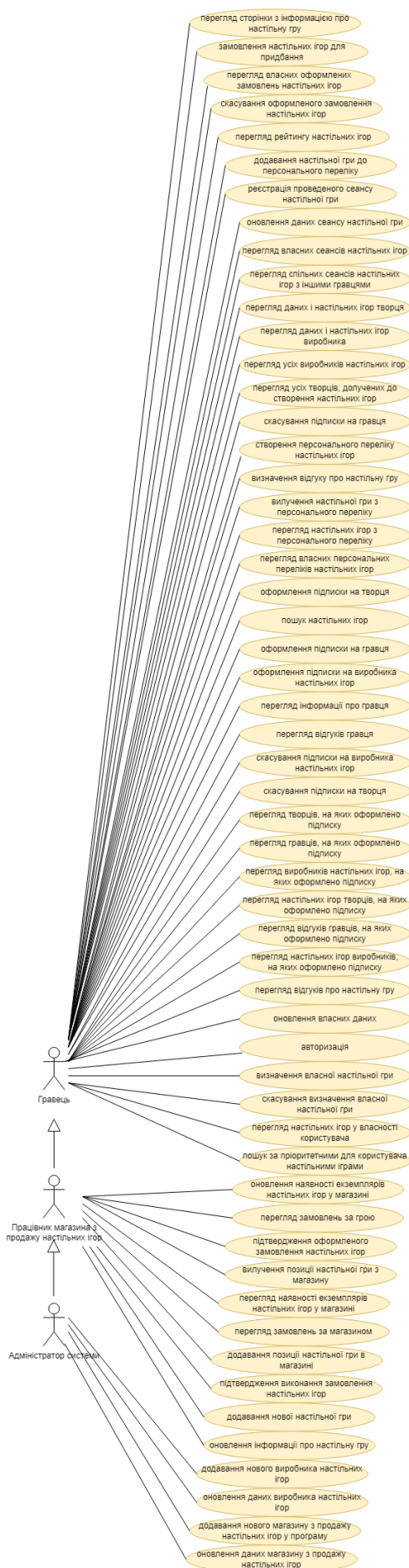


Рисунок 3.1 – Діаграма прецедентів користувачів при роботі з ПЗ

Працівники магазинів з продажу настільних ігор можуть використовувати всі сценарії гравця, а також наступні сценарії:

- перегляд замовлень за магазином;
- перегляд замовлень за грою;
- підтвердження оформленого замовлення настільних ігор;
- підтвердження виконання замовлення настільних ігор;
- перегляд наявності екземплярів настільних ігор у магазині;
- оновлення наявності екземплярів настільних ігор у магазині;
- додавання позиції настільної гри в магазині;
- вилучення позиції настільної гри з магазину;
- додавання нової настільної гри;
- оновлення інформації про настільну гру.

Адміністратори системи можуть використовувати всі сценарії працівника магазину з продажу настільних ігор, а також наступні сценарії:

- додавання нового виробника настільних ігор;
- оновлення даних виробника настільних ігор;
- додавання нового магазину з продажу настільних ігор у програму;
- оновлення даних магазину з продажу настільних ігор.

3.2 Рішення щодо вибору засобів розробки

Мова Python є домінуючою щодо створення засобів машинного навчання, роботи з даними, тому її одразу було вирішено використовувати для створення програмного проєкту, проте вебфреймворк для подальшої розробки міг бути різним. Основними фреймворками для Python є Django [8] і Flask [9]. Порівняння цих фреймворків (табл. 3.1) визначає, що Django має і вбудоване об'єктно-реляційне відображення (ОРВ), і засоби авторизації та роботи адміністратора, тож такий підхід, підсилений відлагодженістю процесів, визначив вибір на користь Django.

Таблиця 3.1 – Порівняння засобів для розробки ПЗ пошуку настільних ігор

Критерій	Django	Flask
Робота з базами даних	Вбудоване ОРВ	ОРВ можливе тільки через розширення
Мова програмування	Python	Python
Авторизація	Вбудований механізм авторизації Django auth	Потрібно встановлювати розширення
Шаблонізатор	Використовується спеціальна мова шаблонів, можна підключати інші шаблонізатори	Jinja2
Можливість написання і підключення засобів машинного навчання	Так	Так

3.3 Визначення структури даних у базі даних

База даних (БД) є основним сховищем, яке вміщує фактично всі дані програми окрім статичних даних вебсторінок та вибірки даних, що використовувалась під час реалізації методу пошуку настільних ігор.

Саму настільну гру в БД виділено у вигляді таблиці Boardgame, сформованої на основі полів:

- id – унікальний номер настільної гри, що є первинним ключем в цій таблиці;
- gamename – назва настільної гри;
- shortinfo – короткий опис настільної гри та її особливостей;
- fullrules – повний опис правил настільної гри;

- basephoto – фотографія, яка є основною для цієї настільної гри;
- maindesigner – головний дизайнер цієї настільної гри, що є зовнішнім ключем, що веде до таблиці Artist;
- developer – головний розробник цієї настільної гри, що є зовнішнім ключем, що веде до таблиці Artist;
- editor – головний редактор цієї настільної гри, що є зовнішнім ключем, що веде до таблиці Artist;
- publisher – виробник цієї настільної гри, що є зовнішнім ключем, що веде до таблиці Publisher;
- originalgame – гра версією якої є дана настільна гра (якщо вона є версією, а не оригіналом, інакше дане поле має бути залишене порожнім), що є зовнішнім ключем, що веде до іншого запису цієї ж таблиці Boardgame;
- mintime – мінімальний час, який можна грати в цю настільну гру (час, за який можна завершити один повний цикл мінімально);
- maxtime – максимальний час, протягом якого можна грати в цю настільну гру (один повний цикл);
- minage – мінімальний вік, з якого можна грати в цю настільну гру;
- minplayers – мінімальна кількість гравців, які можуть одночасно брати участь у цій настільній грі;
- maxplayers – максимальна кількість гравців, які можуть одночасно брати участь у цій настільній грі;
- recommendedtime – час, який рекомендовано витратити на цю настільну гру за інформацію, наданою у відгуках гравців;
- recommendedplayers – кількість гравців, яка оптимально мала б брати участь для гри в цю настільну гру за інформацію, наданою у відгуках гравців;
- rating – середній рейтинг, який цій настільній грі було встановлено у відгуках гравців;
- artwork – середнє значення, яке у відгуках було виставлено цій настільній грі гравцями за показником візуальної краси;

- complexity – середнє значення, яке у відгуках було виставлено цій настільній грі гравцями за показником складності;
- interaction – середнє значення, яке у відгуках було виставлено цій настільній грі гравцями за показником взаємодії з гравцями;
- quality – середнє значення, яке у відгуках було виставлено цій настільній грі гравцями за показником якості компонентів настільної гри;
- video – посилання на відеокліп, де показано цю настільну гру або виконано її огляд;
- price – стандартна поточна вартість цієї настільної гри в магазинах цієї мережі;
- newprice – актуальна поточна вартість цієї настільної гри в магазинах цієї мережі, якщо вона є зниженою в даний момент;
- newtill – дата, до якої використовується знижена вартість цієї настільної гри;
- publishdate – рік, коли настільну гру було створено.

Редактори, дизайнери, розробники представлені в БД окремими записами, щоби накопичувати інформацію про них, а потім використовувати ці дані для впорядкування настільних ігор, в розробці яких вони приймали участь, а також надання інформації про них при переході з відповідних позицій сторінки настільної гри. Такі записи знаходяться в таблиці Artist, сформованої на основі полів:

- id – унікальний номер учасника розробки настільних ігор;
- fullname – повне ім'я учасника розробки настільних ігор;
- description – довідкова інформація про учасника розробки настільних ігор;
- basephoto – фотографія учасника розробки настільних ігор;
- website – адреса вебсайта, який є персональним для цього учасника розробки настільних ігор.

У подальшому для позначення записів у таблиці Artist використано узагальнюючий термін творець.

Оскільки один і той же учасник може виконувати різну роль в різних настільних іграх, то серед полів немає поля, яке б визначало жорстко роль, адже це поняття, яке зв'язує конкретного учасника розробки настільних ігор з конкретною грою.

Виробники настільних ігор є компаніями або маленькими стартапами, які можуть виготовляти різні настільні ігри, тому доцільно було інформацію про них також винести в окрему таблицю Publisher, сформовану на основі полів:

- id – унікальний номер виробника настільних ігор;
- firmname – назва виробника настільних ігор;
- description – довідкова інформація про виробника настільних ігор;
- basephoto – зображення, яким показано фірму-виробника настільних ігор;
- website – адреса вебсайта, який є корпоративним сайтом виробника настільних ігор.

Для демонстрації настільної гри може використовуватися ціла множина фотографій, тому було створено окрему таблицю Boardgamephoto, яка і має містити самі фотографії з зображенням настільної гри, зроблені як магазином, так і користувачами (гравцями) програми. Дана таблиця сформована на основі полів:

- id – унікальний номер фотографії настільної гри;
- photoauthor – користувач, який завантажив дану фотографію в програму, що є зовнішнім ключем, що веде до таблиці Gamer;
- photofile – сама фотографія, де показано цю настільну гру;
- dateadded – час коли цю фотографію було завантажено в програму користувачем.

Оскільки про настільні ігри в програмі не тільки надається інформація, але і має надаватися можливість замовити екземпляри, то в БД було виділено таблицю з даними магазинів (такий магазин може бути і один), які продають ці настільні ігри через програму. При цьому використовується

принцип єдиної ціни, тобто вартість у всіх магазинах має бути однаковою, тому дані про вартість винесені в таблицю з самою настільною грою, яку було описано вище. Магазины, що продають настільні ігри, знаходяться в таблиці Gameshop, сформованій на основі полів:

- id – унікальний номер магазину, що продає настільні ігри через програму;
- shopname – назва магазину, що продає настільні ігри через програму;
- basephoto – фотографія з зображенням магазину, що продає настільні ігри через програму;
- postaddress – поштова адреса, за якою знаходиться магазин, що продає настільні ігри через програму;
- description – довідкова інформація про магазин, що продає настільні ігри через програму;
- city – місто, де знаходиться магазин, що продає настільні ігри через програму.

Оскільки передбачається невелика кількість магазинів, а всі вони тоді логічно знаходяться в одній країні, то вирішено не виносити самі міста до окремої таблиці, а визначати їх програмно, як перелік міст, які є у певній країні.

Для зберігання даних про те, яка кількість екземплярів кожної настільної гри є в магазині, створено таблицю Boardgameshop, сформовану на основі полів:

- id – унікальний номер запису про кількість екземплярів настільної гри в магазині;
- gameshop – магазин, що продає настільні ігри через програму і в якому є така настільна гра в наявності або вона може бути в певний період часу, що є зовнішнім ключем, що веде до таблиці Gameshop;

– boardgame – настільна гра, яка є в цьому магазині або може бути в ньому в наявності на певний час, що є зовнішнім ключем, що веде до таблиці Boardgame;

– quantity – кількість екземплярів гри, яка є доступною для продажу через цей магазин на даний момент.

Через те, що в розробці настільної гри можуть брати участь декілька учасників, а в таблиці з даними про саму настільну гру можна вказати тільки одного розробника, дизайнера тощо, а більше вказувати недоцільно, адже настільна гра може мати і взагалі одного розробника, дизайнера тощо, то додаткові учасники розробки настільних ігор винесені в таблицю БД BoardgameArtist, сформовану на основі полів:

– id – унікальний номер запису про додаткового учасника розробки настільних ігор;

– boardgame – настільна гра, в розробці якої взяв участь цей творець, що є зовнішнім ключем, що веде до таблиці Boardgame;

– artist – творець, який взяв участь в розробці цієї настільної гри, що є зовнішнім ключем, що веде до таблиці Artist;

– details – деталі щодо участі цього творця в розробці цієї настільної гри.

Усі настільні ігри потрібно певним чином об'єднувати. Одним шляхом об'єднання є об'єднання різних версій однієї і тієї самої настільної гри. Іншим – виділення категорій настільних ігор, які знаходяться в таблиці Gamecategory, сформованій на основі полів:

– id – унікальний номер категорії настільних ігор;

– categoryname – назва категорії настільної гри;

– description – опис особливостей, які дозволяють віднести настільну гру до цієї категорії.

Зв'язування ж категорій настільних ігор з самими настільними іграми, адже кожна настільна гра може відноситися до різних категорій, а кожна

категорія фактично має представлятися різними настільними іграми, відбувається через таблицю Boardgamecategory, сформовану на основі полів:

- id – унікальний номер запису про категорію настільної гри;
- boardgame – настільна гра, для якої встановлена категорія, що є зовнішнім ключем, що веде до таблиці Boardgame;
- gamecategory – категорія, до якої належить ця настільна гра, що є зовнішнім ключем, що веде до таблиці Gamecategory.

Усі настільні ігри мають свої особливості, але і певні загальні принципи. Тільки частина загальної ідеї виражається категорією, адже це ніяк не визначає фактично концепцію. Але при цьому настільні ігри нерідко мають певні загальні принципи. Наприклад, деякі настільні ігри можуть потребувати використання грального кубика та його викидання для визначення зокрема кількості ходів, які має зробити гравець, або іншої необхідної для реалізації гри кількості. Такі принципи і називаються механізмами. Загальні механізми настільних ігор знаходяться в таблиці Gamemechanism, сформованій на основі полів:

- id – унікальний номер запису про механізм;
- mechname – назва цього механізму;
- description – опис принципів, на яких цей механізм базується в настільних іграх зазвичай.

У одній настільній грі можуть задіюватися різні механізми, тому зв'язування настільних ігор з механізмами відбувається в таблиці Boardgamemechanism, сформованій на основі полів:

- id – унікальний номер запису про використання ігрового механізму в настільній грі;
- boardgame – настільна гра, в якій використовується цей ігровий механізм, що є зовнішнім ключем, що веде до таблиці Boardgame;
- gamemechanism – ігровий механізм, який використовується в цій настільній грі, що є зовнішнім ключем, що веде до таблиці Boardgamemechanism;

– details – опис деталей застосування саме цього ігрового механізму в цій настільній грі.

Користувачі програми можуть бути як безпосередньо просто гравцями чи клієнтами магазину, так і працівниками магазину, але оскільки вони загалом мають однакову інформацію про себе в програмі, то їх було об'єднано в єдину таблицю Gamer, сформовану на основі полів:

- id – унікальний номер облікового запису, який має користувач програми;
- username – унікальний текстовий ідентифікатор облікового запису (логін);
- password – пароль, який використовується для забезпечення авторизації облікового запису в програмі;
- displayedname – ім'я, яке показується в програмі щодо цього користувача іншим користувачам;
- description – опис себе, своїх інтересів щодо настільних ігор та загалом користувачем;
- basephoto – фотографія або зображення, яким візуалізує себе користувач;
- shopworker визначає, чи це звичайний користувач (гравець), чи це працівник магазину.

Відгуки про настільні ігри створюються користувачами і визначаються загальною частиною, а також значеннями за набором критеріїв. Значення за критеріями є окремими складовими, адже користувач може встановити для них значення за бажанням. Тобто він може надати тільки загальну інформацію, оцінку, але за бажанням може оцінити гру і за окремими критеріями. При цьому оцінити можна настільну гру і тільки за одним критерієм, і за декількома. Тож перша частина відгуку, яка є загальною, знаходиться в таблиці Boardgamerating, сформованій на основі полів:

- id – унікальний номер відгуку про настільну гру;

- boardgame – настільна гра, стосовно якої створено цей відгук, є зовнішнім ключем, що веде до таблиці Boardgame;
- gamer – користувач, який створив цей відгук про настільну гру, є зовнішнім ключем, що веде до таблиці Gamer;
- ratingdate – час, коли було створено відгук про настільну гру;
- rating – значення, на яке було загалом оцінено настільну гру (передбачається оцінювання максимум в 5 балів);
- ratingtext – текстова частина відгуку про настільну гру;
- owner визначає, чи автор відгуку про настільну гру є власником цієї гри, чи ні;
- played визначає, чи автор відгуку про настільну гру грав у неї особисто, чи ні;
- want визначає, чи автор відгуку про настільну гру має активне бажання придбати цю гру (якщо він її ще не має), чи ні;
- recommendedtime – час, який проводити за грою в цю настільну гру вважає оптимальним цей автор відгуку про настільну гру;
- recommendedplayers – кількість гравців, участь яких у цій настільній грі вважає оптимальним цей автор відгуку про настільну гру.

Оцінювання за окремими критеріями, що є додатковою частиною відгуку про настільну гру, яку користувач заповнює за наявності додаткового бажання, знаходяться в таблиці Boardgameratingcritetion, сформованій на основі полів:

- id – унікальний номер оцінювання настільної гри за критерієм;
- boardgamerating – відгуку про настільну гру, якого стосується це оцінювання, є зовнішнім ключем, що веде до таблиці Boardgamerating;
- criterion – критерій, за яким було виконано оцінювання настільної гри (набір критеріїв задається програмно, був перелічений у таблиці Boardgame);
- criterionvalue – значення оцінки, яке було виставлено цим користувачем за цим критерієм стосовно настільної гри.

Оскільки користувач можуть не тільки шукати інформацію про настільні ігри, але і замовляти їх, то окремо було виділено таблицю `Gamerorder` з полями:

- `id` – унікальний номер замовлення на придбання настільних ігор;
- `gamer` – користувач, який створив це замовлення на придбання настільних ігор, є зовнішнім ключем, що веде до таблиці `Gamer`;;
- `orderdate` – час, коли було збережено замовлення на придбання настільних ігор початково користувачем;
- `state` – стан, в якому знаходиться замовлення на придбання настільних ігор;
- `price` – сумарна вартість замовлення на придбання настільних ігор;
- `gameshop` – магазин, в якому продаються настільні ігри, щодо якого і було сформовано замовлення (якщо значення не встановлено, то відбувається працівниками магазинів, які пов’язані зі всіма магазинами, визначення того, який магазин має бути відповідальним, а тому відбувається зв’язування);
- `deliveryaddress` – адреса доставки настільних ігор, якщо замовлення не буде забрано клієнтом самостійно.

