**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ**

**Інститут (факультет)** **автоматизації і комп'ютерних систем імені проф. І.В. Ельперіна**
**Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки**

| **«До захисту в ЕК»** | **«До захисту допущено»** |
|---|---|
| Директор інституту(декан факультету) | Завідувач кафедри |
| _____ Андрій ФОРСЮК | _____ Сергій ГРИБКОВ |
| (підпис) (ім'я та прізвище) | (підпис) (ім'я та прізвище) |
| «08» грудня 2025р. | «08» грудня 2025р. |

# КВАЛІФІКАЦІЙНА РОБОТА
# НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності 122 Комп'ютерні науки
(код та назва спеціальності)
освітньо-професійної програми Управління інформацією та аналітика даних
на тему: Інформаційна система аналізу ринку комп'ютерних ігор

Виконав: здобувач 2 курсу, групи КН-2-2М

Анісімов Олексій Юрійович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Харкянен Олена Валеріївна _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти _____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

*Я як здобувач Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав і не одержував недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

Здобувач _____
(підпис)

Київ — 2025р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) автоматизації і комп'ютерних систем імені проф. І.В. Ельперіна
Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь магістр
Спеціальність 122 Комп'ютерні науки
(код і назва)
**Освітньо-професійна програма** Управління інформацією та аналітика даних
(назва)

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри інформаційних технологій, штучного інтелекту і кібербезпеки**

_____ **Сергій ГРИБКОВ**
**«05» листопада 2025 року**

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Анісімова Олексія Юрійовича
**(прізвище, ім'я, по батькові)**

1. Тема роботи Інформаційна система аналізу ринку комп'ютерних ігор
керівник роботи Харкянен Олена Валеріївна, к.т.н., доцент,
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «05» листопада 2025 року №906-кс
2. Строк подання здобувачем роботи 1 грудня 2025 року
3. Вихідні дані до роботи Відкриті інформаційні джерела на тему відеоігор, набір даних про відеоігри з джерел відкритих даних, набір даних про комп'ютерні ігри з інтернет-магазину комп'ютерних ігор Steam
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1) Дослідження історії і тенденцій розвитку відеоігор
2) Дослідження впливу різних факторів на розробку відеоігор
3) Аналіз набору даних про відеоігри
4) Створення додатку для аналізу ринку відеоігор
5. Перелік графічного матеріалу
Кадри з історичних відеоігор, графічне представлення результатів аналізу даних схема бази даних, інтерфейс користувача розробленого додатку

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | Харкянен О.В., доцент НУХТ | 01.10.2025 | 06.10.2025 |
| 2 | Харкянен О.В., доцент НУХТ | 05.11.2025 | 13.11.2025 |
| 3 | Харкянен О.В., доцент НУХТ | 13.11.2025 | 22.11.2025 |
| 4 | Харкянен О.В., доцент НУХТ | 03.11.2025 | 25.11.2025 |
| | | | |
| | | | |

7. Дата видачі завдання: 01 жовтня 2025 року

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів виконання кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Дослідження та аналіз предметної сфери | 01.10.2025 – 06.10.2025 | Виконано |
| 2 | Дослідження історії і тенденцій розвитку відеоігор | 05.11.2025 – 13.11.2025 | Виконано |
| 3 | Дослідження впливу різних факторів на розробку відеоігор | 05.11.2025 – 13.11.2025 | Виконано |
| 4 | Аналіз набору даних про відеоігри | 13.11.2025 – 22.11.2025 | Виконано |
| 5 | Розробка системи | 03.11.2025 – 23.11.2025 | Виконано |
| 6 | Створення додатку для аналізу ринку відеоігор | 03.11.2025 – 24.11.2025 | Виконано |
| 7 | Оформлення кваліфікаційної роботи | 26.11.2025 – 29.11.2025 | Виконано |
| 8 | Оформлення презентації | 30.11.2025 – 01.12.2025 | Виконано |
| | | | |
| | | | |
| | | | |
| | | | |

**Здобувач** _____ ___Олексій АНІСІМОВ____
(підпис) (ім'я та прізвище)

**Керівник роботи** _____ ___Олена ХАРКЯНЕН_____
( підпис ) (ім'я та прізвище)

# АНОТАЦІЯ

Анісімов Олексій Юрійович — Інформаційна система аналізу ринку комп'ютерних ігор.

Метою кваліфікаційної роботи є дослідження історії появи відеоігор, впливів різних факторів на їх розвиток, а також дослідження впливу відеоігор на сфери культури і комп’ютерних технологій.

В роботі представлена розроблена інформаційна система для полегшення аналізу ринку комп’ютерних ігор.

Інформаційна система створена у вигляді додатку з використанням мови програмування Python, графічного фреймворку Qt і системи керування базами даних SQLite. Для інтеграції Qt в Python застосована бібліотека PySide, а для інтеграції SQLite в Python бібліотека sqlite3.

**Ключові слова:** ВІДЕОІГРИ, КОМП’ЮТЕРНІ ІГРИ, РИНОК КОМП’ЮТЕРНИХ ІГОР, РИНОК ВІДЕОІГОР, ІСТОРІЯ, АНАЛІТИЧНА СИСТЕМА

## SUMMARY

Anisimov Oleksii Yuriiovych — Information System for Computer Game Market Analysis.

The purpose of the graduate work is the research of the history of emergence of the video games, the influences of various factors on their development, as well as research of the influences of video games on the fields of culture and computer technologies.

The graduate work presents an information system developed for assistance in the computer game market analysis.

The information system is created in the form of an application using the Python programming language, the Qt graphical framework, and the SQLite database management system. Qt was integrated into Python using PySide library, and SQLite was integrated into Python using sqlite3 library.

**Keywords:** VIDEO GAMES, COMPUTER GAMES, COMPUTER GAME MARKET, VIDEO GAME MARKET, HISTORY, ANALYTIC SYSTEM

# ЗМІСТ

# ВСТУП

***Актуальність і практична значущість теми***.

На даний момент, сфера відеоігор визнана впливовою і важливою частиною культури і технологій. Ігрова індустрія як за популярністю так і за доходами може бути порівняна з індустріями музики та кіно, відеоігри використовуються у якості навчального засобу і симуляції, елементи ігрового дизайну інтегруються у неігрові контексти.

***Мета*** дослідження полягає у вивченні історії появи та розвитку сфери відеоігор, аналіз тенденцій впливу на неї різноманітних факторів навколишнього культурного середовища. Розробка інформаційної системи, здійснена в процесі дослідження, призначена для полегшення аналізу сучасного ринку комп'ютерних ігор і надання можливості формування рекомендацій для прийняття рішень розробникам і користувачам.

***Завдання дослідження.***

Для виконання поставленої мети були виконані наступні завдання:

- дослідження історії появи та тенденцій розвитку та сфери відеоігор;
- дослідження жанрів відеоігор, їх розвитку і класифікації;
- дослідження різноманітних впливів на розвиток відеоігор і культурне середовище навколо таких факторів як цільова платформа, її технічні характеристики і можливості, пристрої введення/виведення, носії даних, використовувані інструменти розробки і методи доставлення, економічні, культурні, історичні та інші чинники;
- аналіз набору даних з інформацією про відеоігри у середовищі Jupyter мовою Phyton з метою виявлення різноманітних тенденцій;
- створення інформаційної системи з метою аналізу тенденцій розвитку відеоігор і формування рекомендацій для прийняття управлінських рішень.

***Об'єкт дослідження*** – сфера відеоігор в цілому, і ринок комп'ютерних ігор зокрема.

***Предмет дослідження*** – сучасні методи аналізу даних для їх збереження, дослідження і візуального представлення користувачу у вигляді необхідному для прийняття рішень.

***Методи дослідження.***

У процесі дослідження були використані методи: аналізу літературних джерел для дослідження історії та тенденцій розвитку відеоігор, системний аналіз, теорія баз і сховищ даних, методи аналізу великих наборів даних.

***Наукова новизна.*** Досліджена тенденція розвитку класифікації комп'ютерних ігор за жанрами і показана можливість її доповнення за рахунок виявлення тенденцій у комбінуванні жанрів.

***Практичне значення одержаних результатів.***

Розроблена інформаційна система спрощує процес аналізу даних ринку комп'ютерних ігор, що може бути використане для виявлення тенденцій і виключень з них. Розуміння ринку комп'ютерних ігор є корисним для розробників відеоігор, гравців та дослідників, які здійснюють їх пошук за різноманітними параметрами.

***Особистий внесок здобувача.*** Усі основні положення і результати дисертаційної роботи, що захищаються, одержані автором самостійно.

***Апробація***. Наукові та практичні результати кваліфікаційної роботи доповідались та обговорювались на XII Міжнародній науково-технічній Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (м. Київ, Україна, 2025 р.) та Другій міжнародній науково-практичній конференції «Штучний інтелект та інформаційні технології» (Київ, Україна, 2025).

***Публікації.***

1. Анісімов О. Ю. Інформаційна система аналізу ринку комп'ютерних ігор / Анісімов О. Ю., Харкянен О.В. // XII Міжнародна науково-технічна Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення

систем керування організаційно-технічними та технологічними комплексами», 27 листопада 2025 р (Київ, Україна). К.: НУХТ, 2025.

2. Анісімов О. Ю. Опис ігор за допомогою Game description language / Анісімов О. Ю., Харкянен О. В. // Наукові праці Другої міжнар. наук.-практ. конф. «Штучний інтелект та інформаційні технології» (AIIT-2025), 3–4 червня 2025 р. (Київ, Україна). К.: НУХТ, 2025. 352 с.

3. Анісімов О. Ю. Шляхи опису складних поведінок у відеоіграх / Анісімов О. Ю., Харкянен О. В. // Наукові праці Другої міжнар. наук.-практ. конф. «Штучний інтелект та інформаційні технології» (AIIT-2025), 3–4 червня 2025 р. (Київ, Україна). К.: НУХТ, 2025. 352 с.

***Зв'язок роботи з науковими програмами, планами, темами кафедри, університету, іншої наукової установи.*** Кваліфікаційна робота виконувалась згідно з планом науково-дослідних робіт кафедри інформаційних технологій, штучного інтелекту і кібербезпеки Національного університету харчових технологій: № 0125U003889. Дослідження та впровадження сучасних методів аналізу даних у харчову промисловість.

**Кваліфікаційна робота** складається зі вступу, 4 розділів, висновків, 147 сторінок, 30 рисунків, 11 таблиць, 4 додатків, 88 джерел.

# РОЗДІЛ 1. ІСТОРІЯ І ТЕНДЕНЦІЯ РОЗВИТКУ ВІДЕОІГОР

## 1.1. Загальний огляд відеоігор і комп'ютерних ігор

Дещо складно визначити значення терміну "гра". Філософ Людвіг Вітгенштайн (Ludwig Wittgenstein) у своїй роботі, "Філософські дослідження" (Philosophische Untersuchungen), що була видана в 1953 році, аргументував, що існуючи ігри занадто різноманітні і не має єдиного визначення, котре могло б описати усі існуючи ігри. Замість цього, він пропонував розглядати ігри, як родину зі схожими рисами. Щось вважається грою, якщо воно подібне до інших ігор [1, с.31-34].

Історик і теоретик культури Йоган Гейзинга (Johan Huizinga) у книзі Homo Ludens визначав такі ознаки гри: добровільність; несправжність, гра не є частиною звичайного життя і учасники гри усвідомлюють її нереальність; обмеженість у місці і часі, гра не йде вічно і повинна у якийсь момент закінчитися; порядок, гра має деякі правила і учасники гри повинні визнавати та виконувати дані правила; іншість, під час гри правила життя поза грою призупиняються, і гра не має матеріальної вигоди [2, с.7-14]. Дане визначення є досить широким, і включаю у тому числі і явища, котрі традиційно не вважаються іграми, такі як ритуали і мистецтво [2, с. 10, 158-172].

Зазвичай ігри граються заради розваги, навчання або соціальних взаємодій.

Військові ігри з'явилися в Прусії у кінці 18-го сторіччя, і були створені з метою навчати офіцерів військовій справі і дослідженні тактик і стратегій. Ранні військові ігри були створені під сильним впливом шах, але були дуже сильно адаптовані під умови реального бою. Найвідомішою військовою грою є крігшпіль (Kriegspiel). Перша версія крігшпілю була розроблена Георгом Леопольдом фон Райсвіцом (George Leopold von Reisswitz) у 1811 році, і була позитивно сприйнята пруською королівською родиною. Незважаючи на це, Георг Леопольд зупинив розробку гри у 1816 році, але його син, Георг Генріх Рудольф Йоханн фон Райсвіц (Georg Heinrich Rudolf Johann von Reisswitz) продовжив розробку гри, і в 1824 році вона була схвалена як навчальний засіб для військових. Хоча крігшпіль був

створений для навчання військових, він також набув популярності і як форма розваги. Крігшпіль був однією з найвпливовіших ігор, і все ще існують спільноти гравців у крігшпіль. Військові і цивільні ігри нерідко впливали одні одних, і дизайнери військових ігор нерідко також розробляли цивільні ігри. Прикладом впливу військових ігор на цивільні є одна з найвпливовіших ігор, Dungeons & Dragons, котра розпочалася як варіація з фантастичними елементами минулої гри її авторів, Chainmail, котра, у свою чергу, була грою, що симулювала середньовічні бої і мала впливи від інших військових ігор [3, 4].

Подібно до складності з визначенням ігор загалом, відеоігри і комп'ютерні ігри також складно визначити. Можливо, відеогру можна визначити як щось, що водночас є грою і програмою для обчислювальної машини.

В деяких мовах "комп'ютерна гра" нерідко використовується як загальний термін для всіх відеоігор, в той час як у деяких під комп'ютерними іграми мають на увазі саме ігри для персональних комп'ютерів. Так, наприклад, "комп'ютерна гра" нерідко використовується як загальний термін для всіх відеоігор у таких мовах, як британська англійська (computer game), німецька (Computerspiel), польська (gra komputerowa), тощо. Окремо можна відокремити також японську мову, де основний загальний термін для відеоігор — це "コンピュータゲーム" (Konpyu-ta Ge-mu), в той час як для ігор для персональних комп'ютерів використовується термін "パソコンゲーム" (Pasokon Ge-mu). В той же час, у багатьох з перерахованих мов також використовується термін "відеогра", навіть якщо він є менш популярним.

### 1.2. Виникнення відеоігор

Історія відеоігор тісно переплетена з історією розвитку програмування і комп'ютерів.

У 1947 році, Томас Толівер Голдсміт молодший (Thomas Toliver Goldsmith Jr.) і Естл Рей Манн (Estle Ray Mann), фізики, що працювали на посадах

розробників в DuMont Laboratories, що займалася розробленням телевізорів, подали заявку на отримання патенту на "Розважальний пристрій на базі електронно-променевої трубки" (Cathode-ray tube amusement device), що був зареєстрованим у 1948 році. Даний пристрій симулює і візуалізує артилерійський постріл снаряду, що вибухає через деякий час. Гравець може налаштовувати траєкторію і час вибуху. На екран фізично накладаються зображення мішені, і ціль гравця — попасти по мішені. Дані пристрої ніколи не вироблялися і не рекламувалися, і не відомо, чи був створений прототип чи розробка була чисто теоретичною. Наскільки відомо, даний пристрій не мав впливу на майбутні розробки. Нема консенсусу, чи є даний пристрій відеогрою чи ні. Тим не менш, можна помітити схожі риси з деякими ранніми відеоіграми і пристроями для них [5-7].

Більшість того, що можна класифікувати як самі ранні комп'ютерні ігри, були спроби написати програми, що здатні грати у вже існуючи ігри. Вони зазвичай створювалися як дослідницькі проекти або як демонстрація потужностей комп'ютера [8].

У 1940 році, ядерний фізик Едвард Кондон (Edward Condon) розробив і запатентував електромеханічну машину Німатрон (Nimatron), що вміє грати в нім. Німатрон був створений для участі у Всесвітньої виставці 1939 року в Нью-Йоркі (1939 New York's World Fair), що тривала до 1940 року. Німатрон вплинув на майбутні спроби створити програми, що здатні грати в нім [9].

У 1948 році Алан Тюрінг (Alan Turing) і Девід Чампернаун (David Champernowne) розробили алгоритм для гри в шахи Turochamp. Turochamp відомий як одна з найперших спроб створити програму для гри в шахи. Алгоритм тестувався вручну, через складності алгоритму у Тюрінга не вийшло імплементувати його для комп'ютерів того часу [10].

Також у 1947-1948 Дональд Мікі (Donald Michie) і Шон Вайлі (Shaun Wylie) розробили алгоритм Machiavelli для гри в шахи. Планувалося змагання між Machiavelli і Turochamp, і Тюрінг починав імплементувати обидва алгоритмі, але не встиг закінчити до своєї смерті у 1954 році [11].

У 1951 році Дітріх Прінц (Dietrich Prinz) розробив програму, що здатна вирішувати шахові задачі, де потрібно поставити мат за два хода. Ця програма є першою відомою імплементованою шаховою програмою. При цьому, вона не здатна грати у повноцінну партію шах і не знає більш специфічних ходів, таких як рокірування [12].

У 1950 році, на Канадській національній виставці був представлений комп'ютерний апарат для гри у хрестики-нулики Bertie the Brain, розроблений Джозефом Кейтсом (Joseph Kates) (при народженні Йозеф Кац (Josef Katz)). Bertie the Brain був здатним на ідеальну гру у хрестики-нулики і мав налаштування складності гри [8].

У 1952 році Александер Шафто "Сенді" Дуглас (Alexander Shafto "Sando" Douglas) під час здобуття ступеню доктору математики в Кембриджському університеті (University of Cambridge) як частину свого дипломного проекту на тему людино-машинних взаємодій створив гру OXO, комп'ютерну версію хрестиків-нуликів з графічним зображенням стану гри. До того, зазвичай у комп'ютерних іграх графічна складова була відсутня, і OXO нерідко вважається першою комп'ютерною грою, що має візуальну складову. [8].

У 1952 році, Крістофер Стрейчі (Christopher Strachy) створив програму, що здатна грати у шашки. Програма для гри в шашки Стрейчі надихнула Артура Самуеля (Arthur Samuel) на створення своєї програми для гри в шашки. Програма Самуеля була першою відомою програмою, де використовувалося машинне навчання [8].

У 1955 році, Operations Research Office розробили військову гру Hutspiel для аналогового комп'ютера Goodyear Electronic Differential Analyzer (GEDA) для допомоги американській армії у дослідженні використання тактичної ядерної зброї і авіаційної підтримки у Західній Європі на випадок радянського вторгнення. У Hutspiel приймають участь два гравця: один грає за BLUE (базуються на силах НАТО), інший за RED (базується на радянських силах). Оригінальна версія симулювала події у режимі реального часу, якщо один з

гравців не поставив її на паузу для розробки стратегій та віддачі команд. Пізніші версії є покроковими, де один крок представляє деякий часовий інтервал [8, 13].

Були також створені ігри для симуляції інших процесів, окрім прямих бойових дій. Наприклад, RAND Corporation розробила настільну гру Monopologs, що симулює систему постачання Повітряних сил, і має метою навчити гравця логістиці [8].

У 1957 році American Management Association створила комп'ютерну гру для тренування менеджерів The Top Management Decision Simulation [8, 14].

У 1958 році в Технологічному інституті Карнегі (Carnegie Institute of Technologies) командою, що складалася з Калмана Коена (Kalman Cohen), Річарда Сайерта (Richard Cyert) і Вільяма Діла (William Dill), була розроблена програма The Management Game для тренування менеджерів [8].

У 1952 році Willow Run побудували два комп'ютера, MIDAC (Michigan Digital Automatic Computer) і MIDSAC (Michigan Digital Special Automatic Computer). У 1954, ці комп'ютери були вперше представлені публіці, і для демонстрації їх можливостей були розроблені декілька ігор. Для демонстрації можливостей MIDSAC, Вільям Джордж Браун (William George Brown) і Тед Льюїс (Ted Lewis) створили віртуальний більярд, де гравець може управляти кієм за допомогою джойстика, і графічне відображення стану гри оновлювалося у режимі реального часу. До того, якщо гра і мала графічну складову, то вона зазвичай була достатньо статичною, в той час як віртуальний більярд для MIDSAC мав неперервно оновлювався і створював ілюзію постійного руху [8, 15].

У 1958 році, Вільям Гігінботам (William Higinbotham) і Роберт Дворак (Robert Dvorak) в Брукгейвенській національній лабораторії (Brookhaven National Laboratory) створили гру Tennis for Two. Вільям Гігінботам був частиною Мангеттенского проекту, і займався в Лос Аламосе розробкою електронних компонентів першої атомної бомби. Побачивши руйнівний потенціал атомної бомби, Гігінботам став ярим прихильником ядерного нерозповсюдження і роззброєння, а також мирного використання атомної енергії. Після війни,

Гігінботам став головою відділу приладобудування Брукгейвенської національної лабораторії. Брукгейвенська національна лабораторія займалася ядерною енергетикою, і для покращення відношень з громадськістю, почала проводити щорічно проводити екскурсії по лабораторії для громадян, багато з котрих були школярами та студентами. Для дня відвідувача 1958 року, Гігінботам створив відеогру Tennis for Two. Гра мала двох гравців, котрі контролюють грою за допомогою контролерів з круглою ручкою і кнопкою. Ручкою можна було налаштовувати напрям і швидкість, а кнопкою можна використовувати, щоб вдаряти по м'ячу. Поведінка тенісного м'яча базувалася на алгоритмах для розрахування траєкторій балістичних ракет. Гра візуалізувалася за допомогою осцилографа, де тенісний корт зображується збоку. В 1959 році, гра була дещо оновлена, була добавлена можливість змінювати гравітацію, симулюючи гру в теніс на інших планетах. Після дня відвідувача 1959 року, Tennis for Two був розібраний на запчастини, котрі були використані для інших проектів. Невідомо, наскільки впливовим був Tennis for Two, і у свій час він мав обмежену популярність. На протязі достатньо довгого часу Tennis for Two вважався першою відеогрою. Гігінботам особисто не вважав, що Tennis for Two був чимось особливим, адже він базувався на вже існуючих і достатньо стандартних розробках, і не був зацікавленим у тому, щоб його пам'ятали за створення відеогри, замість цього він віддавав перевагу тому, щоб його пам'ятали за його роботу у сфері нерозповсюдження ядерної зброї [16, 17].

У 1956 році, в MIT (Massachusetts Institute of Technology) в Лабораторії Лінкольна (Lincoln Laboratory) був створений комп'ютер TX-0 (Transistorized Experimental computer zero), що був одним з перших комп'ютерів на транзисторах. TX-0 базувався на ламповому комп'ютері Whirlwind I, але, коли Whirlwind I займав цілий поверх (приблизно 190 квадратних метрів), TX-0 поміщався в одній кімнаті, і, не зважаючи на таку різницю в розмірі, був швидше за Whirlwind I. Створенням TX-0 керували Веслі Алісон Клар (Wesley Allison Clark) і Кен Олсен (Ken Olsen). Після створення комп'ютера TX-2, TX-0 був перенесеним до Кембриджського кампусу MIT, до Електротехнічного факультету (Electrical

Engineering Department). Там TX-0 спільно підтримувався Науково-дослідницькою лабораторією електроніки (Research Laboratory of Electronics) і Лабораторією електронних систем (Electronic Systems Laboratory). Там надавався прямий доступ до TX-0 дослідникам і студентам [18, 19, 20].

Зручність і доступність роботи з TX-0 створило культуру експериментаторства та відкритості серед студентів MIT. Прикладом цієї культури є студентська організація Tech Model Railroad Club (TMRC) [21, 22].

У 1957 році, Кен Олсен і Харлан Андерсон (Harlan Anderson) заснували DEC (Digital Equipment Corporation). В 1959 році, DEC випустили комп'ютер PDP-1 (Programmed Data Processor-1), комерційний комп'ютер, заснований на TX-0 і TX-2. В 1961 році, DEC подарували PDP-1 MIT, де ентузіасти переключили свою увагу з TX-0 на PDP-1 [22, 23].

В даному контексті, у 1962 році для PDP-1 вийшла гра Spacewar! (рис. 1.1). Гра була розроблена Стівеном "Слагом" Расселом (Stephen "Slug" Russell), Мартіном Грецом (Martin Graetz) і Уейном Війтаненом (Wayne Wiitanen). У цій грі є два кораблі, Голка (the needle), що має голкоподібну форму і чий дизайн базується на реальній ракеті Redstone, і Клин (the wedge), що має трохи округлену форму і чий дизайн базується на кораблі з коміксів про Бака Роджерса (Buck Rogers). Ця гра розрахована на двох гравців і відбувається у режимі реального часу. Серед доступних дій у гравця є можливість повернути за годинниковою стрілкою, повернути проти годинникової стрілки, стріляти і рухатися вперед. У гравців є обмежена кількість снарядів і палива. Якщо снаряди закінчилися, то гравець не здатний стріляти. Якщо паливо закінчилося, то гравець не може рухатися вперед. Рухання вперед споживає паливо, в той час як повороти не споживають паливо. Якщо корабель виходить за межі екрану, то той з'являється з іншої сторони екрану. Увесь доступний простір покриває гравітаційна зона зірки, що знаходиться посередині екрану, і гравітація тягне кораблі до зірки. Гравітація не впливає на снаряди. Гра була натхнена серії науково-фантастичних книг Lensman і Skylark за авторством Едварда Елмера "Дока" Сміта (Edward Elmer "Doc" Smith), а саме тенденціями злодіїв переслідувати героїв, котрим потрібно

викручуватися з подібних проблем, і описами космічних подій. Як оригінальні розробники так і інші ентузіасти постійно модифікували і дороблювали цю гру. Наприклад, оригінально не планувалося, що дія буде відбуватися у гравітаційній зоні зірки, і у перших версіях це було відсутнє, але Ден Едвардс (Dan Edwards) і Грец потім її добавили, що стало важливою частиною гри. Spacewar! була першою комп'ютерною грою, що широко поширювалася, в той час як комп'ютерні ігри до того зазвичай не поширювалися далеко від своїх творців. У Spacewar! був відкритий код, що призвело до великої кількості модифікацій, копіювань, копіювань з модифікаціями, портування під інші системи, створення ігор, натхнених Spacewar!, тощо [22, 24].

Рисунок 1.1 — Spacewar!

Багато в чому Spacewar! сприймається як поворотний момент у історії відеоігор. Це одна з перших комп'ютерних ігор, що набула широкого поширення. Це одна з перших комп'ютерних ігор, котра не є адаптацією існуючої традиційної гри, і не є тренінгом для керівників. Ігровий дизайн є оригінальним і практично неможливим для реалізації у формі традиційної гри. Основним натхненням цієї гри є не інші ігри, а книги, почуття від читання деяких сцен з них вона намагається відтворити. В той час як раніше згадані ігри нерідко ставали недоступними і забувалися, у Spacewar! грали ще десятиріччями. Також десятиріччями її клонували і модифікували. Найвідоміший набір правил не був запланований до початку розробки, а з'явився в процесі розробці через те, що у деяких людей були ідеї, як можна покращити цю гру.

Скоріше за все, у таких відмінностях є декілька причин. Однією з них є технічний прогрес. З більш зручними і доступними комп'ютерами, розробка програм стає простішою, що дозволяє розробляти більше програм, котрі не мають практичного призначення, в той час як порівняно низька ціна дозволила більшій кількості навчальних закладів закупляти комп'ютери, що, у свою чергу, відкриває доступ до ще більшої кількості дослідників і студентів. Іншою причиною може бути те, що розробники Spacewar! народилися пізніше за багатьох розробників раніше згаданих ігор. Якщо розробники минулого покоління в основному були свідками і нерідко учасниками зародження програмування, то розробники Spacewar! виросли у світі, де програмування вже існує, через що у них могла сформуватися дещо інша перспектива. Ще однією причиною може бути специфічна культура що породила Spacewar!, що, у свою чергу, могла бути сформована їх досвідом буття молодими програмістами. Культура відкриття і експериментаторства призвела до того, що розробка Spacewar! була більш колективним і хаотичним процесом, аніж розробки минулих ігор.

У 1962 році, рада кооперативних освітніх служб (Board of Cooperative Educational Services) штату Нью-Йорк почала переговори з дослідниками з IBM щодо використання комп'ютерів у навчанні для допомоги сільським школам. У

1963 році цей проект розпочався, і у ході переговорів та під впливом статті Річарда Л. Меєра (Richard L. Meier) "Навчання за допомогою участі у мікросимуляціях соціальних організацій" (Teaching through Participation in Micro-simulations of Social Organization), настільної гри Монополія і недостатністю представлення догрецьких цивілізацій у шкільний програмі вирішили створити модель шумерської цивілізації для навчання базовим навичкам економіки у вигляді гри The Sumerian Game, що була створена в 1964 році. За дизайн та сценарій відповідала вчителька Мейбл Аддіс (Mabel Addis), за програмування відповідав програміст Вільям Маккей (William McKay). Гра була написана мовою Fortran. Гру можна назвати текстовою стратегією, уся інформація представляється гравцю у вигляді тексту і гравець управляє грою шляхом набором тексту. Гравцю потрібно займатися управлінням ресурсами і реагувати на події. Гравець приймає на себе роль короля-священника міста Лагаш (Lagash), але в процесі гри змінюється, якого саме: гравець починає як Лудуга I (Luduga I), потім Лудуга II, потім Лудуга III. Перед початком гри, планувалося, щоб викладачі показували школярам спеціально підготовлену презентацію. The Sumerian Game є першою відомою навчальною грою розрахованою на дітей як цільову аудиторію і першою відомою грою з історією. Гра була впливовою і мала декілька імітацій, таких як King of Sumeria від Дага Даймента (Doug Dyment), Hammurabi від Девіда Г. Ала (David H. Ahl), тощо. Оригінальна версія The Sumerian Game була загублена, але на базі відомої інформації, такої як опис у звіті щодо результатів дослідження про ефективність цієї гри як навчального засобу, була створена відбудована версія, котра на момент написання цього тексту присутня на платформі Steam і є безкоштовною [25-27].

У 1951 році, працівнику військового контрактора Lorel Electronics, Ральфу Генрі Баеру (Ralph Henry Baer), при народженні Рудольф Генріх Баер (Rudolph Heinrich Baer), що працював над телевізорами, прийшла ідея добавити інтерактивності до телевізору. Lorel Electronics не були зацікавлені у даній ідеї. В 1958 році, Ральф Баер приєднався до компанії Sanders Associates, що також була військовим контрактором. В 1966 році, Баер був підвищений до керівника відділу

проектування обладнання. Також в 1966 році Баер згадав про свою ідею інтерактивного телевізора, і прийшов до ідеї, що можна створити ігровий пристрій, котрий буде передавати сигнал до телевізора і показувати гру. До розробок Баера, у якості екрана для відеоігор зазвичай використовувалися електронно-променеві трубки або осцилографи, в той час як текст зазвичай друкувався. У тому ж році Баер розробив перший прототип і представив його корпоративному директору з досліджень і розробки компанії Sanders Associates, Герберту Кампману (Herbert Campman). Той схвалив проект. В 1967 році, прототип пристрою вже був достатньо повноцінний, що Кампман погодився, що цей продукт вже має достатній потенціал для продажі. Після цього почався пошук партнерів. Sanders Associates був військовим контрактором, і не мав зв'язків та досвіду щодо продажі споживчих продуктів, тому було вирішено ліцензувати пристрій інший компанії. В 1969 році, був створений фінальний прототип пристрою, так звана "Коричнева коробка" (Brown Box). Даний пристрій був здатний зображати до трьох плям на телевізорі, і графічні обмеження компенсувалися за допомогою фізично накладених на телевізор накладень. У пристрій було вбудована достатньо велика кількість ігор, і пристрій підтримував три контролера: світловий пістолет, джойстики і пульт з трьома ручками. В 1969 році, прототип був представлений перед представниками компанії Magnavox, і 1971 році був заключений ліцензійний договір між Sanders Associates і Magnavox. Після цього, розробка пристрою проводилася командою інженерів Magnavox. В процесі вони прибрали деякі ігри, і замість переключення між іграми за допомогою перемикачів або диску впровадили систему ігрових карток, що являють з себе друковані плати, що вказують, які комбінації поведінок повинні виконуватися. Пристрою була дана назва Magnavox Odyssey, і його випустили у продаж у 1972 році. З усіх контролерів, оригінально був включеним лише пульт з трьома ручками, хоча потім також почали продавати світловий пістолет. В 1975 році Magnavox зупинив виробництво Magnavox Odyssey. Magnavox Odyssey вважається першою ігровою консоллю, і ігри для Magnavox Odyssey були одними з перших комерційних відеоігор. Гра Настільний теніс (Table Tennis), єдина гра

що не потребувала фізичної накладки на екран, була достатньо впливова на наступні адаптації настільного тенісу у відеоігрову форму [28, 29].

Аркадні ігри, тобто ігрові автомати, котрі працюють після вкидання грошей, з'явилися в кінці 19го сторіччя. Ранніми прикладами аркадних ігор є тести на сили та слотові автомати. В 1930х пінбол стає популярною формою аркадних ігор [30, 31].