Безпосередньо дані вмісту замовлення, тобто настільні ігри, які були внесені в замовлення, знаходяться в таблиці `Gamerordergame`, сформованій на основі полів:

- `id` – унікальний номер запису про позицію настільної гри в замовленні;
- `boardgame` – настільна гра, яка входить до замовлення, є зовнішнім ключем, що веде до таблиці `Boardgame`;
- `gameorder` – замовлення на придбання настільних ігор, в яке входить ця настільна гра, є зовнішнім ключем, що веде до таблиці `Gamerorder`;
- `number` – кількість екземплярів цієї настільної гри, яка має бути доставлена або віддана за цим замовленням;
- `price` – вартість одного екземпляра цієї настільної гри.

Як було визначено в розділі 1, однією з особливостей програми є можливість створення власних переліків настільних ігор, таким чином, наприклад, визначаючи, які настільні ігри користувач бажає придбати, якими він цікавиться в певній категорії, інформація про які його цікавить. Безпосередньо самі ці сформовані переліки, тобто без вмісту переліків, а загальна інформація про них, знаходиться в таблиці Gamerlist, сформованій на основі полів:

- id – унікальний номер персонального переліку настільних ігор користувача;
- gamer – користувач-гравець, який створив і якому належить цей перелік настільних ігор, є зовнішнім ключем, що веде до таблиці Gamer;
- listname – назва персонального переліку настільних ігор користувача;
- description – опис персонального переліку настільних ігор користувача.

Безпосередньо самі настільні ігри, які користувач включив до свого переліку, знаходяться в таблиці Listgame, сформованій на основі полів:

- id – унікальний номер запису про входження настільної гри в персональний перелік настільних ігор користувача;
- gamerlist – персональний перелік настільних ігор користувача, до якого входить ця настільна гра, є зовнішнім ключем, що веде до таблиці Gamerlist;
- boardgame – настільна гра, яка входить в цей персональний перелік, є зовнішнім ключем, що веде до таблиці Boardgame.

Реєстрація сеансів, проведених за цією настільною грою, тобто іншими словами випадків, коли користувач грав у цю настільну гру, відбувається через таблицю Boardgamelog, сформовану на основі полів:

- id – унікальний номер реєстрації сеансу, проведеного за настільною грою користувачем;

- `gamer` – користувач-гравець, який грав у цю настільну гру, є зовнішнім ключем, що веде до таблиці `Gamer`;
 - `boardgame` – настільна гра, в яку грав цей користувач у зазначеному випадку, є зовнішнім ключем, що веде до таблиці `Boardgame`;
 - `playdate` – дата, коли цей користувач грав у цю настільну гру;
 - `gametimes` – кількість разів, яку в цю дату було зіграно в цю настільну гру за участю цього користувача;
 - `address` – адреса, за якою відбулась ця гра, якщо її вказує користувач (може бути, наприклад, певним розважальним центром чи якимось іншим загальновідомим місцем, яке не є саме по собі адресою);
 - `playtime` – ігровий час;
 - `comment` – коментар щодо цього випадку від самого користувача, який у випадку надання доступу до цієї інформації іншим користувачам, буде їм видним;
 - `personalcomment` – прихований коментар щодо цього випадку від самого користувача, який у випадку надання доступу до цієї інформації іншим користувачам, буде для них закритим і не відображатися.
- Під час реєстрації сеансу користувач може визначити, з якими користувачами він грав у цю настільну гру в цей раз, після чого вони зможуть бачити введену ним інформацію (окрім прихованого коментаря). Дані про те, який користувач брав участь у якій саме настільній грі, знаходяться в таблиці `Boardgameloggamer`, сформованій на основі полів:
- `id` – унікальний номер запису про участь іншого користувача в сеансі гри користувача в настільну гру;
 - `boardgamelog` – реєстрація сеансу, проведеного за настільною грою, в якому взяли участь цей користувач-гравець, є зовнішнім ключем, що веде до таблиці `Boardgamelog`;
 - `gamer` – користувач-гравець, який приймав участь і грі в настільну гру разом з автором реєстрації, є зовнішнім ключем, що веде до таблиці `Gamer` (рис. 3.2).

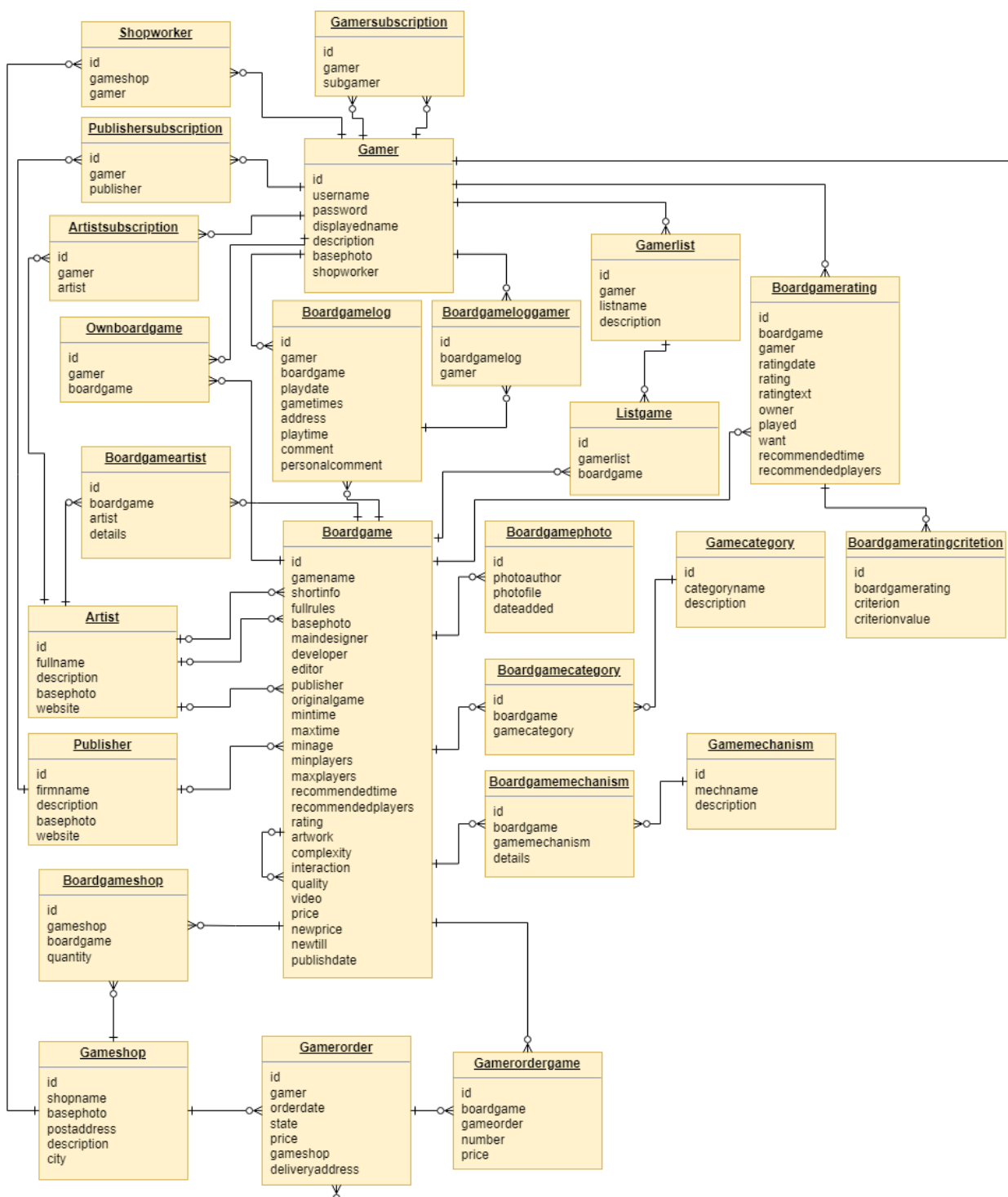


Рисунок 3.2 – Схема БД

Для спрощення внесення цих даних у програмі можна визначити, якими настільними іграми володіє сам користувач-гравець. Такі дані знаходяться в таблиці Ownboardgame, сформованій на основі полів:

- id – унікальний номер запису про володіння настільною грою користувачем-гравцем;

- gamer – користувач-гравець, який володіє цією настільною грою, є зовнішнім ключем, що веде до таблиці Gamer;

- boardgame – настільна гра, якою володіє користувач-гравець, є зовнішнім ключем, що веде до таблиці Boardgame.

Оскільки користувачі можуть бути працівниками магазину з продажу настільних ігор, а деякі з них можуть мати відношення до декількох магазинів, то інформація про доступ користувачів до магазинів з продажу настільних ігор визначається в таблиці Shopworker, сформованій на основі полів:

- id – унікальний номер запису про доступ користувача-працівника до магазину з продажу настільних ігор;

- gameshop – магазин настільних ігор, до якого має доступ цей користувач-працівник, є зовнішнім ключем, що веде до таблиці Gameshop;

- gamer – користувач-працівник, який має доступ до цього магазину настільних ігор, є зовнішнім ключем, що веде до таблиці Gamer.

Окрім всього цього користувачі в програмі можуть також підписуватися на творців, виробників настільних ігор та на інших гравців.

Підписки користувачів-гравців на творців, дані про яких містяться в програмі, знаходяться в таблиці Artistsubscription, сформованій на основі полів:

- id – унікальний номер запису про підписку користувача-гравця на творця;

- gamer – користувач-гравець, який підписався на цього творця, є зовнішнім ключем, що веде до таблиці Gamer;

- artist – творець, на якого ця підписка створена, є зовнішнім ключем, що веде до таблиці Artist.

Підписки користувачів-гравців на виробників настільних ігор, дані про яких містяться в програмі, знаходяться в таблиці Publishersubscription, сформованій на основі полів:

- id – унікальний номер запису про підписку користувача-гравця на виробника настільних ігор;
- gamer – користувач-гравець, який підписався на цього виробника настільних ігор, є зовнішнім ключем, що веде до таблиці Gamer;
- publisher – виробник настільних ігор, на якого ця підписка створена, є зовнішнім ключем, що веде до таблиці Artist.

Підписки користувачів-гравців на інших гравців знаходяться в таблиці Gamersubscription, сформованій на основі полів:

- id – унікальний номер запису про підписку користувача-гравця на іншого гравця;
- gamer – користувач-гравець, який підписався на цього гравця, є зовнішнім ключем, що веде до таблиці Gamer;
- subgamer – користувач-грав, на якого ця підписка створена, є зовнішнім ключем, що веде до таблиці Gamer (рис. 3.2).

Реалізацію роботи з БД вирішено виконати через систему MySQL, що є активно використовуваним рішенням для такої роботи зокрема і в потужних рішеннях, створених у вигляді вебзастосунків.

3.4 Висновки за розділом 3

Діаграма прецедентів, мова Python, фреймворк Django, схема БД є результатами виконання розділу 3.

4 РЕАЛІЗАЦІЯ ПРОГРАМИ

4.1 Визначення послідовності роботи з програмою

Уся послідовність роботи з програмою виражена відповідною функціональною схемою (рис. 4.1).

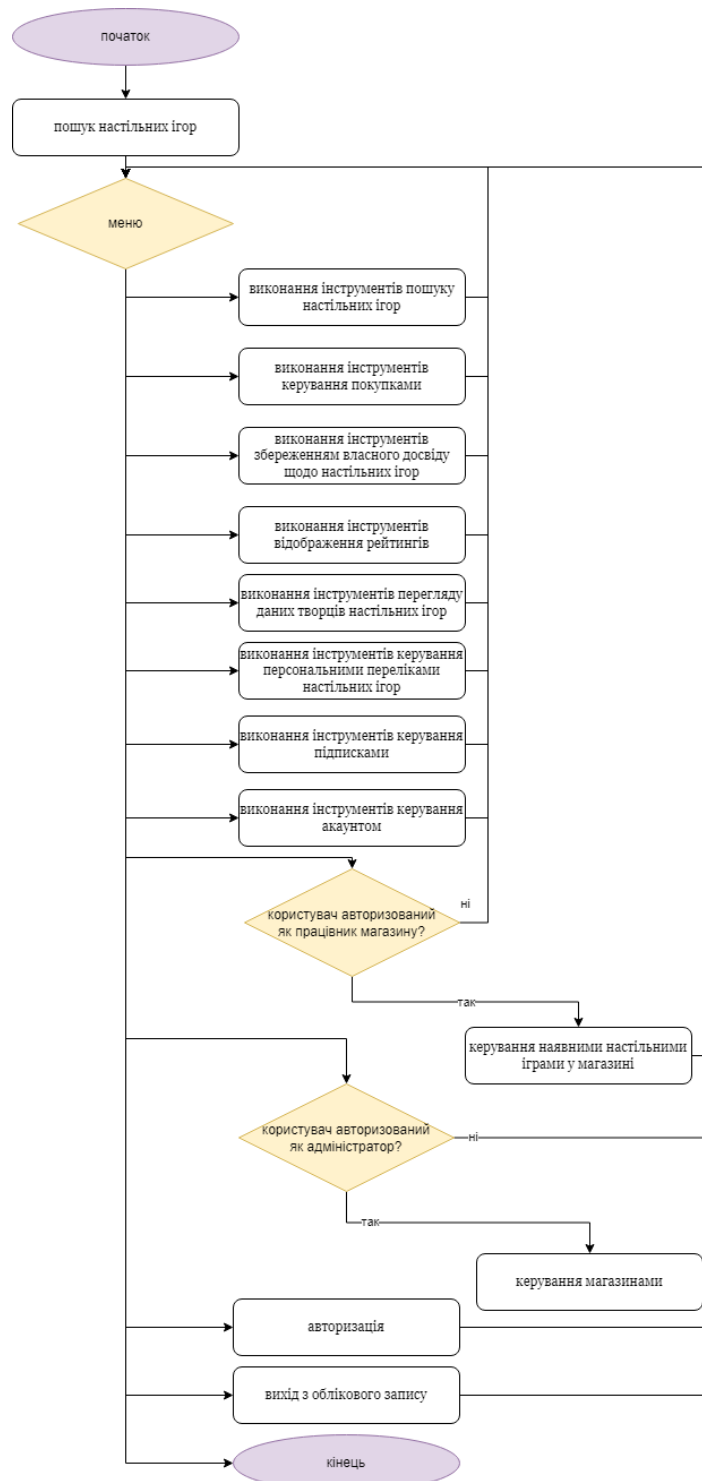


Рисунок 4.1 – Функціональна схема програми пошуку настільних ігор

Функціонування програми у випадку, якщо користувач її розпочинає стандартним зверненням за адресою вебсервера, розпочинається з пошуку настільних ігор, але далі користувач з меню може обрати будь-яку іншу функцію, в тому числі керування наявними настільними іграми у магазині, якщо він авторизований і є працівником цього магазину, керування магазинами, якщо він є адміністратором.

4.2 Особливості реалізації програми

Django як вебфреймворк визначає структуру вебзастосунку на основі моделей, представлень, шаблонів вебсторінок. Шаблони вебсторінок створені на основі вільного шаблону [10]. Далі описані створені моделі та представлення у складі розробленого вебзастосунку.

Модель настільної гри забезпечена створенням класу моделі Boardgame, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут gamename визначається полем CharField, що є назвою настільної гри;
- атрибут shortinfo визначається полем CharField, що є коротким описом настільної гри та її особливостей;
- атрибут fullrules визначається полем TextField, що є повним описом правил настільної гри;
- атрибут basephoto визначається полем ImageField, що є фотографією, яка є основною для цієї настільної гри;
- атрибут maindesigner визначається полем ForeignKey, що є головним дизайнером цієї настільної гри, зовнішнім ключем, що веде до моделі Artist відповідно;
- атрибут developer визначається полем ForeignKey, що є головним розробником цієї настільної гри, зовнішнім ключем, що веде до моделі Artist відповідно;

- атрибут `editor` визначається полем `ForeignKey`, що є головним редактором цієї настільної гри, зовнішнім ключем, що веде до таблиці `Artist` відповідно;
- атрибут `publisher` визначається полем `ForeignKey`, що є виробником цієї настільної гри, зовнішнім ключем, що веде до таблиці `Publisher` відповідно;
- атрибут `originalgame` визначається полем `ForeignKey`, що є грою версією якої є дана настільна гра (якщо вона є версією, а не оригіналом, інакше дане поле має бути залишене порожнім), є зовнішнім ключем, що веде до іншого запису цієї ж моделі `Boardgame`;
- атрибут `mintime` визначається полем `TimeField`, що є мінімальним часом, який можна грати в цю настільну гру (час, за який можна завершити один повний цикл мінімально);
- атрибут `maxtime` визначається полем `TimeField`, що є максимальним часом, протягом якого можна грати в цю настільну гру (один повний цикл);
- атрибут `minage` визначається полем `SmallIntegerField`, що є мінімальним віком, з якого можна грати в цю настільну гру;
- атрибут `minplayers` визначається полем `SmallIntegerField`, що є мінімальною кількістю гравців, які можуть одночасно брати участь у цій настільній грі;
- атрибут `maxplayers` визначається полем `SmallIntegerField`, що є максимальною кількістю гравців, які можуть одночасно брати участь у цій настільній грі;
- атрибут `recommendedtime` визначається полем `TimeField`, що є часом, який рекомендовано витратити на цю настільну гру за інформацію, наданою у відгуках гравців;
- атрибут `recommendedplayers` визначається полем `SmallIntegerField`, що є кількістю гравців, яка оптимально мала б брати участь для гри в цю настільну гру за інформацію, наданою у відгуках гравців;

- атрибут `rating` визначається полем `FloatField`, що є середнім рейтингом, який цій настільній грі було встановлено у відгуках гравців;
- атрибут `artwork` визначається полем `FloatField`, що є середнім значенням, яке у відгуках було виставлено цій настільній грі гравцями за показником візуальної краси;
- атрибут `complexity` визначається полем `FloatField`, що є середнім значенням, яке у відгуках було виставлено цій настільній грі гравцями за показником складності;
- атрибут `interaction` визначається полем `FloatField`, що є середнім значенням, яке у відгуках було виставлено цій настільній грі гравцями за показником взаємодії з гравцями;
- атрибут `quality` визначається полем `FloatField`, що є середнім значенням, яке у відгуках було виставлено цій настільній грі гравцями за показником якості компонентів настільної гри;
- атрибут `video` визначається полем `CharField`, що є посиланням на відеокліп, де показано цю настільну гру або виконано її огляд;
- атрибут `price` визначається полем `FloatField`, що є стандартною поточною вартістю цієї настільної гри в магазинах цієї мережі;
- атрибут `newprice` визначається полем `FloatField`, що є актуальною поточною вартістю цієї настільної гри в магазинах цієї мережі, якщо вона є зниженою в даний момент;
- атрибут `newtill` визначається полем `DateTimeField`, що є датою, до якої використовується знижена вартість цієї настільної гри;
- атрибут `categories` визначається полем `ManyToManyField`, що є переліком категорій, до яких відноситься ця настільна гра, відповідно визначає відношення багато до багатьох до моделі `Gamecategory`;
- атрибут `publishdate` визначається полем `SmallIntegerField`, що є роком створення настільної гри.