У 1971 році, компанія Nutting Associates випустила аркадну гру Computer Space, розроблену Ноланом Бушнелом (Nolan Bushnell) і Тедом Дабні (Ted Dabney). Ця гра є першою відомою аркадною відеогрою. Computer Space був випущений до Magnavox Odyssey, через це Computer Space вважається першою комерційною грою. Computer Space був створений під сильним впливом Spacewar!. В Computer Space є ракета, що управляється гравцем, і дві летючі тарілки, що управляються програмою. Ракета і летючі тарілки можуть стріляти одні в одних, і обидві сторони набирають очко за кожний раз коли вони збивають противника. Раунд триває 90 секунд, якщо у гравця більше очок, аніж летючі тарілки, то гравцю дається ще 90 секунд, інакше гра закінчується. Хоча, наскільки відомо, Computer Space не був повним провалом, але також не виправдав очікування Nutting Associates [32].

У 1972 році, Нолан Бушнел і Тед Дабні заснували компанію Atari. Був найнятий Аллан Алкорн (Allan Alcorn). Як тестове завдання, йому доручили зробити клон гри Настільний теніс для Magnavox Odyssey. Таким чином Алкорн створив аркадну гру Pong. Прототип аркадного кабінету був поставлений у бар, де став достатньо популярним. Atari почали виробництво і продаж аркадних кабінетів. Pong вважається першою популярною аркадною відеогрою, і Pong підштовхнув інших виробників аркадних ігор також спробувати створення відеоігор [33].

У 1970х, популярність аркадних відеоігор зростала. В 1978, вийшла інша відома і впливова аркадна відеогра, Space Invaders (рис. 1.2). Space Invaders були розроблені компанією Taito, дизайнером і програмістом був Нішікадо Томохіро

(西角 友宏, Nishikado Tomohiro). Space Invaders були однією з ранніх аркадних відеоігор, що використовували центральні процесори, і одна з перших відомих японських аркадних відеоігор, що використовували центральні процесори. Space Invaders були натхнена грою від Atari, Breakout (1976). Гравець контролює гармату, що знаходиться внизу екрану, і може рухати її вправо-вліво, а також стріляти з неї. У верхній половині екрану знаходяться прибульці, котрі рухаються вправо-вліво, повільно рухаються вниз, і час від часу стріляють у бік гравця. У нижній половині екрану, над гравцем, є декілька укріплень. Якщо постріл потрапляє по укріпленню, то постріл зникає а укріплення отримує пошкодження. Як гравець, так і прибульці можуть шкодити укріплення. За знищення прибульців надаються бали, різна кількість за різни типи. У гравця є декілька життів, і той губить їх, якщо постріл прибульця влучає в гравця. За кожні 1500 очок гравець отримує бонусне життя. Гра закінчується, якщо у гравця закінчуються життя, або якщо прибульці доходять до низу екрану. Чим менше прибульців, тим швидше вони рухаються. Спочатку це було спричинено тим, що, чим більше прибульців, тим більше потрібно їх промальовувати, що потребує час, через що їх рухи сповільнені. Тим не менш, це стало важливою частиною ігрового процесу Space Invaders, і вважається першим прикладом динамічної зміни складності гри у процесі гри. При знищенні усіх прибульців, починається наступна хвиля прибульців [34-36].


Рисунок 1.2 — Space Invaders

Space Invaders були дуже популярними і впливовими, і допомогли популяризувати відеоігри. Вони надихнули багатьох людей займатися розробкою ігор. Шігеру Міямото (宮本 茂, Miyamoto Shigeru), відомий за серії ігор Mario і the Legend of Zelda, казав, що, до того, як він побачив Space Invaders, він не був зацікавлений у відеоіграх [37]. Близнюки Олівер, Ендрю Олівер (Andrew Oliver) і Філіп Олівер (Philip Oliver), відомі за серію ігор Dizzy, казали, що в дитинстві були сильно зацікавлені іграми типу Space Invaders і Pac-Man, і намагалися відтворити подібні ігри [38-39].

У 1960-х і 1970-х, розробка комп'ютерних ігор ставала все більш доступною, особливо для студентів і ентузіастів. Поява і розповсюдження мов програмування високого рівня, типу FORTRAN, BASIC, Forth і C спростила процес програмування, навіть якщо в той час використання мов програмування високого рівня призводила до менш оптимізованих програм аніж якби використовувалася мова асемблеру. Поява поділу часу, розподілу обчислювальних ресурсів між декількома користувачами, дозволила декільком користувачам одночасно користуватися одним комп'ютером, і, як результат, зробила університетські комп'ютери більш доступними для студентів. Розвиток

комп’ютерних мереж сприяв розповсюдженню програм між університетами і навіть дозволив створення перших мережевих багатокористувацьких ігор.

У 1971 році, була випущена перша версія гри Star Trek, неофіційної адаптації однойменного шоу, що була написана Майком Мейфілдом (Mike Mayfield) мовою Basic для комп’ютера Sigma 7. Мейфілд використовував комп’ютер, що знаходився в Університеті Каліфорнії в Ірвайні (University of California, Irvine). Мейфілд на момент написання гри не був дослідником або студентом, а все ще навчався у школі, і його доступ до університетського комп’ютера був недозволеним. Мейфілд бажав створити гру, подібну до Spacewar!, але не мав доступу до відеотерміналу, а лише до телепрінтера. Мейфілд і його друзі обговорювали ідеї для гри, що не потребує відеотерміналу, котрі Мейфілд потім намагався реалізовувати. Потім, тим же літом, незадовго після закінчення школи, Мейфілд купив програмований калькулятор HP-35, створений компанією HP (Hewlett-Packard). Він приходив декілька разів до місцевого офісу HP по допомогу з програмуванням і йому дозволили користуватись офісним комп’ютером, якщо той портує Star Trek для їх комп’ютера. Мейфілд переписав цю гру на діалекті Basic, що використовувався на комп’ютерах HP, і програма була добавлена у публічну бібліотеку HP. Ця версія є найдавнішою збереженою версією цієї гри, адже версія для Sigma 7 була загублена. На базі цієї гри було створена велика кількість клонів і портів для великої кількості систем, найвідомішим з котрих є розширена версія Super Star Trek, розроблена Бобом Лідомом (Bob Leedom) для комп’ютера Data General Nova. Гравець грає за космічний корабель USS Enterprise, і ціллю гри є знищення усіх кораблів прибульців клінгонів (Klingon) за обмежений проміжок часу. Галактика розділена на сітку квадрантів 8 на 8, кожний з яких, у свою чергу, поділений на сітку секторів 8 на 8. Кожна гра починається з випадковою кількістю клінгонів, зірок і дружніх баз. Гра запам’ятовує кількість клінгонів, зірок і дружніх баз у квадранті, але їх позиції змінюються кожний раз при вході в квадрант. Гравець контролює гру за допомогою вводу текстових команд. Можливий виклик мапи квадранту, в котрому знаходишся, котра буде побудована

з текстових елементів. У гравця є два типи зброї: фазери, котрі не потрібно цілити, але їх ефективність зменшується з відстанню до противника, і потрібно вказувати, скільки енергії вкладаєте у фазерний постріл; і торпеди, чия ефективність не зменшується з відстанню до противника, але їх потрібно цілити, використовуючи полярні координати. Гра є покроковою. Чим далі гравець рухається, тим більше часу відіймається. Рух і атаки споживають енергію, і для її відновлення потрібно відвідувати дружні бази. Гра закінчується, якщо USS Enterprise знищено, якщо усіх клінгонів знищено, або якщо витікає час [40, 41].

У 1971 році, Волтер Брайт (Walter Bright), у віці приблизно 12 років, розробив настільну стратегічну гру Empire, натхнену настільною грою RISK і фільмом "Битва за Британію" (Battle for Britain). На думку Брайта, його настільна гра була провальною, адже гра вимагала відстеження величезної кількості даних та другої людини, що готова присвятити декілька годин на буття опонентом. У школі, Брайт вивчив мову програмування Basic. Брайт також був зацікавлений у грі Hamurabi (клон the Sumerian Game), і займався модифікацією Hamurabi. Брайт поступив у Каліфорнійський технологічний інститут (California Institute of Technology), що нерідко скорочується як Калтех (Caltech). З часом, Брайту набридло продовжувати модифікувати Hamurabi, і він згадав про свою настільну гру. Зрозумівши, що створення комп’ютерної версії вирішить обидві проблеми, він прийнявся за імплементацію. Розробка проводилася на PDP-10, і Брайт починав, використовуючи мову Basic, але в процесі розробки прийшов до висновку, що дану гру не вийде імплементувати за допомогою Basic, і переписав мовою FORTRAN. Розробка над цією версією гри була закінчена у 1977 році. У 1983 році, Брайт портував Empire на персональний комп’ютер Heathkit H11, переписавши на мову асемблера PDP-11, анонсував у журналі BYTE, але через малу популярність платформи у нього вийшло продати лише дві копії. Потім, він переписав ще раз, мовою C для IBM PC, і також анонсував у журналі BYTE, на цей раз отримавши величезну кількість замовлень. Натхнений успіхом, Брайт вирішив знайти видавника, але лише компанія Intelsoft була зацікавлена. Intelsoft найняла програміста Марка Балдвіна (Mark Baldwin), щоб той добавив графічний

інтерфейс, і випустила у 1987 році під назвою Empire: Wargame of the Century. У всіх версіях гри головне, що показується гравцю — це мапа (рис. 1.3). У версіях гри, розроблених Брайтом, мапа будується з текстових символів (версія для IBM PC також використовує різні кольори для різних елементів мапи, як для текстового символу так і для фону), в той час як версії, розроблені без прямої участі Брайта, замість текстових символів використовуються тайлова графіка. Показується лише та територія, де сили гравця (військові одиниці або міста) вже були, в той час як територія, де сили гравця ще не були покриті туманом війни і заповнені чорним кольором. Ландшафт і розташування сил сторін генерується випадково на початку гри. Обидві сторони починають гру, маючи під контролем одне місто. Гра є покроковою. Місто може бути назначеним на виробництво військових одиниць, таких як армії (armies), літаки-винищувачі (fighters), есмінці (destroyers), крейсери (cruisers), лінкори (battleships), підводні човна (submarines), транспортери військ (troop transporters), авіаносці (aircraft carriers). Виробництво різних військових одиниць займає різну кількість кроків. Гравець може рухати свої військові одиниці, і можуть знаходити інші міста, котрі на початку гри є незалежними. Армії здатні завойовувати міста, завойовані міста можна також назначити на виробництво військових одиниць. Гра закінчується, якщо одна сторона завойовує усі міста [42-44].

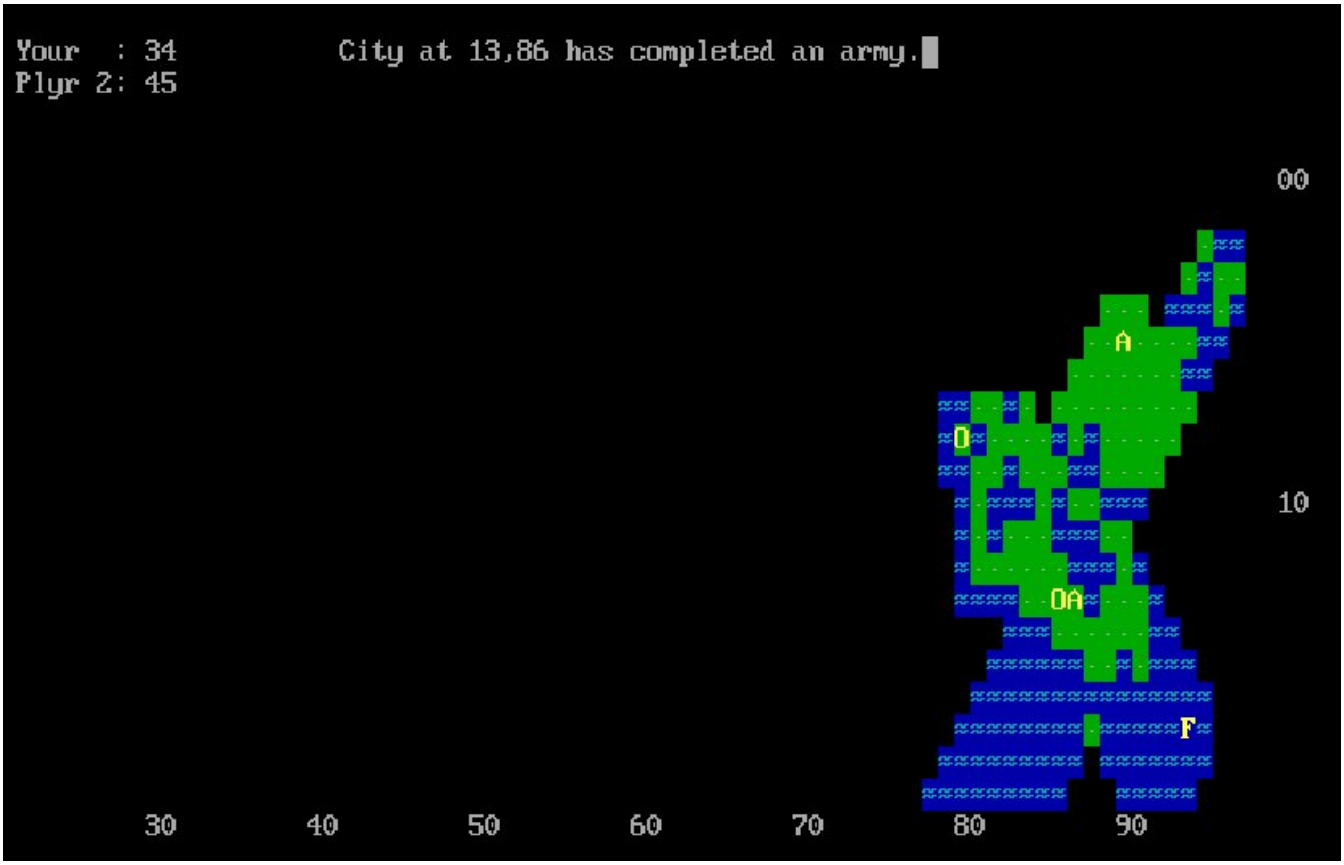



Рисунок 1.3 — Empire для IBM PC (1985 року)

Empire була ранньою стратегічною комп'ютерною грою, і мала достатньо сильний вплив на інші комп'ютерні стратегії. Сід Меєр (Sid Meier) визнавав Empire, разом з SimCity і Populous, як найбільші впливи на свою гру Sid Meier's Civilization 1991 року (перша гра з однойменної серії ігор). Згідно Брюсу Шеллі (Bruce Shelley), котрий приймав участь у розробці Sid Meier's Civilization, частиною процесу дизайну Civilization було складання 10 змін, котрі могли б покращити Empire [45].

У 1970-х, Сілас Уорнер (Silas Warner) розробив гру RobotWar для комп'ютеру PLATO. У 1981, Уорнер портував цю гру на персональний комп'ютер Apple II і вона була видана компанією Muse Software (рис. 1.4). Гра складається з програмування бойових роботів і тестування даних роботів на полі бою. Немає розділення між своїми і чужими роботами, і усі роботи сприймають один одного як ворогів. Виграє останній незнищенний робот. Роботи програмуються простою мовою програмування, що може нагадувати BASIC або FORTH, що компілюється

в об’єктний код. На диску гри простір для збереження програм роботів обмежений, але можливо зберегти як source code так і скомпільовані програми на окремі диски. RobotWar є однією з перших відомих комп’ютерних ігор, що використовують програмування як частину ігрового дизайну і є першим відомим комерційним представником подібних ігор. RobotWar мала декілька імітаторів і породила нішевий жанр відеоігор [46-48].

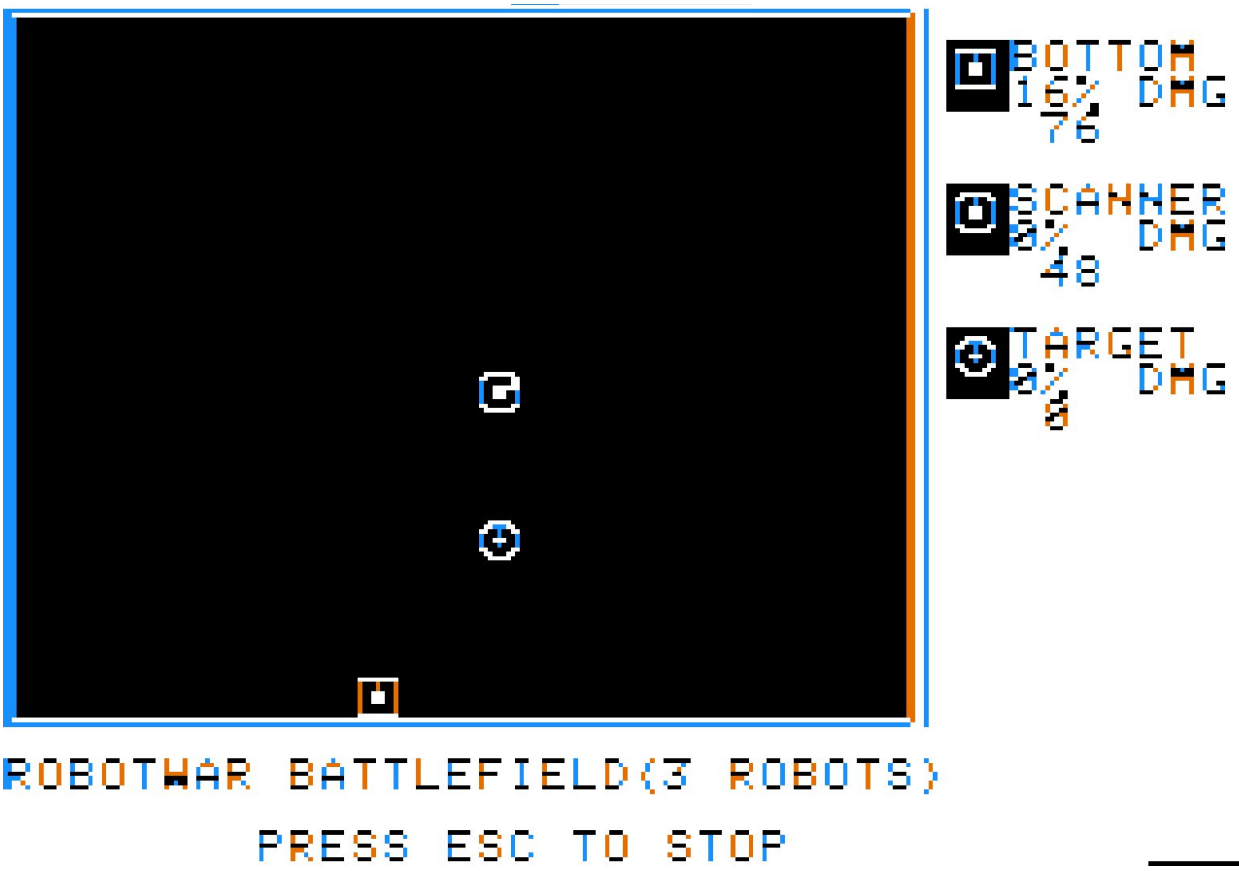



Рисунок 1.4 — RobotWar для Apple II

Настільна рольова гра, Dungeons & Dragons, була дуже популярною серед студентів та комп’ютерних ентузіастів. Тож не дивно, що велика кількість студентів і ентузіастів намагалися відтворити Dungeons & Dragons для комп’ютерів. Однак, особливості настільних рольових ігор робить практично неможливим повноцінно перенести досвід подібних ігор у формат відеоігор. Різні спроби адаптацій фокусувалися на різні елементи, через це ігри, натхнені Dungeons & Dragons, є неймовірно різноманітними. Ці ігри, у свою чергу, надихали ще більшу кількість ігор.

Так, наприклад, серед одних тільки ігор, що були розроблені в Іллінойсі на комп'ютерній системі PLATO були подібні спроби адаптації Dungeons and Dragons: pedit5, розроблена Расті Резерфордом (Rusty Rutherford) в 1975 році, і є найдавнішою спробою адаптувати Dungeons and Dragons, що збереглася; dnd, розроблена Ґері Уазенхантом (Gary Whisenhunt) і Реєм Вудом (Ray Wood), випущена у 1976 році і створена під деяким впливом pedit5; Moria, розроблена Кеветом Данкомбом (Kevet Duncombe) і Джімом Баттіном (Jim Battin), розробка почалася у 1975 році, датою релізу вважають 1978, і мала останнє оновлення у 1984 році, що є ранньою мережевою багатокористувацькою грою, де навколишнє середовище від лиця першої особи ігрового персонажу зображується за допомогою простої трьохвимірної графіки; тощо. Велика кількість ігор того періоду не зберіглися. Дані ігри були дуже вражаючими для свого часу. Цікаво, що автори Moria не були натхнені Dungeons and Dragons напрямом, а були натхнені розповідями авторів dnd [49-51].

Одна з найвпливовіших ігор, натхнених Dungeons and Dragons, була Colossal Cave Adventure (також відома як просто Adventure), розроблена Віллом Краудером (Will Crowther) за допомогою мови програмування FORTRAN, випущена у 1976 році для PDP-10. Окрім Dungeons and Dragons, Colossal Cave Adventure також була натхнена досвідом подорожей по печерам. Частково мотивацією Краудера було створити щось, що могло б сподобатися його дітям, від котрих він відчував відчуженість після розлучення зі своєю дружиною. Згадані раніше ігри, що імітували досвід Dungeons and Dragons, фокусувалися на реалізації ігрових механік, і наратив та атмосфера були на низькому пріоритеті. Colossal Cave Adventure, з іншого боку, має порівняно прості ігрові механіки: гравцю показується текст, гравець пише команду одним-двома словами природною мовою, гра реагує на команду гравця і показує новий текст залежно від написаної команди. Гравцю не кажуть, які команди доступні в даний момент, і йому потрібно здогадуватися про це, використовуючи логічне мислення. Подібний момент в Colossal Cave Adventure і багатьох її нащадках є трохи неоднозначним: з одного боку, відсутність списку можливих команд призводить

до того, що у гравця може статися відчуття, наче можлива будь-яка дія, і може мотивувати креативне мислення для розв’язання проблем; з іншого боку, є ризик того, що гравець загубиться і не буде знати, що робити, або ще гірше, якщо гравець сформулює команду з правильним значенням, але з формулюванням, що не було передбачене дизайнером, що може привести до загубленості гравця, перебору формулювань і знищення відчуття правдоподібності. В Colossal Cave Adventures є немало головоломок, де гравець стикається з деякою проблемою, і потрібно знайти шлях її вирішення за допомогою доступних ресурсів і інформації. Сюжет гри полягає у тому, що гравець блукає печерами і шукає скарби. У грі є фантастичні елементи. Гра була дуже впливова, і породила жанр, котрий на її честь назвали Adventure (пригодницька гра, відома у колишньому Радянському Союзі як "квест") [52, 53].

До 1970-х, комп’ютери були занадто дорогими та громіздкими, щоб виправдовувати затрати для отримання комп’ютер для особистого використання. Зазвичай комп’ютери розроблялися або закуплялися для організацій, з задумом, що достатньо велика група людей. Але з розвитком технологій, з’являлися все більш і більш компактні і дешеві комп’ютери, і з часом ідея комп’ютера для особистого використання вже була менш непрактичною. Kenbak-1, розроблений Джоном В. Бланкенбейкером (John V. Blankenbaker) і випущений у 1971 році компанією Kenbak Corporation вважається деякими першим особистим комп’ютером. Kenbak-1 розроблявся з розрахунком на школи а не окремих користувачів, і позиціонувався як дешевий комп’ютер для навчання роботи з комп’ютерами. Kenbak-1 не використовував мікропроцесор, адже був розроблений незадовго до того, як перші мікропроцесори були доступні. Було продано лише приблизно 40 одиниць до того, як виробник закрився у 1973 році. Іншим кандидатом на позицію першого персонального комп’ютера є Micral N, продукт компанії Réalisation d'Études Électroniques, що був розроблений Андре Чионг (André Truong) і Франсуа Гернель (François Gernelle) і випущений у 1973 році. Micral N використовував мікропроцесор Intel 8008. Micral N не позиціонувався як комп’ютер для особистого використання, а як дешевий

комп’ютер для автоматизації виробництва. У Франції було продано приблизно 500 одиниць, і, за оцінками, загальний обсяг продажів складав менше 2000 одиниць. Mark-8 — дизайн персонального комп’ютера на базі Intel 8008, створений Джонатаном Тітусом (Jonathan Titus) у 1974 році. Mark-8 не вироблявся комерційно, але інструкції по збору продавалися за 5$ журналом Radio-Electronics. Цільовим маркетом були ентузіасти, котрі самі окремо куплять деталі та збирали комп’ютер для особистого використання [54].

У 1974 році, Артур Сальсберг (Arthur Salsberg), директор редакції журналу Popular Electronics, бажав створити статтю про будування комп’ютера вдома для відновлення популярності журналу. Оригінальною ідеєю був дуже простий комп’ютер на базі Intel 8008 для ентузіастів, що бажають навчитися будувати комп’ютери. Проте, плани змінилися, коли редакція журналу побачили статтю свого суперника, Radio-Electronics, про Mark-8, що позиціонувався як повноцінний комп’ютер на базі Intel 8008. Редакція Popular Electronics вирішила замість простого комп’ютера на базі Intel 8008 зробити повноцінний комп’ютер на базі Intel 8080, що буде на рівні комерційних моделей. Редакція Popular Electronics сконтактувала компанію MITS (Micro Instrumentation and Telemetry Systems) щоб домовитися щодо розробки і виробництво конструкторських наборів таких комп’ютерів. У 1974 році, Едвард Робертс (Edward Roberts), засновник MITS, і Білл Єйтс (Bill Yates) завершили прототип і відправили до Popular Electronics, але пошта загубила цей прототип по дорозі. Через це, для статті в Popular Electronics замість фотографії прототипу використовувалася фотографія пустої коробки з вимикачами та світлодіодами. MITS побудували другий прототип, котрий в цей раз вдалося успішно доставити до Popular Electronics. Даному комп’ютеру була дана назва Altair 8800, і у січні 1975 року Popular Electronics заявили про запуск цього комп’ютера. Altair 8800 став дуже популярним серед ентузіастів і було продано велику кількість комп’ютерних конструкторів. До роботи над цим проектом, MITS були близькі до банкрутства, але успіх Altair 8800 спас компанію. Altair 8800 став відомий як перший комерційно успішний персональний комп’ютер. З часом, почали появлятися інші персональні комп’ютери [55].

У 1975 році, Пол Аллен (Paul Allen) прочитав статтю в Popular Electronics про Altair 8800, і, після того, як він показав її своєму другові, Біллу Гейтсу (Bill Gates), вони вирішили написати інтерпретатор мови Basic для цієї системи. Після представлення інтерпретатора Робертсу, той погодився розширювати цей інтерпретатор, відомий як Altair BASIC. В тому ж році, Білл Гейтс і Пол Аллен заснували Microsoft [55].

Написання комп'ютерних ігор було дуже популярним серед комп'ютерних ентузіастів, тому, звісно, коли з'явилися доступні персональні комп'ютери, на них швидко почали розробляти ігри. Спочатку, велика кількість з них базувалася на вже існуючих відеоіграх, котрі були розроблені на університетських комп'ютерах або для аркад. Програми нерідко поширювалися через журнали і книги. Досить відомою є книга "BASIC Computer Games", написана Девідом Алем, що компілює різні ігри написані мовою BASIC (про котрі він раніше також писав у книзі "101 BASIC Computer Games") і перекладені на діалект Microsoft BASIC [56]. Найвідомішими іграми, що були присутні у цій книзі, є Hammurabi і Super Star Trek [26, 41, 56].

Першою комерційною грою для персональних комп'ютерів вважається Microchess, розроблений Пітером Дженінгсом (Peter Jennings) для персонального комп'ютера KIM-1, випущений у 1976 році. Купивши KIM-1, Дженінгс вирішив написати програму для гри в шахи. Алгоритм гри в шахи був заснований на стратегіях, описані у книзі "Моя система" (Mein System) шаховим гравцем Ароном Німцовичем (Ārons Nimcovičs). Оригінальна версія Microchess не мала візуалізації дошки, програма лише вказувала свої ходи за допомогою шахової нотації, але пізніші версії добавили віртуальну дошку. Через те, що Microchess був комерційним продуктом, Дженінгс придав немалу увагу написанню посібника. Посібник включав у себе код програми, і в 1976 і на початку 1977 покупець отримував лише посібник, і очікувалося, що той перепише код з посібника до свого комп'ютера. Потім, Дженінгс почав також продавати трохи дорожчу версію вже записану на касету. Microchess був комерційно успішним

продуктом, через це Дженінгс звільнився зі своєї роботи і у 1977 році заснував компанію Micro-Ware [57].

У 1977 році студенти Тім Андерсон (Tim Anderson), Марк Бланк (Marc Blank), Брюс Даніелс (Bruce Daniels), і науковий співробітник Дейв Леблінг (Dave Lebling) розробили гру Zork, сильно натхнену грою Colossal Cave Adventure, для комп'ютера PDP-10 мовою програмування MDL. Вирішивши, що вони бажають продовжити працювати разом, вони в 1979 році заснували компанію Infocom. Компанія була заснована без планів займатися розважальним бізнесом, але було вирішено, що Zork, портований на персональні комп'ютери, буде їх першим продуктом. Для легкості портування між різними комп'ютерними системами, було вирішено написати віртуальну машину, названу Z-Machine, що інтерпретує файли історії, створені за допомогою спеціальною мови програмування, ZIL (Zork Implementation Language), що була заснована на MDL. Завдяки такому підходу, для портування на нову систему достатньо лише портувати Z-Machine, в той час як файли історії нема потреби змінювати. Іншим наслідком є те, що для створення нової гри подібного типу достатньо лише переписати файли історії, в той час як змінювати Z-Machine нема потреби. Z-Machine можна вважати одним з перших ігрових рушіїв, і Infocom використовувала її і мову ZIL для своїх наступних ігор. З часом також були написані інтерпретатори ZIL і за межами Infocom, що сильно допомагає зі збереженням цих ігор. Також з часом мова ZIL стала достатньо популярною для написання текстових пригодницьких ігор типу Zork або Colossal Cave Adventure. Zork є однією з найпопулярніших текстових пригодницьких ігор і вважається класикою [58-61].

В 1973 році, після успіху Pong, Atari активно розширювалася і наймала якнайбільше людей і засновувала закордонні відділи, такі як Atari Japan. Те, як це проводилося, мало деякі проблеми. Наприклад, Atari найняли якнайбільше виробничого персоналу і недоплачували їм, що призвело до проблем з якістю аркадних кабінетів і скарг від клієнтів. Щоб вирішити цю проблему, Atari найняла сторонніх експертів і почала провадити пільги для персоналу. При розширенні в Японію також були проблеми, пов'язані у тому числі з незнанням японських

регуляцій і порушенні правил. В 1973 році, Atari сформували підставну компанію Kee Games під керівництвом друга співзасновника Atari Бушнела, Джозефа Кінана (Joseph Keenan), що зображала з себе конкурента Atari. В 1973-1974, Atari і Kee Games випустили велику кількість клонів Pong. В 1974 році, Atari також розробили гру Gran Trak 10, гоночна гра, де гравець управляв машиною за допомогою руля, педалей і перемикача передач. Розробка мала багато проблем: досвідчені менеджери були відсутні, або через те, що їх відправили до Kee Games, або, у випадку з Алкорном, через відпустку; новий голова інжинірингу був недосвідченим, через що ставалися відставання від графіку; проблеми з дизайном і перевищення бюджету; були проблеми зі замовленням електросхем, а постачальник в якийсь момент навіть відмовився виробляти їх через нестандартний дизайн; розробники були недосвідчені у перетворенні прототипу на фінальний продукт, і вже після вироблення 100 аркадних кабінетів через їх низьку якість Алкорну прийшлося повернутися з відпустки і виправити недоліки; і, найбільша проблема, через бухгалтерську помилку і непорозуміння, ці аркадні автомати продавалися по ціні, меншім за їх собівартість. Через перелічені проблеми у 1974 році Atari знаходились на межі банкрутства. Atari Japan була продана компанії Namco. Знаходячись у екстремальній ситуації, Atari стали більше експериментувати: якщо раніше більшість ігор, котрі розроблялися, були варіаціями Pong, то на прикінці 1974 року, Atari випустили аркадну гру з використанням світлового пістолету Qwak! та аркадну гру без екрану але з кнопками, що можуть світитися, де потрібно повторяти послідовність Touch Me. Kee Games, з іншого боку, працювали з максимальною ефективністю. У 1974 році була випущена гра Tank, одна з небагатьох оригінальних розробок Kee Games. Гра була дуже успішна, що спасло Atari від банкрутства. В кінці 1974 року, Atari і Kee Games формально об’єдналися в одну компанію. З 1974 по 1975, Atari розробляли домашню версію Pong, котру випустили у кінці 1975. Домашня версія Pong мала величезний попит. Таким чином, Atari вступили у маркет відеоігор для дому. Почуваючи тиск від конкурентів на маркеті аркадних ігор і бажаючи не залежати від занадто дорогих аркадних кабінетів, в 1976 році Atari почала розробку ігрової

консолі [62-64].