Модель творця забезпечена створенням класу моделі `Artist`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `fullname` визначається полем `CharField`, що є повним іменем учасника розробки настільних ігор;
- атрибут `description` визначається полем `CharField`, що є довідковою інформацією про учасника розробки настільних ігор;
- атрибут `basephoto` визначається полем `CharField`, що є фотографією учасника розробки настільних ігор;
- атрибут `website` визначається полем `CharField`, що є адресою вебсайта, який є персональним для цього учасника розробки настільних ігор.

Модель виробника настільних ігор забезпечена створенням класу моделі `Publisher`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `firmname` визначається полем `CharField`, що є назвою виробника настільних ігор;
- атрибут `description` визначається полем `CharField`, що є довідковою інформацією про виробника настільних ігор;
- атрибут `basephoto` визначається полем `CharField`, що є зображенням, яким показано фірму-виробника настільних ігор;
- атрибут `website` визначається полем `CharField`, що є адресою вебсайта, який є корпоративним сайтом виробника настільних ігор.

Модель фотографії настільної гри забезпечена створенням класу моделі `Boardgamephoto`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `photoauthor` визначається полем `CharField`, що є користувачем, який завантажив дану фотографію в програму, зовнішнім ключем, що веде до моделі `Gamer`;
- атрибут `photofile` визначається полем `CharField`, що є самою фотографією, де показано цю настільну гру;
- атрибут `dateadded` визначається полем `CharField`, що є часом, коли цю фотографію було завантажено в програму користувачем.

Модель магазину, що продає настільні ігри, забезпечена створенням класу моделі Gameshop, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут shopname визначається полем CharField, що є назвою магазину, що продає настільні ігри через програму;
- атрибут basephoto визначається полем CharField, що є фотографією з зображенням магазину, що продає настільні ігри через програму;
- атрибут postaddress визначається полем CharField, що є поштовою адресою, за якою знаходиться магазин, що продає настільні ігри через програму;
- атрибут description визначається полем CharField, що є довідковою інформацією про магазин, що продає настільні ігри через програму;
- атрибут city визначається полем CharField, що є містом, де знаходиться магазин, що продає настільні ігри через програму;
- атрибут workers визначається полем ManyToManyField, що є переліком користувачів-працівників, для яких визначено доступ до цього магазину, відповідно визначає відношення багато до багатьох до моделі Gamer.

Модель настільних ігор у магазині забезпечена створенням класу моделі Boardgameshop, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут gameshop визначається полем ForeignKey, що є магазином, що продає настільні ігри через програму і в якому є така настільна гра в наявності або вона може бути в певний період часу, є зовнішнім ключем, що веде до моделі Gameshop;
- атрибут boardgame визначається полем ForeignKey, що є настільною грою, яка є в цьому магазині або може бути в ньому в наявності на певний час, є зовнішнім ключем, що веде до моделі Boardgame;

- атрибут `quantity` визначається полем `SmallIntegerField`, що є кількістю екземплярів гри, яка є доступною для продажу через цей магазин на даний момент.

Модель додаткових учасників розробки настільних ігор забезпечена створенням класу моделі `BoardgameArtist`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `boardgame` визначається полем `ForeignKey`, що є настільною грою, в розробці якої взяв участь цей творець, є зовнішнім ключем, що веде до моделі `Boardgame`;

- атрибут `artist` визначається полем `ForeignKey`, що є творцем, який взяв участь в розробці цієї настільної гри, є зовнішнім ключем, що веде до моделі `Artist`;

- атрибут `details` визначається полем `CharField`, що є деталями щодо участі цього творця в розробці цієї настільної гри.

Модель можливої категорії настільної гри забезпечена створенням класу моделі `BoardgameCategory`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `categoryname` визначається полем `CharField`, що є назвою категорії настільної гри;

- атрибут `description` визначається полем `CharField`, що є описом особливостей, які дозволяють віднести настільну гру до цієї категорії.

Модель загальних механізмів настільних ігор забезпечена створенням класу моделі `Gamemechanism`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `mechname` визначається полем `CharField`, що є назвою цього механізму;

- атрибут `description` визначається полем `CharField`, що є описом принципів, на яких цей механізм базується в настільних іграх зазвичай.

Модель механізмів, використаних у конкретній настільній грі, забезпечена створенням класу моделі Boardgamemechanism, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут boardgame визначається полем ForeignKey, що є настільною грою, в якій використовується цей ігровий механізм, є зовнішнім ключем, що веде до моделі Boardgame;
- атрибут gamemechanism визначається полем ForeignKey, що є ігровим механізмом, який використовується в цій настільній грі, є зовнішнім ключем, що веде до моделі Boardgamemechanism;
- атрибут details визначається полем CharField, що є описом деталей застосування саме цього ігрового механізму в цій настільній грі.

Модель користувача програми побудована на основі базового класу Django моделі User та в підсумку забезпечена створенням класу моделі Gamer, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут username визначається полем CharField, що є унікальним текстовим ідентифікатором облікового запису (логіном);
- атрибут password визначається полем CharField, що є паролем, який використовується для забезпечення авторизації облікового запису в програмі;
- атрибут displayedname визначається полем CharField, що є іменем, яке показується в програмі щодо цього користувача іншим користувачам;
- атрибут description визначається полем CharField, що є описом себе, своїх інтересів щодо настільних ігор та загалом користувачем;
- атрибут basephoto визначається полем ImageField, що є фотографією або зображенням, яким візуалізує себе користувач;
- атрибут shopworker визначається полем BooleanField, що визначає, чи це звичайний користувач (гравець), чи це працівник магазину;
- атрибут ownngames визначається полем ManyToManyField, що є переліком настільних ігор, якими володіє користувач-гравець, відповідно визначає відношення багато до багатьох до моделі Boardgame;

- атрибут `subscribedgamers` визначається полем `ManyToManyField`, що є переліком гравців, на які підписався цей користувач-гравець, відповідно визначає відношення багато до багатьох до моделі `Gamer`;

- атрибут `subscribedartists` визначається полем `ManyToManyField`, що є переліком творців, на які підписався цей користувач-гравець, відповідно визначає відношення багато до багатьох до моделі `Artist`;

- атрибут `subscribedpublishers` визначається полем `ManyToManyField`, що є переліком виробників настільних ігор, на які підписався цей користувач-гравець, відповідно визначає відношення багато до багатьох до моделі `Publisher`.

Модель відгуків про настільні ігри забезпечена створенням класу моделі `Boardgamerating`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `boardgame` визначається полем `ForeignKey`, що є настільною грою, стосовно якої створено цей відгук, є зовнішнім ключем, що веде до моделі `Boardgame`;

- атрибут `gamer` визначається полем `ForeignKey`, що є користувачем, який створив цей відгук про настільну гру, є зовнішнім ключем, що веде до моделі `Gamer`;

- атрибут `ratingdate` визначається полем `DateTimeField`, що є часом, коли було створено відгук про настільну гру;

- атрибут `rating` визначається полем `SmallIntegerField`, що є значенням, на яке було загалом оцінено настільну гру;

- атрибут `ratingtext` визначається полем `CharField`, що є текстовою частиною відгуку про настільну гру;

- атрибут `owner` визначається полем `BooleanField`, що визначає, чи автор відгуку про настільну гру є власником цієї гри, чи ні;

- атрибут `played` визначається полем `BooleanField`, що визначає, чи автор відгуку про настільну гру грав у неї особисто, чи ні;

- атрибут `want` визначається полем `BooleanField`, що визначає, чи автор відгуку про настільну гру має активне бажання придбати цю гру (якщо він її ще не має), чи ні;

- атрибут `recommendedtime` визначається полем `TimeField`, що є часом, який проводити за грою в цю настільну гру вважає оптимальним цей автор відгуку про настільну гру;

- атрибут `recommendedplayers` визначається полем `SmallIntegerField`, що є кількістю гравців, участь яких у цій настільній грі вважає оптимальним цей автор відгуку про настільну гру.

Модель оцінювань за окремими критеріями для відгуків про настільні ігри забезпечена створенням класу моделі `Boardgameratingcritetion`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `boardgamerating` визначається полем `ForeignKey`, що є відгуком про настільну гру, якого стосується це оцінювання, є зовнішнім ключем, що веде до моделі `Boardgamerating`;

- атрибут `criterion` визначається полем `SmallIntegerField`, що є критерієм, за яким було виконано оцінювання настільної гри;

- атрибут `criterionvalue` визначається полем `SmallIntegerField`, що є значенням оцінки, яке було виставлено цим користувачем за цим критерієм стосовно настільної гри.

Модель замовлення на придбання настільних ігор забезпечена створенням класу моделі `Gamerorder`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `gamer` визначається полем `ForeignKey`, що є користувачем, який створив це замовлення на придбання настільних ігор, є зовнішнім ключем, що веде до моделі `Gamer`;

- атрибут `orderdate` визначається полем `DateTimeFieldForeignKey`, що є часом, коли було збережено замовлення на придбання настільних ігор початково користувачем;

- атрибут `state` визначається полем `SmallIntegerField`, що є станом, в якому знаходиться замовлення на придбання настільних ігор;
- атрибут `price` визначається полем `FloatField`, що є сумарною вартістю замовлення на придбання настільних ігор;
- атрибут `gameshop` визначається полем `ForeignKey`, що є магазином, в якому продаються настільні ігри, щодо якого і було сформовано замовлення, є зовнішнім ключем, що веде до моделі `Gameshop`;
- атрибут `deliveryaddress` визначається полем `CharField`, що є адресою доставки настільних ігор, якщо замовлення не буде забрано клієнтом самостійно.

Модель вмісту замовлення на придбання настільних ігор забезпечена створенням класу моделі `Gamerordergame`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `boardgame` визначається полем `ForeignKey`, що є настільною грою, яка входить до замовлення, є зовнішнім ключем, що веде до моделі `Boardgame`;
- атрибут `gameorder` визначається полем `ForeignKey`, що є замовленням на придбання настільних ігор, в яке входить ця настільна гра, є зовнішнім ключем, що веде до моделі `Gamerorder`;
- атрибут `number` визначається полем `SmallIntegerField`, що є кількістю екземплярів цієї настільної гри, яка має бути доставлена або віддана за цим замовленням;
- атрибут `price` визначається полем `FloatField`, що є вартістю одного екземпляра цієї настільної гри.

Модель персонального переліку настільних ігор користувача забезпечена створенням класу моделі `Gamerlist`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `gamer` визначається полем `ForeignKey`, що є користувач-гравець, який створив і якому належить цей перелік настільних ігор, є зовнішнім ключем, що веде до моделі `Gamer`;

- атрибут `listname` визначається полем `CharField`, що є назвою персонального переліку настільних ігор користувача;
- атрибут `description` визначається полем `CharField`, що є описом персонального переліку настільних ігор користувача;
- атрибут `games` визначається полем `ManyToManyField`, що є переліком настільних ігор, які включені до цього персонального переліку, тобто фактично є вмістом цього персонального переліку настільних ігор, відповідно визначає відношення багато до багатьох до моделі `Boardgame`.

Модель реєстрації сеансів, проведених за настільною грою, забезпечена створенням класу моделі `BoardgameLog`, яка у своїй структурі була визначена через такі атрибути класу:

- атрибут `gamer` визначається полем `ForeignKey`, що є користувачем-гравцем, який грав у цю настільну гру, є зовнішнім ключем, що веде до моделі `Gamer`;
- атрибут `boardgame` визначається полем `ForeignKey`, що є настільною грою, в яку грав цей користувач у зазначеному випадку, є зовнішнім ключем, що веде до моделі `Boardgame`;
- атрибут `playdate` визначається полем `DateField`, що є датою, коли цей користувач грав у цю настільну гру;
- атрибут `gametimes` визначається полем `SmallIntegerField`, що є кількістю разів, яку в цю дату було зіграно в цю настільну гру за участю цього користувача;
- атрибут `address` визначається полем `CharField`, що є адресою, за якою відбулась ця гра, якщо її вказує користувач;
- атрибут `playtime` визначається полем `TimeField`, що є ігровим часом;
- атрибут `comment` визначається полем `CharField`, що є коментарем щодо цього випадку від самого користувача, який у випадку надання доступу до цієї інформації іншим користувачам, буде їм видним;
- атрибут `personalcomment` визначається полем `CharField`, що є прихованим коментарем щодо цього випадку від самого користувача, який у

випадку надання доступу до цієї інформації іншим користувачам, буде для них закритим і не відображатися;

– атрибут `gamers` визначається полем `ManyToManyField`, що є переліком користувачів-гравців, які брали участь у цьому сеансі настільної гри, відповідно визначає відношення багато до багатьох до моделі `Gamer`.

Перегляд власних оформлених замовлень настільних ігор забезпечено функцією `owngameordersview`, при цьому стан потрібних замовлень можна задати параметром `orderstate`, а якщо значення дорівнює `-1`, то відображаються у шаблон сторінки `gameorders.html` всі замовлення.

Перегляд сторінки з інформацією про настільну гру забезпечено функцією `gameinfoview`, а номер настільної гри визначається параметром `boardgame`. У шаблон вебсторінки `game.html` повертаються дані самої настільної гри за ключем `game`, 5 найновіших відгуків з повними даними цих відгуків за ключем `ratings`, 5 останніх реєстрацій сеансів за цією грою, включаючи власні сеанси та спільні сеанси, за ключем `logs`, дані про наявність цієї гри в магазинах за ключем `shops`. За кожними об'єктами тут, як і в інших функціях, також повертається персональна інформація. Наприклад, якщо подаються дані гри, то надаються позначки щодо того, чи було створено відгук самим авторизованим користувачем на цю настільну гру, чи він володіє цією настільною грою, чи він вніс її у свої персональні переліки і відповідно які саме. За творцями, наприклад, повертаються дані про те, чи підписаний цей користувач на цього творця.

Перегляд власних сеансів настільних ігор забезпечено функцією `owngamerlogsview`. Найновіші вносяться першими, всі дані передаються в шаблон `ownlogs.html`.

Перегляд спільних сеансів настільних ігор з іншими гравцями забезпечено функцією `sharedgamerlogsview`. Визначаються всі сеанси, в яких було відзначено в якості учасників авторизованого користувача. Всі дані передаються в шаблон `sharedlog.html`.

Перегляд даних і настільних ігор виробника забезпечено функцією `publisherinfoview`, де параметр `publisherinfo` задає унікальний номер виробника.

Перегляд даних і настільних ігор творця забезпечено функцією `artistinfoview`, де параметр `artistinfo` задає унікальний номер творця.

Перегляд відгуків гравця забезпечено функцією `gamerratingsview`, де параметр `ratinggamer` задає логін гравця.

Перегляд гравців, на яких оформлено підписку забезпечено функцією `gamergamersview`. Всі дані передаються в шаблон `subgamers.html`.

Перегляд настільних ігор творців, на яких оформлено підписку, забезпечено функцією `subartistgamesview`, де параметр `order_principle` задає порядок сортування: 0 – за датою видання від найновіших, 1 – за рейтингом від найбільшого, 2 – за рейтингом візуальної краси, 3 – за рейтингом складності, 4 – за рейтингом взаємодії, 5 – за рейтингом якості компонентів, 6 – за вартістю від найбільшої, 7 – за вартістю від найменшої. Вибираються тільки ті настільні ігри, в яких творець був дизайнером, редактором, розробником або виконував іншу роль, а творцем є будь-хто з тих творців, на кого підписався авторизований користувач.

Перегляд настільних ігор виробників, на яких оформлено підписку, забезпечено функцією `subpublishergamesview`. Параметр `order_principle` також задає порядок сортування.

Перегляд відгуків гравців, на яких оформлено підписку забезпечено функцією `subgameratingsview`. Дані передаються в шаблон вебсторінки `subgameratings.html`.

Перегляд відгуків про настільну гру забезпечено функцією `boardgameratingsview`, де параметр `boardgame` задає номер настільної гри, відгуки за якою потрібно показати.

Перегляд настільних ігор у власності користувача забезпечено функцією `ownboardgamesview`, параметр `order_principle` якої визначає напрямок сортування.

Перегляд настільних ігор з персонального переліку забезпечено функцією `listboardgamesview`, де параметр `gamelist` визначає номер переліку, за яким потрібно показати настільні ігри, а параметр `order_principle` визначає напрямок сортування.

Перегляд рейтингу настільних ігор за заданим номером ігрового механізму `gamemechanism` забезпечено функцією `mechanismboardgamesratingview`. Параметр `order_principle` визначає принцип складання рейтингу найкращих таких настільних ігор: 0 – загальний, 1 – візуальна краса, 2 – складність, 3 – взаємодія з іншими гравцями, 4 – якість виконання компонентів гри. Відповідно перегляд рейтингу настільних ігор за заданим номером виробника `publisher` забезпечено функцією `publisherboardgamesratingview`. Параметр `order_principle` аналогічно виконує сортування. Перегляд рейтингу настільних ігор за заданим номером категорії `gamecategory` забезпечено функцією `categoryboardgamesratingview`. Перегляд загального рейтингу настільних ігор забезпечено функцією `allboardgamesratingview`. Порядок сортування визначається аналогічно.

Перегляд виробників настільних ігор, на яких оформлено підписку, забезпечено функцією `gamerpublishersview`. Дані повертаються в шаблон сторінки `subpublishers.html`.

Перегляд творців, на яких оформлено підписку, забезпечено функцією `subartistsview`. Дані повертаються в шаблон сторінки `subartists.html`.

Перегляд усіх виробників настільних ігор забезпечено функцією `allpublishersview`. Дані повертаються в шаблон сторінки `publishers.html`.

Перегляд усіх творців, долучених до створення настільних ігор, забезпечено функцією `allartistsview`. Дані повертаються в шаблон сторінки `artists.html`.

Пошук настільних ігор забезпечено функцією `boardgamessearchview`, для якої через адресу звернення задаються такі параметри:

- мінімальний потрібний час гри `mintime` у цю настільну гру;
- максимальний потрібний час гри `maxtime` у цю настільну гру;

- мінімальний необхідний вік `minage` для гри в цю настільну гру;
- мінімальна кількість гравців `minpl`, що можуть грати в цю настільну гру;
- максимальна кількість гравців `maxpl`, що можуть грати в цю настільну гру;
- мінімальний потрібний загальний рейтинг настільної гри `minrat`;
- мінімальний потрібний рейтинг настільної гри `minart` за візуальною красою;
- мінімальний потрібний рейтинг настільної гри `mincompl` за складністю;
- мінімальний потрібний рейтинг настільної гри `minq` за якістю компонентів;
- мінімальний потрібний рейтинг настільної гри `mininter` за взаємодією з іншими гравцями;
- мінімальна вартість настільної гри `minprice`;
- максимальна вартість настільної гри `maxprice`;
- мінімальний рік випуску настільної гри `minyear`;
- максимальний рік випуску настільної гри `maxyear`;
- назва настільної гри або фраза з назви `name`;
- номер видавця настільної гри `publ` (цей і будь-які інші наступні параметри можуть бути пропущені, тоді настільні ігри за цим параметром не відфільтровуються);
- номер категорії настільної гри `categ`;
- номер ігрового механізму, що має бути задіяний у настільній грі, `mechan`;
- номер запису про творця, який має бути долучений до створення цієї настільної гри, як редактор, дизайнер, розробник або в іншій ролі, `art`;
- напрямок сортування `ord`.

Перегляд інформації про гравця забезпечено функцією `gamerinfoview`, в якій параметром `ratinggamer` задається логін гравця як користувача, а

повертаються в шаблон вебсторінки `gamerinfo.html` дані самого гравця за ключем `gamer`, останні створені ним відгуки за ключем `ratings` та останні його сеанси за настільними іграми, які були спільними з авторизованим користувачем, за ключем `logs`.

Перегляд власних персональних переліків настільних ігор забезпечено функцією `owngamelistsview`. Дані повертаються в шаблон сторінки `ownlists.html`.

Оформлення підписки на гравця забезпечено функцією `subscribegamerview`, де параметр `gamer` визначає логін цього гравця.

Оформлення підписки на виробника настільних ігор забезпечено функцією `subscribepublisherview`, де параметр `publisher` визначає номер виробника.

Оформлення підписки на творця забезпечено функцією `subscribeartistview`, де параметр `artist` визначає номер творця.

Скасування підписки на гравця забезпечено функцією `deletegamersubscribe`, де параметр `gamer` визначає логін гравця, на якого було виконано підписку.

Скасування підписки на виробника настільних ігор забезпечено функцією `deletepublishersubscribe`, де параметр `publisher` визначає номер виробника настільних ігор, на якого було виконано підписку.

Скасування підписки на творця забезпечено функцією `deleteartistsubscribe`, де параметр `artist` визначає номер запису творця, на якого було виконано підписку.

Визначення власної настільної гри забезпечено функцією `createowngameview`, де параметр `game` визначає номер настільної гри.

Скасування визначення власної настільної гри забезпечено функцією `deleteowngameview`, де параметр `boardgame` визначає номер настільної гри.

Замовлення настільних ігор для придбання забезпечено функцією `creategamerorderview`. Скасування оформленого замовлення настільних ігор забезпечено функцією `cancelgamerorder`, де параметр `gamerorder` визначає

номер замовлення, а скасування може бути виконано тільки працівником магазину з продажу настільних ігор, адміністратором або користувачем-гравцем, який створив це замовлення.

Визначення відгуку про настільну гру забезпечено функцією `createboardgameratingview`.

Реєстрація проведеного сеансу настільної гри забезпечено функцією `createboardgamelog`. Оновлення даних сеансу настільної гри забезпечено функцією `editboardgamelog`, де параметр `gamelog` визначає номер самого сеансу.

Створення персонального переліку настільних ігор забезпечено функцією `creategamelistview`.

Додавання настільної гри до персонального переліку забезпечено функцією, де параметр `gamelist` визначає номер переліку настільних ігор, а `game` – номер самої настільної гри.

Вилучення настільної гри з персонального переліку забезпечено функцією `deletelistgameview`, де параметр `gamelist` визначає номер переліку настільних ігор, а `game` – номер самої настільної гри.

Реалізацію самого методу пошуку настільних ігор виконано через використання класу `KMeans` з підпаketу `cluster` пакету `scikit-learn`.

4.3 Висновки за розділом 4

Усі функції програми було забезпечено через реалізацію Django-застосунку, в якому розроблено ряд функцій представлень, які вибирають відповідні дані з моделей та надають їх у шаблони вебсторінок, ряд класів, які реалізують самі моделі за структурою БД.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програми

Програма призначена для надання інформації про настільні ігри, забезпечення можливості замовити настільні ігри в магазинах мережі, пошуку нових настільних ігор, в тому числі через надання пріоритетних щодо вибору настільних ігор, реєстрації власних настільних ігор у програмі, надання відгуків про власний досвід, ознайомлення з інформацією щодо досвіду інших користувачів.

5.2 Виконання програми

Виконання програми має забезпечуватися на серверній стороні запуском серверу та відповідно запуском самого Django-застосунку, після чого подальша взаємодія відбувається зі сторони клієнта так само як і з будь-яким вебзастосунком. У випадку введення адреси вебсервера без уточнення адрес окремими сторінками відкривається користувачу одразу сторінка пошуку настільних ігор. Ця сторінка містить зліва форму пошуку, а у іншій частині результати пошуку. Форма пошуку включає як параметри:

- мінімальний потрібний час гри у цю настільну гру;
- максимальний потрібний час гри у цю настільну гру;
- мінімальний необхідний вік для гри в цю настільну гру;
- мінімальна кількість гравців, що можуть грати в цю настільну гру;
- максимальна кількість гравців, що можуть грати в цю настільну гру;
- мінімальний потрібний загальний рейтинг настільної гри;
- мінімальний потрібний рейтинг настільної гри за візуальною красою;
- мінімальний потрібний рейтинг настільної гри за складністю;

- мінімальний потрібний рейтинг настільної гри за якістю компонентів;
- мінімальний потрібний рейтинг настільної гри за взаємодією з іншими гравцями;
- мінімальна вартість настільної гри;
- максимальна вартість настільної гри;
- мінімальний рік випуску настільної гри;
- максимальний рік випуску настільної гри;
- назва настільної гри або фраза з назви;
- виробник настільної гри;
- категорія настільної гри;
- ігровий механізм, що має бути задіяний у настільній грі;
- творець, який має бути долучений до створення цієї настільної гри, як редактор, дизайнер, розробник або в іншій ролі;
- напрямок сортування.

Від результатів пошуку можна перейти одразу до сторінки самої настільної гри (рис. 5.1). Ця сторінка містить інформацію про гру, включаючи назву, короткий опис, вартість у магазинах, середню оцінку, надає можливість одразу замовити настільну гру, додаючи її в кошик, що відображається згори у правій стороні (доступна кількість залежить від обраного з переліку магазину), а при натисканні на нього можна відкрити підготовлене замовлення, звідки його можна оформити. Нижче знаходиться кнопка для створення відгуку, поруч з якою відображається оцінка з відгуку авторизованого користувача, якщо він вже створював відгук, кнопка для додавання настільної гри у власний перелік, натискаючи на яку, можна задати з випадającego меню потрібний перелік, кнопка для визначення наявності у себе відповідної настільної гри, кнопка для створення логу (сеансу цієї настільної гри). Нижче розташовано інформацію про гру, яку не вказано у верхній частині, що міститься в БД, також останні відгуки з

можливістю перейти до сторінки перегляду всіх відгуків, записи логів, до яких має стосунок користувач.

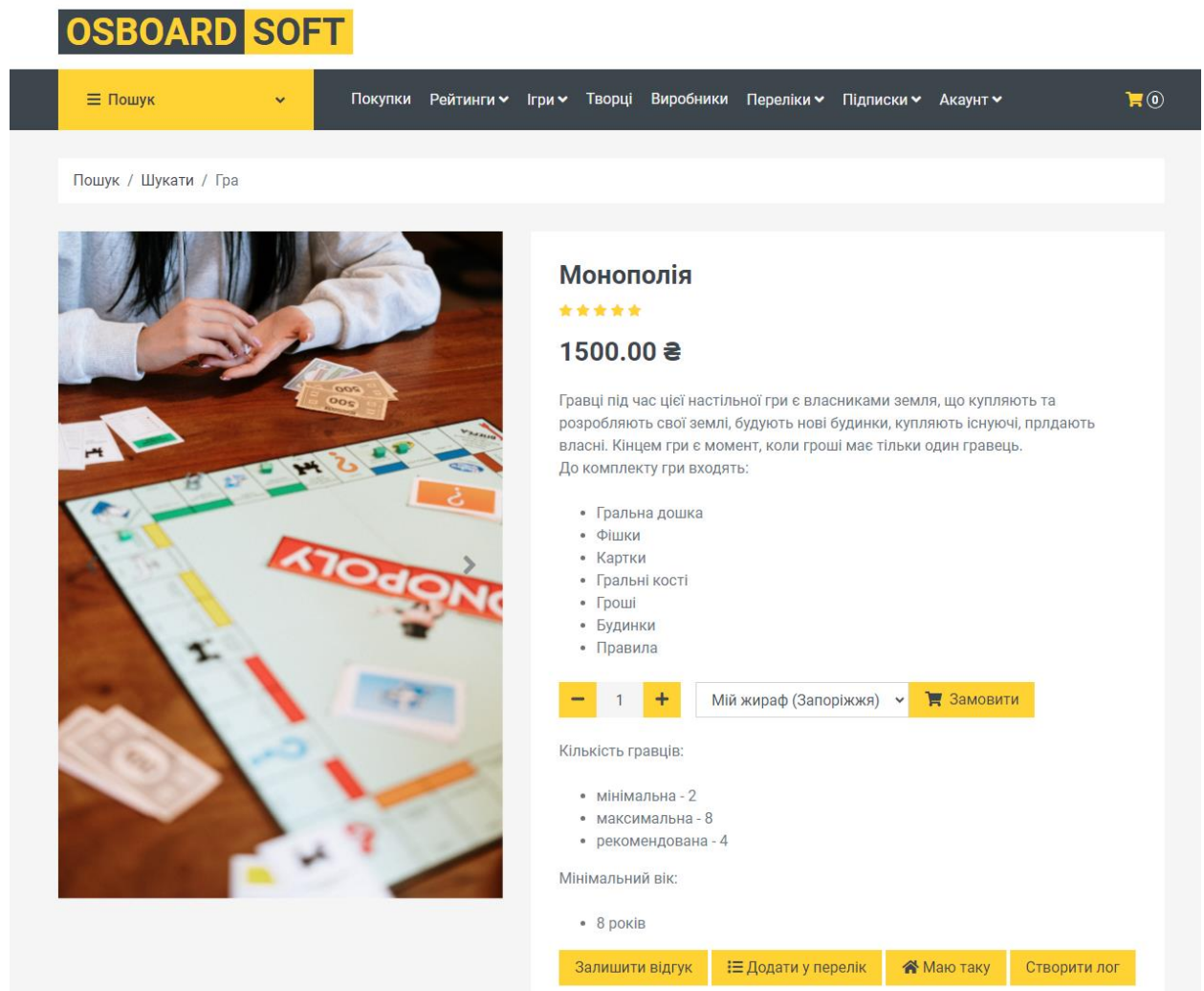


Рисунок 5.1 – Сторінка настільної гри в програмі

Для того щоб перейти до наявних у користувача переліків, потрібно відкрити розділ Переліки, звідки можна створити перелік, переглянути створені персональні переліки настільних ігор, переходячи до вмісту конкретного переліку. Відповідно підписки, які можна в програмі оформити на творців настільних ігор, на їх виробників (інформація про виробника на сторінці настільної гри розташована нижче окремим блоком, адже містить поле для оформлення підписки) розташовані у розділі меню Підписки.

Окрім сторінки пошуку настільних ігор є дуже подібна за інтерфейсом сторінка пріоритетного пошуку, яка працює на основі методу пошуку настільних ігор, що було запропоновано. Для цього користувач має бути спершу авторизований та мати щонайменше 1 позитивний відгук про настільну гру. Після цього він може виконувати пошук настільних ігор, але вже не за всіма іграми, а тими, які відносяться до того ж кластера, що і основна кількість тих настільних ігор, які сподобались користувачу. Ця сторінка також доступна з розділу Пошук у меню (рис. 5.2).

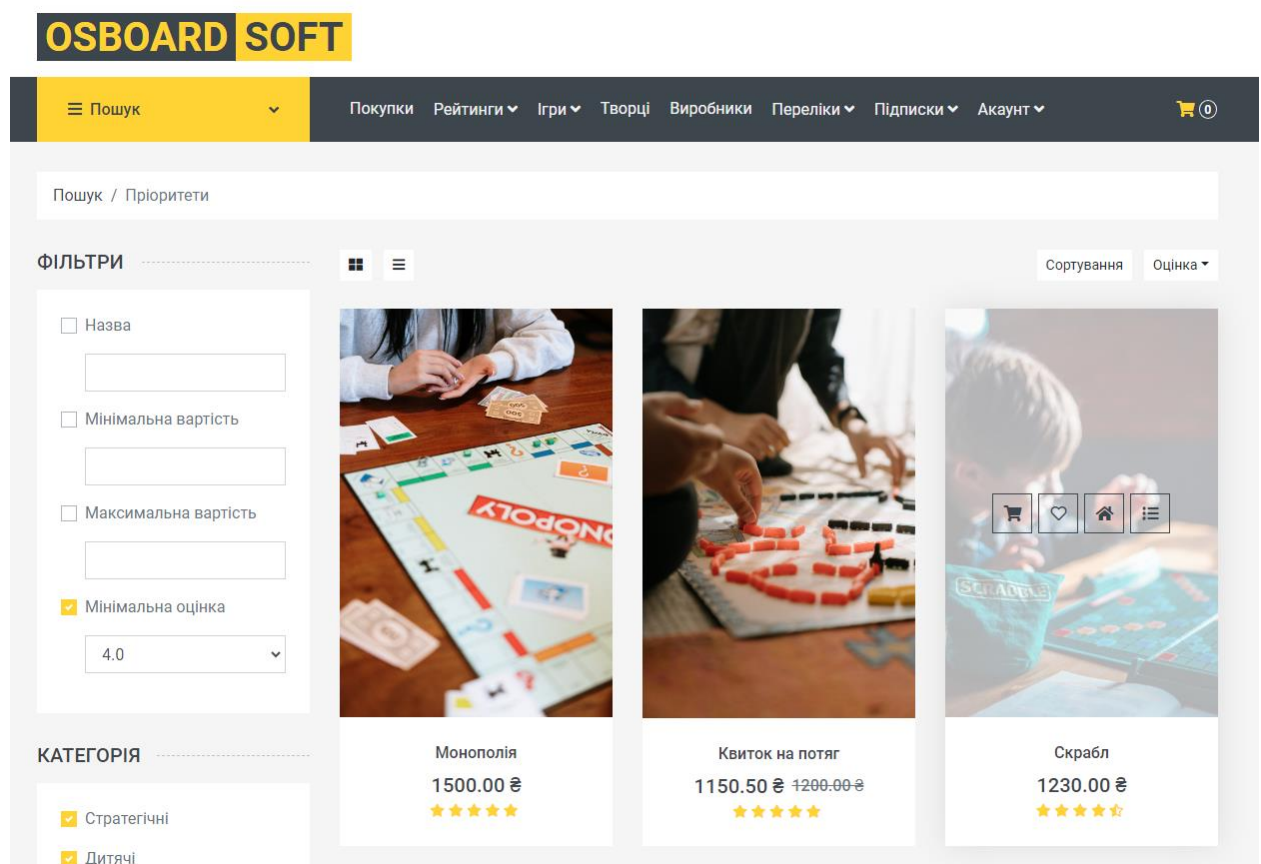


Рисунок 5.2 – Пошук за пріоритетними для користувача настільними іграми

Отримані пріоритетні настільні ігри далі можна сортувати або фільтрувати необхідним чином. При наведенні на картку настільної гри в результатах можна використати додаткове меню, що послідовно дозволяє додати настільну гру до замовлення, перейти до створення відгуку про неї,

внести дані про те, що ця гра наявна у користувача, додати гру до одного з персональних переліків настільних ігор користувача.

Меню також містить розділи для переходу до власних вже створених замовлень через розділи Покупки, переходу до переліку виробників настільних ігор з розділу Виробники, творців з розділу Творці, рейтингів настільних ігор з розділу Рейтинги. Розділ Ігри містить підрозділи для перегляду власних ігор, перегляду власних сеансів та сеансів ігор, в яких брав участь користувач.

Для працівника магазину доступний додатковий розділ Магазин, який для адміністратора складається з додаткового підрозділу, що дозволяє переглядати всі магазини і додавати нові (рис. 5.3). Працівник магазину працює з даними тільки магазину, до якого має доступ.