Розробка ігрової консолі виявилася досить дорогою справою, і, заради фінансування, Бушнел домовився з Warner Communications, котрі були зацікавлені на проникнення на ринок відеоігор, про придбання Atari. В 1977 році, консоль була видана під назвою Atari VCS (Video Computer System), хоча у 1982 році вона була перейменована на Atari 2600. Atari VCS використовувала картриджі. Основний контролер складався з джойстика і кнопки. Більшість ранніх ігор була адаптаціями аркадних ігор, таких як Pong, Breakout, Space Invaders, тощо. Цікавою є гра Adventure, що була розроблена Ворреном Робінеттом (Warren Robinett) і була натхнена грою Colossal Cave Adventure, але, можливо через те, що можливості Atari VCS не дуже підходили під потреби гри типу Colossal Cave Adventure, те, як Робінетт імітував почуття гри Colossal Cave Adventure не використовуючи тексту було досить унікальним. [62-64].

У 1979 році, група програмістів, незадоволені політикою Atari не вказувати авторство своїх розробників, покинули її. Девід Крейн (David Crane), Ларрі Каплан (Larry Kaplan), Алан Міллер (Alan Miller), і Боб Вайтхед (Bob Whitehead) заснували Activision, що стало першою компанією окрім Atari що розробляла ігри для Atari VCS. З часом, інші компанії також почали розробляти для консолі Atari [62].

Нове десятиріччя, 1980-ті, ігрова індустрія зустріла вже придбавши знайому форму. Так, звичайно, ігрова індустрія не перестає розвиватися і змінюватися, нерідко під впливом технологічних проривів, культурних подій і інших форм медіа, і у наступних десятиріччях також будуть досить важливі зміни що матимуть великий вплив на відеоігри, такі як поширення портативних комп’ютерних пристроїв та поява інтернету, але велика кількість базових тенденцій та динаміки сформулювалися ще до 1980-х, і вже у 1970-х почали з’являтися компанії, що спеціалізуються в основному на відеоіграх, в той час як існуючі компанії почали звертати увагу на цю молоду розважальну індустрію.

### 1.3. Висновки до розділу 1

Історія відеоігор тісно і нерозривно пов’язана з історією інформаційних технологій і культури. Більшість ранніх відеоігор були створені дослідниками як адаптації існуючих ігор (наприклад, шах, нім, хрестики-нулики, більярд, теніс тощо), зазвичай як дослідницькі проекти або демонстрація можливостей науки. Деякі з даних проектів мали значний вплив на розвиток комп’ютерних наук (наприклад, програма для гри у шашки Артура Самуеля була першою відомою програмою, де використовувалося машинне навчання). З розвитком технологій, розробка і розповсюдження відеоігор ставали більш і більш доступними, що призвело до появи нових типів ігор. Так, наприклад, гра Spacewar! не була заснована на існуючих до того іграх, і замість цього надихалася науково-фантастичними книгами, і метою розробників було передати почуття бойових сцен з популярних книжок за допомогою комп’ютерної гри. Spacewar! було б практично неможливо розробити у форматі традиційних ігор, адже вона використовувала специфічні можливості комп’ютера. Поява і поширення мов програмування полегшало розробку ігор з великою кількістю тексту. Цікавим фактом є те, що перша ігрова консоль, Magnavox Odyssey, була концептуалізована як інтерактивний телевізор, а не ігровий комп’ютер.

# РОЗДІЛ 2. ВПЛИВ РІЗНИХ ФАКТОРІВ НА РОЗРОБКУ ВІДЕОІГОР

## 2.1. Жанри відеоігор і проблеми з їх визначенням

Жанр — неформальна категорія форми творів мистецтва зі суспільно узгодженими конвенціями. Жанри є плинними, розуміння про них може змінюватися з часом, можуть виділятися нові жанри або забуватися старі, і твір, котрий історично приписувався як представник одного жанру, з часом може сприйматися як представник іншого. Категоризація творів за жанрами є суб'єктивним процесом, хоча були спроби дослідників створити жорсткі критерії належності ігри до певного жанру.

Поділ відеоігор на жанри потенційно полегшує гравцям процес знаходження відеоігор в яких вони зацікавлені; розробникам процес створення дизайну ігор, на основі використання досвіду попередніх розробок; покращує обізнаність гравців з конвенціями жанру; а також полегшує процес обговорення відеоігор гравцями.

Зазвичай жанри відеоігри класифікуються за спільними рисами ігрового дизайну або "генеалогічно", на базі ланцюгів натхнень між різними іграми. Видатними виключеннями є такі жанри, як горор (horror), ігри котрого створюються з фокусом на залякування гравця і/або створення атмосфери жаху, і стелс (stealth), в іграх котрого гравцям потрібно використовувати скритність. Подібні риси реалізуються різними шляхами, і обидва ці жанри складно представити як єдині родини ігор, що пов'язані ланцюгами натхнення (наприклад, Castle Wolfenstein (1981), Metal Gear (1987) і Thief: The Dark Project (1998) є достатньо відомими стелс-іграми, але, наскільки відомо, не впливали одна на одну). Також, елементи цих жанрів нерідко використовують як епізодичний елемент у іграх, котрі в цілому не вважаються частинами даних жанрів.

Іноді назва жанру створюється видавництвом під час рекламної компанії, як сталося з Survival Horror, жанром ігор, що створюють атмосферу дискомфорту шляхом обмеження ресурсів і, іноді, дещо незручних ігрових механік. Ця назва була створена компанією Capcom як частина рекламної компанії і опис їх нової

гри, Resident Evil, що була випущена у 1996 році. З часом, цей термін почали використовувати не тільки щодо Resident Evil та її імітаторів, а і з іншими іграми, що мали подібні філософії ігрового дизайну [65].

Показовою є історія терміну метроїдванія (Metroidvania), котрим зазвичай описують ігри-платформери, зазвичай двохвимірні, з ігровим світом, що складається з взаємопов'язаних зон, доступ до котрих відкривається у дещо нелінійному шляху в процесі гри, зазвичай методом здобуття нових здібностей для ігрового персонажа, що дозволяють потрапити у недоступні раніше зони. Цей термін оригінально використовував Скотт Шаркі (Scott Sharkey) для опису нових на той час ігор серії Castlevania (особливо гру Castlevania: Symphony of the Night), що були створені під керівництвом Ігараші Коджі (五十嵐 孝司, Igarashi Koji) і мали більш нелінійну структура, аніж більшість минулих ігор серії Castlevania, нагадуючи багатьом гравцям про серію ігор Metroid. Джеремі Паріш (Jeremy Parish) розширив цей термін, включивши усі ігри, що мали подібний ігровий дизайн, і популяризував його. Велика кількість ігор, що з'явилися задовго до появи цього терміну, стали вважатися частиною цього жанру, і також стали з'являтися нові ігри у цьому жанрі. Сам Ігараші, дізнавшись про появу терміну "метроїдванія", сприйняв його дуже позитивно, але зауважив, що для нього основним натхненням була The Legend of Zelda, а не Metroid [66-68].

Іноді розробники ігор з самого початку планують створити гру деякого жанру, а іноді гра розробляється без прив'язки до жанру. Так, наприклад, Moon Studios, розробники серії ігор Ori, заявляли, що для них метою розробки цією серії ігри було створення ідеальної метроїдванії, з фокусом на покращення елементів, котрі на їх думку, були недостатньо проробленими у інших метроїдваніях. Ori and the Blind Forest (2015) фокусувався на контролі руху ігрового персонажу, в той час як Ori and the Will of the Wisps (2020) фокусувався на бойовій системі. З іншого боку, Team Cherry, розробники Hollow Knight (2017), розробляли гру не прив'язуючись до конвенцій жанру і навіть зазвичай не називали свою гру метроїдванією під час розробки, незважаючи на те, що вона мала багато спільних

елементів з іграми цього жанру і серед її натхнень були ігри, що можуть вважатися метроїдваніями. Іронічно, при обговоренні Ori and the Blind Forest зазвичай в основному звертали увагу на плавність рухів ігрового персонажу, той час як елементи метроїдванії сприймалися вторинними, в той час як Hollow Knight став найбільш впливовою сучасною метроїдванією [69].

Напевно, одним з найбільш відомих жанрів, чиї межі стали дуже розмитими є Roguelike, і найбільш відомою спробою створити жорстке визначення жанру відеоігор є Берлінська інтерпретація, що описує характеристики Roguelike.

Засновником жанру Roguelike зазвичай вважають гру Rogue 1980 року. Rogue була розроблена Майклом Тойем (Michael Toy), Гленом Вічманом (Glenn Wichman) і Кеном Арнольдом (Ken Arnold). Основними впливами на Rogue були Dungeons and Dragons і Colossal Cave Adventure. Графіка гри будується з текстових символів, де різні символи символізують різні об’єкти (наприклад, "@" є ігровим персонажем). Світ гри процедурно генерується на початку гри. Якщо ігровий персонаж помирає, то гравець губить увесь прогрес і йому потрібно починати з самого початку. Гравцю присутній достатньо велика кількість дій, кожна з котрих назначена на окрему клавішу. Гра має доволі велику кількість достатньо складних систем. Бойовий процес, проте, є достатньо простим: якщо істота (включаючи ігрового персонажа) спробує рухатися на місце, де знаходиться інша істота (включаючи ігрового персонажа), то перша істота замість руху спробує атакувати другу істоту. Також гравець може екіпірувати лук і стріляти стрілами (за допомогою команд екіпіровки і кидання предметів). Дана гра була дуже впливовою і з’явилася величезна кількість імітаторів, але цей жанр в основному залишався нішевим [70].

У 2008 році, на International Roguelike Developer Conference 2008 у результаті обговорень було створення визначення жанру Roguelike, що складалася з 9 ключових характеристик і 6 незначних характеристик [71].

Ключові характеристики включають: випадкова генерація навколишнього середовища; смерть назавжди (permadeath) — гравець губить увесь прогрес у випадку смерті, і не очікується, що гравцю вийде пройти гру з першого разу;

покроковість; гра відбувається на сітці клітинок, усі істоти (включаючи гравця) займають одну клітину; відсутність різних режимів гри — рух, битва і інші дії відбуваються у одному режимі, і гравець може в будь-який момент виконати будь-яку дію; гра складається зі складних систем, що може приводити до емерджентних поведінок; гравцю потрібно ефективно розпоряджатися ресурсами; фокус на протистояння гравця і світу, відсутність відношень між іншими істотами; гра потребує обережне дослідження навколишнього середовища [71].

Незначні характеристики включають: гра є однокористувацькою і орієнтованою на гравця; інші істоти функціонують за правилами, подібних до тих, за котрими функціонує ігровий персонаж; тактичне випробування, гравцям потрібно розробляти тактики для здобуття прогресу; графіка будується з текстових символів; рівні складаються з кімнат та коридорів між ними; характеристики персонажа показуються гравцю у вигляді чисел [71].

Берлінська інтерпретація є найбільш відомим визначенням Roguelike, але також критикувалися як застаріла і занадто обмежувальна. Навіть деякі класики жанру, такі як ADOM (1994), Angbang (1993) і Crawl (2006) не відповідають повністю Берлінський інтерпретації через присутність декількох режимів [71].

Іронічно, але саме у 2008 році вийшла гра, котра популяризувала комбінування елементів Roguelike з іншими жанрами, і призвела до появи величезної кількості ігор з елементами Roguelike, котрі не відповідають Берлінській інтерпретації: Spelunky, розроблена Дереком Ю (Derek Yu). Spelunky комбінує інтуїтивно зрозумілий ігровий процес платформерів з випадковою генерацією рівнів і постійною смертю Roguelike. Також, між різними зонами гравець зустрічає Тунельника (Tunnel Man), котрому може заплатити за побудування тунелю до даної зони, і прогрес, пов’язаний з Тунельником, не губиться при смерті ігрового персонажу. Якщо тунель був побудований, то при початку нової гри гравець може почати гру з тієї зони, до котрої веде тунель замість самого початку [72].

Гібридні Roguelike стали дуже популярними, в той час як більш "чисті" залишилися нішевими. Як результат, зараз Roguelike ігри є дуже різноманітними і

зазвичай перехрещуються з іншими жанрами, а спільні риси у більшості Roguelike – це випадкова генерація ігрового світу і загублення прогресу у випадку програшу (хоча нерідко також присутні окремі системи, що зберігають свій прогрес між різними спробами і впливають на наступні спроби, подібно до Тунельнику з Spelunky).

Зазвичай не очікується, що гравцям вийде пройти Roguelike з першого разу. Очікується, що гравець буде робити багато спроб, програвати, знову пробувати і ставати краще і краще у даній грі. Сюжети зазвичай мінімальні, але бувають виключення.

З іншого боку, в Японії з'явилася інша генеалогічна лінія Roguelike ігор з дещо іншими пріоритетами. У 1983 році вийшла гра Torneko's Great Adventure, перша гра дуже популярної і впливової серії ігор Mystery Dungeon, що породила свою гілку жанру Roguelike (котру можемо називати японськими Roguelike).

Зазвичай в японських є два режимі гри, котрі ми можемо назвати "містом" (town) і "підземеллям" (dungeon) (звичайно, вони не обов'язково є саме містом і підземеллям). Підземелля нагадує своїм ігровим процесом класичні Roguelike, з рівнем розділеним на сітку клітинок і бойовою системою, що складається зі спроб рухатися на зайняту противником клітину, де дії відбуваються покроково. У випадку програшу, ігровий персонаж губить усі предмети, що має з собою, і повертається до міста. Ігровий персонаж може також добровільно повернутися до міста, у такому випадку той приносить здобуті предмети з підземелля. Місто, у свою чергу, є мирною локацією, нерідко з дружними неігровими персонажами і можливостями продажу та покупки предметів. Перед виходом до підземелля, гравець може обрати обмежену кількість предметів, котрі бере до підземелля. Японські Roguelike нерідко мають гарно промальовану графіку і сюжет.

Японські Roguelike дещо цікаві тим, що одразу наслідують Rogue і при тому вони водночас не відповідають Берлінській інтерпретації, зберігають риси котрі зазвичай відсутні у гібридних Roguelike, і загублення прогресу у випадку програшу виконане так, що сильно змінює пріоритети гри. Якщо для інших Roguelike, очікується, що гравець буде пробувати грати, програвати, покращувати

свої навички щоб просунутися далі, аніж раніше, знову програвати і так до кінця гри або життя гравця, то японські Roguelike мають набагато більше фокусу на те, щоб знати, коли повернутися назад. Чим далі гравець зайде у підземелля, тим більше той може знайти ресурсів, але тим більше ризикує загубити усе, що має з собою. Чи буде гравець ризикувати і проходити далі, чи поверне назад щоб покращити шанси наступної спроби?

Одним з найскладнішим для визначення і водночас один з найпопулярніших жанрів відеоігор є RPG (Role-Playing Game, рольова гра). RPG є одним з найрізноманітніших жанрів відеоігор, і нерідко, у вакуумі, дві гри, обидві з котрих вважаються RPG, можуть майже не мати нічого спільного. RPG має величезну кількість піджанрів, у котрих родинні подібності дещо більш помітні. Причиною цього є те, що RPG, як жанр комп'ютерних ігор, розпочався як адаптація настільних рольових ігор (TTRPG, Tabletop Role-Playing Game), особливо Dungeons and Dragons. Настільні рольові ігри дещо легше визначити, це ігри, у котрих гравці приймають роль видуманих персонажів і приймають участь у інтерактивній історії. Настільні рольові ігри зазвичай розраховані на декілька гравців, одна людина виконує роль, яку зазвичай називають Майстером Гри (Game Master), і контролює практично усе у грі за межами ігрових персонажів і забезпечує виконання правил. Настільні рольові ігри відчуваються як комбінації гри і створення історії у реальному часу. Через популярність настільних рольових ігор серед комп'ютерних ентузіастів і студентів, була величезна кількість спроб адаптувати досвід гри у настільні рольові ігри для комп'ютерних систем. Однак, через природу настільних рольових ігор їх практично неможливо повноцінно адаптувати у формат відеоігор. Різні спроби адаптації фокусувалися на різні елементи, і, як результат, спроби адаптації ігор типу Dungeons and Dragons були досить різноманітні. Ці ігри, у свою чергу, надихали інших розробників на створення нових ігор, вже необов'язково з прив'язкою до настільних ігор. З часом RPG ігри також почали інкорпорувати елементи з інших жанрів, а деякі ігри інших жанрів також почали інкорпорувати елементи з RPG.

Так сталося, що один з небагатьох елемент, що присутній у більшості RPG, це підвищення рівню (leveling up) або система досвіду (experience system): у ігрового персонажу є деякі характеристики; в процесі гри за деякі події (зазвичай перемогу над противником або важливі події в сюжеті) гравцю даються очки досвіду (experience points, EXP), і при наборі деякої кількості очок досвіду стається підвищення рівня персонажу, що, у свою чергу, приводить до підвищення характеристик персонажу. Деталі подібних систем можуть сильно варіюватися. Деякі вважають подібну систему рисою, за котрою можна визначити, чи належить гра до жанру RPG. Тим не менш, подібні системи використовувались також у іграх, котрі занадто складно назвати RPG, такі як багатокористувацький режим Call of Duty. З іншого боку, ігри, котрі нерідко вважаються RPG, можуть не мати системи досвіду, наприклад, The Tower of Druaga (1984), що досить сильно вплинула на інші RPG ігри [73].

З іншого боку, наприклад, ігри серії The Legend of Zelda зазвичай не вважаються RPG. Це аргументується відсутністю системи досвіду (за винятком Zelda II: Adventure of Link). Також розробники ігор серії The Legend of Zelda зазвичай не називали їх RPG (за винятком The Legend of Zelda: Ocarina of Time, що в Японії описувалася як Action RPG). Не зважаючи на це, ігри серії The Legend of Zelda іноді мають разючу схожість з трендами RPG того часу. Так, наприклад, перша гра серії, The Legend of Zelda (1986) дуже сильно нагадує RPG ігри The Tower of Druaga (1984) і Hydlide (1984), що були досить популярними під час розробки цієї гри. Друга гра серії, Zelda II: The Adventures of Link (1987), також дещо слідує трендам RPG ігор того часу: на відміну від першої гри, і подібно до Dragon Buster (1984) і Romancia (1986), гра має елементи платформера, під час проходження рівнів, знаходження у містах, та боїв камера знаходиться збоку; також, подібно до Dragon Quest (1986), гра має режим глобальної мапи, де, коли ігровий персонаж знаходиться за межами рівнів, міст і боїв, то показується мапа світу де гравець контролює рухом непропорційно зображеного ігрового персонажа, і іноді можуть ставатися випадкові напади противників. Серія ігор The Legend of Zelda була також досить впливова, як серед RPG так і за межами цього

жанру. Отже, незважаючи на те, що традиційно The Legend of Zelda не вважається RPG, вона є частиною генеалогії жанру RPG [74, 75].

Іншими іграми, котрі мають близькі родинні зв'язки з RPG, але зазвичай не вважаються RPG, є пригодницькі ігри. Colossal Cave Adventure також був однією зі спроб адаптувати Dungeons and Dragons у формат комп'ютерної гри, і наступні пригодницькі ігри, такі як Zork, також нерідко мали прямий вплив Dungeons and Dragons. Історично, RPG і пригодницькі ігри нерідко впливали на і надихали одні одних. Не зважаючи на це, традиційно пригодницькі ігри не вважаються піджанром RPG.

Також іноді виділяють жанр RPG Horror. Це не поєднання жанрів RPG і Horror (то буде Horror RPG), а жанр ігор у жанрі Horror, що були розроблені за допомогою інструментів для розробки RPG ігор типу RPG Maker або котрі імітують стиль подібних ігор.

### 2.2. Вплив інструментів і методів розробки на ігровий дизайн

Цільова платформа, її технічні характеристики і можливості, її пристрої ведення та виведення, носії даних, використовувані інструменти розробки і методи доставлення — усе це має серйозний вплив на те, які ігри можливо і доцільно розробляти. Таким чином, усі перелічені фактори впливають на ігровий дизайн.

Мова асемблеру близька до машинного коду і вона дозволяє дуже точний контроль над роботою комп'ютера. Мови програмування високого рівня, з іншого боку, повинні компілюватися або інтерпретуватися, вони зазвичай легші у використанні, але якість програми досить сильно залежать від компілятора або інтерпретатора. Зараз, зазвичай зручність використання мов програмування високого рівню і висока якість компіляторів робить написання програм асемблером досить непрактичним. Однак, раніше нерідко для створення оптимізованих програм потребувався точність контролю, котру мови програмування високого рівня не могли забезпечити.

Серед ранніх текстових ігор досить велика кількість була написана за допомогою мов програмування високого рівня. Наприклад, The Sumerian Game (1964) була написана мовою Fortran, Star Trek (1971) був написаний мовою BASIC, Colossal Cave Adventure (1977) була написана мовою Fortran, і Zork (1977, 1980) оригінально був написаний мовою MDL, а потім переписаний спеціально створеною для цієї гри мовою ZIL. Через це можемо прийти до висновку, що для даних ігор точність контролю виконання програми було не настільки важливим, адже вони не виконували занадто складних операцій, на відміну від спрощення процесу розробки. Це на відміну від, скажімо, Spacewar! (1962), що була написана мовою асемблера і мала графіку. Враховуючи, що перші відомі текстові ігри вже були написані за допомогою мов програмування високого рівня, можна припустити, що саме використання мов програмування високого рівня зробило доцільним розробку текстових ігор, дозволивши розробникам зосередитися на загальній картині, а не заморочуватися з кожною деталлю реалізації.

В процесі портування Zork під персональні комп'ютери, була створена віртуальна машина Z-Machine і мова програмування ZIL. Це спростило процес портування гри під різні системи, але також дозволило оптимізувати мову програмування під потреби гри. Взаємодія гри з машиною була дещо відокремлення від логіки самої гри. Це також облегшило повторне використання цих інструментів для наступних ігор. Z-Machine була раннім прикладом ігрового рушія, але не єдиним. Так, Mystery Home, вироблений та випущений Sierra On-Line (що на той момент називалася On-Line Systems) в 1980 році, була розроблена використовуючи ADL (Adventure Development Language), котрий також використовувався для декілька наступних ігор, в 1984 році Sierra On-Line випустили гру King's Quest, створену за допомогою AGI (Adventure Game Interpreter) і Game Adaptation Language, і в 1987 році Lucas Arts випустили гру Maniac Mansion, що була створена використовуючи рушій SCUMM (Script Creation Utility for Maniac Mansion), котрий вони використовували до 1998 року. Для цих перерахованих ігор для спрощення процесу розробки були розроблені більш спеціалізовані інструменти, котрі потім використовувалися для наступних

ігор. Спеціалізовані інструменти дозволили легше розробляти набагато складніші ігри, аніж якби вони програмувалися мовою асемблера.

З часом, використання ігрових рушіїв стало стандартною практикою у розробці ігор.

### 2.3. Вплив пристроїв вводу і виводу на ігровий дизайн

Пристрою вводу і виводу є важливими, адже вони визначають, що комп'ютер може виводити і як комп'ютером можна керувати. Наприклад, Майк Мейфілд через відсутність відеотерміналу не міг грати у Spacewar!, адже не було куди виводити графіку, і тому став розробляти гру, в котру можна грати з телепрінтером, Star Trek. Star Trek як виводився у вигляді тексту, так і контролювався за допомогою текстових команд. Можна припустити, що покроковість Start Trek могла бути спричинена форматом текстової гри, адже контролювати гру у режимі реального часу за допомогою тексту зазвичай незручно [40].

Цікавим для дослідження є приклад гри Adventure, розробленої Ворреном Робінеттом і випущеної у 1980 році. У 1978 році, працівник Atari, Робінетт пограв у Colossal Cave Adventure, і вирішив адаптувати її під Atari 2600. З цим одразу виникло декілька проблем: картридж для Atari 2600 мав лише 4 KB, що було недостатньо для збереження великих об'ємів тексту, і Atari 2600 контролювався за допомогою контролера, що складався зі джойстика та однієї кнопки, що не дуже зручно використовувати для вводу тексту. Отже, перед Робінеттом стояла задача адаптувати текстову гру, що контролювалася шляхом команд природної мови, для консолі, не використовуючи текст взагалі і контролюючи лише за допомогою джойстика та кнопки. Кімнати зображалися візуально, де форми, залиті кольором, представляли об'єкти. За допомогою джойстика можна було рухатися, в той час як кнопка використовувалась для підіймання або опускання об'єктів. Наприклад, якщо гравець підібрав меч, і дракон стикається з цим мечем, то дракон помирає. У грі є два типи противників: дракон (три екземпляри), що

намагається вбити ігрового персонажу, і кажани, що намагаються вкрасти об’єкти. Adventure для Atari є досить вражаючою, і як гра для Atari 2600, і як адаптація Colossal Cave Adventure. Adventure для Atari вдається передати відчуття від гри у Colossal Cave Adventure, включаючи прості і інтуїтивно зрозуміли шляхи взаємодій зі світом, незважаючи на обмеження консолі [76, 77].

Якщо дати ентузіастам достатньо часу, то вони зазвичай зможуть найти рішення будь-яким технічним проблемам. В 1997 році, задовго після кінця релевантності Atari 2600, Грег Траутмен (Greg Troutmen) розробив Dark Mage, текстову пригодницьку гру для Atari 2600. Гравцю даються описи текстом, і той обирає перелік дій за допомогою комбінації джойстика та кнопки. У стандартному режимі, якщо затиснути джойстик у деякому напрямі і паралельно з цим натиснути кнопку, то ігровий персонаж спробує піти до сусідньої кімнаті, що знаходиться у тому напрямі. Якщо натиснути кнопку, не затискаючи джойстик у якомусь напрямі, то виконується команда "оглянутися" (look). Після виконання огляду, дається список команд з котрих можна вибирати за допомогою джойстика і підтвердити за допомогою кнопки. Гра дуже проста, але враховуючи, що вона працює на платформі, характеристики котрої роблять розробку подібних ігор дуже непрактичним, це трохи вражає [78].

Гра Dark Mage (1997) була розроблена вже задовго після кінця релевантності Atari 2600 (1976-1983, хоча виробництво закінчилося у 1992), коли майже єдині, хто ще розробляв ігри для Atari 2600 були ентузіасти, що досліджували межі можливого застарілої консолі і шляхи обійти технічні обмеження. Хоча гра Adventure для Atari і була натхнена текстовою пригодницькою грою, вона через технічні обмеження мала дещо інший формат і майже не мала тексту. Отже, через технічні характеристики Atari 2600, один з найпопулярніших і найвпливовіших жанрів того часу не знайшов собі місця на платформі.

Через різність форм вводу, ні текстові пригодницькі ігри (що зазвичай управляються за допомогою вводу текстом команд), ні графічні пригодницькі ігри (що зазвичай управляються за допомогою мишки) зазвичай не були популярні на

консолях, не зважаючи на їх популярність на персональних комп'ютерах. Були деякі порти (наприклад, Maniac Mansion для NES, 1988 рік) і навіть деякі оригінальні ігри (Clock Tower, для Super Famicom, 1995, котра, незважаючи на те, що була розроблена і видана ексклюзивно в Японії для консолі, у плані ігрового процесу нагадувала західні графічні ігри, де курсор замість мишки управлявся за допомогою кнопок на контролері), але загалом ці жанри були дещо менш популярними. Видатними виключеннями є японські текстові пригодницькі ігри, котрі набагато частіше портувалися під консолі, і введення команд за допомогою тексту замінявся вибором команд зі списку (найранішим відомим портом подібного типу є гра The Portopia Serial Murder Case, що оригінально була розроблена для персонального PC-6001 комп'ютеру і випущена у 1983 і потім, у 1985, була портована до Famicom). З часом, подібний стиль введення команд став прийнятим і серед японських текстових пригодницьких ігор для персональних комп'ютерів. Більшість японських текстових пригодницьких ігор не були локалізовані за межами Японії. Потім в Японії також з'явився піджанр текстових пригодницьких ігор, де замість вибору команд що робити, фокус змістився на прийняття рішень у деякі моменти історії, що став відомим як візуальні новели (Visual Novel, VN, в Японії відомі як NVL). Іноді, особливо за межами Японії, усі японські текстові пригодницькі ігри також називаються візуальними новелами.

Інші жанри, що управляються в основному мишкою та клавіатурою, наприклад, стратегічні ігри, також були дещо менш популярними на консолях, хоча це дещо залежить від піджанрів.

Цікавим прикладом розробки представника жанру для платформи, що не дуже підходить для цього жанру, є The Tower of Druaga (1984), розроблена під керівництвом Ендо Масанобу (遠藤 雅伸, Endou Masanobu) компанією Namco. RPG ігри зазвичай займають достатньо довгий час, мають довгі сесії, і фокусуються на прогресі персонажа. Через це, вони зазвичай потребують функціонал збереження прогресу. Аркадні ігри, з іншого боку, зазвичай фокусуються на короткі сесії і не мають можливості зберегти прогрес. Аркадні

ігри здаються одним з найменш придатних форматів для RPG. Тим не менш, Ендо Масанобу, поставив собі мету адаптувати досвід гри в RPG у формат аркадної гри. Під впливом RPG гри для персональних комп'ютерів Wizardry, настільної RPG Dungeons and Dragons, і аркадної гри Pac-Man, гра The Tower of Druaga була створена. У даній грі, гравцю потрібно продертися через 60 поверхів вежі. Поверхи випадково генеруються. На кожному поверсі є двері до наступного поверху і ключ, котрий потрібно знайти, щоб бути здатним пройти на наступний поверх. Також на кожному поверсі є секретні предмети, для отримання котрих потрібно зробити деякі дії, що різняться між поверхами. Ці предмети можуть впливати на здібності ігрового персонажу і деякі з них необхідні для перемоги у грі. У грі є противники та бойова система. The Tower of Druaga є достатньо складною грою, і самому дізнатися про її численні секрети, деякі з котрих необхідні для проходження гри, практично неможливо. Проходження The Tower of Druaga було колективним досвідом, де гравці намагалися пройти гру, обмінювалися досвідом і давали поради один одному, після чього знову пробували пройти гру. The Tower of Druaga була дуже популярною і впливовою грою в Японії, але вона не була випущена за межами Японії [73].

Колись портування програми на іншу платформу було практично створенням нової але майже ідентичної програми. З часом, портувати програми з однієї платформи на іншу ставало все легше і легше, і версії програм для різних платформ почали базуватися на одній кодовій базі.

Ранні порти ігор нерідко дуже сильно відрізнялися одна від одної, і деякі ігри створювалися виключно для однієї платформи. З часом, різні версії ігор почали нагадувати одна одну набагато більше і, поступово, розробка ігор лише для однієї платформи стало сприйматися дещо недоцільним, особливо якщо у цей проект було вкладено багато ресурсів.

Як наслідок, ігри почали розроблятися з розрахунком на декілька платформ. Одним з наслідків було те, що ігри для персональних комп'ютерів почали нагадувати ігри для консолей, адже легше адаптувати управління, розраховане на

контролер, під клавіатуру і мишу, аніж адаптувати управління, розраховане на клавіатуру і/або мишу, під контролер.

Наприклад, Dragon Age: Origins (2009) був останньою значною грою від Bioware, що розроблялася з фокусом на персональні комп'ютери. Управління проводилося за допомогою клавіатури і мишки. При портуванні на консолі, значні елементи гри, від управління до дизайну рівнів, були спрощені. Незважаючи на те, що версія для персональних комп'ютерів була дещо краще сприйнята аніж версія для консолей, наступні ігри серії базувалися на дизайні консольної версії.

### 2.4. Вплив носіїв даних на ігровий дизайн

Носії даних також можуть мати сильний вплив на дизайн гри.

Наприклад, у 1986 році Nintendo випустила Famicom Disk System, периферійний апарат для своєї консолі Famicom (локалізований як NES або Nintendo Entertainment System за межами Японії), що дозволяє використовувати дискети. Дискети, у порівнянні з картриджами, що використовувалися для Famicom/NES, були дешевші, у них була більша місткість, їх можна було перезаписувати, що дозволяло зберігати прогрес. Основним недоліком дискет було те, що зчитування з них даних займає більше часу. Щоб показати здібності Famicom Disk System, Nintendo розробили The Legend of Zelda, що була випущена у 1986 році. На відміну від минулих консольних ігор, що були обмежені малим розміром даних і неможливості зберігати прогрес, і як наслідок, розроблялися з розрахунком, що їх можливо пройти за одну сесію і більшість часу гравець буде робити саме намагаючись завершити гру, The Legend of Zelda мала великий (для свого часу) відкритий світ, дослідження котрого скоріше за все займе у гравця декілька сесій. Впровадження дискет для консолі дозволило розробляти ігри, котрі були б раніше не доцільні для картриджів [74, 79].

У картриджів, однак, була одна важлива перевага: складаючись з електросхеми з чіпами, їх конструкцію можна модифікувати, розширюючи їх можливості. Можливо розширювати доступну пам'ять на картриджі шляхом

добавлення нових чипів пам’яті, між котрими можливо переключатися (хоча програмування ігор з розрахунком на такий метод дещо складніше, аніж якщо використовувати дискети). Проблему загублення прогресу при завершенні роботи вирішували, добавляючи у картридж батарейку. Як результат, гра типу The Legend of Zelda, стала можливою на картриджах. Іронічно, але The Legend of Zelda, що створювалася як гра для дискет, була випущена за межами Японії на картриджах, а Famicom Disk System, що колись вважався майбутнім для компанії, був випущений лише в Японії і Гонконзі, і навіть в Японії фокус змістився назад до картриджів [74, 79].