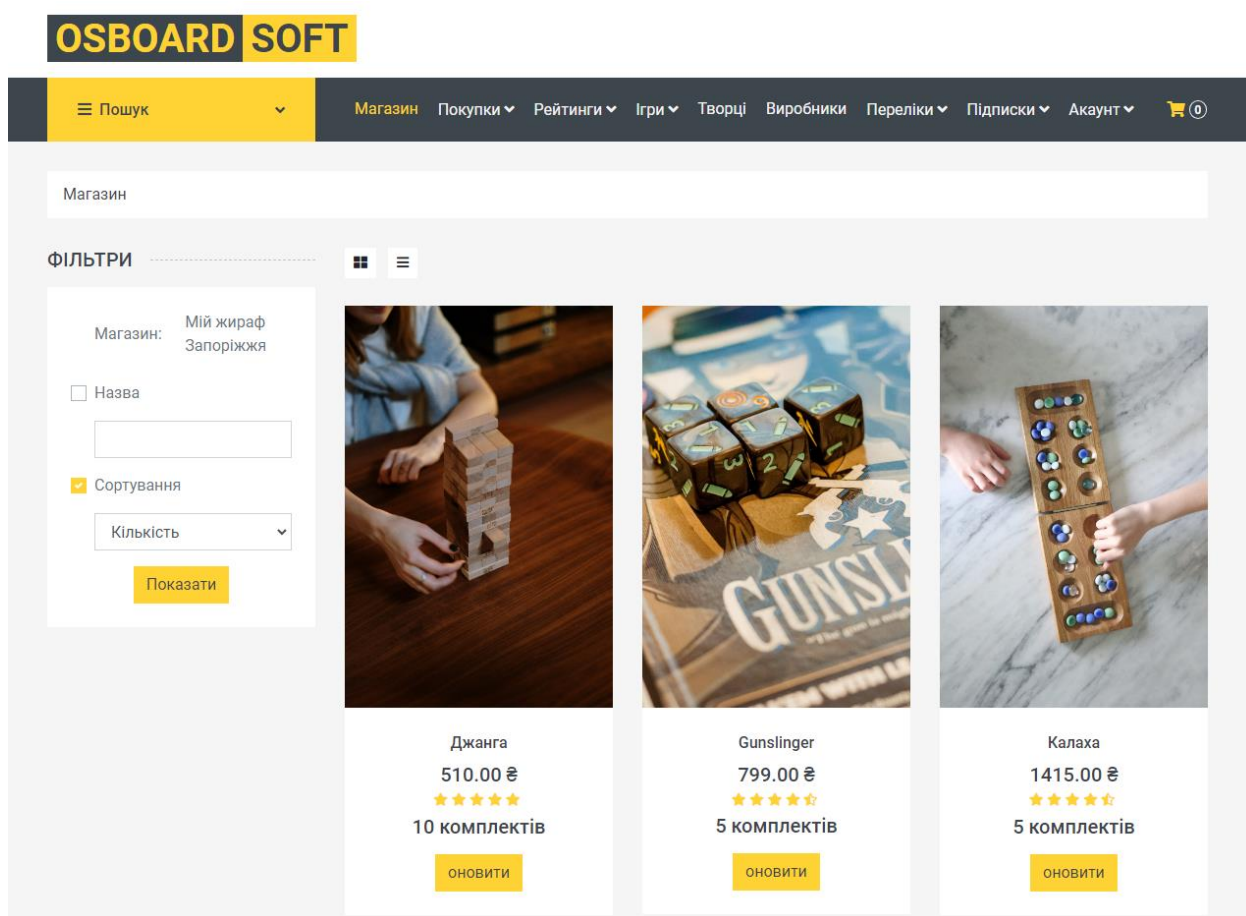


Рисунок 5.3 – Відображення кількості настільних ігор за конкретними позиціями у магазині

За магазином працівнику відображаються наявні в ньому настільні ігри, які можна відсортувати або виконати серед них пошук за назвою. З кожною настільною грою можна натиснути на кнопку Оновити та внести нову кількість екземплярів гри, що доступна в цьому магазині.

5.3 Експериментальне дослідження

Експериментальне дослідження було направлено на дослідження результатів роботи методу пошуку настільних ігор, який було запропоновано в розділі 2. Оскільки оцінити результати пошуку можна тільки індивідуально, то було виконано дослідження, яке дозволяє перевірити якість розбиття даних вибірки на кластери. Для цього застосована вибірка даних [11] з відкритого доступу. З цієї вибірки даних були відібрані тільки ті настільні ігри, які мають вказані значення за кожним з параметрів, які було виділено у розділі 2, окрім того було відібрано тільки ті настільні ігри, які мають середній бал вище 8 (у вибірці оцінювання виконувалось за шкалою з максимумом на рівні 10).

Після того, як вибірка описаним чином була сформульована, вона була розділена на певну кількість кластерів на основі ознак, які визначені методом пошуку настільних ігор. Для перевірки того, як впливає кількість кластерів на результати, було проведено створення кластерів на основі визначення кількості кластерів, рівною 5, 10, 20, 50. Також було визначено і більшу кількість, але результати вже не приводяться в даному розділі через велику кількість.

Для кожного кластера було обчислено відстань між всіма парами екземплярів у цьому кластері та окремо відстань між всіма парами екземплярів цього кластера та всіх інших кластерів. Значення було отримано за всіма ознаками на основі обчислення суми квадратів різниці. Тоді таке сумарне значення було додано за всіма парами екземплярів, а отримане значення розділене на кількість таких пар.

Серед всіх значень стосовно кожного кластера було виділено таким чином отримане значення для екземплярів всередині кластера, а також мінімальне та максимальне значення серед попарних обчислень між екземплярами цього кластера та всіх інших. Окремо було визначено всі ознаки, в тому числі категоріальні, та всі ознаки без категоріальних. За наявності категоріальних ознак різниця доіврнювала 1, якщо значення не співпадали, 0 – у протилежному випадку. Для механізмів (2 ознаки) два співпадіння давали 0, 1 – 0,5 (далі значення піднесене у квадрат), жодного – 1.

Випадки для 5 кластерів (табл. 5.1) і 10 кластерів (табл. 5.2) вказують, що екземпляри всередині кластерів розташовані ближче один до одного ніж до екземплярів інших кластерів.

Таблиця 5.1 – Результати кластеризації вибірки даних за визначення 5 кластерів з врахуванням категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,065886569	0,096531324	0,132287873
1	0,048799196	0,087153027	0,132287873
2	0,075442241	0,076436714	0,109538686
3	0,067813131	0,074353886	0,096531324
4	0,067974446	0,074353886	0,10294081

Таблиця 5.2 – Результати кластеризації вибірки даних за визначення 10 кластерів з врахуванням категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,071920938	0,07484534	0,115361577
1	0,050638066	0,066987337	0,114421428
2	0,059104643	0,068812458	0,11177276
3	0,036908444	0,052463273	0,137773509
4	0,052551324	0,052563273	0,134594513
5	0,060222628	0,066987337	0,105945315
6	0,051162263	0,052107181	0,112372777
7	0,065179404	0,085678915	0,137773509
8	0,029300184	0,052107181	0,114144527
9	0,067382675	0,070016472	0,110308165

Також було перевірено результати кластеризації вибірки даних за визначення 20 кластерів з врахуванням категоріальних значень (табл. 5.3) Вони також вказали на те, що екземпляри всередині кластера розташовані ближче один до одного ніж екземпляри різних кластерів відносно всіх 20 сформованих кластерів.

Таблиця 5.3 – Результати кластеризації вибірки даних за визначення 20 кластерів з врахуванням категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,052648993	0,052661214	0,138488246
1	0,05590697	0,063471364	0,161322581
2	0,035970093	0,041598166	0,155540739
3	0,027885218	0,03670077	0,148527171
4	0,061114668	0,066930872	0,17965986
5	0,054782486	0,065479961	0,172585875
6	0,052878246	0,0633039	0,167982377
7	0,052517715	0,052572971	0,138762187
8	0,030919672	0,041598166	0,164705914
9	0,028917691	0,03670077	0,140442148
10	0,047811976	0,079987831	0,220183786
11	0,044822804	0,050952855	0,135671267
12	0,029287606	0,0633039	0,159793861
13	0,064203555	0,073767539	0,16901711
14	0,056072974	0,064447212	0,166952642
15	0,001391224	0,118060623	0,220183786
16	0,072067751	0,079028766	0,160012289
17	0,045352601	0,065562591	0,160072811
18	0,044844145	0,046540288	0,154937034
19	0,052921413	0,05415841	0,136621764

Випадок для 50 кластерів представлений скорочено (табл. 5.4).

Таблиця 5.4 – Результати кластеризації вибірки даних за визначення 50 кластерів з врахуванням категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,058523089	0,068611472	0,167349228
1	0,035584778	0,041119445	0,16481309
6	0,004177684	0,055726535	0,244156612
7	0,060727105	0,069044881	0,166798676
8	0,03239263	0,032577559	0,161328051
9	0,040931323	0,05034206	0,174909468
10	0,040018165	0,046197071	0,165103282
11	0,055761866	0,080494944	0,18142924
15	0,04143402	0,042499999	0,165414274
16	0,044402791	0,04822399	0,16662575
17	0,031582123	0,03202623	0,181823962
18	0,026321812	0,034618923	0,157798456
32	0,026655394	0,051087978	0,164603714
33	0,021590766	0,034618923	0,161789266
34	0,042168848	0,051595479	0,1767716
35	0,022290927	0,032577559	0,165655216
36	0,030348247	0,033358977	0,166485013
37	0,008618737	0,021319715	0,169759902
48	0,047891966	0,068615837	0,162340392
49	0,039303626	0,053961708	0,179644904

Після цього було відкинуто при обчисленні відстані категоріальні значення і залишено тільки ті, які були початково цифровими. У такому випадку для 5 кластерів результати також підтвердили виявлену раніше логіку (табл. 5.5).

Таблиця 5.5 – Результати кластеризації вибірки даних за визначення 5 кластерів без врахування категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,018690493	0,018765676	0,045121679
1	0,007231448	0,013765676	0,039866154
2	0,012414237	0,017877944	0,049347963
3	0,010619221	0,039866154	0,07163846
4	0,015848758	0,03621042	0,07163846

Для 10 кластерів перевірка не виявила відхилень (табл. 5.6), максимальне значення є значно віддаленішим, однак і мінімальне також перевищує середнє значення всередині кластера, хоча для деяких кластерів ці значення і можуть бути достатньо близькими між собою. Проте і у випадку використання категоріальних значень для обчислення відстані, і без використання такої різниці все ж є, а відповідно розподіл даних вибірки на кластери можна вважати достатньо ефективним. Його можна використовувати для роботи з даними настільних ігор, визначаючи пріоритетні для користувачів.

Таблиця 5.6 – Результати кластеризації вибірки даних за визначення 10 кластерів без врахування категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,006536123	0,007463535	0,045658654
1	0,00681873	0,007524489	0,046347439
2	0,007676481	0,013674826	0,077992307
3	0,00693751	0,007524489	0,043693815
4	0,009258336	0,019114841	0,060039888
5	0,01530865	0,037742552	0,077992307
6	0,006730017	0,007463535	0,044670125
7	0,004772316	0,013674826	0,075927235
8	0,01537305	0,016045623	0,05009809
9	0,019589581	0,02419853	0,050483707

Випадок для 20 кластерів також зберіг тенденцію, виявлену в ситуації за використання категоріальних значень і для ситуації без використання категоріальних значень (табл. 5.7). Усе це дозволяє стверджувати, що впровадження методу пошуку настільних ігор, який було запропоновано в роботі й який було інтегровано в програму, для практичних ситуацій є можливим.

Таблиця 5.7 – Результати кластеризації вибірки даних за визначення 20 кластерів без врахування категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,003259386	0,009409526	0,134769071
1	0,004660229	0,006096029	0,109407315
2	0,00402105	0,00542262	0,10941134
3	0,010632237	0,007759041	0,10753857
4	0,011665963	0,026616077	0,177654697
5	0,007876332	0,011990032	0,121456789
6	0,007527908	0,012742581	0,118605146
7	0,010114956	0,009456571	0,105419593
8	0,003542826	0,0074365	0,112037291
9	0,007346778	0,009488825	0,11275446
10	0,005951335	0,011544072	0,142709745
11	0,012911435	0,009952531	0,114757391
12	0,009232111	0,008044237	0,111435646
13	0,003666863	0,00542262	0,113512768
14	0,005247608	0,006432035	0,106974679
15	0,004514718	0,005781835	0,138429125
16	0,000178901	0,105419593	0,177654697
17	0,005929331	0,041911367	0,14574355
18	0,006793911	0,012742581	0,126744694
19	0,005965617	0,005781835	0,13626142

Випадок для 50 кластерів представлений скорочено (табл. 5.8).

Таблиця 5.8 – Результати кластеризації вибірки даних за визначення 50 кластерів без врахування категоріальних значень

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,006044477	0,006830941	0,116904284
1	0,003737658	0,005006013	0,109845528
2	0,003730777	0,007415138	0,137785606
3	0,003394293	0,004511569	0,110002551
4	0,002281221	0,00309499	0,111586686
8	0,003151137	0,00345028	0,108824468
9	0,004883636	0,005912332	0,115939464
10	0,00223303	0,00316532	0,135605874
11	0,007695563	0,007780807	0,121061572
12	0,002303355	0,00309499	0,11016134
13	0,00272687	0,005316697	0,135149297
14	0,003190848	0,003787341	0,113551662
20	0,005584351	0,015038192	0,149654936
21	0,002621265	0,003588571	0,108061376
22	0,005774979	0,007634026	0,101876297
33	0,005644246	0,009030349	0,119862524
34	0,004798548	0,006830941	0,111875426
35	0,004341608	0,00845715	0,145572663
36	0,003790775	0,004601654	0,115052891
37	0,007032945	0,010958501	0,135279894
47	0,00512696	0,005157392	0,144184419

5.4 Висновки за розділом 5

Проведена перевірка роботи програми для користувача в різних режимах, виділення принципів роботи з програмою у виконанні користувача, виконане експериментальне дослідження є результатами виконання розділу 5. Експериментальне дослідження стосувалось перевірки розбиття даних на кластери. Впровадження методу пошуку настільних ігор, який було запропоновано в роботі й який було інтегровано в програму, для практичних ситуацій є можливим. Його можна використовувати для роботи з даними настільних ігор, визначаючи пріоритетні для користувачів з можливістю виконання подальшого пошуку.

ВИСНОВКИ

У роботі виконано розробку ПЗ пошуку настільних ігор для підвищення зручності користувачів через визначення пріоритетних для них настільних ігор.

Результати роботи включають безпосередньо саму програму пошуку настільних ігор та результати експериментального дослідження кластеризації даних настільних ігор.

Розроблена програма пошуку настільних ігор призначена для надання інформації про настільні ігри, забезпечення можливості замовити настільні ігри в магазинах мережі, пошуку нових настільних ігор, в тому числі через надання пріоритетних настільних ігор щодо вибору, реєстрації власних настільних ігор у програмі, надання відгуків про власний досвід, ознайомлення з інформацією щодо досвіду інших користувачів.

Наукова новизна роботи полягає в запропонованому методі пошуку настільних ігор, в якому визначається попередній набір даних, встановлюються персональні переваги користувача на основі відгуків про настільні ігри, виконується кластеризація даних на основі методу k-середніх, відбувається визначення пріоритетних настільних ігор для користувача та виконується відбір серед них за параметрами пошуку, встановленими користувачем.

Проведене експериментальне дослідження стосувалось перевірки розбиття даних на кластери. Було перевірено відстань між екземплярами всередині кластерів та між різними кластерами для різної кількості кластерів, за якими відбувалось розбиття даних. Впровадження методу пошуку настільних ігор, який було запропоновано в роботі й який було інтегровано в програму, для практичних ситуацій є можливим. Його можна використовувати для роботи з даними настільних ігор, визначаючи пріоритетні для користувачів з можливістю виконання подальшого пошуку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Peiser, J. We're in a golden age of board games. It might be here to stay [Electronic resource] / J. Peiser. – Access mode : <https://www.washingtonpost.com/business/2022/12/24/board-game-popularity/>.
2. Surprising Board Games Statistics to Look Out for in 2023 [Electronic resource]. – Access mode : <https://playtoday.co/blog/stats/board-games-statistics/>.
3. BoardGameGeek. Gaming Unplugged Since 2000 [Electronic resource]. – Access mode : <https://boardgamegeek.com/>.
4. Zatu Games. Board Games & Collectables | Seek Your Adventure [Electronic resource]. – Access mode : <https://www.board-game.co.uk/>.
5. Настільні ігри. Купити настільну гру в Україні, Києві за низькою ціною. Інтернет магазин Geekach Shop [Електрон. ресурс]. – Режим доступу : <https://geekach.com.ua/>.
6. K-Means Clustering in Machine Learning : A Deep Dive [Electronic resource]. – Access mode : <https://datascientest.com/en/k-means-clustering-in-machine-learning-a-deep-dive>.
7. Understanding K-means Clustering in Machine Learning [Electronic resource]. – Access mode : <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>.
8. Mele, A. Django 4 By Example : Build powerful and reliable Python web applications from scratch [Text] / Antonio Mele, Bob Belderbos. – Birmingham : Packt Publishing, 2022. – 766 p.
9. Farrell, D. The Well-Grounded Python Developer : How the pros use Python and Flask [Text] / D. Farrell. – New York : Manning, 2023. – 296 p.
10. MultiShop – Online Shop Website Template [Electronic resource]. – Access mode : <https://htmlcodex.com/online-shop-website-template/>.
11. Samarasinghe, D. BoardGameGeek Dataset on Board Games [Electronic resource] / D. Samarasinghe. – Access mode : <https://dx.doi.org/10.21227/9g61-bs59>.

ДОДАТОК А
Технічне завдання

Вступ

Популярність настільних ігор у сучасному світі вказує з одного боку на те, що все більша кількість учасників з'являється в цій галузі, а відповідно вони потребують самих настільних ігор, їхнього більшого різноманіття, а з іншого боку більше різноманіття є ознакою того, що пошук потрібних варіантів ускладнюється, а тому підбір тих настільних ігор, які можуть бути цікавими користувачу, які йому варто придбати, стає складнішим. Тому для реалізації цього процесу недостатньо тільки наявності відповідних фільтрів пошуку, самого пошуку, а потрібно також мати ПЗ, яке буде надавати певні рекомендації, надавати загалом інформацію про ігри, надавати можливості ці ігри замовити.

Метою роботи є розробити ПЗ пошуку настільних ігор для підвищення зручності користувачів через визначення пріоритетних для них настільних ігор.

A.1 Підстави для розробки

Завдання на роботу визначено самим виданим завданням від 4 вересня 2023 року, що має входити до складу пояснювальної записки, та наказом № 437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка», де затверджено тему роботи: «Дослідження та програмна реалізація методів пошуку настільних ігор».

A.2 Призначення розробки

Програма призначена для надання інформації про настільні ігри, забезпечення можливості замовити настільні ігри в магазинах мережі, пошуку нових настільних ігор, в тому числі через надання пріоритетних настільних ігор щодо вибору, реєстрації власних настільних ігор у програмі,

надання відгуків про власний досвід, ознайомлення з інформацією щодо досвіду інших користувачів.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для того щоб досягти сформульовано призначення розробки в підсумку, потрібно виконати такі функціональні вимоги:

- перегляд сторінки з інформацією про настільну гру;
- замовлення настільних ігор для придбання;
- перегляд власних оформлених замовлень настільних ігор;
- перегляд замовлень за магазином;
- перегляд замовлень за грою;
- скасування оформленого замовлення настільних ігор його власником або працівником відповідного магазину;
- підтвердження оформленого замовлення настільних ігор;
- підтвердження виконання замовлення настільних ігор;
- перегляд рейтингу настільних ігор загалом, за категорією, за механізмом, за виробником;
- перегляд наявності екземплярів настільних ігор у магазині;
- оновлення наявності екземплярів настільних ігор у магазині;
- додавання позиції настільної гри в магазині;
- вилучення позиції настільної гри з магазину;
- додавання нової настільної гри;
- оновлення інформації про настільну гру;
- визначення відгуку про настільну гру;
- реєстрація проведеного сеансу настільної гри;
- оновлення даних сеансу настільної гри;
- перегляд власних сеансів настільних ігор;
- перегляд спільних сеансів настільних ігор з іншими гравцями;

- перегляд даних і настільних ігор творця;
- перегляд даних і настільних ігор виробника;
- додавання нового виробника настільних ігор;
- оновлення даних виробника настільних ігор;
- перегляд усіх виробників настільних ігор;
- перегляд усіх творців, долучених до створення настільних ігор;
- пошук настільних ігор;
- створення персонального переліку настільних ігор;
- додавання настільної гри до персонального переліку;
- вилучення настільної гри з персонального переліку;
- перегляд настільних ігор з персонального переліку;
- перегляд власних персональних переліків настільних ігор;
- перегляд інформації про гравця;
- перегляд відгуків гравця;
- оформлення підписки на гравця;
- оформлення підписки на виробника настільних ігор;
- оформлення підписки на творця;
- скасування підписки на гравця;
- скасування підписки на виробника настільних ігор;
- скасування підписки на творця;
- перегляд творців, на яких оформлено підписку;
- перегляд гравців, на яких оформлено підписку;
- перегляд виробників настільних ігор, на яких оформлено підписку;
- перегляд настільних ігор творців, на яких оформлено підписку;
- перегляд відгуків гравців, на яких оформлено підписку;
- перегляд настільних ігор виробників, на яких оформлено підписку;
- перегляд відгуків про настільну гру;
- додавання нового магазину з продажу настільних ігор у програму;
- оновлення даних магазину з продажу настільних ігор;
- оновлення власних даних;

- авторизація;
- визначення власної настільної гри;
- скасування визначення власної настільної гри;
- перегляд настільних ігор у власності користувача;
- перегляд власних зареєстрованих сеансів (всіх або заданої гри);
- перегляд спільних зареєстрованих сеансів настільної гри;
- пошук за пріоритетними для користувача настільними іграми.

A.3.2 Умови експлуатації та вимоги

Умови експлуатації програми визначаються тими засобами, які мають бути обрані під час проєктування програми, включаючи зокрема підтримку обраної мови програмування, організацію роботи серверу визначеним чином, забезпечення на ньому роботи системи керування базами даних, яку має бути обрано, а також будь-яких додаткових засобів, які обрані та використані під час розробки програми та потребують інсталяції.

A.4 Порядок контролю та приймання

Контроль виконання заявлених складових роботи має відбуватися згідно з завданням, виданим на роботу, що є однією з підстав для розробки. Виконання роботи має бути завершено протягом 12 тижнів з проведенням захисту перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from .models import Publisher
from .models import Gamer
from django.shortcuts import render
from .models import Gamerlist
from .models import Boardgamerating
from .models import Boardgamecategory
from .models import Boardgamemechanism
from .models import Artist
from .models import Boardgameratingcritetion
from .models import Gamerorder
from .models import Gamerordergame
from django.db.models import Q
from .models import Boardgameartist

def gamerinfoview(request, ratinggamer):
    try:
        info_gamer = Gamer.objects.get(username =
ratinggamer)
        info_gamer =
info_gamer.select_related("subscribedgamers")
        except Gamer.DoesNotExist:
            return render(request,
"osboardsoft/gamerinfo.html", {})
        game_ratings = Boardgamerating.objects.filter(gamer =
info_gamer)
        game_ratings = game_ratings.select_related("boardgame")
        game_ratings = game_ratings.order_by("-ratingdate")
        game_ratings = game_ratings.values()[:5]
        for gamerating in game_ratings:
            gamerating.criterions =
Boardgameratingcritetion.objects.filter(boardgamerating =
gamerating)
            game_logs = []
            if request.user.is_authenticated:
                gamer_auth = Gamer.objects.filter(username =
request.user.username)
                gamer_auth =
gamer_auth.prefetch_related("owngames")
                owngames = gamer_auth.owngames
                gamelogs = Boardgamelog.objects.filter(gamer =
info_gamer)
                gamelogs = gamelogs.prefetch_related("gamers")
                gamelogs = gamelogs.order_by("-playtime")
                if gamelogs and gamelogs.count() > 0:
                    cases = 0
                    for gamelog in gamelogs:
                        if gamer_auth in gamelog.gamers:
                            cases = cases + 1
                            game_logs.append(gamelog)
                    if cases == 5:
                        break

```

```

        for gamerating in game_ratings:
            if gamerating.boardgame in owngames:
                gamerating.own = True
            else:
                gamerating.own = False
        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
        gamerlists = gamerlists.prefetch_related("games")
        for gamerating in game_ratings:
            gamerating.inlist = []
            for gamerlist in gamerlists:
                if gamerating.boardgame in gamerlist.games:
                    gamerating.inlist.append(gamerlist)
        for gamerating in game_ratings:
            ownratings =
Boardgamerating.objects.filter(boardgame = gamerating.boardgame,
gamer = gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            gamerating.ratings = ownratings.values()[0]
            if gamer_auth in info_gamer.subscribedgamers:
                gamer_auth.subscribed = True
        return render(request, "osboardsoft/gamerinfo.html", {
"game": info_gamer, "ratings": game_ratings, "logs": game_logs
})

```

```

def allpublishersview(request):
    publishers_list = Publisher.objects.all()
    publishers_list = publishers_list.order_by("firmname")
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("subscribedpublishers")
        subscribedpublishers =
gamer_auth.subscribedpublishers
        for publisher in publishers_list:
            if publisher in subscribedpublishers:
                publisher.subscribed = True
            else:
                publisher.subscribed = False
        return render(request, "osboardsoft/publishers.html",
{"publishers": publishers_list})

```

```

def allboardgamesratingview(request, order_principle = 0):
    boardgames = Boardgame.objects.all()
    boardgames = boardgames.select_related("publisher")
    if order_principle == 0:
        boardgames = boardgames.order_by("-rating")
    elif order_principle == 1:
        boardgames = boardgames.order_by("-artwork")
    elif order_principle == 2:

```

```

        boardgames = boardgames.order_by("-complexity")
    elif order_principle == 3:
        boardgames = boardgames.order_by("-interaction")
    elif order_principle == 4:
        boardgames = boardgames.order_by("-quality")
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("owngames")
        owngames = gamer_auth.owngames
        for boardgame in boardgames:
            if boardgame in owngames:
                boardgame.own = True
            else:
                boardgame.own = False
        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
        gamerlists = gamerlists.prefetch_related("games")
        for boardgame in boardgames:
            boardgame.inlist = []
            for gamerlist in gamerlists:
                if boardgame in gamerlist.games:
                    boardgame.inlist.append(gamerlist)
        for boardgame in boardgames:
            ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            boardgame.ratings = ownratings.values()[0]
        return render(request, "osboardsoft/rating.html",
{"boardgames": boardgames})

def allartistsview(request):
    artists_list = Artist.objects.all()
    artists_list = artists_list.order_by("fullname")
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("subscribedartists")
        subscribedartists = gamer_auth.subscribedartists
        for artist in artists_list:
            if artist in subscribedartists:
                artist.subscribed = True
            else:
                artist.subscribed = False
        return render(request, "osboardsoft/artists.html",
{"artists": artists_list})

def owngamelistsview(request):

```

```

        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamelists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamelists = gamelists.prefetch_related("games")
            gamelists = gamelists.order_by("-id")
            return render(request, "osboardsoft/ownlists.html",
{"lists": gamelists})

    def categoryboardgamesratingview(request, gamecategory,
order_principle = 0):
        try:
            category =
Boardgamecategory.objects.get(gamecategory = gamecategory)
            except Boardgamecategory.DoesNotExist:
                return render(request,
"osboardsoft/categoryrating.html", {})
            boardgames = Boardgame.objects.filter(categories__in =
category)
            boardgames = boardgames.select_related("publisher")
            if order_principle == 0:
                boardgames = boardgames.order_by("-rating")
            elif order_principle == 1:
                boardgames = boardgames.order_by("-artwork")
            elif order_principle == 2:
                boardgames = boardgames.order_by("-complexity")
            elif order_principle == 3:
                boardgames = boardgames.order_by("-interaction")
            elif order_principle == 4:
                boardgames = boardgames.order_by("-quality")
            if request.user.is_authenticated:
                gamer_auth = Gamer.objects.filter(username =
request.user.username)
                gamer_auth =
gamer_auth.prefetch_related("owngames")
                owngames = gamer_auth.owngames
                for boardgame in boardgames:
                    if boardgame in owngames:
                        boardgame.own = True
                    else:
                        boardgame.own = False
                gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
                gamerlists = gamerlists.prefetch_related("games")
                for boardgame in boardgames:
                    boardgame.inlist = []
                    for gamerlist in gamerlists:
                        if boardgame in gamerlist.games:
                            boardgame.inlist.append(gamerlist)
                for boardgame in boardgames:

```

```

        ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
        ownratings = ownratings.order_by("-ratingdate")
        boardgame.ratings = ownratings.values()[0]
    return render(request,
"osboardsoft/categoryrating.html", {"boardgames": boardgames})

    def subartistsview(request):
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("subscribedartists")
            subscribedartists = gamer_auth.subscribedartists
            subscribedartists = subscribedartists.order_by("-
id")
            return render(request, "osboardsoft/subartists.html",
{"artists": subscribedartists})

    def publisherboardgamesratingview(request, publisher,
order_principle = 0):
        try:
            publisher = Publisher.objects.get(id = publisher)
        except Publisher.DoesNotExist:
            return render(request,
"osboardsoft/publisherrating.html", {})
        boardgames = Boardgame.objects.filter(publisher =
publisher)
        if order_principle == 0:
            boardgames = boardgames.order_by("-rating")
        elif order_principle == 1:
            boardgames = boardgames.order_by("-artwork")
        elif order_principle == 2:
            boardgames = boardgames.order_by("-complexity")
        elif order_principle == 3:
            boardgames = boardgames.order_by("-interaction")
        elif order_principle == 4:
            boardgames = boardgames.order_by("-quality")
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            for boardgame in boardgames:
                if boardgame in owngames:
                    boardgame.own = True
                else:
                    boardgame.own = False

```

```

        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
        gamerlists = gamerlists.prefetch_related("games")
        for boardgame in boardgames:
            boardgame.inlist = []
            for gamerlist in gamerlists:
                if boardgame in gamerlist.games:
                    boardgame.inlist.append(gamerlist)
        for boardgame in boardgames:
            ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            boardgame.ratings = ownratings.values()[0]
        return render(request,
"osboardsoft/publisherrating.html", {"boardgames": boardgames})

    def gamerpublishersview(request):
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("subscribedpublishers")
            subscribedpublishers =
gamer_auth.subscribedpublishers
            subscribedpublishers =
subscribedpublishers.order_by("-id")
            return render(request,
"osboardsoft/subpublishers.html", {"pubslishers":
subscribedpublishers})

    def mechanismboardgamesratingview(request, gamemechanism,
order_principle = 0):
        try:
            mechanism = Boardgamemechanism.objects.get(id =
gamemechanism)
        except Boardgamemechanism.DoesNotExist:
            return render(request,
"osboardsoft/mechanismrating.html", {})
        mechanisms =
Boardgamemechanism.objects.filter(gamemechanism = mechanism)
        mechanisms = mechanisms.select_related("boardgame")
        mechanisms = mechanisms.values_list("boardgame__id")
        mechanisms = mechanisms.distinct()
        boardgames = Boardgame.objects.filter(id__in =
mechanisms)
        boardgames = boardgames.select_related("publisher")
        if order_principle == 0:
            boardgames = boardgames.order_by("-rating")
        elif order_principle == 1:
            boardgames = boardgames.order_by("-artwork")

```

```

elif order_principle == 2:
    boardgames = boardgames.order_by("-complexity")
elif order_principle == 3:
    boardgames = boardgames.order_by("-interaction")
elif order_principle == 4:
    boardgames = boardgames.order_by("-quality")
if request.user.is_authenticated:
    gamer_auth = Gamer.objects.filter(username =
request.user.username)
    gamer_auth =
gamer_auth.prefetch_related("owngames")
    owngames = gamer_auth.owngames
    for boardgame in boardgames:
        if boardgame in owngames:
            boardgame.own = True
        else:
            boardgame.own = False
    gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
    gamerlists = gamerlists.prefetch_related("games")
    for boardgame in boardgames:
        boardgame.inlist = []
        for gamerlist in gamerlists:
            if boardgame in gamerlist.games:
                boardgame.inlist.append(gamerlist)
    for boardgame in boardgames:
        ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
        ownratings = ownratings.order_by("-ratingdate")
        boardgame.ratings = ownratings.values()[0]
    return render(request,
"osboardsoft/mechanismrating.html", {"boardgames": boardgames})

def listboardgamesview(request, gamelist, order_principle =
5):
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("owngames")
        try:
            gamelist = Gamerlist.objects.get(id = gamelist,
gamer = gamer_auth)
            gamelist = gamelist.prefetch_related("games")
            gamelist =
gamelist.prefetch_related("games__publisher")
        except Gamerlist.DoesNotExist:
            return render(request,
"osboardsoft/listgames.html", {})
        boardgames = gamelist.games

```

```

        if order_principle == 0:
            boardgames = boardgames.order_by("-rating")
        elif order_principle == 1:
            boardgames = boardgames.order_by("-artwork")
        elif order_principle == 2:
            boardgames = boardgames.order_by("-complexity")
        elif order_principle == 3:
            boardgames = boardgames.order_by("-
interaction")
        elif order_principle == 4:
            boardgames = boardgames.order_by("-quality")
        elif order_principle == 5:
            boardgames = boardgames.order_by("-id")
        owngames = gamer_auth.owngames
        for boardgame in boardgames:
            if boardgame in owngames:
                boardgame.own = True
            else:
                boardgame.own = False
        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
        gamerlists = gamerlists.prefetch_related("games")
        for boardgame in boardgames:
            boardgame.inlist = []
            for gamerlist in gamerlists:
                if boardgame in gamerlist.games:
                    boardgame.inlist.append(gamerlist)
        for boardgame in boardgames:
            ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            boardgame.ratings = ownratings.values()[0]
        return render(request,
"osboardsoft/listgames.html", {"boardgames": boardgames})

    def ownboardgamesview(request, order_principle = 0):
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            gamer_auth =
gamer_auth.prefetch_related("owngames__publisher")
            boardgames = gamer_auth.owngames
            if order_principle == 0:
                boardgames = boardgames.order_by("-
publishdate")
            elif order_principle == 1:
                boardgames = boardgames.order_by("-rating")
            elif order_principle == 2:

```

```

        boardgames = boardgames.order_by("-artwork")
    elif order_principle == 3:
        boardgames = boardgames.order_by("-complexity")
    elif order_principle == 4:
        boardgames = boardgames.order_by("-
interaction")
    elif order_principle == 5:
        boardgames = boardgames.order_by("-quality")
    elif order_principle == 6:
        boardgames = boardgames.order_by("-newprice")
    elif order_principle == 7:
        boardgames = boardgames.order_by("newprice")
    gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
    gamerlists = gamerlists.prefetch_related("games")
    for boardgame in boardgames:
        boardgame.inlist = []
        for gamerlist in gamerlists:
            if boardgame in gamerlist.games:
                boardgame.inlist.append(gamerlist)
    for boardgame in boardgames:
        ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
        ownratings = ownratings.order_by("-ratingdate")
        boardgame.ratings = ownratings.values()[0]
    return render(request, "osboardsoft/owngames.html",
{"boardgames": boardgames})

def boardgameratingsview(request, boardgame):
    try:
        boardgame = Boardgame.objects.get(id = boardgame)
    except Boardgame.DoesNotExist:
        return render(request,
"osboardsoft/gameratings.html", {})
    gameratings_list =
Boardgamerating.objects.filter(boardgame = boardgame)
    gameratings_list =
gameratings_list.select_related("gamer")
    gameratings_list = gameratings_list.order_by("-
ratingdate")
    for gamerating in gameratings_list:
        gamerating.criterions =
Boardgameratingcritetion.objects.filter(boardgamerating =
gamerating)
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("subscribedgamers")
            subscribedgamers = gamer_auth.subscribedgamers

```