Серед інших прикладів використання цієї властивості картриджів є гра Star Fox, що була розроблена Nintendo і Argonaut Software, і випущена у 1993 році для Super Famicom/Super NES, що використовувала графічний чіп для обраховування трьохвимірної графіки, і Game Boy Camera, картридж з вбудованою камерою.

Незважаючи на те, що її конкуренти вже використовували CD диски (що можуть містити приблизно 650-850 мегабайтів пам’яті), у 1996 році Nintendo випустили Nintendo 64 з підтримкою картриджів і лише картриджів (що можуть містити приблизно 4-64 мегабайтів пам’яті). Через це, велика кількість розробників була не дуже зацікавлена у розробці ігор для Nintendo 64, включаючи тих, хто традиційно фокусувався на консолях від Nintendo. Так, наприклад, саме відсутність підтримки CD дисків вважається основною причиною того, що Squaresoft, що до того мала близькі відношення з Nintendo, переключила свій фокус на PlayStation від Sony. Їх гра, Final Fantasy VII, що вийшла в 1997 році, була розміром в 1,317 мегабайтів, котрі були розподілені між трьома дисками, і через це була неможливої для реалізація на Nintendo 64. У 1999 році, Nintendo випустили периферійний апарат 64DD для Nintendo 64, що дозволяв використовувати дискети (що можуть містити приблизно 64 мегабайтів пам’яті), для котрого вийшло лише 9 відомих ігор.

### 2.5. Методи доставлення ігор до гравця

Методи доставлення ігор до гравця також є важливим.

Одним з перших методів масового поширення ігор було друкування їх коду у журналі. Іншим було замовлення поштою. Обидва з цих методів дозволяють осягнути достатньо обмежену аудиторію.

Найпопулярнішим методом продавати ігри було продавати ігри в магазинах. Однак, полиці магазинів обмежені, тому вони зазвичай не продають ігри, в шансах продажу котрих невпевнені. Через це, розробникам нерідко потрібно мати видавника, бажано котрий має достатній вплив серед магазинів, щоб їх ігри опинилися на полицях.

В Японії є таке цікаве явище, як доджін (同人, doujin), що приблизно відповідає поняттю "самвидав", коли людина або мала група (доджін групи зазвичай називаються "колами" (サークル, circle)) виробля’ твір і власноруч продає. Історія японського самвидаву протягається з 19го сторіччя, але важливим поворотним моментом була поява регулярної конвенції Comiket у 1975 році, що фокусується на самвидавні роботи. Зазвичай саме на конвенціях доджін автори продають свої твори. Зазвичай твори можуть будь-якої форми, такої як комікси, книги, музика, відеоігри чи навіть анімація. Через те, що ці твори зазвичай не регулюють, вони можуть мати сексуальний контент, жорстокість та порушення авторських прав. Деякі достатньо відомі автори починали зі створення доджін творів, і деякі професійні автори також приймають участь у створенні доджін творів. Деякі доджін кола, наприклад, розробники візуальних новел Type-Moon, стають повноцінними компаніями.

Поява інтернету мала величезний вплив на процес розповсюдження ігор.

Деякі розробники почали впроваджувати свої сайти для продажу ігор. Це можна сприймати як продовження практики замовлення поштою, особливо враховуючи те, що найперші сайтів, через які можна було купити ігри, це сайти на яких користувач оформлював замовлення і йому надсилали копію гри поштою. З

часом, замість надсилання гри поштою її почали завантажувати через мережу інтернет.

Деякі ігри публікувалися на форумах. Зазвичай це були некомерційні ігри, які могли бути досить експериментальними.

Почали з’являтися вебсайти, що дозволяють користувачам публікувати свій контент, включаючи розроблені ігри. Зазвичай ігри, що видавалися через такі сайти, є некомерційними. Подібні сайти мають свою спільноту і культуру. Дані середовища непогані для розробників-початківців, адже у розповсюдження своїх робіт низький бар’єр входу і вони можуть стати частиною спільноти. Прикладами подібних сайтів є Newgrounds (1995) і itch.io (2013).

Також є інтернет-магазини, де користувач може купити ігри. Бар’єр входу більше, аніж у сайтах з фокусом на користувацький контент, але зазвичай інтернет-магазини набагато популярніші серед публіки, що готова покупати ігри. Через те, що, на відміну від фізичних магазинів, нема обмеження щодо місця на полицях та можливе розповсюдження будь-якої кількості копій, інтернет-магазини більш дружні до розробників без видавника. Прикладами інтернет-магазинів є Steam (2003), GOG.com (2008), DLSite (1996).

### 2.6. Регіональні особливості

В різних регіонах часто бувають свої особливі тенденції спричинені економічними, історичними, культурними, і, іноді, навіть медичними чинниками.

Але, нерідко стається те, що Феліпе Пепе (Felipe Pepe), бразильській ентузіаст історії відеоігор і автор книги "The CRPG Book", називає джентрифікацією історії відеоігор (Gentrification of the Video Game History, також колонізацією історії відеоігор, Colonização da História dos Video Games). Обговорення відеоігор зазвичай проводиться базуючись на домінантному наративі (що, в основному, базується на Американській перспективі і має дуже сильний фокус на комерції, зокрема це консольні ігри). Через це, досвід за

межами цього наративу зазвичай не береться до уваги. Таким чином, поширюється гомогенізована версія історія, що нехтує місцевими реаліями [80].

Нерідко нехтують частиною історії відеоігор, що не стосується офіційного виданих ігор, таких як модифікації ігор, піратство і неофіційні адаптації та продовження.

При тому, у Східній Європі піратство дуже сильно вплинуло на простір відеоігор.

Так, наприклад, Марцін Івінський (Marcin Iwiński), співзасновник польського видавника і розробника ігор CD Projekt Red, визнавав своє минуле як продавця іноземних ігор, здобутих без дозволу правовласника, і коментував, що велика кількість інших засновників польської ігрової індустрії також починали з піратства [81].

Однією з перших українських комп'ютерних ігор є гра «Пригоди піонерки Ксенії», що була розроблена у 1990 і випущена у 1991 році Юрієм Лесюком, Сергієм Самойленко і Владиславом Коломийцем. Вона являла собою модифіковану версію гри Captain Comic, що була розроблена у 1988 році американцем Майклом Деніо (Michael Denio) [82].

Першою грою, розробленою українською компанією GSC Game World, був WarCraft 2000: Nuclear Epidemic, випущений у 1998 році. Гра була неофіційним, неліцензованим і некомерційним продовженням серії ігор WarCraft від Blizzard. Рушій, створений для цієї гри, був також використаний у першій оригінальній грі компанії – «Козаки: Європейські війни» [83].

Український російськомовний сайт Anivisual був, скоріше за все, найбільшою спільнотою, присвяченою візуальним новелам у колишньому Радянському Союзі. Дана спільнота займалася як локалізацією і розповсюдженням візуальних новел, створених за межами Східної Європи, а також розробкою оригінальних творів і створенням навчальних матеріалів щодо розробки візуальних новел [84].

Однак, потрібно визнати: незважаючи на те, що сайт був українським, майже увесь контент був російськомовним, і велику частину спільноти складали

росіяни. На даному сайту було немало посилань на російські ресурси і сервіси, особливо Яндекс.Диск. Те, що сайт український було неочевидним до початку повномасштабного вторгнення у 2022 році, після якого організатори почали намагатися привертати увагу до війни і збирати гроші на підтримку України. На даний момент сайт закритий і функціонує лише у вигляді архіву.

В цілому, незважаючи на те, що Україна здобула незалежність ще у 1991 році, українська ігрова індустрія була тісно переплетена з російською.

Серія ігор Metro, одна з найбільш відомих серій ігор, що були розроблені в Україні, є адаптацією серії книг російського автора.

Велика кількість українських ігор видавалася російськими видавництвами навіть в Україні, наприклад, такі ігри як Анабіоз: Сон Розуму, Pirate Hunter. Somali Trap, Вівісектор: Звір Усередині тощо. Також нерідко перевага озвучування надавалася російській мові.

Дивно, українська компанія з продажу ігор «Світ Ігор» мала сайт, що до 2014 року був представлений українською мовою, але після 2014 року виключно російською [85, 86].

Можна зробити висновок, що і колонізаційні зв’язки, і сусідство з гегемоном нерідко залишають глибокі сліди на культурі, що можуть ще досить довго не зникати. Після початку повномасштабного вторгнення велика частина українських розробників почала набагато більше позиціонувати себе саме як українці.

### 2.7. Висновки до розділу 2

У другому розділі були проаналізовані впливи різних факторів на ігровий дизайн і культуру.

На ігровий дизайн і культуру впливають різні фактори, такі як цільова платформа, її технічні характеристики і можливості, пристрої вводу та виведення, носії даних, використовувані інструменти розробки і методи доставки, економічні, культурні та історичні чинники, тощо. При цьому, вплив нерідко є не

однобічним, і спосіб розробки ігор може впливати на вибір платформи і інструментів та на створення і розвиток цих платформ і інструментів.

Враховуючи, що у різних спільнотах, регіонах і країнах присутні різні економічні, культурні, історичні та інші чинники, логічно, і їх історія має свою специфіку. Тим не менш, присутня тенденція надавати пріоритет домінантному наративу, що фокусується лише на малому шматку історії і нехтує іншими перспективами.

# РОЗДІЛ 3. АНАЛІЗ НАБОРУ ДАНИХ ПРО ВІДЕОІГРИ

## 3.1. Огляд набору даних про відеоігри

З метою дослідження і моніторингу популярності відеоігор проаналізуємо масив даних, що містить інформацію про відеоігри, включно з їх жанрами, рейтингом, кількістю рекомендацій тощо.

Набір даних, наведений у таблиці 3.1 завантажено з джерела відкритих даних – міжнародної платформи Kaggle, що використовується для спілкування і співпраці спеціалістів з Data Science. Вихідний набір даних сформований на базі даних з сайту rawg.io [87].

Таблиця 3.1 — Вихідний набір даних

| Поле | Опис |
|---|---|
| id | Унікальний числовий індекс гри |
| slug | Унікальний текстовий індекс гри |
| name | Назва гри |
| metacritic | Рейтинг гри на Metacritic |
| released | Дата випуску гри |
| tba | Чи дата випуску ще не анонсована |
| updated | Дата останнього оновлення |
| website | Вебсайт |
| rating | Рейтинг гри користувачами rawg.io |
| rating_top | Найбільший рейтинг |
| playtime | Час проходження гри |
| achievements_count | Кількість досягнень у грі |
| ratings_count | Кількість рейтингів |
| suggestions_count | Кількість рекомендацій користувачами |

| Поле | Опис |
|---|---|
| | rawg.io |

Продовження таблиці

| Поле | Опис |
|---|---|
| game_series_count | Кількість ігор у серії |
| reviews_count | Кількість оглядів користувачами rawg.io |
| platforms | Платформи, на котрих гра була випущена |
| developers | Розробники гри |
| genres | Жанри гри |
| publishers | Видавники гри |
| esrb_rating | Віковий рейтинг ESRB |
| added_status_yet | Кількість користувачів rawg.io, що поставили грі статус "Not player" |
| added_status_owned | Кількість користувачів rawg.io, що поставили грі статус "Owned" |
| added_status_beaten | Кількість користувачів rawg.io, що поставили грі статус "Completed" |
| added_status_toplay | Кількість користувачів rawg.io, що поставили грі статус "To play" |
| added_status_dropped | Кількість користувачів rawg.io, що поставили грі статус "Played but not beaten" |
| added_status_playing | Кількість користувачів rawg.io, що поставили грі статус "Playing" |

### 3.2. Проведення аналізу набору даних про відеоігри

Аналіз вихідних даних проводився з використанням інтерактивного середовища Jupyter. Програмний код для проведення аналізу набору даних наведений у Додатку А.

Розглянемо задачі аналізу даних для наведеного у таблиці 3.1 вихідного масиву даних.

Задача 1. Аналіз перетину різних жанрів відеоігор.

Розглянемо стовпчикову діаграму, що відображує залежність кількості ігор від жанру (рис. 3.1). Жовтим кольором показана кількість відеоігор, які відносяться до декількох жанрів, а червоним кольором показана кількість ігор, які відносяться до одного жанру.

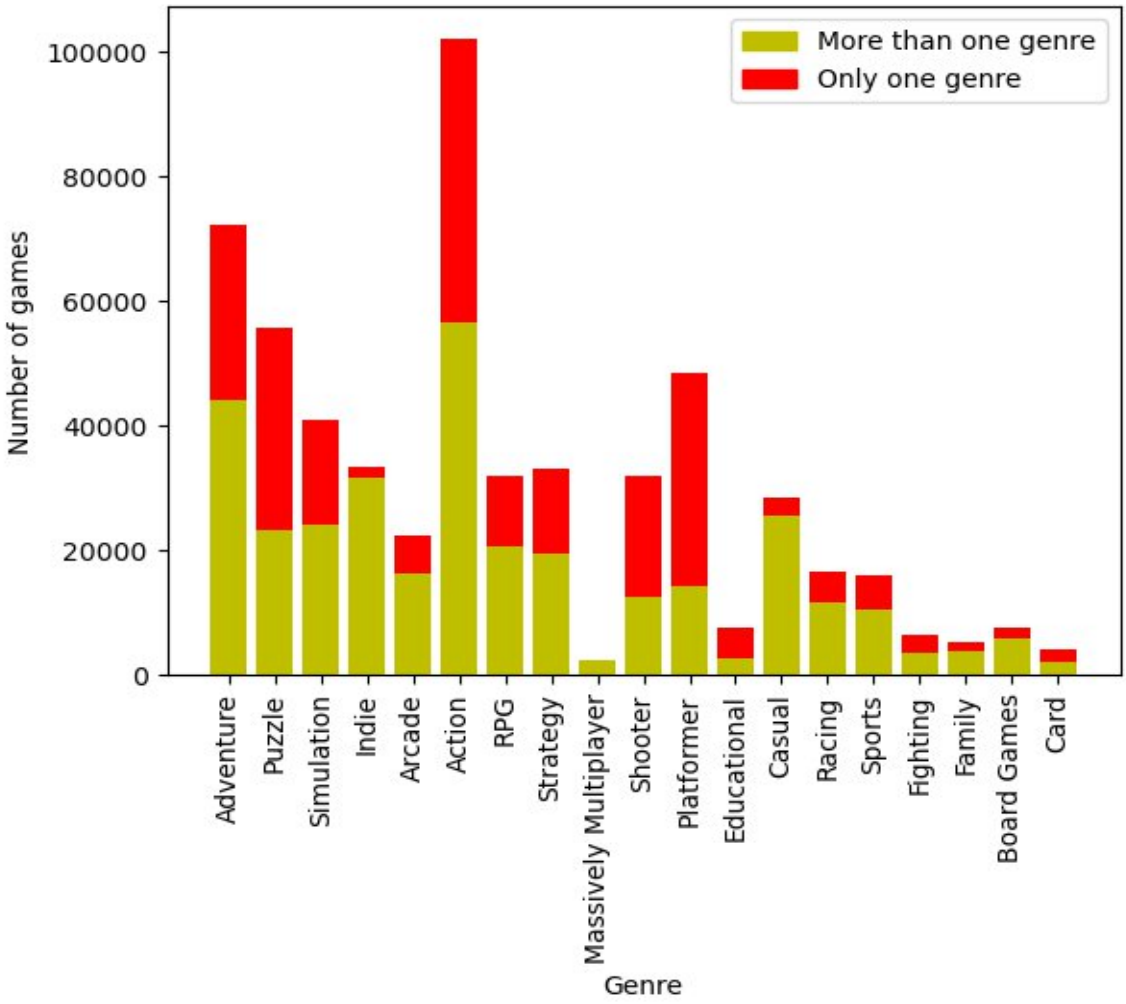



Рисунок 3.1 — Співвідношення ігор з одним вказаним жанром до ігор з декількома вказаними жанрами

Аналіз наведеної діаграми показав, що розподіл ігор, що належать до одного жанру, і ігор, що належать до декількох жанрів не рівномірний. Деякі жанри, наприклад, платформери, шутери, головоломки, мають більше ігор у яких вказаний лише даний жанр, в той час як ігри, у яких у жанрі вказаний виключно Massively Multiplayer, практично відсутні. Такі жанри, як Indie, Casual, Family, Board Games зазвичай вказані разом з іншими жанрами.

З цього можна зробити висновок, що деякі жанри краще комбінуються з іншими. Такі жанри як Indie, Casual, Family, Board Games і Massively Multiplayer, зазвичай є не єдиним жанром гри. Гра Indie визначається незалежністю розробника від великих компаній та належністю до субкультури незалежно розроблених ігор; гра Casual  фокусом на цільову аудиторію за межами існуючих фанатів відеоігор; гра Family  тим, що ігри даного жанру підходять для дітей як за тематикою і сюжетом, так і за ігровим процесом; гра Board Games визначається тим, що ігри даного жанру адаптують або нагадують настільні ігри, а Massively Multiplayer  тим, що у грі присутні онлайн компоненти і велика кількість взаємодій між різними гравцями. Дані жанри визначаються не ігровим процесом або генеалогією, і тому їх недостатньо для категоризації гри.

Наявність великої кількості ігор, що належать до декількох жанрів, показує доречність подальших досліджень щодо тенденцій за комбінаціями жанрів.

Задача 2. Аналіз кількості ігор за комбінаціями жанрів.

Вказані кількості ігор зі заданими комбінаціями жанрів (рис. 3.2) і вказані процентні відношення кількості ігор зі заданою комбінацією жанрів до кількості ігор, у котрих вказаний хоча б один зі заданих жанрів (рис. 3.3).

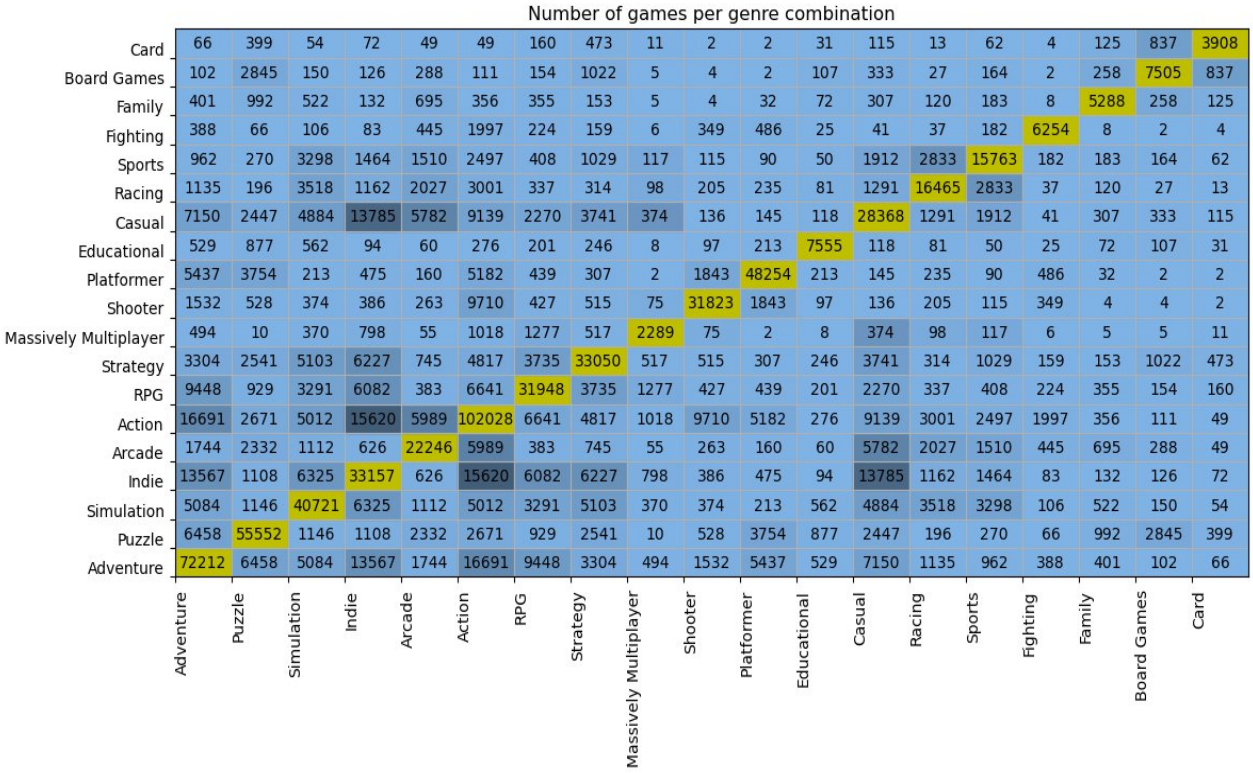



Рисунок 3.2 — Кількість ігор з заданою комбінацією жанрів

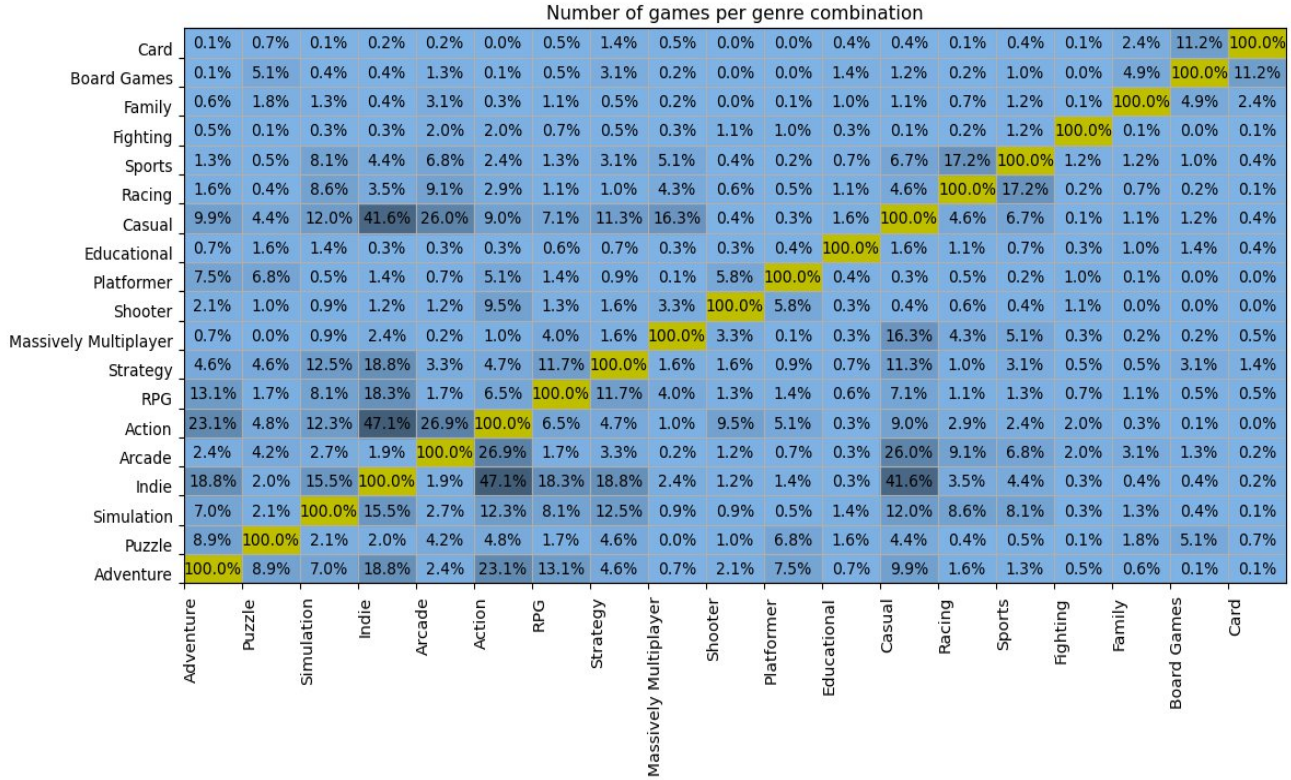



Рисунок 3.3 — Процентне відношення ігор з заданою комбінацією жанрів

Аналіз наведених діаграм показав, що багато жанрів комбінуються один з одним. Наприклад, 47.1% ігор з назвами «Action» або «Indie», належать двом жанрам, що показує, що між іграми «Action» і «Indie» є великий перетин. Аналогічно з іграми «Casual» і «Indie», у яких перетин 41.6%. При цьому, перетин ігор «Casual» і «Action» становить лише 9%. Попри стереотипи, перетин між іграми «Indie» і «Platformer» досить малий 1.4%.

Таким чином, були визначені частоти комбінацій різних жанрів. Завдяки цьому, можна визначати частоту комбінацій різних жанрів і використовувати при проведенні наступних досліджень.

Задача 3. Аналіз середнього рейтингу гри за комбінаціями жанрів.

Показані середні рейтинги ігор за комбінаціями жанрів (рис. 3.4).

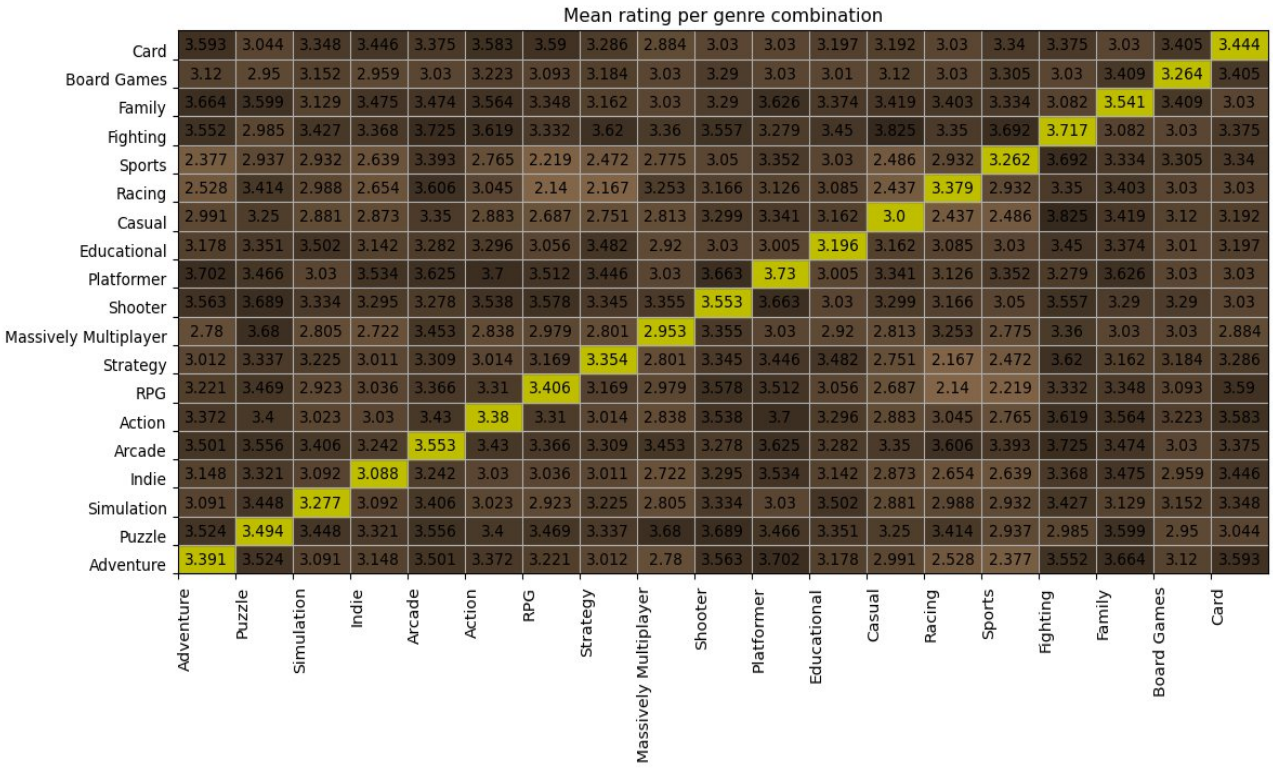



Рисунок 3.4 — Середній рейтинг ігор з заданою комбінацією жанрів

Середні рейтинги у ігор різних жанрів відрізняються несуттєво. Серед комбінацій жанрів з досить низьким рейтингом більшість ігор мають рідкі комбінації жанрів (наприклад, жанри Strategy і Racing, з середнім рейтингом 2.167 і перетином 1%). З іншого боку, рідка комбінація жанрів не обов’язково має

низький рейтинг (наприклад, це жанри Fighting і Casual, з перетином 0.1% і середнім рейтингом 3.825). Можливо, у більш рідкісних комбінаціях жанрів через малу кількість ігор окремі ігри мають більше впливу на середній показник. Але такі жанри, як Racing і Sports мають більше низьких середніх рейтингів, в той час як Fighting має більше високих середніх рейтингів. Racing і Sports мають мало перетинів з іншими жанрами, в той час як в жанрі Fighting написано мало ігор.

Можна зробити висновок, що вплив жанрів на рейтинг гри є порівняно незначним.

Задача 4. Аналіз середньої кількості рекомендацій за комбінаціями жанрів.

Показана середня кількість рекомендацій ігор з заданими комбінаціями жанрів (рис. 3.5).

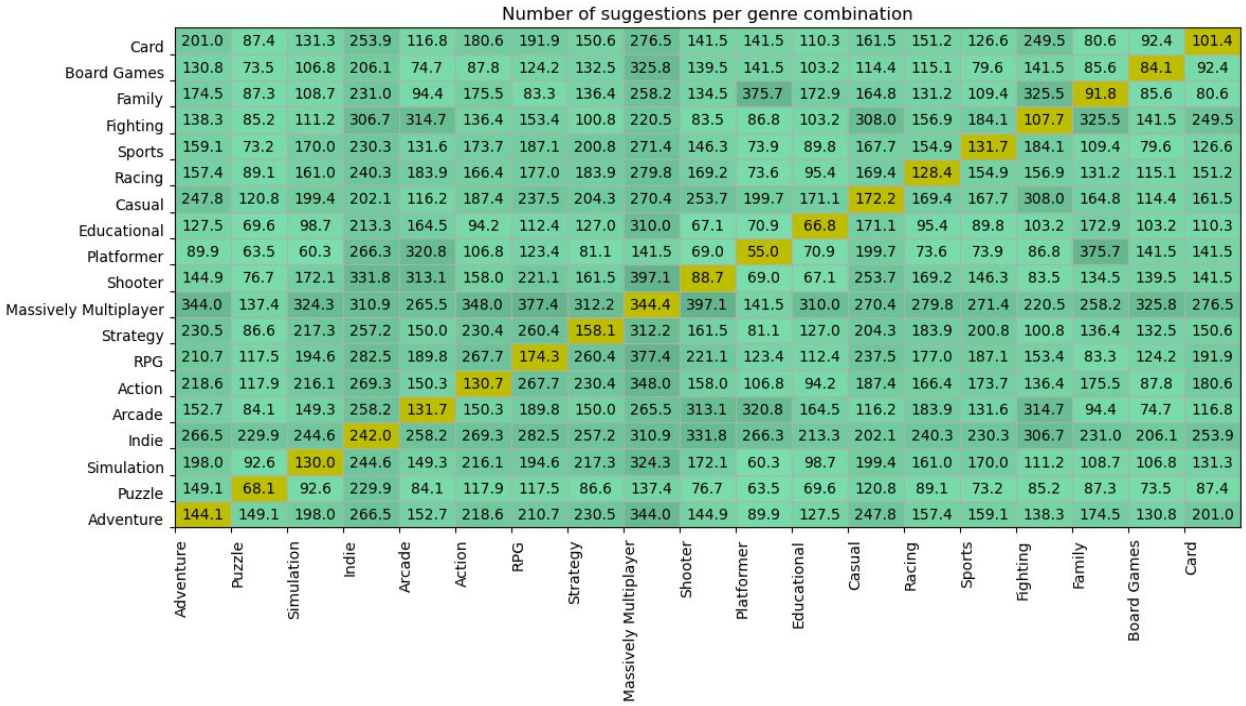



Рисунок 3.5 — Середня кількість рекомендацій ігор з заданою комбінацією жанрів

При аналізі можна побачити декілька цікавих тенденцій. Наприклад, кількість рекомендацій набагато вище, якщо один з жанрів є Massively Multiplayer (виключеннями є комбінації з Platformer і Puzzle). Також кількість рекомендацій вище у Indie ігор. Окрім цього, у комбінацій Arcade з жанрами Shooter, Platformer,

Fighting також вище кількість рекомендацій. Просліджуються також досить високі кількості рекомендацій у таких комбінацій, як Family і Platformer, Casual і Fighting, Fighting і Family.

Висока кількість рекомендацій у Massively Multiplayer іграх може підтверджувати концепцію, що у що завгодно веселіше грати з друзями. Окрім того, Massively Multiplayer ігри дещо залежні від кількості гравців, тому в їх інтересах заохочувати інших гравців до гри.

Високу кількість рекомендацій у Indie іграх можна пояснити культурними особливостями. Indie ігри за визначенням розробляються порівняно малими командами, і зазвичай не мають підтримки великих видавництв. Через це вони дещо обмежені у можливостях маркетингу, і їх популярність нерідко залежить від гравців, котрі поширяють між собою Indie ігри, які їм подобаються. Через це, гравці у Indie ігри нерідко відчувають більше відповідальності щодо поширення ігор, які їм подобаються, аніж гравці у ігри від великих видавництв.