```

        for gamerating in gameratings_list:
            if gamerating.gamer in subscribedgamers:
                gamerating.subscribed = True
            else:
                gamerating.subscribed = False
        return render(request, "osboardsoft/gameratings.html",
{"ratings": gameratings_list})

def subgameratingsview(request):
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("subscribedgamers")
        gameratings_list =
Boardgamerating.objects.filter(gamer__in =
gamer_auth.subscribedgamers)
        gameratings_list =
gameratings_list.select_related("gamer")
        gameratings_list = gameratings_list.order_by("-
ratingdate")
        for gamerating in gameratings_list:
            gamerating.criteria =
Boardgameratingcritetion.objects.filter(boardgamerating =
gamerating)
        return render(request,
"osboardsoft/subgameratings.html", {"ratings":
gameratings_list})

def subpublishergamesview(request, order_principle = 0):
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("subscribedpublishers")
        pubgames_list =
Boardgame.objects.filter(publisher__in =
gamer_auth.subscribedpublishers)
        boardgames =
pubgames_list.select_related("publisher")
        if order_principle == 0:
            boardgames = boardgames.order_by("-
publishdate")
        elif order_principle == 1:
            boardgames = boardgames.order_by("-rating")
        elif order_principle == 2:
            boardgames = boardgames.order_by("-artwork")
        elif order_principle == 3:
            boardgames = boardgames.order_by("-complexity")
        elif order_principle == 4:

```

```

        boardgames = boardgames.order_by("-
interaction")
    elif order_principle == 5:
        boardgames = boardgames.order_by("-quality")
    elif order_principle == 6:
        boardgames = boardgames.order_by("-newprice")
    elif order_principle == 7:
        boardgames = boardgames.order_by("newprice")
    return render(request,
"osboardsoft/subpublishergames.html", {"games": boardgames})

def subartistgamesview(request, order_principle = 0):
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("subscribedartists")
        designergames_list =
Boardgame.objects.filter(maindesigner__in =
gamer_auth.subscribedartists)
        designergames_list =
designergames_list.values_list("id")
        developergames_list =
Boardgame.objects.filter(developer__in =
gamer_auth.subscribedartists)
        developergames_list =
developergames_list.values_list("id")
        editorgames_list =
Boardgame.objects.filter(editor__in =
gamer_auth.subscribedartists)
        editorgames_list =
developergames_list.editorgames_list("id")
        artistgames_list =
Boardgameartist.objects.filter(artist__in =
gamer_auth.subscribedartists)
        artistgames_list =
artistgames_list.select_related("boardgame")
        artistgames_list =
artistgames_list.values_list("boardgame__id")
        boardgames = Boardgame.objects.filter(id__in =
[designergames_list, developergames_list, editorgames_list,
artistgames_list])
        boardgames = boardgames.select_related("publisher")
        if order_principle == 0:
            boardgames = boardgames.order_by("-
publishdate")
        elif order_principle == 1:
            boardgames = boardgames.order_by("-rating")
        elif order_principle == 2:
            boardgames = boardgames.order_by("-artwork")
        elif order_principle == 3:

```

```

        boardgames = boardgames.order_by("-complexity")
    elif order_principle == 4:
        boardgames = boardgames.order_by("-
interaction")
    elif order_principle == 5:
        boardgames = boardgames.order_by("-quality")
    elif order_principle == 6:
        boardgames = boardgames.order_by("-newprice")
    elif order_principle == 7:
        boardgames = boardgames.order_by("newprice")
    return render(request,
"osboardsoft/subartistgamesview.html", {"games": boardgames})

    def gamergamersview(request):
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("subscribedgamers")
            subscribedgamers = gamer_auth.subscribedgamers
            subscribedgamers = subscribedgamers.order_by("-id")
            return render(request, "osboardsoft/subgamers.html",
{"gamers": subscribedgamers })

    def gamerratingsview(request, ratinggamer):
        try:
            rating_gamer = Gamer.objects.get(username =
ratinggamer)
            rating_gamer =
rating_gamer.select_related("subscribedgamers")
        except Gamer.DoesNotExist:
            return render(request,
"osboardsoft/gamerratings.html", {})
        game_ratings = Boardgamerating.objects.filter(gamer =
rating_gamer)
        game_ratings = game_ratings.select_related("boardgame")
        game_ratings = game_ratings.order_by("-ratingdate")
        for gamerating in game_ratings:
            gamerating.criterions =
Boardgameratingcritetion.objects.filter(boardgamerating =
gamerating)
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            if gamer_auth in rating_gamer.subscribedgamers:
                rating_gamer.subscribed = True
            else:
                rating_gamer.subscribed = False

```

```

        for gamerating in game_ratings:
            if gamerating.boardgame in owngames:
                gamerating.own = True
            else:
                gamerating.own = False
        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
        gamerlists = gamerlists.prefetch_related("games")
        for gamerating in game_ratings:
            gamerating.inlist = []
            for gamerlist in gamerlists:
                if gamerating.boardgame in gamerlist.games:
                    gamerating.inlist.append(gamerlist)
        for gamerating in game_ratings:
            ownratings =
Boardgamerating.objects.filter(boardgame = gamerating.boardgame,
gamer = gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            gamerating.ratings = ownratings.values()[0]
        return render(request, "osboardsoft/gamerratings.html",
{"ratings": game_ratings })

def artistinfoview(request, artistinfo):
    try:
        info_artist = Artist.objects.get(id = artistinfo)
    except Artist.DoesNotExist:
        return render(request,
"osboardsoft/artistinfo.html", {})
    designergames_list =
Boardgame.objects.filter(maindesigner = info_artist)
    designergames_list =
designergames_list.values_list("id")
    developergames_list =
Boardgame.objects.filter(developer = info_artist)
    developergames_list =
developergames_list.values_list("id")
    editorgames_list = Boardgame.objects.filter(editor =
info_artist)
    editorgames_list =
developergames_list.editorgames_list("id")
    artistgames_list =
Boardgameartist.objects.filter(artist = info_artist)
    artistgames_list =
artistgames_list.select_related("boardgame")
    artistgames_list =
artistgames_list.values_list("boardgame__id")
    boardgames = Boardgame.objects.filter(id__in =
[designergames_list, developergames_list, editorgames_list,
artistgames_list])
    boardgames = boardgames.select_related("publisher")
    boardgames = boardgames.order_by("-rating")

```

```

        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            for boardgame in boardgames:
                if boardgame in owngames:
                    boardgame.own = True
                else:
                    boardgame.own = False
            gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamerlists = gamerlists.prefetch_related("games")
            for boardgame in boardgames:
                boardgame.inlist = []
                for gamerlist in gamerlists:
                    if boardgame in gamerlist.games:
                        boardgame.inlist.append(gamerlist)
            for boardgame in boardgames:
                ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
                ownratings = ownratings.order_by("-ratingdate")
                boardgame.ratings = ownratings.values()[0]
            return render(request, "osboardsoft/artistinfo.html", {
"artist": info_artist, "games": boardgames})

def publisherinfoview(request, publisherinfo):
    try:
        info_publisher = Publisher.objects.get(id =
publisherinfo)
    except Publisher.DoesNotExist:
        return render(request,
"osboardsoft/publisherinfo.html", {})
    boardgames = Boardgame.objects.filter(publisher =
info_publisher)
    boardgames = boardgames.order_by("-rating")
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gamer_auth =
gamer_auth.prefetch_related("owngames")
        owngames = gamer_auth.owngames
        for boardgame in boardgames:
            if boardgame in owngames:
                boardgame.own = True
            else:
                boardgame.own = False
        gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)

```

```

        gamerlists = gamerlists.prefetch_related("games")
        for boardgame in boardgames:
            boardgame.inlist = []
            for gamerlist in gamerlists:
                if boardgame in gamerlist.games:
                    boardgame.inlist.append(gamerlist)
        for boardgame in boardgames:
            ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            boardgame.ratings = ownratings.values()[0]
        return render(request,
"osboardsoft/publisherinfo.html", { "publisher": info_publisher,
"games": boardgames})

    def owngamerlogsview(request):
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            gamelogs = Boardgamelog.objects.filter(gamer =
gamer_auth)
            gamelogs = gamelogs.prefetch_related("gamers")
            gamelogs = gamelogs.select_related("boardgame")
            gamelogs = gamelogs.order_by("-playtime")
            for gamelog in gamelogs:
                if gamelog.boardgame in owngames:
                    gamelog.own = True
                else:
                    gamelog.own = False
            gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamerlists = gamerlists.prefetch_related("games")
            for gamelog in gamelogs:
                gamelog.inlist = []
                for gamerlist in gamerlists:
                    if gamelog.boardgame in gamerlist.games:
                        gamelog.inlist.append(gamerlist)
            for gamelog in gamelogs:
                ownratings =
Boardgamerating.objects.filter(boardgame = gamerating.boardgame,
gamer = gamer_auth)
                ownratings = ownratings.order_by("-ratingdate")
                gamelog.ratings = ownratings.values()[0]
            return render(request, "osboardsoft/ownlogs.html",
{"logs": gamelogs })

    def sharedgamerlogsview(request):

```

```

        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            gamelogs = Boardgamelog.objects.filter(gamers__in =
gamer_auth)
            gamelogs = gamelogs.prefetch_related("gamers")
            gamelogs = gamelogs.select_related("boardgame")
            gamelogs = gamelogs.order_by("-playtime")
            for gamelog in gamelogs:
                if gamelog.boardgame in owngames:
                    gamelog.own = True
                else:
                    gamelog.own = False
            gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamerlists = gamerlists.prefetch_related("games")
            for gamelog in gamelogs:
                gamelog.inlist = []
                for gamerlist in gamerlists:
                    if gamelog.boardgame in gamerlist.games:
                        gamelog.inlist.append(gamerlist)
            for gamelog in gamelogs:
                ownratings =
Boardgamerating.objects.filter(boardgame = gamerating.boardgame,
gamer = gamer_auth)
                ownratings = ownratings.order_by("-ratingdate")
                gamelog.ratings = ownratings.values()[0]
            return render(request,
"osboardsoft/sharedlog.html", {"logs": gamelogs })

def gameinfoview(request, boardgame):
    try:
        info_boardgame = Boardgame.objects.get(id =
boardgame).prefetch_related("photos")
    except Boardgame.DoesNotExist:
        return render(request, "osboardsoft/game.html", {})
    game_ratings = Boardgamerating.objects.filter(boardgame
= info_boardgame)
    game_ratings = game_ratings.order_by("-ratingdate")
    game_ratings = game_ratings.values()[:5]
    for gamerating in game_ratings:
        gamerating.criteria =
Boardgameratingcritetion.objects.filter(boardgamerating =
gamerating)
        shops = Boardgameshop.objects.filter(boardgame =
info_boardgame)
        shops = shops.select_related("gameshop")
        game_logs = []

```

```

        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            gamelogs = Boardgamelog.objects.filter(gamer =
gamer_auth, boardgame = info_boardgame)
            gamelogs = gamelogs.prefetch_related("gamers")
            gamelogs = gamelogs.order_by("-playtime")
            gamelogs = gamelogs.values_list("id")
            shared_logs = Boardgamelog.objects.filter(gamers =
gamer_auth)
            shared_logs = shared_logs.order_by("-playtime")
            shared_logs = shared_logs.values_list("id")
            gamelogs = Boardgamelog.objects.filter(id__in =
[gamelogs, shared_logs])
            if gamelogs and gamelogs.count() > 0:
                cases = 0
                for gamelog in gamelogs:
                    if gamer_auth in gamelog.gamers or
gamelog.gamer == gamer_auth:
                        cases = cases + 1
                        game_logs.append(gamelog)
                        if cases == 5:
                            break
            if info_boardgame in owngames:
                info_boardgame.own = True
            else:
                info_boardgame.own = False
            gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamerlists = gamerlists.prefetch_related("games")
            for gamerlist in gamerlists:
                if info_boardgame in gamerlist.games:
                    info_boardgame.inlist.append(gamerlist)
            ownratings =
Boardgamerating.objects.filter(boardgame = info_boardgame, gamer
= gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            info_boardgame.ratings = ownratings.values()[0]
            subscribedgamers = gamer_auth.subscribedgamers
            for gamerating in game_ratings:
                if gamerating.gamer in subscribedgamers:
                    gamerating.subscribed = True
                else:
                    gamerating.subscribed = False
            return render(request, "osboardsoft/game.html", {
"game": info_boardgame, "ratings": game_ratings, "logs":
game_logs, "shops": shops })

```

```

def owngameordersview(request, orderstate = -1):
    if request.user.is_authenticated:
        gamer_auth = Gamer.objects.filter(username =
request.user.username)
        gameorders = Gamerorder.objects.filter(gamer =
gamer_auth)
        if state != -1:
            gameorders = gameorders.filter(state =
orderstate)
            gameorders = gameorders.select_related("gameshop")
            gameorders = gameorders.order_by("-orderdate")
            for gameorder in gameorders:
                gameorder_content =
Gamerordergame.objects.filter(gameorder = gameorder)
                gameorder_content =
gameorder_content.select_related("boardgame")
                gameorder.content = gameorder_content
            return render(request, "osboardsoft/gameorders.html", {
"orders": gameorders })

import datetime

def boardgamessearchview(request):
    boardgames = Boardgame.objects.all()
    boardgames = boardgames.select_related('publisher')
    if request.method == 'GET' and 'name' in request.GET:
        gamename = request.GET.get('name', '')
        boardgames = boardgames.filter(gamename__icontains
= gamename)
    if request.method == 'GET' and 'mintime' in
request.GET:
        gamemintime = request.GET.get('mintime', '')
        if gamemintime:
            gamemintime =
datetime.datetime.strptime(gamemintime, '%H:%M').time()
            boardgames = boardgames.filter(mintime__gte =
gamemintime)
    if request.method == 'GET' and 'maxtime' in
request.GET:
        gamemaxtime = request.GET.get('maxtime', '')
        if gamemaxtime:
            gamemaxtime =
datetime.datetime.strptime(gamemaxtime, '%H:%M').time()
            boardgames = boardgames.filter(maxtime__gte =
gamemaxtime)
    if request.method == 'GET' and 'minage' in request.GET:
        gameminage = request.GET.get('minage', '')
        gameminage = int(gameminage)
        if gameminage:
            boardgames = boardgames.filter(minage__lte =
gameminage)

```

```

        if request.method == 'GET' and 'minpl' in request.GET:
            gameminplayers = request.GET.get('minpl', '')
            gameminplayers = int(gameminplayers)
            if gameminplayers:
                boardgames = boardgames.filter(minplayers__lte
= gameminplayers)
        if request.method == 'GET' and 'maxpl' in request.GET:
            gamemaxplayers = request.GET.get('maxpl', '')
            gamemaxplayers = int(gamemaxplayers)
            if gamemaxplayers:
                boardgames = boardgames.filter(maxplayers__lte
= gamemaxplayers)
        if request.method == 'GET' and 'minrat' in request.GET:
            gameminrat = request.GET.get('minrat', '')
            gameminrat = float(gameminrat)
            if gameminrat:
                boardgames = boardgames.filter(rating__gte =
gameminrat)
        if request.method == 'GET' and 'minart' in request.GET:
            gameminart = request.GET.get('minart', '')
            gameminart = float(gameminart)
            if gameminart:
                boardgames = boardgames.filter(artwork__gte =
gameminart)
        if request.method == 'GET' and 'mincompl' in
request.GET:
            gamemincompl = request.GET.get('mincompl', '')
            gamemincompl = float(gamemincompl)
            if gamemincompl:
                boardgames = boardgames.filter(complexity__gte
= gamemincompl)
        if request.method == 'GET' and 'minq' in request.GET:
            gameminq = request.GET.get('minq', '')
            gameminq = float(gameminq)
            if gameminq:
                boardgames = boardgames.filter(quality__gte =
gameminq)
        if request.method == 'GET' and 'mininter' in
request.GET:
            gamemininter = request.GET.get('mininter', '')
            gamemininter = float(gamemininter)
            if gamemininter:
                boardgames = boardgames.filter(interaction__gte
= gamemininter)
        if request.method == 'GET' and 'minprice' in
request.GET:
            gameminprice = request.GET.get('minprice', '')
            gameminprice = float(gameminprice)
            if gameminprice:
                boardgames = boardgames.filter(newprice__gte =
gameminprice)

```

```

        if request.method == 'GET' and 'maxprice' in
request.GET:
            gamemaxprice = request.GET.get('maxprice', '')
            gamemaxprice = float(gamemaxprice)
            if gamemaxprice:
                boardgames = boardgames.filter(newprice__lte =
gamemaxprice)
        if request.method == 'GET' and 'minyear' in
request.GET:
            gameminyear = request.GET.get('minyear', '')
            gameminyear = int(gameminyear)
            if gameminyear:
                boardgames = boardgames.filter(publishdate__gte
= gameminyear)
        if request.method == 'GET' and 'maxyear' in
request.GET:
            gamemaxyear = request.GET.get('maxyear', '')
            gamemaxyear = int(gamemaxyear)
            if gamemaxyear:
                boardgames = boardgames.filter(publishdate__gte
= gamemaxyear)
        if request.method == 'GET' and 'publ' in request.GET:
            publisher = request.GET.get('publ', '')
            publisher = int(publisher)
            if publisher:
                boardgames = boardgames.filter(publisher =
publisher)
        if request.method == 'GET' and 'categ' in request.GET:
            category = request.GET.get('categ', '')
            category = int(category)
            if category:
                category = Gamecategory.objects.get(id =
category)
                boardgames = boardgames.filter(categories =
category)
        if request.method == 'GET' and 'mechan' in request.GET:
            mechanism = request.GET.get('mechan', '')
            mechanism = int(mechanism)
            if mechanism:
                boardmechan =
Boardgamemechanism.objects.filter(gamemechanism__id = mechanism)
                boardmechan =
boardmechan.select_related('boardgame')
                boardmechan =
boardmechan.values_list('boardgame__id')
                boardgames = boardgames.filter(id__in =
boardmechan)
        if request.method == 'GET' and 'art' in request.GET:
            artist = request.GET.get('art', '')
            artist = int(artist)
            if artist:

```