У вихідному наборі даних 32 ігри з комбінацією Family і Platformer, 41 гра з комбінацією Fighting і Casual і 8 ігор з комбінацією Family і Fighting. Вірогідно, що висока кількість рекомендацій у даних комбінацій через малу кількість ігор, і окремі ігри можуть сильно впливати на середнє значення рекомендацій, але також можливо, що велика кількість рекомендацій цих ігор показує незадоволений попит на дані комбінації жанрів.

Можна зробити висновок, що комбінація жанрів може мати достатньо високий вплив на кількість рекомендацій гри. Також присутні деякі комбінації жанрів з малою кількістю ігор але порівняно високою кількістю рекомендацій, що може позначати присутність недорозкритої ніші.

### 3.3. Висновки до розділу 3

У третьому розділі здійснено дослідження тенденцій серед комбінацій жанрів відеоігор, частоти використання жанрів, розглянуто середній рейтинг і середню кількість рекомендацій ігор певної комбінації жанрів. Проаналізовано

набір даних про відеоігри, що включає їх жанри, рейтинг, кількість рекомендацій тощо. Особливу увагу приділено виявленню тенденцій серед комбінації жанрів відеоігор. Виявлено, що між деякими жанрами набагато більше перетинів ніж між іншими, наприклад, Indie і Action чи Indie і Casual (при цьому, між Action і Casual перетин досить незначний). Середній рейтинг ігор зазвичай не дуже відрізняється між різними комбінаціями жанрів. З іншого боку, середня кількість рекомендацій може суттєво відрізнятися за комбінаціями жанрів. Деякі жанри, такі як Indie і Massively Multiplayer, в середньому мають набагато більше рекомендацій користувачів. Деякі комбінації жанрів, що зустрічаються досить рідко, мають високу середню кількість рекомендацій користувачів. Дану тенденцію можна пояснити сильним впливом окремих ігор на середню кількість рекомендацій через малу кількість ігор з даними комбінаціями, або це може бути через присутність попиту на дані комбінації зі занадто малою пропозицією, тобто недорозкритої ніші.

Наприклад, ігри жанру Fighting мають репутацію занадто складних для гравців, які ще не грали у ігри жанру Fighting, що ускладнює їх популяризацію.

Висока середня кількість комбінації жанрів Fighting і Casual при малій кількості представників даної комбінації може сприйматися як підтвердження того, що є попит на ігри у жанрі Fighting, розраховані на аудиторію за межами існуючих фанатів жанрів, і що пропозиція занадто мала.

Звичайно, має сенс зауважити, що занадто великий фокус на тенденції у великому наборі даних може призвести до нехтування тенденціями, що складно представити у наборі даних. Через це, аналіз великого набору даних не варто використовувати як єдину методику дослідження відеоігор, але аналіз великого набору даних дозволяє підтверджувати або спростувати гіпотези та виявити неочевидні тенденції.

Хоча комбінація жанрів зазвичай має малий вплив на рейтинг гри, що показує, що комбінація жанрів не є головним у питанні наскільки гравцю сподобається гра. В той самий час комбінація жанрів може мати достатньо високий вплив на кількість рекомендацій гри, що показує, що комбінації жанрів

можуть мати достатньо сильний вплив на те, наскільки гравець буде зацікавлений щоб грати у гру і рекомендувати її іншим. Окрім того, присутні деякі комбінації жанрів, що мають диспропорційно велику кількість рекомендацій порівняно з кількістю ігор з даною комбінацією жанрів, через що можна припустити, що присутній попит на дані комбінації жанрів зі занадто малою пропозицією, і їх можна вважати недорозкритою нішею.

# РОЗДІЛ 4. СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ РИНКУ КОМП'ЮТЕРНИХ ІГОР

## 4.1. Засоби розробки і реалізації інформаційної системи

Інформаційна система була розроблена у вигляді додатку за допомогою мови програмування Python, графічного фреймворку Qt і системи керування базами даних SQLite.

Мова програмування Python була обрана через простоту та лаконічність її синтаксису, що дозволяє швидко проводити прототипування і розробку додатку, та широкий вибір бібліотек та фреймворків.

Графічний фреймворк Qt був обраний через свою перевіреність (серед програм, що були розроблені використовуючи Qt, є Krita, OBS, Autodesk Maya, тощо), простоту використання і можливість інтеграції у Python.

Система керуваннями базами даних SQLite обрана через можливість локального збереження бази даних на диску, перевіреність (SQLite використовується у Firefox, Chrome, Android, тощо), простоту використання і можливість інтеграції у Python.

Для інтеграції Qt в Python використовувалася бібліотека PySide, а для інтеграції SQLite в Python використовувалася бібліотека sqlite3.

Код додатку представлений у додатку Б.

## 4.2. Структура даних інформаційної системи

Інформаційна система використовує набір даних, що були взяті зі популярного інтернет-магазину ігор Steam за допомогою Steam Spy API [88]. Набір даних був завантажений у форматі JSON і потім переведений у формат базі даних SQLite. Код для завантаження набору даних у форматі JSON представлений у додатку В, а код для переведення у формат бази даних SQLite представлений у додатку Г.

Структура бази даних (рис. 4.1) складається з 10 таблиць.

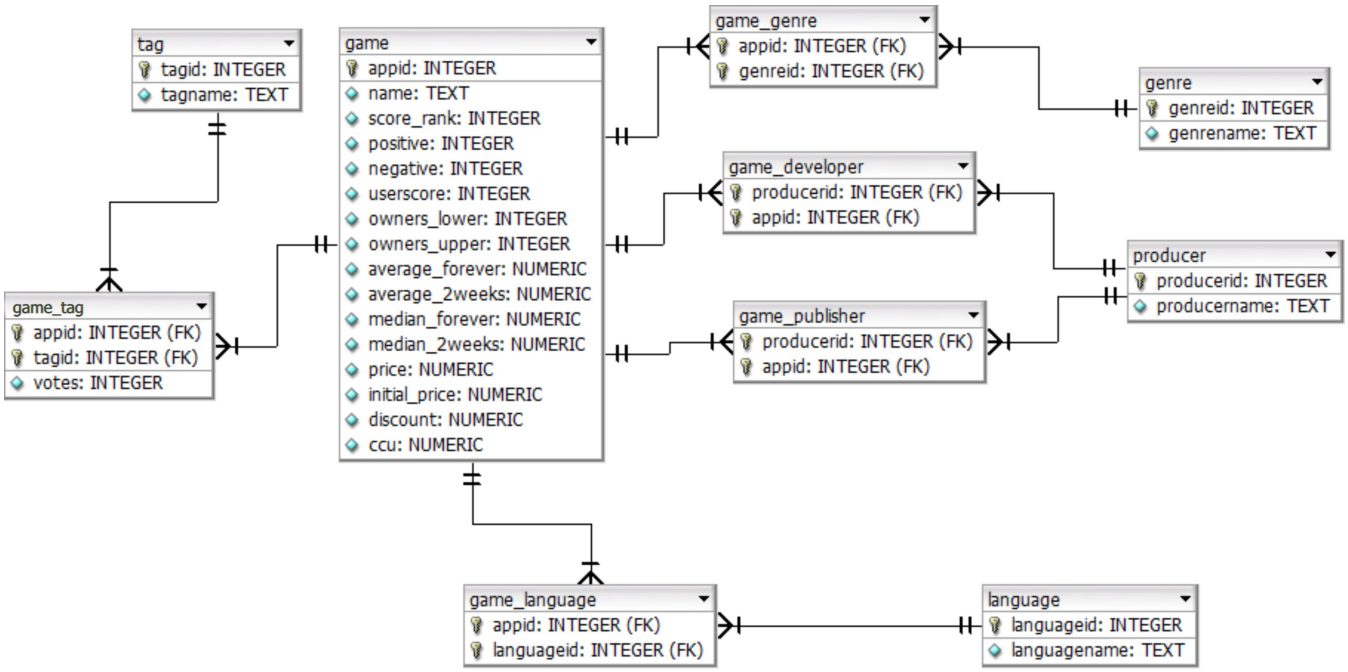



Рисунок 4.1 — Модель бази даних

Розглянемо таблицю 4.1, що включає більшість даних про гру.

Таблиця 4.1 — Структура таблиці game

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| name | Назва гри |
| score_rank | Рейтинг гри на основі відгуків користувачів |
| positive | Кількість позитивних відгуків |
| negative | Кількість негативних відгуків |
| userscore | Процентне відношення позитивних відгуків до загальної кількості відгуків |
| owners_lower | Верхня межа кількості власників додатку |
| owners_upper | Нижня межа кількості власників додатку |
| average_forever | Середній час, проведений у грі (за весь час) |
| average_2weeks | Середній час, проведений у грі (за останні два тижня) |

Продовження таблиці

| Поле | Опис |
|---|---|
| median_forever | Медіана часу, проведеного у грі (за весь час) |
| median_2weeks | Медіана часу, проведеного у грі (за останні два тижня) |
| price | Ціна |
| initial_price | Оригінальна ціна |
| discount | Знижка у процентах |
| ccu | Найбільша кількість одночасно активних користувачів у минулий день |

Розглянемо таблицю 4.1, що описує жанри, які були вказані видавцями гри

Таблиця 4.2 — Структура таблиці genre

| Поле | Опис |
|---|---|
| genreid | Індекс жанру |
| genrename | Назва жанру |

Розглянемо таблицю 4.3, що описує теги, котрі були вказані гравцями.

Таблиця 4.3 — Структура таблиці tag

| Поле | Опис |
|---|---|
| tagid | Індекс тегу |
| tagname | Назва тегу |

Розглянемо таблицю 4.4, що описує виробників гри (розробників та видавників).

Таблиця 4.4 — Структура таблиці producer

| Поле | Опис |
|---|---|
| producerid | Індекс виробника |
| producername | Назва або ім’я виробника |

Розглянемо таблицю 4.5, що описує мови, якими доступна гра.

Таблиця 4.5 — Структура таблиці language

| Поле | Опис |
|---|---|
| languageid | Індекс мови |
| languagename | Назва мови |

Розглянемо таблицю 4.6, що описує відношення між грою та жанром.

Таблиця 4.6 — Структура таблиці game_genre

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| genreid | Індекс жанру |

Розглянемо таблицю 4.7, що описує відношення між грою та тегом.

Таблиця 4.7 — Структура таблиці game_tag

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| tagid | Індекс тегу |
| votes | Кількість користувачів, що вказали даний тег |

Розглянемо таблицю 4.8, що описує відношення між грою та розробником.

Таблиця 4.8 — Структура таблиці game_developer

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| producerid | Індекс розробника |

Розглянемо таблицю 4.9, що описує відношення між грою та видавником.

Таблиця 4.9 — Структура таблиці game_publisher

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| producerid | Індекс видавника |

Розглянемо таблицю 4.10, що описує відношення між грою та мовою.

Таблиця 4.10 — Структура таблиці game_language

| Поле | Опис |
|---|---|
| appid | Індекс гри |
| languageid | Індекс мови |

### 4.3. Інтерфейс користувача інформаційної системи

Важливим пріоритетом при створенні додатку було забезпечення користувача легким, інтуїтивно зрозумілим графічним інтерфейсом.

Додаток дозволяє формувати вибірки даних і представляти їх у зручному вигляді.

Інтерфейс містить два основні елементи: селектор (selector) і візуалізатор (visualiser).

Селектор відповідає за умови для формування набору даних. У формі завжди присутній основний селектор, а селектори типів "AND" і "OR" можуть

мати дочірні. Селектор можна інвертувати, натиснувши на галочку "invert", і тоді умова буде протилежна вказаній.

У селектора також є функція "return_value" для повернення структури даних, що описує задану умову. Функція "buildQuery" формує на базі отриманої структури даних частину запиту SQL, що може бути розміщена у секції "WHERE".

Візуалізатори зазвичай мають параметри, що може обирати користувач і кнопку "Select Data", що формує запит SQL по типу візуалізатора, його параметрів та функції buildQuery на базі умови основного селектора. На основі результатів запиту створюється представлення, вигляд якого залежить від типу візуалізатора.

Селектор може бути таких типів: "AND", "OR", "COMPARISON", "GENRE", "DEVELOPER", "PUBLISHER", "LANGUAGE", "TAG", "NONE", "ALL".

Селектори типів "AND" (рис 4.2), і "OR" (рис. 4.3) мають декілька дочірніх селекторів, котрі можна добавляти або видаляти. Селектор типу "AND" представляє умову, що всі умови дочірніх селекторів виконуються, а селектор типу "OR" представляє умову, що одна з умов дочірніх селекторів виконується.

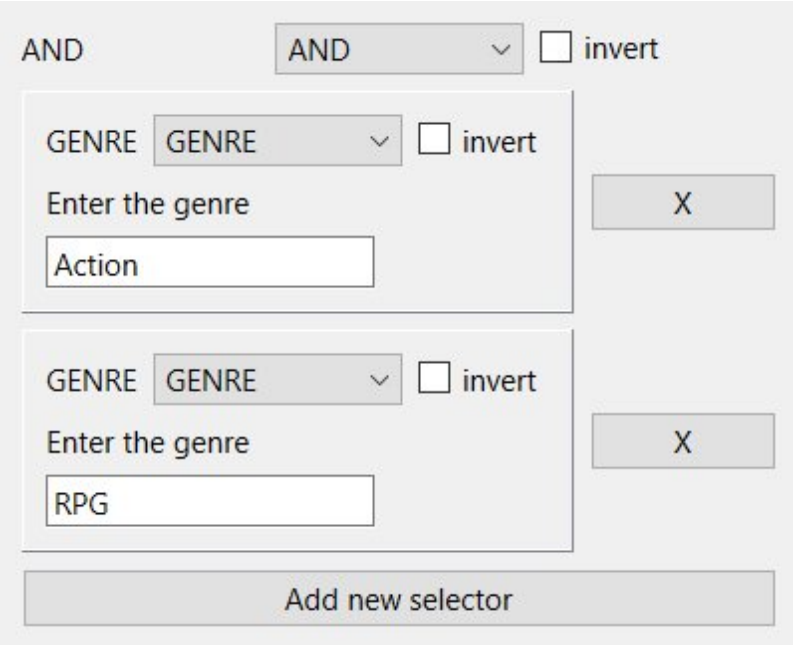



Рисунок 4.2 — Селектор типу "AND"

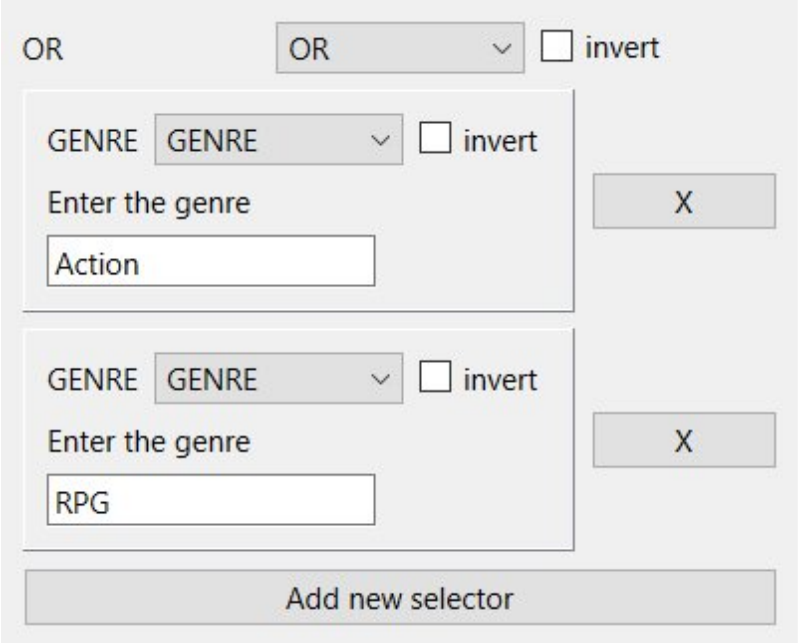



Рисунок 4.3 — Селектор типу "OR"

Селектор типу "COMPARISON" (рис. 4.4) порівнює поле із заданим числовим значенням. Присутні такі порівняння: "more than", "less than", "equal to", "in range". Усі порівняння, окрім "in_range", можна порівняти з одним числовим значенням, в той час як "in_range" може порівнюватись з двома значеннями (рис. 4.5).

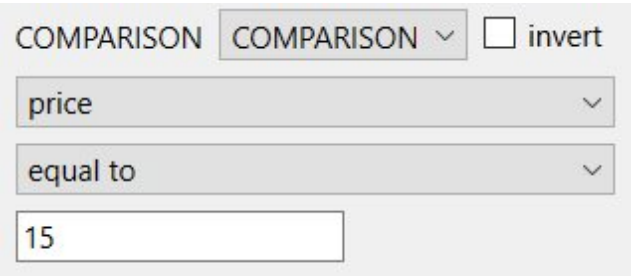



Рисунок 4.4 — Селектор типу "COMPARISON"

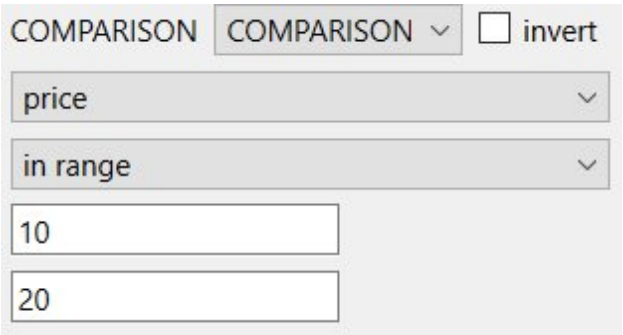
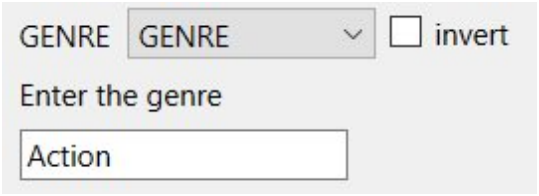



Рисунок 4.5 — Селектор типу "COMPARISON", порівняння "in range"

Селектори типів "GENRE", "DEVELOPER", "PUBLISHER", "LANGUAGE", "TAG" (рис. 4.6) дозволяють перевіряти чи має гра відношення із заданим представником даної категорії (наприклад, чи належить вона до даного жанру, чи була розроблена даними розробниками). При вводі назви представника




пропонуються існуючі назви жанрів.

Рисунок 4.6 — Селектор типу "GENRE"

Селектор типу "NONE" (рис. 4.7) завжди повертає незадоволену умову, а


селектор типу "ALL" (рис. 4.8) завжди повертає задоволену умову.

Рисунок 4.7 — Селектор типу "NONE"

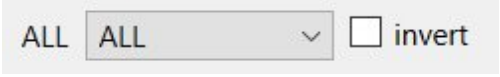



Рисунок 4.8 — Селектор типу "ALL"

Візуалізатор може бути таких типів: "Games Table", "Bar Chart", "Relation Table", "None".

Візуалізатор типу "Games Table" (рис. 4.9) дозволяє виводити дані про гру у вигляді таблиці. Зверху присутній список полів, що будуть виводитися, котрі можна обирати або прибирати за допомогою галочок. Якщо натиснути на кнопку "Select data", то буде сформований запит до бази даних на основі значень обраних полів та умови основного селектора. Повернені дані будуть виведені у таблиці. Максимально виводиться 10 записів, але передбачена можливість перегорнути сторінку таблиці. Кнопки '>' і '<' дозволяють перегорнути сторінку на наступну чи попередню; кнопки '>>' і '<<' дозволяють перейти на першу чи останню сторінку; в поле сторінки можна ввести значення для швидкого переходу на сторінку що відповідає цьому значенню.

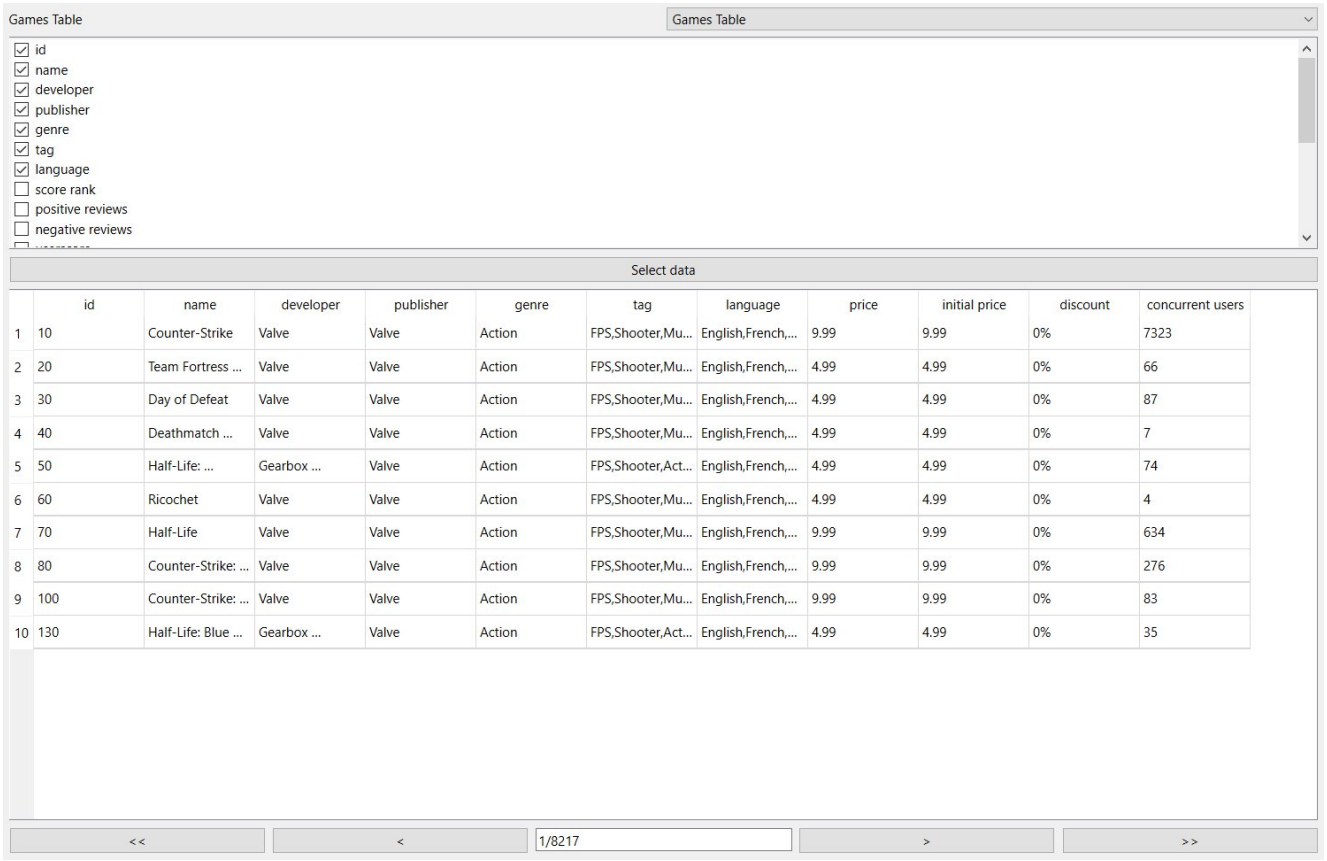



Рисунок 4.9 — Селектор типу "Game Table"

Візуалізатор типу "Bar Chart" (рис. 4.10) дозволяє виводити агреговані дані одного з полів за обраною категорією у вигляді стовпчастої діаграми. Можна обрати тип функції агрегації ("mean", "maximum", "minimum", "sum", "count"), поле ("positive reviews", "negative reviews", "average playtime (forever)", "average

playtime (2 weeks)", "median playtime (forever)", "median playtime (2 weeks)", "price", "initial price", "discount", "concurrent users"), категорію ("genre", "tag", "developer", "publisher", "language"). Якщо обрати функцію агрегації "count", то пропаде можливість вибору поля (рис. 4.11).

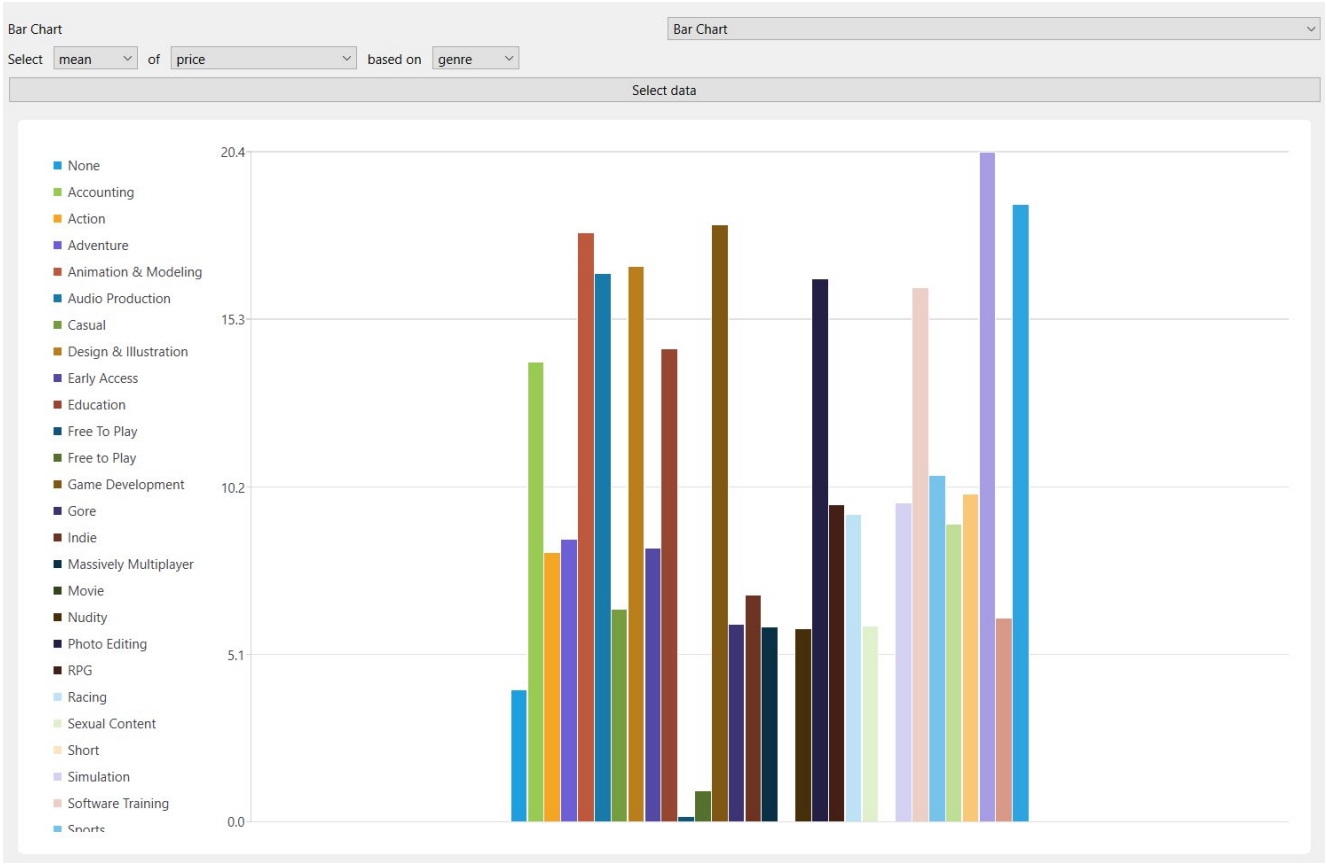



Рисунок 4.10 — Селектор типу "Bar Chart"

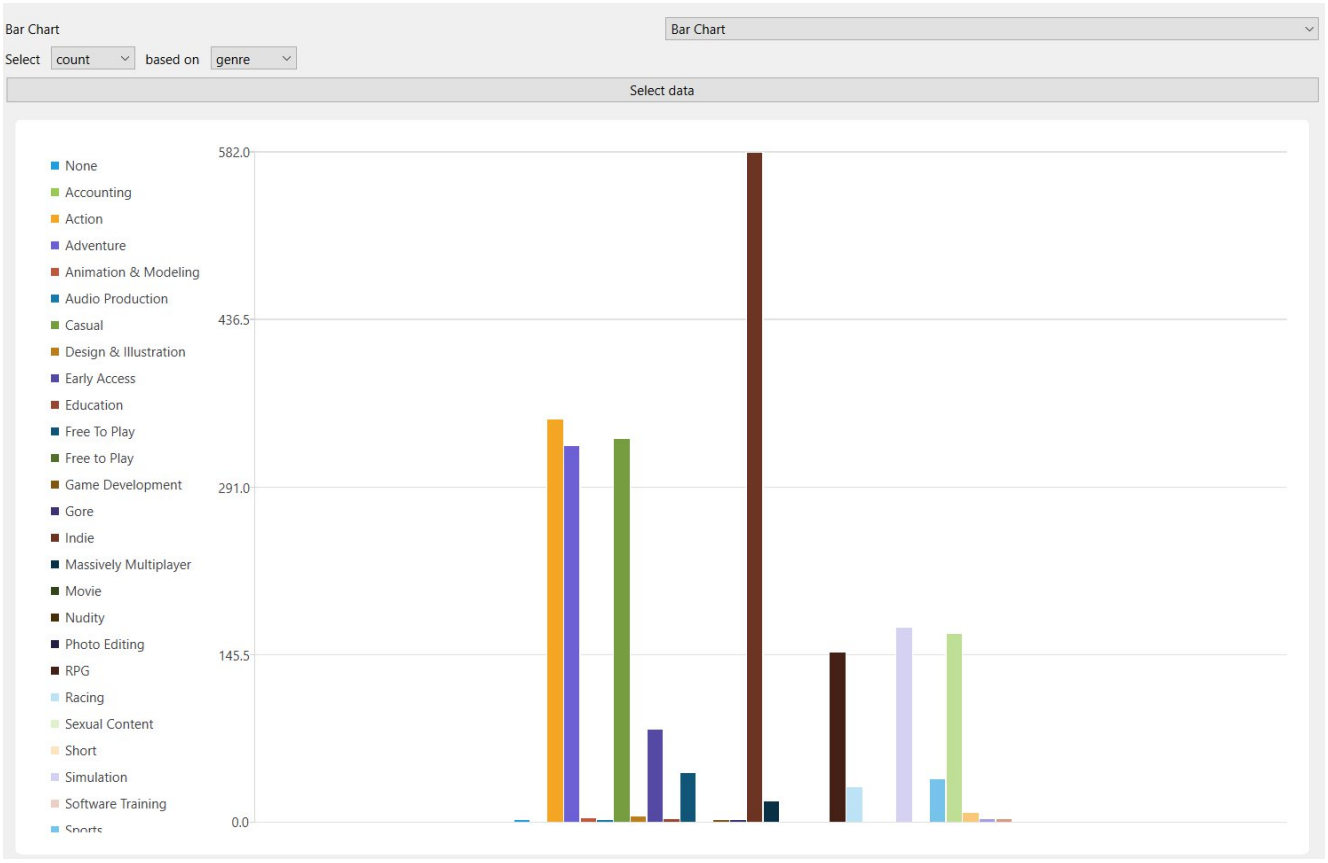


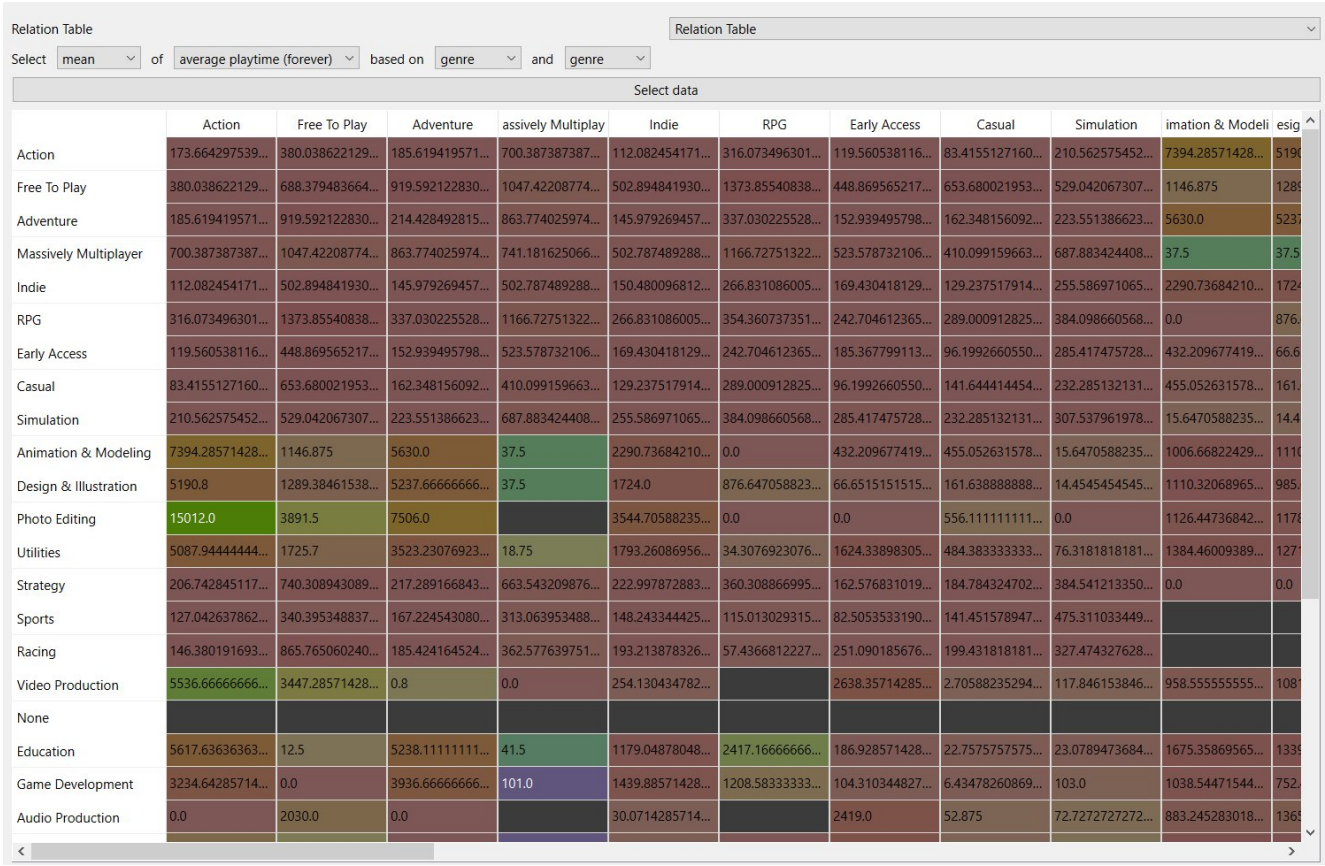

Рисунок 4.11 — Селектор типу "Bar Chart", функція агрегації "count"

Візуалізатор типу "Relation Table" (рис. 4.12) дозволяє виводити агреговані дані одного з полів за двома обраними категоріями у вигляді таблиці. Можна обрати тип функції агрегації ("mean", "maximum", "minimum", "sum", "count"), поле ("positive reviews", "negative reviews", "average playtime (forever)", "average playtime (2 weeks)", "median playtime (forever)", "median playtime (2 weeks)", "price", "initial price", "discount", "concurrent users"), категорії ("genre", "tag", "developer", "publisher", "language").

Відтінок фону клітини залежить від відношення результату функції агрегації до максимального значення обраного поля у перетину даних представників заданих категорій (чим ближче до блакитного, тим більше результат функції агрегації відносно до максимального значення; якщо у фона фіолетовий відтінок, то результат функції агрегації більше за максимальне значення). Насиченість фону клітинки залежить від відношення результату функції агрегації, що записаний у даній клітини, до максимального результату

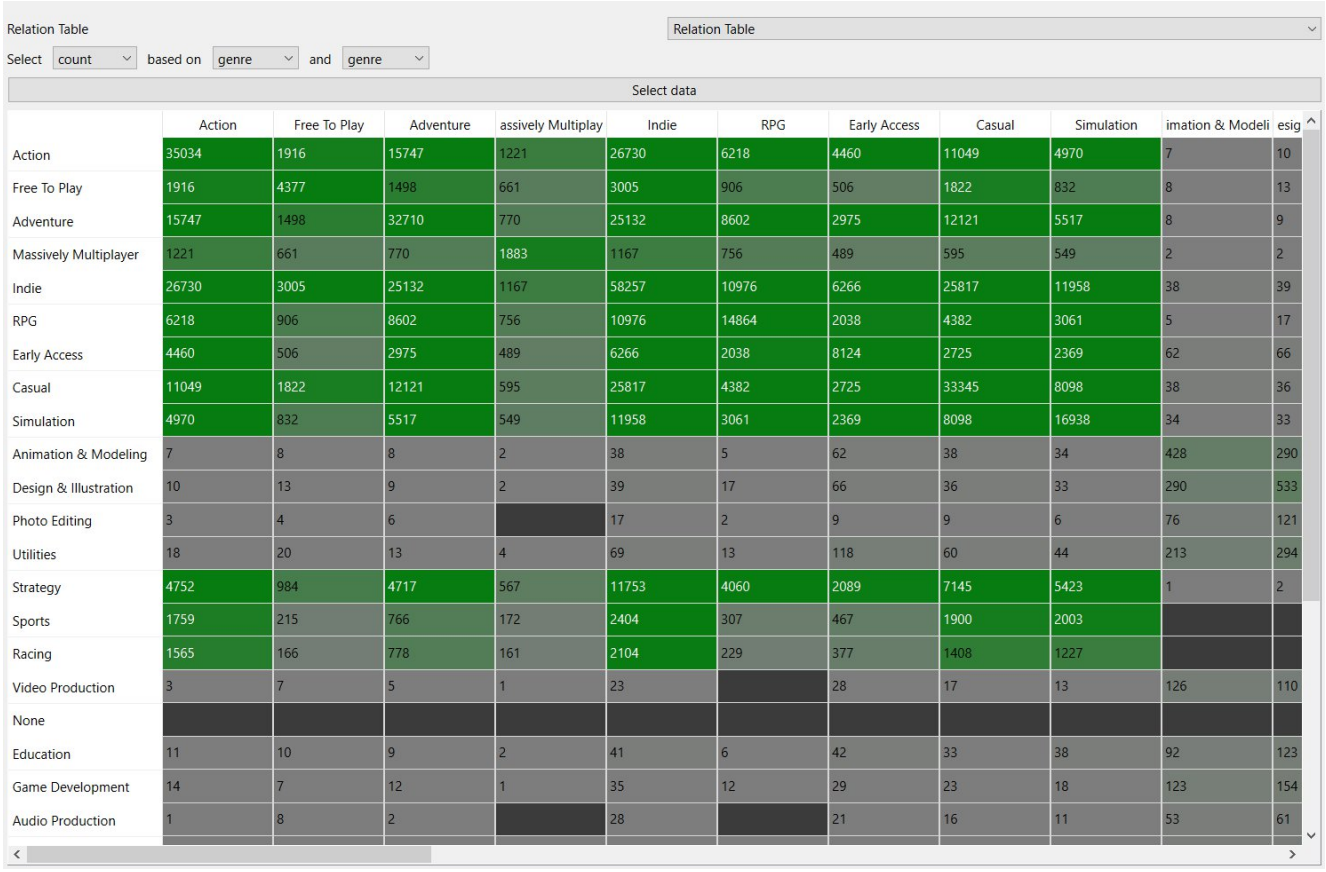
функції агрегації, що записаний у таблиці (чим насиченіше колір, тим більше результат у даній клітини відносно до максимального результату).

Якщо обрати функцію агрегації "count", то пропаде можливість вибору поля (рис. 4.13). Окрім того, відтінок фону клітин завжди зелений, а насиченість фону залежить від відношення кількості ігор у перетині даних представників заданих категорій до середнього значення кількості ігор у всій таблиці (чим насиченіше колір, тим більше результат у даній клітини відносно до середнього результату).




| | Action | Free To Play | Adventure | assively Multiplay | Indie | RPG | Early Access | Casual | Simulation | imation & Modeli | esig |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Action | 173.664297539... | 380.038622129... | 185.619419571... | 700.387387387... | 112.082454171... | 316.073496301... | 119.560538116... | 83.4155127160... | 210.562575452... | 7394.28571428... | 519 |
| Free To Play | 380.038622129... | 688.379483664... | 919.592122830... | 1047.42208774... | 502.894841930... | 1373.85540838... | 448.869565217... | 653.680021953... | 529.042067307... | 1146.875 | 128 |
| Adventure | 185.619419571... | 919.592122830... | 214.428492815... | 863.774025974... | 145.979269457... | 337.030225528... | 152.939495798... | 162.348156092... | 223.551386623... | 5630.0 | 523 |
| Massively Multiplayer | 700.387387387... | 1047.42208774... | 863.774025974... | 741.181625066... | 502.787489288... | 1166.72751322... | 523.578732106... | 410.099159663... | 687.883424408... | 37.5 | 37.5 |
| Indie | 112.082454171... | 502.894841930... | 145.979269457... | 502.787489288... | 150.480096812... | 266.831086005... | 169.430418129... | 129.237517914... | 255.586971065... | 2290.73684210... | 172 |
| RPG | 316.073496301... | 1373.85540838... | 337.030225528... | 1166.72751322... | 266.831086005... | 354.360737351... | 242.704612365... | 289.000912825... | 384.098660568... | 0.0 | 876 |
| Early Access | 119.560538116... | 448.869565217... | 152.939495798... | 523.578732106... | 169.430418129... | 242.704612365... | 185.367799113... | 96.1992660550... | 285.417475728... | 432.209677419... | 66.6 |
| Casual | 83.4155127160... | 653.680021953... | 162.348156092... | 410.099159663... | 129.237517914... | 289.000912825... | 96.1992660550... | 141.644414454... | 232.285132131... | 455.052631578... | 161 |
| Simulation | 210.562575452... | 529.042067307... | 223.551386623... | 687.883424408... | 255.586971065... | 384.098660568... | 285.417475728... | 232.285132131... | 307.537961978... | 15.6470588235... | 14.4 |
| Animation & Modeling | 7394.28571428... | 1146.875 | 5630.0 | 37.5 | 2290.73684210... | 0.0 | 432.209677419... | 455.052631578... | 15.6470588235... | 1006.66822429... | 111 |
| Design & Illustration | 5190.8 | 1289.38461538... | 5237.66666666... | 37.5 | 1724.0 | 876.647058823... | 66.6515151515... | 161.638888888... | 14.4545454545... | 1110.32068965... | 985 |
| Photo Editing | 15012.0 | 3891.5 | 7506.0 | | 3544.70588235... | 0.0 | 0.0 | 556.111111111... | 0.0 | 1126.44736842... | 117 |
| Utilities | 5087.94444444... | 1725.7 | 3523.23076923... | 18.75 | 1793.26086956... | 34.3076923076... | 1624.33898305... | 484.383333333... | 76.3181818181... | 1384.46009389... | 127 |
| Strategy | 206.742845117... | 740.308943089... | 217.289166843... | 663.543209876... | 222.997872883... | 360.308866995... | 162.576831019... | 184.784324702... | 384.541213350... | 0.0 | 0.0 |
| Sports | 127.042637862... | 340.395348837... | 167.224543080... | 313.063953488... | 148.243344425... | 115.013029315... | 82.5053533190... | 141.451578947... | 475.311033449... | | |
| Racing | 146.380191693... | 865.765060240... | 185.424164524... | 362.577639751... | 193.213878326... | 57.4366812227... | 251.090185676... | 199.431818181... | 327.474327628... | | |
| Video Production | 5536.66666666... | 3447.28571428... | 0.8 | 0.0 | 254.130434782... | | 2638.35714285... | 2.70588235294... | 117.846153846... | 958.555555555... | 108 |
| None | | | | | | | | | | | |
| Education | 5617.63636363... | 12.5 | 5238.11111111... | 41.5 | 1179.04878048... | 2417.16666666... | 186.928571428... | 22.7575757575... | 23.0789473684... | 1675.35869565... | 133 |
| Game Development | 3234.64285714... | 0.0 | 3936.66666666... | 101.0 | 1439.88571428... | 1208.58333333... | 104.310344827... | 6.43478260869... | 103.0 | 1038.54471544... | 752 |
| Audio Production | 0.0 | 2030.0 | 0.0 | | 30.0714285714... | | 2419.0 | 52.875 | 72.7272727272... | 883.245283018... | 136 |

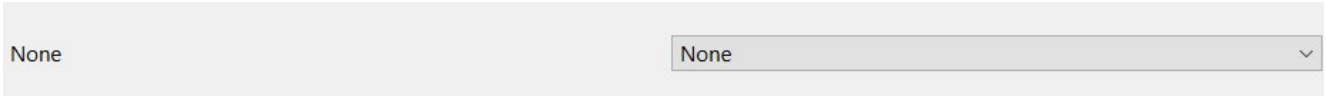
Рисунок 4.12 — Селектор типу "Relation Table"




| | Action | Free To Play | Adventure | assively Multiplay | Indie | RPG | Early Access | Casual | Simulation | imation & Modeli | esig |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Action | 35034 | 1916 | 15747 | 1221 | 26730 | 6218 | 4460 | 11049 | 4970 | 7 | 10 |
| Free To Play | 1916 | 4377 | 1498 | 661 | 3005 | 906 | 506 | 1822 | 832 | 8 | 13 |
| Adventure | 15747 | 1498 | 32710 | 770 | 25132 | 8602 | 2975 | 12121 | 5517 | 8 | 9 |
| Massively Multiplayer | 1221 | 661 | 770 | 1883 | 1167 | 756 | 489 | 595 | 549 | 2 | 2 |
| Indie | 26730 | 3005 | 25132 | 1167 | 58257 | 10976 | 6266 | 25817 | 11958 | 38 | 39 |
| RPG | 6218 | 906 | 8602 | 756 | 10976 | 14864 | 2038 | 4382 | 3061 | 5 | 17 |
| Early Access | 4460 | 506 | 2975 | 489 | 6266 | 2038 | 8124 | 2725 | 2369 | 62 | 66 |
| Casual | 11049 | 1822 | 12121 | 595 | 25817 | 4382 | 2725 | 33345 | 8098 | 38 | 36 |
| Simulation | 4970 | 832 | 5517 | 549 | 11958 | 3061 | 2369 | 8098 | 16938 | 34 | 33 |
| Animation & Modeling | 7 | 8 | 8 | 2 | 38 | 5 | 62 | 38 | 34 | 428 | 290 |
| Design & Illustration | 10 | 13 | 9 | 2 | 39 | 17 | 66 | 36 | 33 | 290 | 533 |
| Photo Editing | 3 | 4 | 6 | | 17 | 2 | 9 | 9 | 6 | 76 | 121 |
| Utilities | 18 | 20 | 13 | 4 | 69 | 13 | 118 | 60 | 44 | 213 | 294 |
| Strategy | 4752 | 984 | 4717 | 567 | 11753 | 4060 | 2089 | 7145 | 5423 | 1 | 2 |
| Sports | 1759 | 215 | 766 | 172 | 2404 | 307 | 467 | 1900 | 2003 | | |
| Racing | 1565 | 166 | 778 | 161 | 2104 | 229 | 377 | 1408 | 1227 | | |
| Video Production | 3 | 7 | 5 | 1 | 23 | | 28 | 17 | 13 | 126 | 110 |
| None | | | | | | | | | | | |
| Education | 11 | 10 | 9 | 2 | 41 | 6 | 42 | 33 | 38 | 92 | 123 |
| Game Development | 14 | 7 | 12 | 1 | 35 | 12 | 29 | 23 | 18 | 123 | 154 |
| Audio Production | 1 | 8 | 2 | | 28 | | 21 | 16 | 11 | 53 | 61 |

Рисунок 4.13 — Селектор типу "Relation Table", функція агрегації "count"

Візуалізатор типу "None" (рис. 4.14) не має параметрів, функціоналу і




візуалізації.

Рисунок 4.14 — Селектор типу "None"

Після запуску програми, за замовченням, тип основного селектора — "AND", і він має два дочірніх селектора, обидва типу "ALL", в той час як за замовченням тип візуалізатора — "Games Table", і усі поля обрані (рис. 4.15).

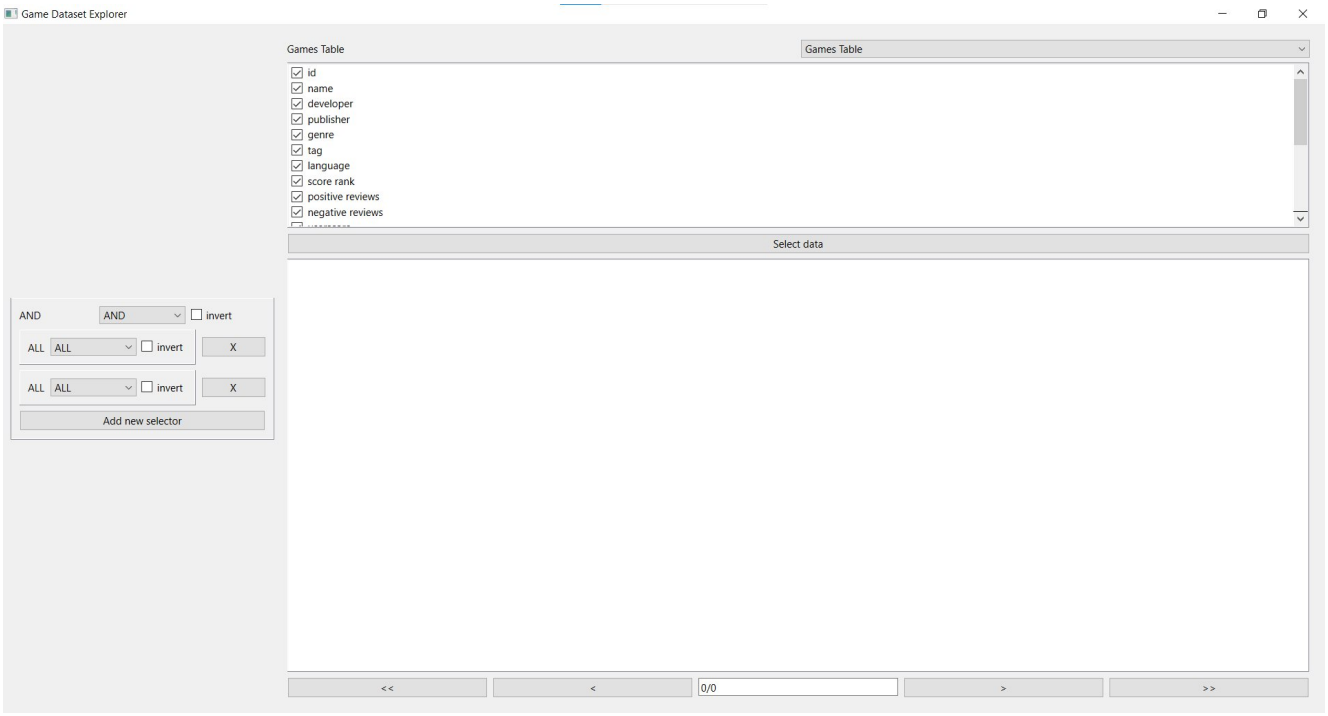
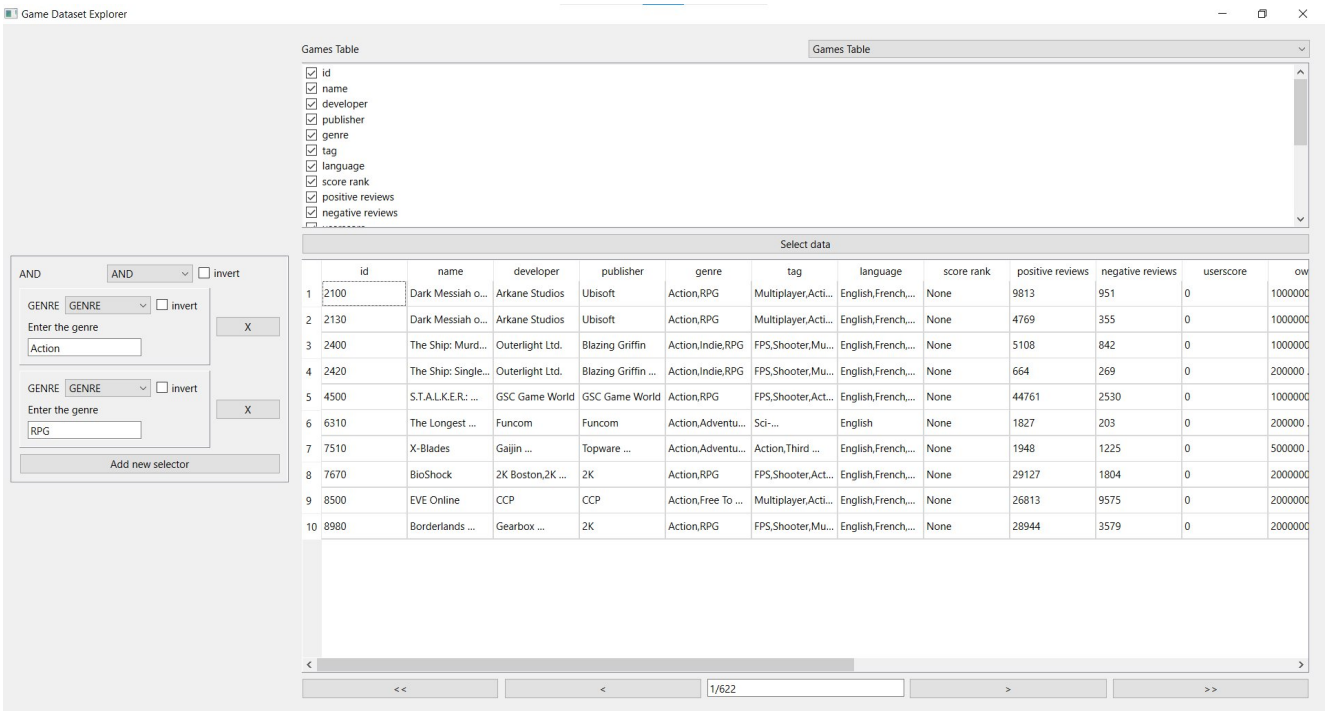



Рисунок 4.15 — Додаток після запуску

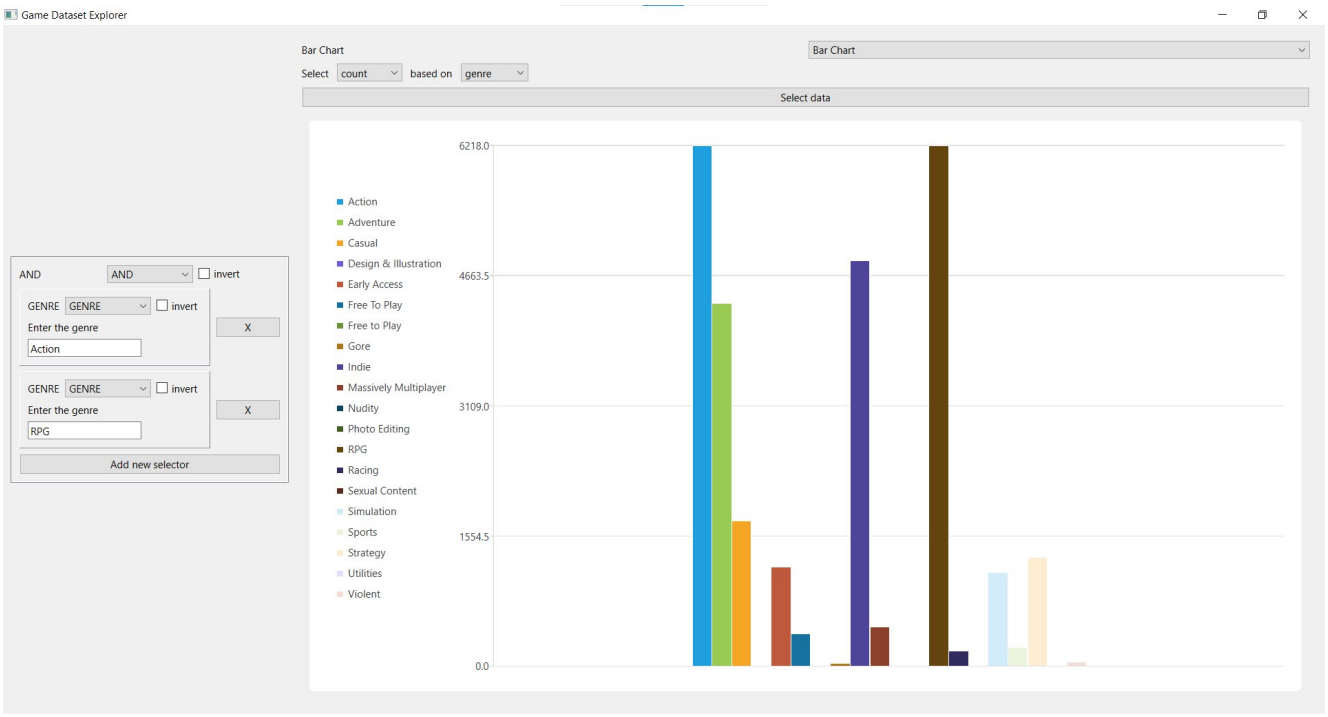
Як приклад використання можемо переглянути інформацію про ігри, що




належать до жанрів Action і RPG (рис. 4.16).

Рисунок 4.16 — Ігри у жанрах Action і RPG

Можемо побачити кількість ігор, що належать до жанрів Action і RPG, розподілену по жанрам, і на рисунку 4.18  середню кількість позитивних відгуків




про ігри що належать до жанрів Action і RPG, розподілену по жанрах (рис. 4.17)).

Рисунок 4.17 — Кількість ігор у жанрах Action і RPG за жанрами

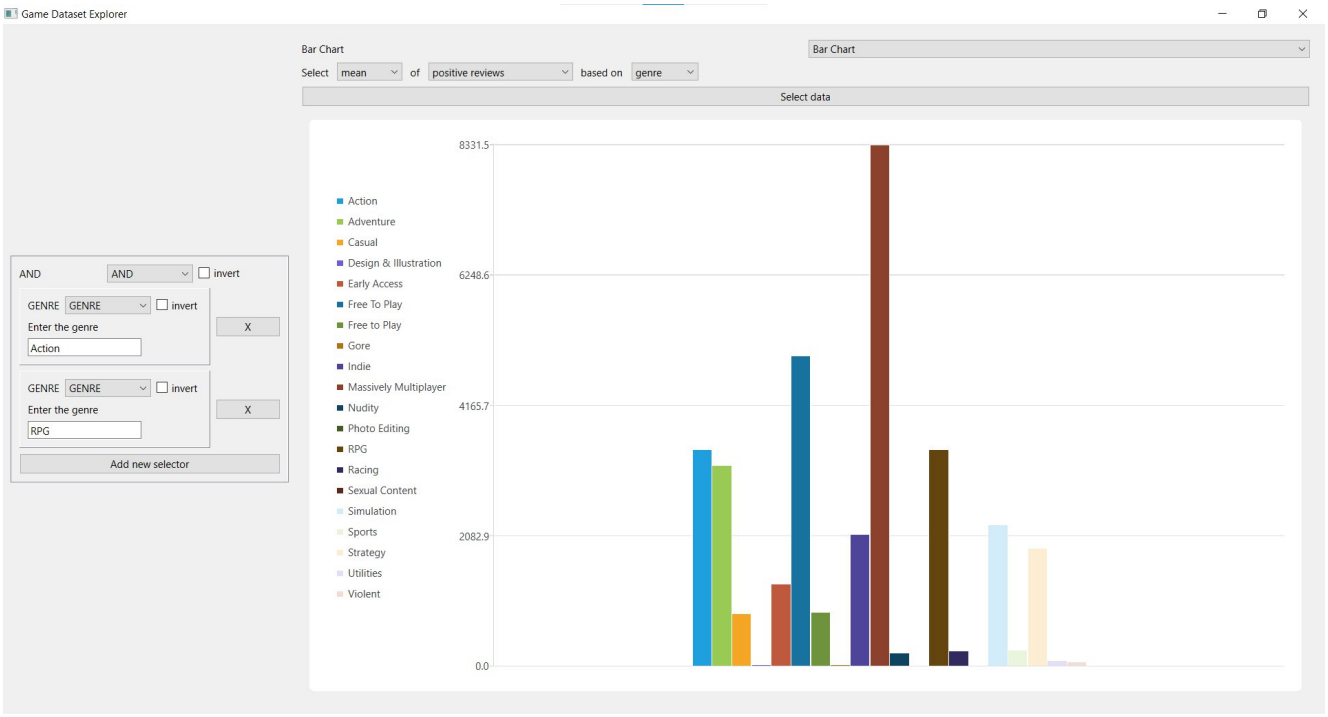



Рисунок 4.18 — Середня кількість позитивних відгуків про ігри

у жанрах Action і RPG

Аналізуючи діаграми можна прийти до висновку, що, хоча серед ігор у жанрах Action і RPG порівняно мало ігор, що можна описати як Massively Multiplayer і Free to Play, вони мають диспропорційно велику кількість позитивних відгуків.

Можемо побачити кількість ігор (рис. 4.19), середню кількість позитивних відгуків (рис. 4.20) і середню кількість одночасно активних користувачів (рис. 4.21) у жанрах Action і RPG за комбінаціями жанрів.

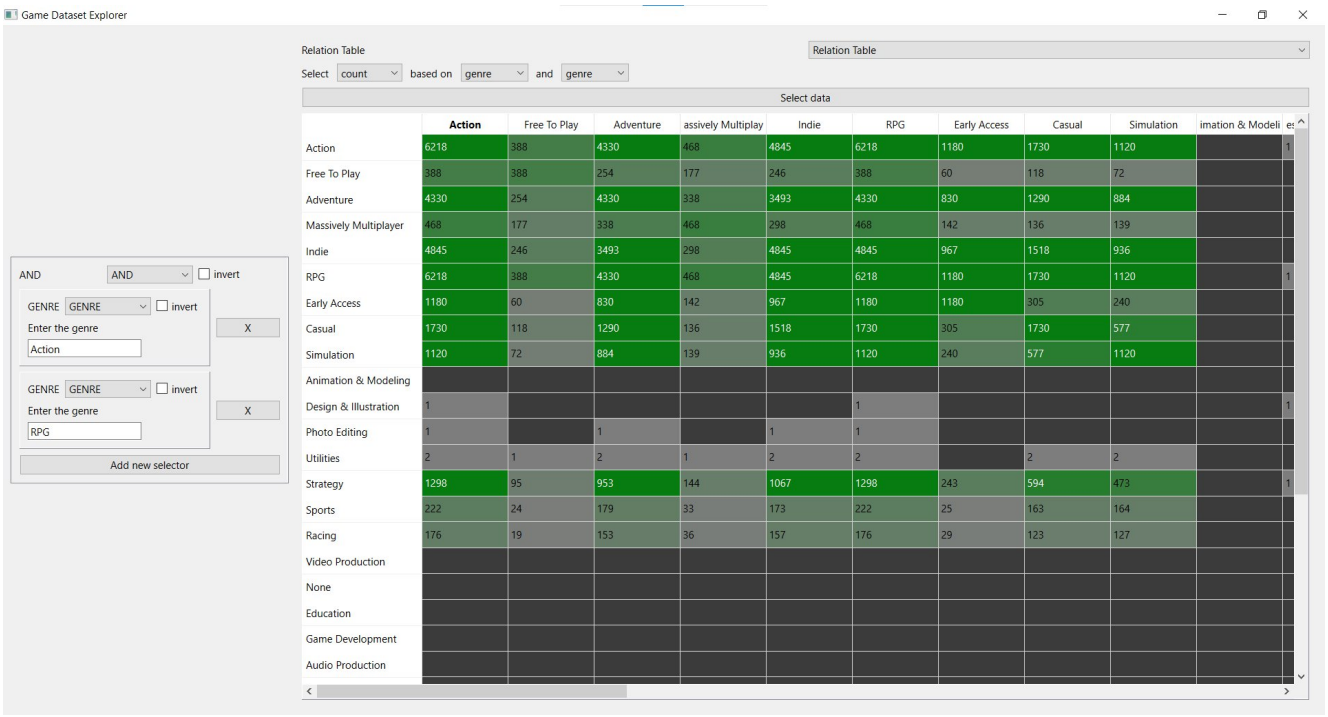



Рисунок 4.19 — Кількість ігор у жанрах Action і RPG за комбінаціями жанрів

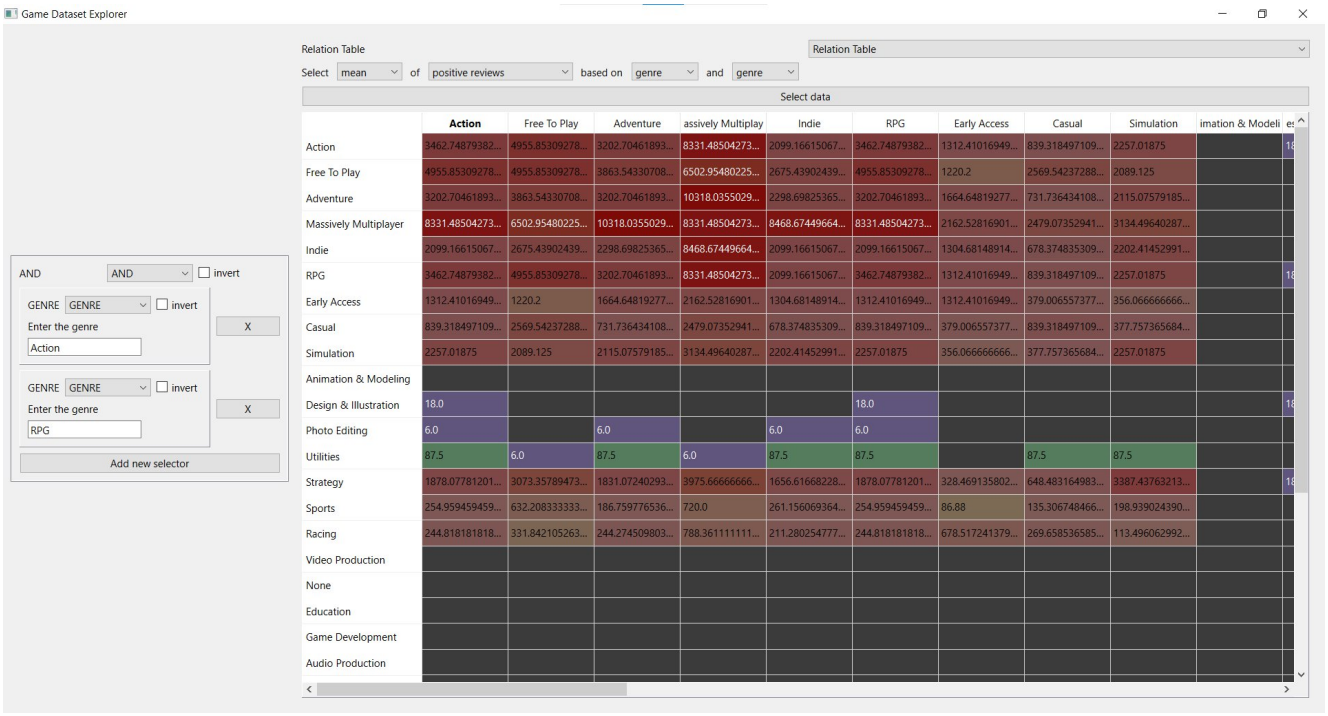



Рисунок 4.20 — Середня кількість позитивних відгуків про ігри у жанрах Action і RPG за комбінаціями жанрів

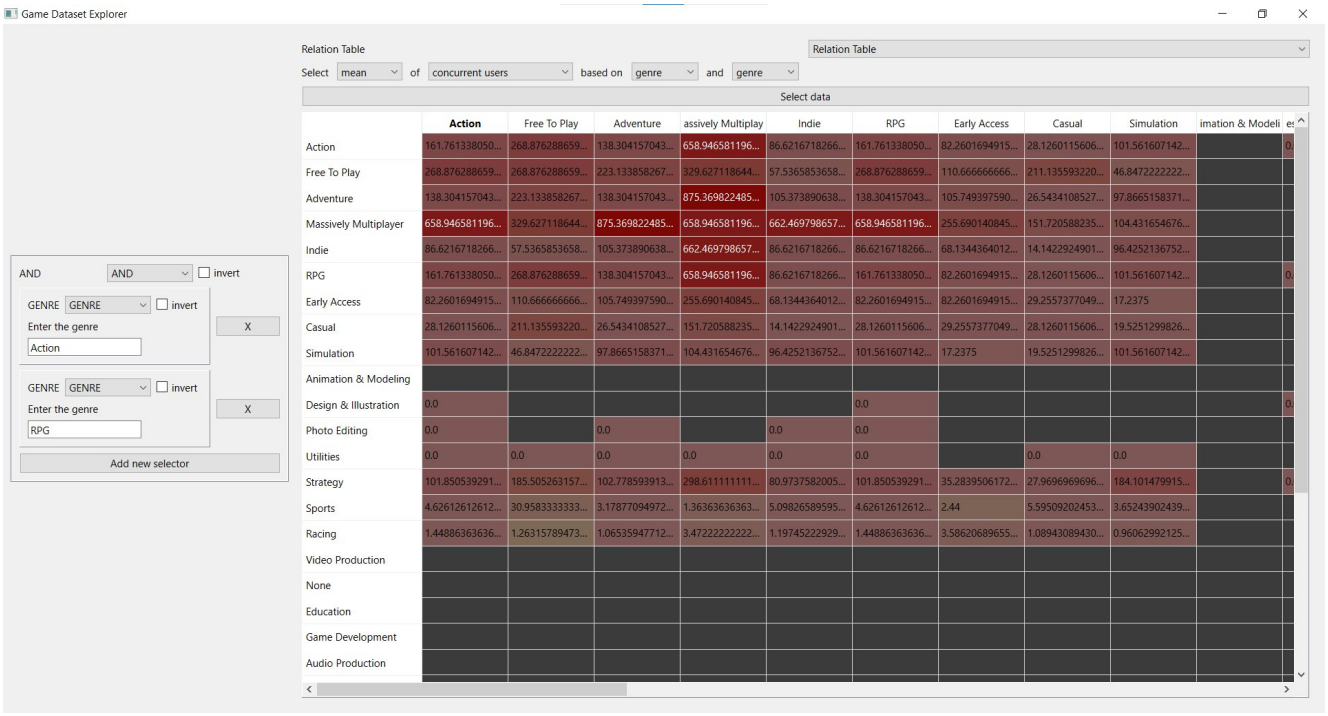



Рисунок 4.21 — Середня кількість одночасно активних користувачів про ігри у жанрах Action і RPG за комбінаціями жанрів

Серед ігор у жанрах Action і RPG, 468 є Massively Multiplayer і 388 є Free to Play, але лише 177 є одразу Massively Multiplayer і Free to Play. Середня кількість позитивних відгуків і одночасно активних користувачів ігор, що, окрім жанрів Action і RPG, належать також до Free to Play або Massively Multiplayer, у середньому вище аніж при інших комбінаціях жанрів. При цьому, хоча середня кількість позитивних відгуків і одночасно активних користувачів у іграх, що належать одночасно жанрам Massively Multiplayer і Free to Play, вище, аніж у Free to Play в цілому, вони нижче, аніж у Massively Multiplayer.