```

        boardgames2 =
boardgames.filter(Q(maindesigner__id = artist) | Q(developer__id
= artist) | Q(editor__id = artist))
        boardgames2 = boardgames2.values_list('id')
        boardgames3 =
Boardgameartist.objects.filter(artist = artist)
        boardgames3 =
boardgames3.select_related('boardgame')
        boardgames3 =
boardgames3.select_related('artist')
        boardgames3 =
boardgames3.values_list('boardgame__id')
        boardgames3 = boardgames3.distinct()
        boardgames = boardgames.filter(id__in =
[boardgames2, boardgames3])
        if request.method == 'GET' and 'ord' in request.GET:
            if order_principle == 0:
                boardgames = boardgames.order_by("-
publishdate")
            elif order_principle == 1:
                boardgames = boardgames.order_by("-rating")
            elif order_principle == 2:
                boardgames = boardgames.order_by("-artwork")
            elif order_principle == 3:
                boardgames = boardgames.order_by("-complexity")
            elif order_principle == 4:
                boardgames = boardgames.order_by("-
interaction")
            elif order_principle == 5:
                boardgames = boardgames.order_by("-quality")
            elif order_principle == 6:
                boardgames = boardgames.order_by("-newprice")
            elif order_principle == 7:
                boardgames = boardgames.order_by("newprice")
        else:
            boardgames = boardgames.order_by("-rating")
        if request.user.is_authenticated:
            gamer_auth = Gamer.objects.filter(username =
request.user.username)
            gamer_auth =
gamer_auth.prefetch_related("owngames")
            owngames = gamer_auth.owngames
            for boardgame in boardgames:
                if boardgame in owngames:
                    boardgame.own = True
                else:
                    boardgame.own = False
            gamerlists = Gamerlist.objects.filter(gamer =
gamer_auth)
            gamerlists = gamerlists.prefetch_related("games")
            for boardgame in boardgames:

```

```

        boardgame.inlist = []
        for gamerlist in gamerlists:
            if boardgame in gamerlist.games:
                boardgame.inlist.append(gamerlist)
        for boardgame in boardgames:
            ownratings =
Boardgamerating.objects.filter(boardgame = boardgame, gamer =
gamer_auth)
            ownratings = ownratings.order_by("-ratingdate")
            boardgame.ratings = ownratings.values()[0]
        publishers = Publisher.objects.all()
        artists = Artist.objects.all()
        categories = Gamecategory.objects.all()
        mechanisms = Gamemechanism.objects.all()
        return render(request, "osboardsoft/search.html", {
"games": boardgames, "publishers": publishers, "artists":
artists, "categories": categories, "mechanisms": mechanisms})

```

ДОДАТОК В
Слайди презентації

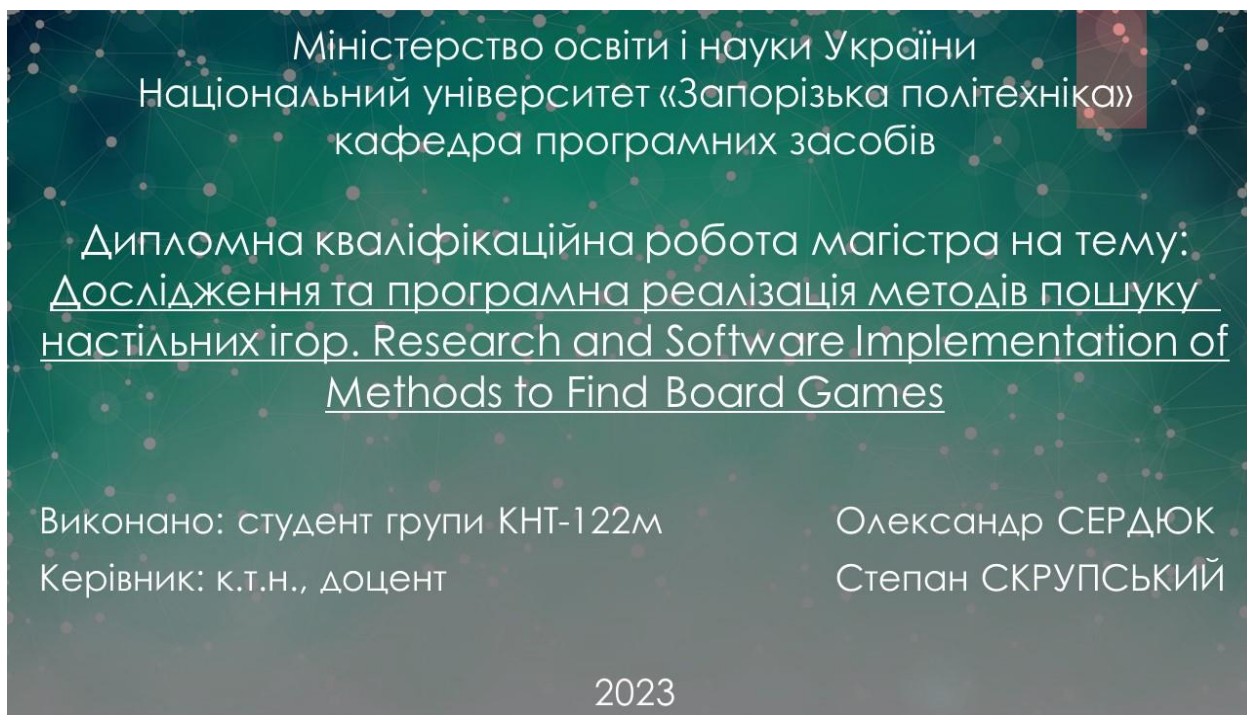


Рисунок В.1 – Титульний слайд

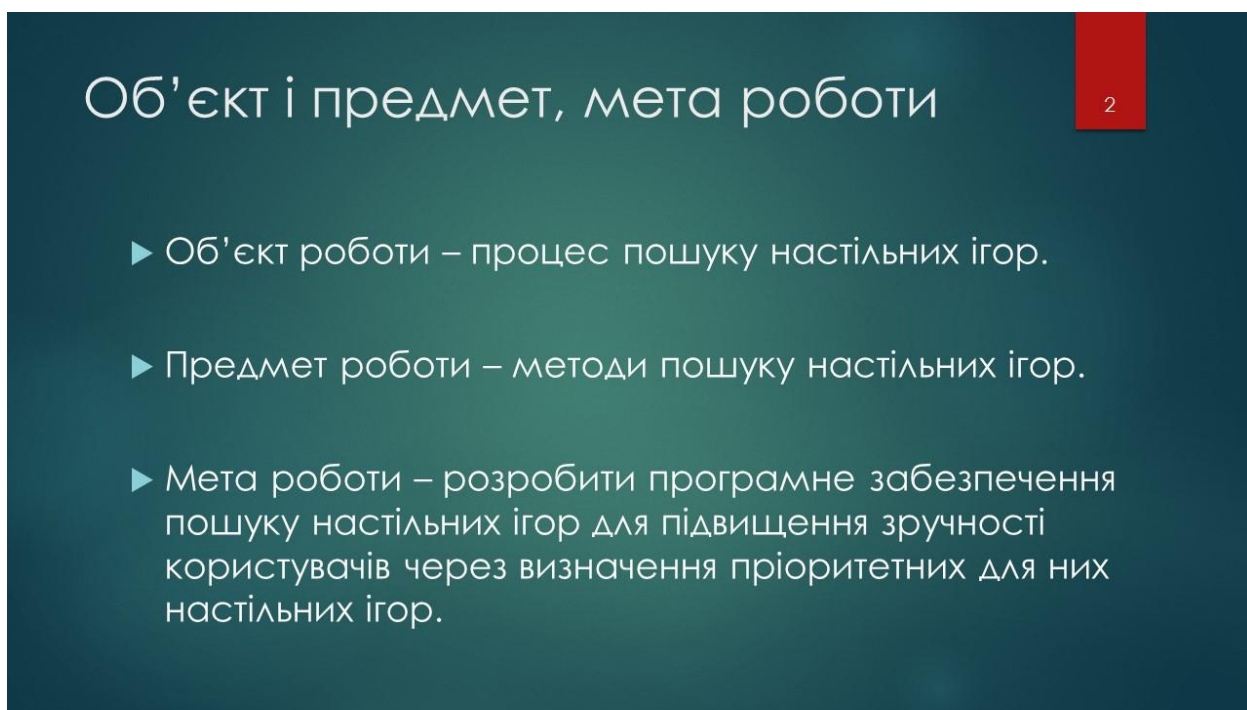


Рисунок В.2 – Об'єкт і предмет, мета роботи

Завдання роботи

3

- ▶ аналіз існуючих програм, які виконують інформування про настільні ігри;
- ▶ дослідження методів і моделей пошуку настільних ігор;
- ▶ проектування програмного забезпечення пошуку настільних ігор;
- ▶ реалізація програми пошуку настільних ігор.

Рисунок В.3 – Завдання роботи

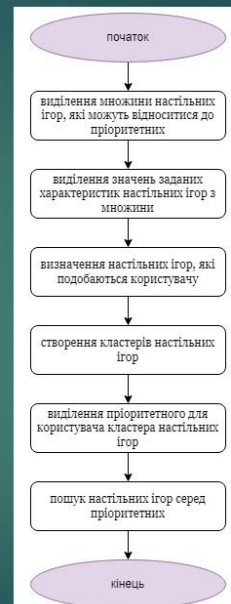
Характеристика розроблюваної та існуючих програм

4

Показник	BoardGameGeek	Zatu Games	Geekach	OSBoardSoft
Пошук настільних ігор	Так	Так, тільки за назвою	Так	Так
Замовлення настільних ігор	Ні	Так	Так	Так
Збереження настільних ігор у програмі	Так, у наперед визначені переліки	Так	Так	Так, у власні переліки
Реєстрація сеансів	Так	Ні	Ні	Так
Рейтинг настільних ігор	За відгуками	За оцінками експертів	За відгуками	За відгуками
Пошук за пріоритетними для користувача настільними іграми	Ні	Ні	Ні	Так

Рисунок В.4 – Характеристика розроблюваної та існуючих програм

Метод пошуку настільних ігор



5

Рисунок В.5 – Метод пошуку настільних ігор

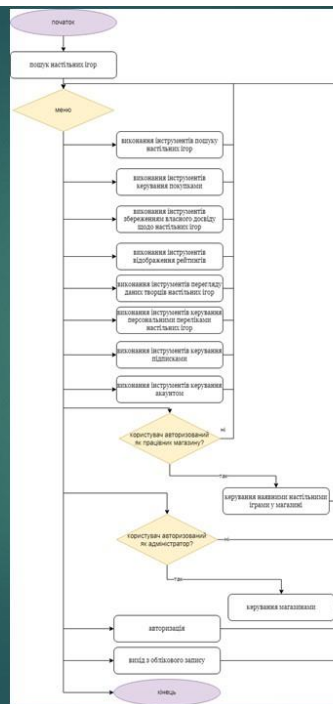
Діаграма прецедентів користувачів при роботі з програмним забезпеченням



6

Рисунок В.6 – Діаграма прецедентів користувачів при роботі з програмним забезпеченням

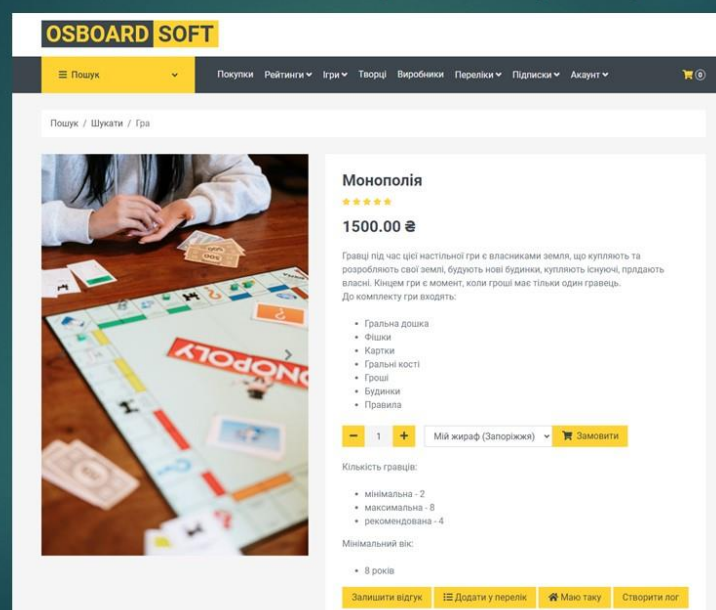
Функціональна схема програми пошуку настільних ігор



9

Рисунок В.9 – Функціональна схема програми пошуку настільних ігор

Сторінка настільної гри в програмі



10

Рисунок В.10 – Сторінка настільної гри в програмі

Пошук за пріоритетними для користувача настільними іграми і робота працівника магазину

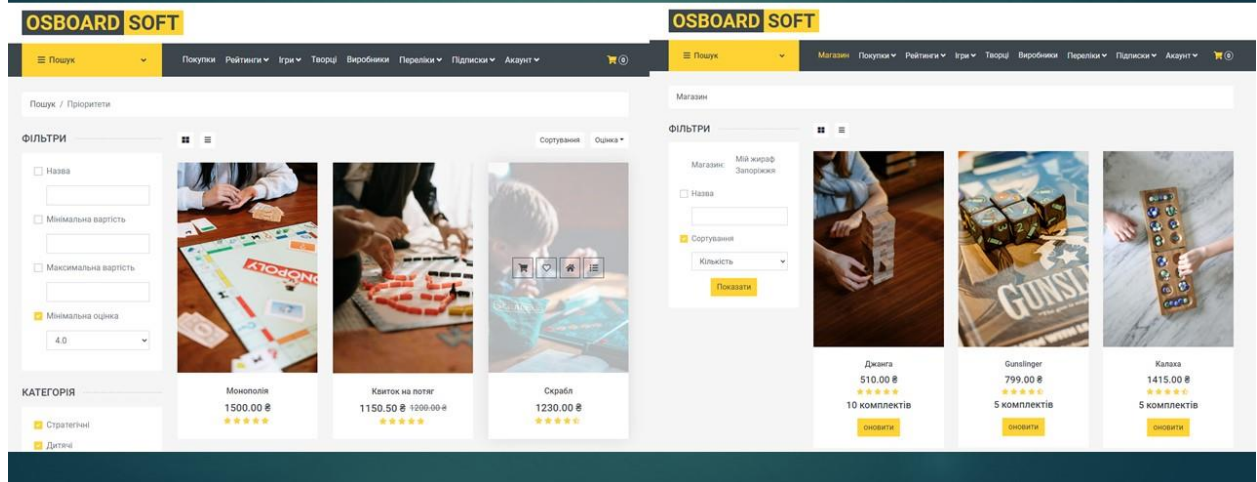


Рисунок В.11 – Пошук за пріоритетними для користувача настільними іграми і робота працівника магазину

Експериментальне дослідження

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,065886569	0,096531324	0,132287873
1	0,048799196	0,087153027	0,132287873
2	0,075442241	0,076436714	0,109538686
3	0,067813131	0,074353886	0,096531324
4	0,067974446	0,074353886	0,10294081

Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів
0	0,071920938	0,07484534	0,115361577
1	0,050638066	0,066987337	0,114421428
2	0,059104643	0,068812458	0,11177276
3	0,036908444	0,052463273	0,137773509
4	0,052551324	0,052563273	0,134594513
5	0,060222628	0,066987337	0,105945315
6	0,051162263	0,052107181	0,112372777
7	0,065179404	0,085678915	0,137773509
8	0,029300184	0,052107181	0,114144527
9	0,067382675	0,070016472	0,110308165

Рисунок В.12 – Експериментальне дослідження

Експериментальне дослідження				13
Мітка кластеру	Середнє значення суми квадратів відстані між екземплярами кластеру	Мінімальне середнє значення суми квадратів відстані між екземплярами різних кластерів	Максимальне середнє значення суми квадратів відстані між екземплярами різних кластерів	
0	0,052648993	0,052661214	0,138488246	
1	0,05590697	0,063471364	0,161322581	
2	0,035970093	0,041598166	0,155540739	
3	0,027885218	0,03670077	0,148527171	
4	0,061114668	0,066930872	0,17965986	
5	0,054782486	0,065479961	0,172585875	
6	0,052878246	0,0633039	0,167982377	
7	0,052517715	0,052572971	0,138762187	
8	0,030919672	0,041598166	0,164705914	
9	0,028917691	0,03670077	0,140442148	
10	0,047811976	0,079987831	0,220183786	
11	0,044822804	0,050952855	0,135671267	
12	0,029287606	0,0633039	0,159793861	
13	0,064203555	0,073767539	0,16901711	
14	0,056072974	0,064447212	0,166952642	
15	0,001391224	0,118060623	0,220183786	
16	0,072067751	0,079028766	0,160012289	
17	0,045352601	0,065562591	0,160072811	
18	0,044844145	0,046540288	0,154937034	
19	0,052921413	0,05415841	0,136621764	

Рисунок В.13 – Експериментальне дослідження

Висновки

- ▶ Розроблена програма пошуку настільних ігор призначена для надання інформації про настільні ігри, забезпечення можливості замовити настільні ігри в магазинах мережі, пошуку нових настільних ігор, в тому числі через надання пріоритетних настільних ігор щодо вибору, реєстрації власних настільних ігор у програмі, надання відгуків про власний досвід, ознайомлення з інформацією щодо досвіду інших користувачів.
- ▶ Наукова новизна роботи полягає в запропонованому методі пошуку настільних ігор, в якому визначається попередній набір даних, встановлюються персональні переваги користувача на основі відгуків про настільні ігри, виконується кластеризація даних на основі методу k-середніх, відбувається визначення пріоритетних настільних ігор для користувача та виконується відбір серед них за параметрами пошуку, встановленими користувачем.

Рисунок В.14 – Висновки