З цього можна зробити висновок, що серед ігор у жанрах Action і RPG, ігри, що також є Free to Play і Massively Multiplayer, є диспропорційно популярними. При тому, серед Massively Multiplayer Action RPG ігор, Free to Play є порівняно непопулярними. Таким чином, можна зробити висновок, що, хоча є сенс розробляти як Massively Multiplayer Action RPG ігри, так і Free to Play Action RPG ігри, не має особливого сенсу розробляти Free to Play Massively Multiplayer Action RPG ігри.

## 4.4. Висновки до розділу 4

В четвертому розділі описаний процес створення інформаційної системи для аналізу набору даних про відеоігри. Додаток розроблений за допомогою мови програмування Python, графічного фреймворку Qt і системи керування базами даних SQLite.

Додаток має легкий для використання інтерфейс, що дозволяє користувачу сформувати вибірку даних і отримати її візуальне представлення.

Серед переваг розробленої інформаційної системи є легкість вибору різноманітних комбінацій параметрів відеоігор для аналізу за допомогою селекторів та інтуїтивна зрозумілість даних, що представляються користувачу з використанням візуалізаторів.

До недоліків можна віднести:

- незручності при використанні візуалізатора "Bar Chart" через наявність занадто схожих кольорів, відсутність підписів і підказок щодо стовпців графіку;
- кольори клітин таблиці при використанні візуалізатора "Relation Table" представляють деяку інформацію, але залежності між цією інформацією і кольорами не завжди очевидні для користувача, що ускладнює її аналіз;
- додаток іноді працює досить повільно, особливо при використанні візуалізатора "Relation Table".

В цілому, інформаційна система представляє достатньо легкий спосіб дослідження набору даних про відеоігри. Порівняно з аналізом на основі використання програмного коду у середовищі Jupyter, розроблена інформаційна система має зручний інтерфейс і надає користувачу різноманітні засоби візуалізації необхідних даних.

# ВИСНОВКИ

В процесі виконання кваліфікаційної роботи здійснено дослідження історії відеоігор і впливів на відеоігри і культуру різних зовнішніх факторів. Виконано аналіз набору даних з інформацією про відеоігри з використанням середовища Jupyter. На основі проведеного дослідження розроблена інформаційна система для полегшення процесу аналізу ринку комп'ютерних ігор.

Історія відеоігор тісно і нерозривно пов'язана з історією технологій і культури. Перші комп'ютерні ігри створювалися як дослідницькі проекти або демонстрація можливостей комп'ютера і зазвичай були адаптаціями існуючих ігор. З розвитком технологій, розробка і розповсюдження відеоігор стали більш доступними, що дозволило створення нових типів ігор, комерційну розробку і розповсюдження відеоігор і сприяла створенню відеоігор як окремої культурної сфери.

На ігровий дизайн і ігрову культуру значний вплив мають різні фактори цільова платформа, її технічні характеристики і можливості, пристрої вводу та виведення, носії даних, використовувані інструменти розробки і методи доставки, економічні, культурні, історичні чинники тощо. У різних спільнотах, регіонах і країнах відеоігри, культура навколо них і історія їх розвитку мають свою специфіку завдяки різним економічним, культурним, історичним та інші чинникам. Тим не менш, присутня тенденція надавати пріоритет домінантному наративу, що фокусується лише на малому шматку цієї історії, і нехтувати іншими перспективами.

Комбінація жанрів гри зазвичай має малий вплив на її рейтинг, це означає, що комбінація жанрів не є головним у питанні наскільки гравцю сподобається гра, але комбінація жанрів може мати достатньо високий вплив на кількість рекомендацій гри. Окрім того, присутні деякі комбінації жанрів, що мають диспропорційно велику кількість рекомендацій порівняно з кількістю ігор з даною комбінацією жанрів. Можна припустити, що попит на дані комбінації жанрів з невеликою пропозицією є незаповненою нішею.

Інформаційна система була розроблена у вигляді додатку за допомогою мови програмування Python, графічного фреймворку Qt і системи керування базами даних SQLite. Для інтеграції Qt в Python використовувалась бібліотека PySide, а для інтеграції SQLite в Python використовувалась бібліотека sqlite3.

Додаток має легкий для користувача інтерфейс, що дозволяє сформувати вибірку даних за допомогою селекторів і представити її за допомогою візуалізатора.

Розроблена інформаційна система представляє достатньо легкий спосіб дослідження набору даних про відеоігор. У порівнянні з аналізом даних у середовищі Jupyter, за допомогою даної інформаційної системи можна набагато швидше і легше обирати та представляти дані, але вона має деякі обмеження при візуалізації інформації. Інформаційна система спрощує процес аналізу ринку комп'ютерних ігор, що може бути використано для виявлення тенденцій і виключень з них; надає можливості кращого розуміння ринку комп'ютерних ігор, що може бути корисним для розробників відеоігор; знаходження ігор за деякими параметрами, що необхідно гравцям та дослідникам.

Подальший розвиток інформаційної системи передбачає оновлення бази даних; формування набору даних на основі декількох джерел, виправлення виявлених недоліків, оптимізацію роботи програми, додавання нових селекторів і візуалізаторів тощо. У цілому, використання інформаційної системи надає користувачу можливість зручно та оперативно аналізувати та візуалізувати великі обсяги інформації про комп'ютерні відеоігри.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wittgenstein L. Philosophical Investigations. URL: https://archive.org/details/philosophicalinvestigations_201911 (дата звернення: 13.11.2025).

2. Huizinga J. Homo Ludens. URL: https://web.archive.org/web/20150328040138/http://art.yale.edu/file_columns/0000/1474/homo_ludens_johan_huizinga_routledge_1949_.pdf (дата звернення: 13.11.2025).

3. Super Bunnyhop. Kriegsspiel! How Napoleon Accidentally Invented Strategy Games, 2025. *YouTube*. URL: https://www.youtube.com/watch?v=s6Am1Gjr74A.

4. People Make Games. The Games Behind Your Government's Next War, 2024. *YouTube*. URL: https://www.youtube.com/watch?v=lYaDXZ2MI-k (дата звернення: 13.11.2025).

5. CATHODE-RAY TUBE AMUSEMENT DEVICE : patent 2,455,992 United States. Applied on 25.01.1947 ; published on 14.12.1948. URL: https://patentimages.storage.googleapis.com/83/75/ec/97d9851b888141/US2455992.pdf (дата звернення: 13.11.2025).

6. Blitz M. The Unlikely Story of the First Video Game. *Popular Mechanics*. 2016. URL: https://www.popularmechanics.com/culture/gaming/a20129/the-very-first-video-game/ (дата звернення: 13.11.2025).

7. Cathode Ray Tube Amusement Device [1947]. *Arcade Idea*. URL: https://arcadeidea.wordpress.com/2022/03/21/cathode-ray-tube-amusement-device-1947/ (дата звернення: 13.11.2025).

8. The Priesthood at Play: Computer Games in the 1950s. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2014/01/22/the-priesthood-at-play-computer-games-in-the-1950s/ (дата звернення: 13.11.2025).

9. NIMATRON: An Early Electromechanical Machine to Play the Game of Nim : History of Information. *History of Information*. URL: https://www.historyofinformation.com/detail.php?id=4012 (дата звернення: 13.11.2025).

10. Friedel F. Reconstructing Turing's "Paper Machine". *Chess News*. URL: https://en.chessbase.com/post/reconstructing-turing-s-paper-machine (дата звернення: 13.11.2025).

11. Machiavelli. *Chessprogramming wiki*. URL: https://www.chessprogramming.org/Machiavelli (дата звернення: 13.11.2025).

12. Mate-in-two. *Chessprogramming wiki*. URL: https://www.chessprogramming.org/Mate-in-two (дата звернення: 13.11.2025).

13. HUTSPIEL a Theater War Game / D. K. Clark et al. 1958. URL: https://archive.org/details/hutspiel-a-theater-war-game (дата звернення: 13.11.2025).

14. American Management Association. Top Management Decision Simulation. 1957. URL: https://archive.org/details/top-management-decision-simulation (дата звернення: 13.11.2025).

15. Digital Video Game Firsts - Michigan Pool (1954). *Now Go Bang! – mass:werk / Blog*. URL: https://www.masswerk.at/nowgobang/2019/michigan-pool (дата звернення: 13.11.2025).

16. Tennis Anyone?. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2014/01/28/tennis-anyone/ (дата звернення: 13.11.2025).

17. Nowak P. Video games turn 50. *CBC*. URL: https://www.cbc.ca/news/science/video-games-turn-50-1.703624 (дата звернення: 13.11.2025).

18. J.T. Gilmore, H.P. Peterson. A functional description of the TX-0 computer. URL: https://bitsavers.trailing-edge.com/pdf/mit/tx-0/6M-4789-1_TX0_funcDescr.pdf (дата звернення: 13.11.2025).

19. The Whirlwind Computer at CHM. *Computer History Museum*. URL: https://computerhistory.org/blog/the-whirlwind-computer-at-chm/ (дата звернення: 13.11.2025).

20. McKenzie J. A. TX-0 Computer History. The Research Laboratory of Electronics at the Massachusetts Institute of Technology, 1999. URL: https://dspace.mit.edu/bitstream/handle/1721.1/4132/RLE-TR-627-42827671.pdf;jsessionid=A1D26CA3AECC69591B6773BF4EE823EC?sequence=1 (дата звернення: 13.11.2025).

21. The Computer Museum Report. 1984. URL: https://bitsavers.org/pdf/mit/tx-0/TX-0_history_1984.txt (дата звернення: 13.11.2025).

22. One, Two, Three, Four I Declare a Space War. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2014/08/07/one-two-three-four-i-declare-a-space-war/ (дата звернення: 13.11.2025).

23. Ben Gurley | PDP-1 Restoration Project. *Computer History Museum*. URL: https://www.computerhistory.org/pdp-1/ben-gurley/ (дата звернення: 13.11.2025).

24. Russell S., Graetz M., Wiitanen W. Spacewar! (PDP-1). 1962.

25. The production and evaluation of three computer-based economics games for the sixth grade. Final report. / R. L. Wing et al. U.S. Departament of Health, Education and Welfare. URL: https://archive.org/details/ERIC_ED014227 (дата звернення: 13.11.2025).

26. The Sumerian Game: The Most Important Video Game You've Never Heard Of. *A Critical Hit!*. URL: https://www.acriticalhit.com/sumerian-game-most-important-video-game-youve-never-heard/ (дата звернення: 13.11.2025).

27. HareSoft. The Sumerian Game (Steam). 2024. URL: https://store.steampowered.com/app/2699250/The_Sumerian_Game/ (дата звернення: 13.11.2025).

28. The Baer Essentials. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2015/10/30/the-baer-essentials/ (дата звернення: 13.11.2025).

29. 1TL200: A Magnavox Odyssey. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2015/11/16/1tl200-a-magnavox-odyssey/ (дата звернення: 13.11.2025).

30. Pay to Play: A Brief History of Coin-Op Machines. *Museum Hack*. URL: https://museumhack.com/coin-op-machines/ (дата звернення: 13.11.2025).

31. Coin-Op Century: A Brief History of the American Arcade - The Strong National Museum of Play. *The Strong National Museum of Play*. URL: https://www.museumofplay.org/blog/coin-op-century-a-brief-history-of-the-american-arcade/ (дата звернення: 13.11.2025).

32. A Nutty Idea. *They Create Worlds*. URL: https://videogamehistorian.wordpress.com/2015/09/03/a-nutty-idea/ (дата звернення: 13.11.2025).

33. Schuyler S. J. The Untold Truth Of Pong - Grunge. *Grunge*. URL: https://www.grunge.com/929429/the-untold-truth-of-pong/ (дата звернення: 13.11.2025).

34. Grannell C. The 'Space Invaders' Creator Reveals the Game’s Origin Story. *WIRED*. URL: https://www.wired.com/story/space-invaders-45-years-tomohiro-nishikado/ (дата звернення: 13.11.2025).

35. Space Invaders: The Iconic Game That Revolutionized the Video Game Industry. *Serpentor's Lair - Your Source for TV, Movie, Comic Book, Cosplay and Toy*

*News!*. URL: https://serpentorslair.com/space-invaders-the-iconic-game-that-revolutionized-the-video-game-industry/ (дата звернення: 13.11.2025).

36. Taito. Space Invaders (Arcade). 1978.

37. 10 Questions for Shigeru Miyamoto. *TIME*. URL: https://web.archive.org/web/20070826025748/http://www.time.com/time/magazine/article/0,9171,1645158,00.html (дата звернення: 13.11.2025).

38. Mason G. Game Developer Interview: The Oliver Twins. *Antstream*. URL: https://www.antstream.com/post/game-developer-interview-the-oliver-twins (дата звернення: 13.11.2025).

39. Lissaman R. Interview with Andrew Oliver – Making video games. University of Warwick. URL: https://cdn.ima.org.uk/wp/wp-content/uploads/2023/09/Interview-with-Andrew-Oliver-from-MT-October-2023.pdf (дата звернення: 13.11.2025).

40. Star Trek. *Games of Fame*. URL: https://gamesoffame.wordpress.com/star-trek/ (дата звернення: 13.11.2025).

41. Smith T. Star Trek: The original computer game. *The Register: Enterprise Technology News and Analysis*. URL: https://www.theregister.com/2013/05/03/antique_code_show_star_trek/ (дата звернення: 13.11.2025).

42. A Brief History of Empire. *EMPIRE, Wargame of the Century*. URL: https://www.classicempire.com/history.html (дата звернення: 13.11.2025).

43. Game 233: Empire. *Data Driven Gamer*. URL: https://datadrivengamer.blogspot.com/2021/01/game-233-empire.html (дата звернення: 13.11.2025).

44. Game #34 : Empire (1977-1987) – The Wargaming Scribe. *The Wargaming Scribe*. URL: https://zeitgame.net/archives/2909 (дата звернення: 13.11.2025).

45. The History of Civilization. *Game Developer*. URL: https://web.archive.org/web/20230603085449/https://www.gamedeveloper.com/design/the-history-of-civilization (дата звернення: 13.11.2025).

46. RobotWar: the Battlefield of the Future. *Core War - Discover the Ultimate Programming Game*. URL: https://corewar.co.uk/robotwar.htm (дата звернення: 13.11.2025).

47. Muse Software. RobotWar Manual. URL: https://archive.org/details/robotwarmanual (дата звернення: 13.11.2025).

48. Muse Software, Warner S. RobotWar (Apple II). 1981.

49. Going Rogue: A Brief History of the Computerized Dungeon Crawl. *IEEE-USA InSight*. URL: https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl/ (дата звернення: 13.11.2025).

50. Game 121: Moria (1975). *The CRPG Addict*. URL: https://crpgaddict.blogspot.com/2013/11/game-121-moria-1975.html (дата звернення: 13.11.2025).

51. Kevet Duncombe on Moria - Spillhistorie.no. *Spillhistorie.no*. URL: https://spillhistorie.no/2016/12/30/kevet-duncombe-on-moria/ (дата звернення: 13.11.2025).

52. A history of 'Adventure'. *rickadams.org*. URL: https://rickadams.org/adventure/a_history.html (дата звернення: 13.11.2025).

53. Game #5: Colossal Cave a.k.a. Adventure. *Retro Games Trove*. URL: https://www.retrogamestrove.com/game-5-colossal-cave-a-k-a-adventure/ (дата звернення: 13.11.2025).

54. PCs That Paved the Way for the Altair. *PC Mag*. URL: https://www.pcmag.com/news/pcs-that-paved-the-way-for-the-altair (дата звернення: 13.11.2025).

55. The Altair 8800: The Machine That Launched the PC Revolution. *PC Mag*. URL: https://www.pcmag.com/news/the-altair-8800-the-machine-that-launched-the-pc-revolution (дата звернення: 13.11.2025).

56. Basic computer games / ed. by A. D. H. 8th ed. Morristown, NJ : Creative Computing Press, 1979. 185 p.

57. History of Microchess - Peter Jennings. *Peter Jennings - Benlo Park*. URL: http://www.benlo.com/microchess/index.html (дата звернення: 13.11.2025).

58. Goninon M. Zork I Review. *Choicest Games*. URL: https://www.choicestgames.com/2024/08/zork-i-review.html (дата звернення: 13.11.2025).

59. History of Infocom. *Infocom - The Master Storytellers*. URL: http://infocom-if.org/company/company.html (дата звернення: 13.11.2025).

60. Lammle R. A Brief History of Zork. *Mental Floss*. URL: https://www.mentalfloss.com/article/29885/eaten-grue-brief-history-zork (дата звернення: 13.11.2025).

61. ZIL and the Z-Machine. *The Digital Antiquarian*. URL: https://www.filfre.net/2012/01/zil-and-the-z-machine/ (дата звернення: 13.11.2025).

62. Atari 2600 - History of Video Game Consoles Guide - IGN. *IGN*. URL: https://www.ign.com/wikis/history-of-video-game-consoles/Atari_2600 (дата звернення: 13.11.2025).

63. The History of Atari: 1971-1977. *Gamasutra*. URL: https://web.archive.org/web/20180912021902/http://www.gamasutra.com/view/feature/130414/the_history_of_atari_19711977.php?print=1 (дата звернення: 13.11.2025).

64. Fisher A. Atari's Hard-Partying Origin Story: An Oral History. *Medium*. URL: https://medium.com/@AdamcFisher/ataris-hard-partying-origin-story-an-oral-history-c438b0ce9440 (дата звернення: 13.11.2025).

65. Fahs T. IGN Presents the History of Survival Horror - IGN. *IGN*. URL: https://www.ign.com/articles/2009/10/30/ign-presents-the-history-of-survival-horror?page=5 (дата звернення: 13.11.2025).

66. Design in Action | Symphony of the Night and the Metroidvania Lie. *USgamer.net*. URL: https://web.archive.org/web/20170120221744/http://www.usgamer.net/articles/design-in-action-|-symphony-of-the-night-and-the-metroidvania-lie (дата звернення: 13.11.2025).

67. Webster A. The enduring influence of Metroid. *The Verge*. URL: https://www.theverge.com/2017/9/14/16303016/metroid-nintendo-influence-legacy (дата звернення: 13.11.2025).

68. 'Metroidvania' should actually be 'Zeldavania'. *Engadget*. URL: https://www.engadget.com/2014-03-24-metroidvania-should-actually-be-zeldavania.html (дата звернення: 13.11.2025).

69. mossbag. Wait... Did Ori and the Will of the Wisps copy Hollow Knight?, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=So9E1egsaHU (дата звернення: 13.11.2025).

70. Epyx Computer Software. Rogue Instruction Manual. URL: https://britzl.github.io/roguearchive/files/misc/EpyxRogueDOSManual/manual.htm (дата звернення: 13.11.2025).

71. Berlin Interpretation. *RogueBasin*. URL: https://roguebasin.com/index.php?title=Berlin_Interpretation (дата звернення: 13.11.2025).

72. Yu D. Spelunky Classic (Windows). 2008.

73. Jeremy Parish | Video Works. The Tower of Druaga retrospective: Demonitization | NES Works Gaiden #38, 2022. *YouTube*. URL: https://www.youtube.com/watch?v=ZhfX1dzjS_0 (дата звернення: 13.11.2025).

74. Jeremy Parish | Video Works. The Legend of Zelda retrospective: The gold standard | NES Works #047, 2019. *YouTube*. URL: https://www.youtube.com/watch?v=W7zx-9jDaOk (дата звернення: 13.11.2025).

75. Jeremy Parish | Video Works. Zelda II: The Adventure of Link retrospective: Hyrule warrior | NES Works #105, 2022. *YouTube*. URL: https://www.youtube.com/watch?v=L-zz4JrqIK0 (дата звернення: 13.11.2025).

76. Inventing the Adventure Game: Chapter 3. *Warren Robinett's Home Page*. URL: http://www.warrenrobinett.com/inventing_adventure/inventing_ch3.htm (дата звернення: 13.11.2025).

77. Vintage Computer Federation. VCF East: Atari Adventure: Colossal Cave without the Text – Wil Lindsay, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=JE5_3aW7oPs (дата звернення: 13.11.2025).

78. Adventure of the Week: Dark Mage (1997). *Gaming After 40*. URL: https://gamingafter40.blogspot.com/2013/06/adventure-of-week-dark-mage-1997.html (дата звернення: 13.11.2025).

79. Displaced Gamers. NES vs Famicom Disk System - Zelda, Disks, Mappers, and "Ports" - Talkin' Code Episode 2, 2022. *YouTube*. URL: https://www.youtube.com/watch?v=UleSwaVH1MY (дата звернення: 13.11.2025).

80. Pepe F. The Gentrification of Video Game History. *Medium*. URL: https://felipepepe.medium.com/the-gentrification-of-video-game-history-dfe11f1e08ae (дата звернення: 13.11.2025).

81. Purchese R. The history of CD Projekt Red. *Eurogamer.net*. URL: https://www.eurogamer.net/seeing-red-the-story-of-cd-projekt (дата звернення: 13.11.2025).

82. документація - Пригоди піонерки Ксені. *Wayback Machine*. URL: https://web.archive.org/web/20220820071803/https://sites.google.com/site/pionerkakseniа/materialy/dokumentacia (дата звернення: 13.11.2025).

83. History. *CossacksHQ International - Home of all GSC strategies*. URL: https://cossackshq.net/gsc-history/ (дата звернення: 13.11.2025).

84. Anivisual.net. URL: https://anivisual.net/ (дата звернення: 13.11.2025).

85. Світ Ігор. *Wayback Machine*. URL: https://web.archive.org/web/20110203031349/https://svitigor.com/ (дата архівації: 03.02.2011, дата звернення: 13.11.2025).

86. Магазин Компьютерных Игр Світ Ігор. *Wayback Machine*. URL: https://web.archive.org/web/20141218092352/https://svitigor.com/ (дата архівації: 18.12.2014, дата звернення: 13.11.2025).

87. Video Game Dataset. *Kaggle*. URL: https://www.kaggle.com/datasets/jummyegg/rawg-game-dataset/data (дата звернення: 01.07.2025).

88. *SteamSpy - All the data and stats about Steam games*. URL: https://steamspy.com/api.php (дата звернення: 19.11.2025).

# ДОДАТКИ

## Додаток А. Аналіз даних у середовищі Jupyter Notebook

Аналіз даних у середовищі Jupyter Notebook:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches
from types import MethodType


    df=pd.read_csv("game_info.csv")


    def contains(self, column, objects, excl=False):
  isnan=False
  if type(objects) is not list:
    if pd.isna(objects):
      excl=True
      isnan=True
    objects=[objects]
  dat=self[column].str.split(r'\|\|')
  if excl==False:
    return pd.Series(data=[set(objects).issubset(l) if type(l) is not float else False for l
in dat])
  if isnan==True:
    return pd.isna(dat)
  if excl==True:
    return pd.Series(data=[set(objects)==set(l) if type(l) is not float else False for l in
dat])
```

```
df.contains=MethodType(contains, df)


        genres_list=df["genres"].str.split(r'\|\|').explode().unique()
for genre in genres_list:
   print((genre))
genres_list


        if True:
   for genre in genres_list:
      df_ex=df[df.contains("genres", genre)]
      df_ex_excl=df_ex[df_ex["genres"]==genre]
      df_ex_nexcl=df_ex[df_ex["genres"]!=genre]
      if (pd.isna(genre)):
         df_ex_excl=df_ex[df_ex["genres"].isna()]
         df_ex_nexcl=df_ex[~df_ex["genres"].isna()]
      print(genre, df_ex.id.count(), df_ex_excl.id.count(), df_ex_nexcl.id.count())
      bar=plt.bar(genre, df_ex_nexcl.id.count(), bottom=0, color='y')
      bar=plt.bar(genre, df_ex_excl.id.count(), bottom=df_ex_nexcl.id.count(),
color='r')
   nexcl_patch=mpatches.Patch(color='y', label="More than one genre")
   excl_patch=mpatches.Patch(color='r', label="Only one genre")
   plt.legend(handles=[nexcl_patch, excl_patch])
   plt.xticks(rotation=90)
   plt.show()


        df_calc={}
if True:
   gen_list=genres_list[~pd.isnull(genres_list)]
```

```
    for i in range(len(gen_list)):
        for j in range(len(gen_list)):
            print(i,j)
            if frozenset([i, j]) in df_calc.keys():
                print("?")
                continue
            else:
                c=[gen_list[i], gen_list[j]] if i!=j else gen_list[i]
                ex=df[df.contains("genres", c)]
                df_calc[frozenset([i, j])]=ex
df_calc


        if True:
    gen_list=genres_list[~pd.isnull(genres_list)]
    calc={}
    fig, ax=plt.subplots(figsize=(13,6))
    for i in range(len(gen_list)):
        for j in range(len(gen_list)):
            #print(i, j)
            v=0
            sum=0
            if frozenset([i, j]) in calc.keys():
                di=calc[frozenset([i, j])]
                v=di[0]
                sum=di[1]
            else:
                ex=df_calc[frozenset([i, j])]
                v=ex.id.count()
                sum=df[df.contains("genres", gen_list[i]) | df.contains("genres",
gen_list[i])].id.count()
```

```
            calc[frozenset([i, j])]=(v, sum)
        col_v=1-(v/sum)
        col=(col_v*0.5, col_v*0.7, col_v*0.9)
        if (v==sum):
            col='y'
        rec_p=mpatches.Rectangle((i,j), 1, 1, color=col)
        ax.add_patch(rec_p)
        ax.text(i+0.5, j+0.5, str(round(v, 3)), va='center', ha='center')
ax.set_xlim(0, len(gen_list))
ax.set_ylim(0, len(gen_list))
ax.set_xticks(np.arange(len(gen_list)), gen_list, ha='left', rotation=90)
ax.set_yticks(np.arange(len(gen_list)), gen_list, va='bottom')
ax.grid()
ax.set_title("Number of games per genre combination")
plt.show()


    if True:
gen_list=genres_list[~pd.isnull(genres_list)]
calc={}
fig, ax=plt.subplots(figsize=(13,6))
for i in range(len(gen_list)):
   for j in range(len(gen_list)):
        #print(i, j)
        v=0
        sum=0
        if frozenset([i, j]) in calc.keys():
            di=calc[frozenset([i, j])]
            v=di[0]
            sum=di[1]
        else:
```

```
            ex=df_calc[frozenset([i, j])]
            v=ex.id.count()
            sum=df[df.contains("genres", gen_list[i]) | df.contains("genres",
gen_list[i])].id.count()
            calc[frozenset([i, j])]=(v, sum)
        col_v=1-(v/sum)
        col=(col_v*0.5, col_v*0.7, col_v*0.9)
        if (v==sum):
            col='y'
        rec_p=mpatches.Rectangle((i,j), 1, 1, color=col)
        ax.add_patch(rec_p)
        ax.text(i+0.5, j+0.5, str(round(v*100/sum, 1))+"%", va='center', ha='center')
    ax.set_xlim(0, len(gen_list))
    ax.set_ylim(0, len(gen_list))
    ax.set_xticks(np.arange(len(gen_list)), gen_list, ha='left', rotation=90)
    ax.set_yticks(np.arange(len(gen_list)), gen_list, va='bottom')
    ax.grid()
    ax.set_title("Number of games per genre combination")
    plt.show()


        if True:
    gen_list=genres_list[~pd.isnull(genres_list)]
    calc={}
    fig, ax=plt.subplots(figsize=(13,6))
    for i in range(len(gen_list)):
        for j in range(len(gen_list)):
            #print(i, j)
            v=0
            sum=0
            if frozenset([i, j]) in calc.keys():
```

```
            di=calc[frozenset([i, j])]
            v=di
        else:
            ex=df_calc[frozenset([i, j])]
            v=ex[ex["ratings_count"]>10].rating.mean()
            calc[frozenset([i, j])]=v
        col_v=1-(v/5)
        col=(col_v*0.9, col_v*0.7, col_v*0.5)
        if (i==j):
            col='y'
        rec_p=mpatches.Rectangle((i,j), 1, 1, color=col)
        ax.add_patch(rec_p)
        ax.text(i+0.5, j+0.5, str(round(v, 3)), va='center', ha='center')
ax.set_xlim(0, len(gen_list))
ax.set_ylim(0, len(gen_list))
ax.set_xticks(np.arange(len(gen_list)), gen_list, ha='left', rotation=90)
ax.set_yticks(np.arange(len(gen_list)), gen_list, va='bottom')
ax.grid()
ax.set_title("Mean rating per genre combination")
plt.show()


    if True:
gen_list=genres_list[~pd.isnull(genres_list)]
calc={}
fig, ax=plt.subplots(figsize=(13,6))
for i in range(len(gen_list)):
   for j in range(len(gen_list)):
        #print(i, j)
        v=0
        sum=0
```

```
        if frozenset([i, j]) in calc.keys():
            di=calc[frozenset([i, j])]
            v=di[0]
            sum=di[1]
        else:
            ex=df_calc[frozenset([i, j])]
            v=ex.suggestions_count.mean()
            sum=df.suggestions_count.max()
            calc[frozenset([i, j])]=(v, sum)
        col_v=1-(v/sum)
        col=(col_v*0.5, col_v*0.9, col_v*0.7)
        if (i==j):
            col='y'
        rec_p=mpatches.Rectangle((i,j), 1, 1, color=col)
        ax.add_patch(rec_p)
        ax.text(i+0.5, j+0.5, str(round(v, 1)), va='center', ha='center')
ax.set_xlim(0, len(gen_list))
ax.set_ylim(0, len(gen_list))
ax.set_xticks(np.arange(len(gen_list)), gen_list, ha='left', rotation=90)
ax.set_yticks(np.arange(len(gen_list)), gen_list, va='bottom')
ax.grid()
ax.set_title("Number of suggestions per genre combination")
plt.show()
```

**Додаток Б. Код додатку**

Код додатку:

```
from PySide6.QtCore import QSize, Qt
from PySide6.QtGui import QAction, QDoubleValidator, QColor
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QMenu, QPushButton, QSizePolicy,
QVBoxLayout, QHBoxLayout, QLayout, QWidget,
    QLineEdit, QLabel, QComboBox, QFrame, QCompleter, QTableWidget,
QTableWidgetItem, QListWidget, QSpinBox,
    QAbstractSpinBox, QCheckBox
)
from PySide6 import QtCharts
from PySide6.QtCharts import QChart, QChartView, QBarSet, QBarSeries,
QBarCategoryAxis
import sqlite3


genre_list=[]
producer_list=[]
language_list=[]
tag_list=[]


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        conn=sqlite3.connect("steam_games.db")
        cursor=conn.cursor()
```

```
        cursor.execute("SELECT DISTINCT genrename FROM genre")
        global genre_list
        genre_list=[i[0] for i in cursor.fetchall()]
        cursor.execute("SELECT DISTINCT producername FROM producer")
        global producer_list
        producer_list=[i[0] for i in cursor.fetchall()]
        cursor.execute("SELECT DISTINCT languagename FROM language")
        global language_list
        language_list=[i[0] for i in cursor.fetchall()]
        cursor.execute("SELECT DISTINCT tagname FROM tag")
        global tag_list
        tag_list=[i[0] for i in cursor.fetchall()]
        conn.close()

        selector=Selector()
        visualiser=Visualiser(selector)
        layout=QHBoxLayout()
        layout.addWidget(selector)
        layout.addWidget(visualiser)
        container=QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)
        self.setWindowTitle("Game Dataset Explorer")
        self.showMaximized()

def buildQuery(struct, top=False):
    ret_val=""
    if (struct["type"]=="AND"):
        args=struct["args"]
        if (len(args)==0):
```

```
            ret_val=""
        else:
            arg_res=[]
            for arg in args:
                arg_res.append(buildQuery(arg))
            ret_val="("+ " AND ".join(arg_res) +")"
    elif (struct["type"]=="OR"):
        args=struct["args"]
        if (len(args)==0):
            ret_val= ""
        else:
            arg_res=[]
            for arg in args:
                arg_res.append(buildQuery(arg))
            ret_val="("+ " OR ".join(arg_res) +")"
    elif (struct["type"]=="GENRE"):
        arg=struct["arg"]
        ret_val= f'(game.appid IN (SELECT game.appid FROM game JOIN
game_genre ON game.appid=game_genre.appid JOIN genre ON
game_genre.genreid=genre.genreid WHERE genre.genrename="{arg}"))'
    elif (struct["type"]=="DEVELOPER"):
        arg=struct["arg"]
        ret_val= f'game.appid IN (SELECT game.appid FROM game JOIN
game_developer ON game.appid=game_developer.appid JOIN producer ON
game_developer.producerid=producer.producerid WHERE
producer.producername="{arg}")'
    elif (struct["type"]=="PUBLISHER"):
        arg=struct["arg"]
        ret_val= f'game.appid IN (SELECT game.appid FROM game JOIN
game_publisher ON game.appid=game_publisher.appid JOIN producer ON
```

```
game_publisher.producerid=producer.producerid WHERE
producer.producername="{arg}")'
        elif (struct["type"]=="LANGUAGE"):
            arg=struct["arg"]
            return f'game.appid IN (SELECT game.appid FROM game JOIN
game_language ON game.appid=game_language.appid JOIN language ON
game_language.languageid=language.languageid WHERE
language.languagename="{arg}")'
        elif (struct["type"]=="TAG"):
            arg=struct["arg"]
            ret_val= f'game.appid IN (SELECT game.appid FROM game JOIN
game_tag ON game.appid=game_tag.appid JOIN tag ON game_tag.tagid=tag.tagid
WHERE tag.tagname="{arg}")'
        elif (struct["type"]=="COMPARISON"):
            subj=struct["subj"]
            comp_type=struct["comp_type"]
            if subj=="owners":
                if comp_type=="in range":
                    field1=min(float(struct["field1"]), float(struct["field2"]))
                    field2=max(float(struct["field1"]), float(struct["field2"]))
                    ret_val= f"(owners_lower>={field1} AND owners_upper<={field2})"
                else:
                    comparisons={"more than":">", "less than":"<", "equal to":"="}
                    comp=comparisons[comp_type]
                    field=float(struct["field"])
                    if comp==">":
                        ret_val= f"owners_upper>{field}"
                    if comp=="<":
                        ret_val= f"owners_lower<{field}"
                    if comp=="=":
```

```
                ret_val= f"(owners_lower<={field} AND owners_upper>={field})"
        else:
            if comp_type=="in range":
                field1=min(float(struct["field1"]), float(struct["field2"]))
                field2=max(float(struct["field1"]), float(struct["field2"]))
                if (subj=="price" or subj=="initialprice"):
                    field1=field1*100
                    field2=field2*100
                ret_val= f"({subj}>={field1} AND {subj}<={field2})"
            else:
                comparisons={"more than":">", "less than":"<", "equal to":"="}
                comp=comparisons[comp_type]
                field=float(struct["field"])
                if (subj=="price" or subj=="initialprice"):
                    field=field*100
                ret_val= f"{subj}{comp}{field}"
    elif (struct["type"]=="NONE"):
        ret_val="0"
    elif (struct["type"]=="ALL" or struct["type"]=="TRUE"):
        ret_val="1"
    else:
        ret_val= str(struct)
    print(ret_val)
    if (struct["invert"]==False):
        return ret_val
    if (struct["invert"]==True):
        return "NOT "+ret_val
    pass

def selectTest(query_condition):
```

```
        conn=sqlite3.connect("steam_games.db")
        cursor=conn.cursor()
        print(query_condition)
        cursor.execute("SELECT * FROM game WHERE "+query_condition+" LIMIT
10")
        ret_val=cursor.fetchall()
        conn.close()
        return ret_val
        pass

    def clearLayout(layout):
        if layout is not None:
            while layout.count():
                i=layout.takeAt(0)
                if i.widget() is not None:
                    i.widget().setParent(None)
                else:
                    clearLayout(i.layout())
        pass



    selector_types=["AND", "OR","COMPARISON", "GENRE", "DEVELOPER",
"PUBLISHER", "LANGUAGE", "TAG", "NONE", "ALL"]


    class Selector(QFrame):
        def __init__(self, selector_type="AND"):
            super().__init__()
```

```
self._selector_type=selector_type
self.type_label=QLabel(self._selector_type)

self.argument_type="None"
self.arguments=[]

self.combo=QComboBox()
self.combo.addItems(selector_types)
for i in range(self.combo.count()):
    if (self.combo.itemText(i)==self.selector_type):
        self.combo.setCurrentIndex(i)
self.combo.currentIndexChanged.connect(self.combo_index_changed)

self.invert_check=QCheckBox("invert")

type_header=QHBoxLayout()
type_header.addWidget(self.type_label)
type_header.addWidget(self.combo)
type_header.addWidget(self.invert_check)

self.content=QVBoxLayout()


layout=QVBoxLayout()
layout.addLayout(type_header)
layout.addLayout(self.content)
self.setFrameStyle(QFrame.StyledPanel | QFrame.Plain)
self.setSizePolicy(QSizePolicy.Minimum, QSizePolicy.Maximum)
self.setLayout(layout)
self.render_content()
```

```
def render_content(self):
    global producer_list
    def deleteItem(i, on_demand=False):
            print(i)
            print(self.arguments)
            print(self.content.itemAt(i))
            if on_demand:
                if self.argument_type=="Selectors":
                    self.arguments.remove(self.content.itemAt(i))
            if (self.content.itemAt(i).widget() is not None):
                self.content.itemAt(i).widget().deleteLater()
                self.content.takeAt(i)
            elif (self.content.itemAt(i).layout() is not None):
                clearLayout(self.content.itemAt(i).layout())
                self.content.takeAt(i)
    if self.selector_type=="AND" or self.selector_type=="OR":
        if self.argument_type!="Selectors":
            for i in reversed(range(self.content.count())):
                deleteItem(i)
            def sel(t="NONE"):
                selector=Selector(t)
                but=QPushButton("X")
                cont=QHBoxLayout()
                cont.addWidget(selector)
                cont.addWidget(but)
                cont.iden=None
                cont.par=None
                cont.button=but
                def deleteSel():
```

```
                    deleteItem(cont.par.arguments.index(cont.iden),
on_demand=True)
                cont.button.clicked.connect(deleteSel)
                cont.s_type="selector with button"
                cont.return_value=selector.return_value
                return cont
            self.arguments=[]
            self.argument_type="Selectors"
            self.arguments=[sel("ALL"), sel("ALL")]
            for ii in range(len(self.arguments)):
                i=self.arguments[ii]
                if hasattr(i, "s_type"):
                    if i.s_type=="selector with button":
                        print(ii)
                        i.par=self
                        i.iden=i
                if (issubclass(type(i), QWidget)):
                    self.content.addWidget(i)
                    continue
                if (issubclass(type(i), QLayout)):
                    self.content.addLayout(i)
                    continue
            self.content.add_button=QPushButton("Add new selector")
            def add_new_selector():
                new_selector=sel("NONE")
                new_selector.par=self
                self.arguments.append(new_selector)
                new_selector.iden=new_selector
                self.content.addLayout(new_selector)
                self.content.add_button.deleteLater()
```

```
                    self.content.add_button=QPushButton("Add new selector")
                    self.content.add_button.clicked.connect(add_new_selector)
                    self.content.addWidget(self.content.add_button)
                self.content.add_button.clicked.connect(add_new_selector)
                self.content.addWidget(self.content.add_button)
            elif self.selector_type=="GENRE":
                if self.argument_type!="Genre":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Genre"
                    self.arguments=QLineEdit()
                    global genre_list
                    self.arguments.setCompleter(QCompleter(genre_list))
                    self.content.addWidget(QLabel("Enter the genre"))
                    self.arguments.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.content.addWidget(self.arguments)
            elif self.selector_type=="DEVELOPER":
                if self.argument_type!="Producer":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Producer"
                    self.arguments=QLineEdit()
                    self.arguments.setCompleter(QCompleter(producer_list))
                    self.content.producer_label=QLabel()
                    self.content.addWidget(self.content.producer_label)
                    self.arguments.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.content.addWidget(self.arguments)
                self.content.producer_label.setText("Enter the developer")
```

```
            elif self.selector_type=="PUBLISHER":
                if self.argument_type!="Producer":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Producer"
                    self.arguments=QLineEdit()
                    self.arguments.setCompleter(QCompleter(producer_list))
                    self.content.producer_label=QLabel()
                    self.content.addWidget(self.content.producer_label)
                    self.arguments.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.content.addWidget(self.arguments)
                self.content.producer_label.setText("Enter the publisher")
            elif self.selector_type=="LANGUAGE":
                if self.argument_type!="Language":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Language"
                    self.arguments=QLineEdit()
                    global language_list
                    self.arguments.setCompleter(QCompleter(language_list))
                    self.content.addWidget(QLabel("Enter the language"))
                    self.arguments.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.content.addWidget(self.arguments)
            elif self.selector_type=="TAG":
                if self.argument_type!="Tag":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Tag"
```

```
                self.arguments=QLineEdit()
                global tag_list
                self.arguments.setCompleter(QCompleter(tag_list))
                self.content.addWidget(QLabel("Enter the tag"))
                self.arguments.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                self.content.addWidget(self.arguments)
            elif self.selector_type=="COMPARISON":
                if self.argument_type!="Comparison":
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Comparison"
                    self.arguments=[]
                    sel_dic={
                        "score rank":"score_rank",
                        "positive reviews": "positive",
                        "negative reviews": "negative",
                        "userscore": "userscore",
                        "owners": "owners",
                        "average playtime (forever)": "average_forever",
                        "average playtime (2 weeks)": "average_2weeks",
                        "median playtime (forever)": "median_forever",
                        "median playtime (2 weeks)": "average_2weeks",
                        "price": "price",
                        "initial price": "initialprice",
                        "discount": "discount",
                        "concurrent users": "ccu"
                        }
                    comparisons=["more than", "less than", "equal to", "in range"]
                    self.arguments.append(QComboBox()) #select subject
```

```
                self.arguments[0].addItems(sel_dic.keys())
                self.arguments.append(QComboBox())
                self.arguments[1].addItems(comparisons) #select type of comparison
                self.arguments.append(QVBoxLayout())
                self.arguments.append("") #argument 3
                self.arguments.append("") #argument 4
                old_index=[0]
                old_index[0]=self.arguments[1].currentIndex()
                def change_comp_type(i):
                    old_type=comparisons[old_index[0]]
                    new_type=comparisons[i]
                    old_index[0]=i
                    self.arguments[3]=new_type #argument 3
                    print(old_type, new_type, self.arguments[2].count())
                    if (new_type=="in range" and old_type!="in range") or
(old_type=="in range" and new_type!="in range") or (self.arguments[2].count()==0):
                        for i in reversed(range(self.arguments[2].count())):
                            print(i)
                            self.arguments[2].itemAt(i).widget().deleteLater()
                        def text_edited(field, new_text):
                            new_text=new_text.replace(",",".")
                            try:
                                float(new_text)
                                field.setText(new_text)
                            except:
                                field.setText("0")
                            pass
                        if new_type=="in range":
                            self.arguments[2].field1=QLineEdit()
                            self.arguments[2].field2=QLineEdit()
```

```
                    #self.arguments[2].field1.setValidator(QDoubleValidator())
                    #self.arguments[2].field2.setValidator(QDoubleValidator())
                    self.arguments[2].field1.editingFinished.connect(lambda:
text_edited(self.arguments[2].field1, self.arguments[2].field1.text()))
                    self.arguments[2].field1.setText("0")
                    self.arguments[2].field2.editingFinished.connect(lambda:
text_edited(self.arguments[2].field2, self.arguments[2].field2.text()))
                    self.arguments[2].field2.setText("0")
                    self.arguments[2].field1.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.arguments[2].field2.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.arguments[2].addWidget(self.arguments[2].field1)
                    self.arguments[2].addWidget(self.arguments[2].field2)
                else:
                    self.arguments[2].field=QLineEdit()
                    #self.arguments[2].field.setValidator(QDoubleValidator())
                    self.arguments[2].field.editingFinished.connect(lambda:
text_edited(self.arguments[2].field, self.arguments[2].field.text()))
                    self.arguments[2].field.setText("0")
                    self.arguments[2].field.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)
                    self.arguments[2].addWidget(self.arguments[2].field)
            pass
        self.arguments[1].currentIndexChanged.connect(change_comp_type)
        change_comp_type(old_index[0])
        def change_subj(i):
            new_type=sel_dic[list(sel_dic.keys())[i]]
            self.arguments[4]=new_type #argument 4
        self.arguments[0].currentIndexChanged.connect(change_subj)
```

```
        change_subj(self.arguments[0].currentIndex())
        self.content.addWidget(self.arguments[0])
        self.content.addWidget(self.arguments[1])
        self.content.addLayout(self.arguments[2])
    elif self.selector_type=="TEXT":
        if self.argument_type!="Text":
            for i in reversed(range(self.content.count())):
                deleteItem(i)
            self.argument_type="Text"
            self.arguments=QLineEdit()
            self.content.addWidget(self.arguments)
    else:
        for i in reversed(range(self.content.count())):
            deleteItem(i)
        self.argument_type="None"
    '''def setSize(it):
        if (it.widget() is not None):
            try:
                it.setSizePolicy(QSizePolicy.Maximum, QSizePolicy.Maximum)
            except:
                pass
        elif (it.layout() is not None):
            for i in range(it.count()):
                setSize(it.itemAt(i))
            pass
        pass
    for i in range(self.content.count()):
        it=self.content.itemAt(i)
        setSize(it)
        #it.setSizePolicy(QSizePolicy.Minimum, QSizePolicy.Minimum)'''
```

```
        pass

    def return_value(self):
        ret_val={}
        if self.selector_type=="AND":
            print("AND")
            ret_val["type"]="AND"
            args=[]
            for i in self.arguments:
                if hasattr(i, "return_value"):
                    args.append(i.return_value())
            ret_val["args"]=args
        elif self.selector_type=="OR":
            print("OR")
            ret_val["type"]="OR"
            args=[]
            for i in self.arguments:
                if hasattr(i, "return_value"):
                    args.append(i.return_value())
            ret_val["args"]=args
        elif self.selector_type=="GENRE":
            print("GENRE")
            ret_val["type"]="GENRE"
            arg=self.arguments.text()
            ret_val["arg"]=arg
        elif self.selector_type=="DEVELOPER":
            print("DEVELOPER")
            ret_val["type"]="DEVELOPER"
            arg=self.arguments.text()
            ret_val["arg"]=arg
```

```
elif self.selector_type=="PUBLISHER":
    print("PUBLISHER")
    ret_val["type"]="PUBLISHER"
    arg=self.arguments.text()
    ret_val["arg"]=arg
elif self.selector_type=="LANGUAGE":
    print("LANGUAGE")
    ret_val["type"]="LANGUAGE"
    arg=self.arguments.text()
    ret_val["arg"]=arg
elif self.selector_type=="TAG":
    print("TAG")
    ret_val["type"]="TAG"
    arg=self.arguments.text()
    ret_val["arg"]=arg
elif self.selector_type=="COMPARISON":
    print("COMPARISON")
    ret_val["type"]="COMPARISON"
    comp_type=self.arguments[3]
    ret_val["comp_type"]=comp_type
    if (comp_type=="in range"):
        ret_val["field1"]=self.arguments[2].field1.text()
        ret_val["field2"]=self.arguments[2].field2.text()
    else:
        ret_val["field"]=self.arguments[2].field.text()
    ret_val["subj"]=self.arguments[4]
    print(ret_val)
else:
    print(self.selector_type+": "+"NOT IMPLEMENTED")
    ret_val["type"]=self.selector_type
```

```
            ret_val["comment"]="NOT IMPLEMENTED"
        ret_val["invert"]=self.invert_check.isChecked()
        return(ret_val)
        pass

    def type_changed(self, old_value, new_value):
        self.type_label.setText(new_value)
        self.render_content()

    def combo_index_changed(self, index):
        self.selector_type=selector_types[index]
        pass
    @property
    def selector_type(self):
        return self._selector_type
    @selector_type.setter
    def selector_type(self, new_value):
        old_value=self._selector_type
        self._selector_type=new_value
        self.type_changed(old_value, new_value)

visualiser_types=["Games Table", "Bar Chart", "Relation Table", "None"]

class Visualiser(QWidget):
    def __init__(self, selector, visualiser_type="Games Table"):
        super().__init__()
        self.selector=selector

        self._visualiser_type=visualiser_type
        self.type_label=QLabel(self._visualiser_type)
```

```
        self.argument_type="None"
        self.arguments=[]

        self.combo=QComboBox()
        self.combo.addItems(visualiser_types)
        for i in range(self.combo.count()):
            if (self.combo.itemText(i)==self.visualiser_type):
                self.combo.setCurrentIndex(i)
        self.combo.currentIndexChanged.connect(self.combo_index_changed)

        type_header=QHBoxLayout()
        type_header.addWidget(self.type_label)
        type_header.addWidget(self.combo)

        self.content=QVBoxLayout()

        layout=QVBoxLayout()
        layout.addLayout(type_header)
        layout.addLayout(self.content)
        self.setSizePolicy(QSizePolicy.Minimum, QSizePolicy.Minimum)
        self.layout=layout
        self.setLayout(layout)
        self.render_content()

    def render_content(self):
        def deleteItem(i):
                print(i)
                if (self.content.itemAt(i).widget() is not None):
                    self.content.itemAt(i).widget().deleteLater()
```

```
                    self.content.takeAt(i)
                elif (self.content.itemAt(i).layout() is not None):
                    clearLayout(self.content.itemAt(i).layout())
                    self.content.takeAt(i)
            if (self.visualiser_type=="Games Table"):
                if self.argument_type!="Games_Table":
                    self.arguments=[]
                    for i in reversed(range(self.content.count())):
                        deleteItem(i)
                    self.argument_type="Games_Table"
                table=QTableWidget()
                lis=QListWidget()
                sel_dict={
                    "id": "appid",
                    "name": "name",
                    "developer": "(SELECT GROUP_CONCAT(producer.producername)
FROM producer WHERE producer.producerid IN (SELECT producer.producerid
FROM game_developer JOIN producer ON
game_developer.producerid=producer.producerid WHERE
game_developer.appid=game.appid))",
                    "publisher": "(SELECT GROUP_CONCAT(producer.producername)
FROM producer WHERE producer.producerid IN (SELECT producer.producerid
FROM game_publisher JOIN producer ON
game_publisher.producerid=producer.producerid WHERE
game_publisher.appid=game.appid))",
                    "genre": "(SELECT GROUP_CONCAT(genre.genrename) FROM
genre WHERE genre.genreid IN (SELECT genre.genreid FROM game_genre JOIN
genre ON game_genre.genreid=genre.genreid WHERE
game_genre.appid=game.appid))",
```

```
            "tag": "(SELECT GROUP_CONCAT(tag.tagname) FROM tag WHERE
tag.tagid IN (SELECT tag.tagid FROM game_tag JOIN tag ON
game_tag.tagid=tag.tagid WHERE game_tag.appid=game.appid))",
            "language": "(SELECT GROUP_CONCAT(language.languagename)
FROM language WHERE language.languageid IN (SELECT language.languageid
FROM game_language JOIN language ON
game_language.languageid=language.languageid WHERE
game_language.appid=game.appid))",
            "score rank":"score_rank",
            "positive reviews": "positive",
            "negative reviews": "negative",
            "userscore": "userscore",
            "owners": "(owners_lower ||' .. ' || owners_upper) owners",
            "average playtime (forever)": "average_forever",
            "average playtime (2 weeks)": "average_2weeks",
            "median playtime (forever)": "median_forever",
            "median playtime (2 weeks)": "average_2weeks",
            "price": "price*0.01",
            "initial price": "initialprice*0.01",
            "discount": "discount || '%'",
            "concurrent users": "ccu"
            }
        lis.addItems(sel_dict.keys())
        for i in range(lis.count()):
            print(lis.item(i).text())
            lis.item(i).setCheckState(Qt.Checked)
        selectButton=QPushButton("Select data")
        self.arguments={"data":[], "page":0}
        shown_items=10
        def display_table(page=1):
```

```
                page=page-1
                selected_data=self.arguments["data"]
                data_max=len(selected_data)
                table.clearContents()
                table.setRowCount(min(shown_items, data_max-page*shown_items))
                i_local_index=0
                if page==-1:
                    page=0
                for i_index in range(page*shown_items, min((page+1)*shown_items,
data_max)):
                    i_data=selected_data[i_index]
                    for j_index, j_data in enumerate(i_data):
                        table.setItem(i_local_index, j_index,
QTableWidgetItem(str(j_data)))
                    table.setVerticalHeaderItem(i_local_index,
QTableWidgetItem(str(i_index+1)))
                    i_local_index+=1
                pass
            page_selector=QHBoxLayout()
            page_selector.page=QSpinBox()
            page_selector.page.setMaximum(0)
            page_selector.page.setMinimum(0)
            page_selector.page.setSuffix("/"+str(page_selector.page.maximum()))
            page_selector.page.valueChanged.connect(display_table)
            page_selector.page.setButtonSymbols(QAbstractSpinBox.NoButtons)
            page_selector.pageBack=QPushButton("<")
            page_selector.pageBack.clicked.connect( lambda:
page_selector.page.stepBy(-1) )
            page_selector.pageForward=QPushButton(">")
```

```
            page_selector.pageForward.clicked.connect( lambda:
page_selector.page.stepBy(1) )
            page_selector.pageToMinimum=QPushButton("<<")
            page_selector.pageToMinimum.clicked.connect( lambda:
page_selector.page.setValue(page_selector.page.minimum()) )
            page_selector.pageToMaximum=QPushButton(">>")
            page_selector.pageToMaximum.clicked.connect( lambda:
page_selector.page.setValue(page_selector.page.maximum()) )
            page_selector.addWidget(page_selector.pageToMinimum)
            page_selector.addWidget(page_selector.pageBack)
            page_selector.addWidget(page_selector.page)
            page_selector.addWidget(page_selector.pageForward)
            page_selector.addWidget(page_selector.pageToMaximum)
            def selectData():
                table.clear()
                conn=sqlite3.connect("steam_games.db")
                cursor=conn.cursor()
                sel_cols=[]
                header_cols=[]
                for i in range(lis.count()):
                    item=lis.item(i)
                    if (item.checkState()==Qt.Checked):
                        header_cols.append(item.text())
                        sel_cols.append(sel_dict[item.text()])
                sel_col_string=", ".join(sel_cols)
                query=f"SELECT {sel_col_string} FROM game WHERE
"+buildQuery(self.selector.return_value())#+" LIMIT 10"
                cursor.execute(query)
                selected_data=cursor.fetchall()
                self.arguments["data"]=selected_data
```

```
                conn.close()
                table.setColumnCount(len(sel_cols))
                table.setHorizontalHeaderLabels(list(header_cols))
                print(len(selected_data))
                if (len(selected_data)>0):
                    page_selector.page.setMinimum(1)
                    page_selector.page.setValue(1)
                    page_selector.page.setMaximum(int( ( len(selected_data)-
len(selected_data)%shown_items) /shown_items)+1)
                else:
                    page_selector.page.setMinimum(0)
                    page_selector.page.setMaximum(0)
                    page_selector.page.setValue(0)
                page_selector.page.setSuffix("/"+str(page_selector.page.maximum()))
                display_table()

                pass
            selectButton.clicked.connect(selectData)
            lis.setSizePolicy(QSizePolicy.Minimum, QSizePolicy.Maximum)
            self.content.addWidget(lis)
            self.content.addWidget(selectButton)
            self.content.addWidget(table)
            self.content.addLayout(page_selector)
            pass
        elif (self.visualiser_type=="Bar Chart"):
            if self.argument_type!="Bar_Chart":
                self.arguments=[]
                for i in reversed(range(self.content.count())):
                    deleteItem(i)
                self.argument_type="Bar_Chart"
```

```
        aggs={"mean":"AVG", "maximum":"MAX", "minimum":"MIN",
"sum":"SUM", "count":"COUNT"}
        sel_dic={
            #"score rank":"score_rank",
            "positive reviews": "positive",
            "negative reviews": "negative",
            "userscore": "userscore",
            #"owners": "owners",
            "average playtime (forever)": "average_forever",
            "average playtime (2 weeks)": "average_2weeks",
            "median playtime (forever)": "median_forever",
            "median playtime (2 weeks)": "average_2weeks",
            "price": "price",
            "initial price": "initialprice",
            "discount": "discount",
            "concurrent users": "ccu"
            }
        cats=["genre", "tag", "developer", "publisher", "language"]
        agg_combo=QComboBox()
        agg_combo.addItems(aggs)
        sel_combo=QComboBox()
        sel_combo.addItems(list(sel_dic.keys()))
        cats_combo=QComboBox()
        cats_combo.addItems(cats)
        chart=QChart()
        chartView=QChartView(chart)
        combo_layout=QHBoxLayout()
        combo_layout.addWidget(QLabel("Select "))
        combo_layout.addWidget(agg_combo)
        of_label=QLabel(" of ")
```

```
combo_layout.addWidget(of_label)
combo_layout.addWidget(sel_combo)
combo_layout.addWidget(QLabel(" based on "))
combo_layout.addWidget(cats_combo)

def agg_changed(i):
    new_agg=list(aggs.keys())[i]
    if (new_agg=="count"):
        sel_combo.setVisible(False)
        of_label.setVisible(False)
    else:
        sel_combo.setVisible(True)
        of_label.setVisible(True)
    pass
agg_combo.currentIndexChanged.connect(agg_changed)

selectButton=QPushButton("Select data")
def selectData():
    chart.removeAllSeries()
    conn=sqlite3.connect("steam_games.db")
    cursor=conn.cursor()
    cat=cats[cats_combo.currentIndex()]
    what_name=f"{cat}.{cat}name"
    if cat=="developer" or cat=="publisher":
        what_name="producer.producername"
    agg=aggs[list(aggs.keys())[agg_combo.currentIndex()]]
    sel=sel_dic[list(sel_dic.keys())[sel_combo.currentIndex()]]
    what_agg=f"{agg}({sel})"
    if sel=="price" or sel=="initialprice":
        what_agg=what_agg+"/100"
```

```
                print(what_agg)
                where=buildQuery(self.selector.return_value())
                from_q=f"game JOIN game_{cat} ON game.appid=game_{cat}.appid
JOIN {cat} ON game_{cat}.{cat}id={cat}.{cat}id"
                if cat=="developer":
                    from_q=f"game JOIN game_developer ON
game.appid=game_developer.appid JOIN producer ON
game_developer.producerid=producer.producerid"
                if cat=="publisher":
                    from_q=f"game JOIN game_publisher ON
game.appid=game_publisher.appid JOIN producer ON
game_publisher.producerid=producer.producerid"
                query=f"SELECT {what_name}, {what_agg} FROM {from_q} WHERE
{where} GROUP BY {what_name} LIMIT 100"
                print(query)
                cursor.execute(query)
                data=cursor.fetchall()
                conn.close()
                series=QBarSeries()
                for i in data:
                    name=i[0]
                    data_entry=i[1]
                    if name=="":
                        name="None"
                    bset=QBarSet(name)
                    bset.append(data_entry)
                    series.append(bset)
                chart.addSeries(series)
                print(chart.legend())
                chart.legend().setAlignment(Qt.AlignLeft)
```

```
                chart.legend().setInteractive(True)
                chart.legend().update()
                chart.createDefaultAxes()
                chart.removeAxis(chart.axes(Qt.Horizontal)[0])
                chartView.setRubberBand(QChartView.VerticalRubberBand)
            selectButton.clicked.connect(selectData)
            combo_layout.setAlignment(Qt.AlignLeft)
            self.content.addLayout(combo_layout)
            self.content.addWidget(selectButton)
            self.content.addWidget(chartView)
            pass
        elif (self.visualiser_type=="Relation Table"):
            if self.argument_type!="Relation_Table":
                self.arguments=[]
                for i in reversed(range(self.content.count())):
                    deleteItem(i)
                self.argument_type="Relation_Table"

            aggs={"mean":"AVG", "maximum":"MAX", "minimum":"MIN",
"sum":"SUM", "count":"COUNT"}
            sel_dic={
                    #"score rank":"score_rank",
                    "positive reviews": "positive",
                    "negative reviews": "negative",
                    "userscore": "userscore",
                    #"owners": "owners",
                    "average playtime (forever)": "average_forever",
                    "average playtime (2 weeks)": "average_2weeks",
                    "median playtime (forever)": "median_forever",
                    "median playtime (2 weeks)": "average_2weeks",
```

```
                "price": "price",
                "initial price": "initialprice",
                "discount": "discount",
                "concurrent users": "ccu"
                }
            cats=["genre", "tag", "developer", "publisher", "language"]
            cat_lists={"genre": genre_list, "tag":tag_list, "developer": producer_list,
"publisher": producer_list, "language":language_list}
            cat_where={
                "genre": "SELECT game.appid FROM game JOIN game_genre ON
game.appid=game_genre.appid JOIN genre ON game_genre.genreid=genre.genreid
WHERE genre.genrename=",
                "developer": "SELECT game.appid FROM game JOIN
game_developer ON game.appid=game_developer.appid JOIN producer ON
game_developer.producerid=producer.producerid WHERE producer.producername=",
                "publisher": "SELECT game.appid FROM game JOIN game_publisher
ON game.appid=game_publisher.appid JOIN producer ON
game_publisher.producerid=producer.producerid WHERE producer.producername=",
                "language": "SELECT game.appid FROM game JOIN game_language
ON game.appid=game_language.appid JOIN language ON
game_language.languageid=language.languageid WHERE language.languagename=",
                "tag": "SELECT game.appid FROM game JOIN game_tag ON
game.appid=game_tag.appid JOIN tag ON game_tag.tagid=tag.tagid WHERE
tag.tagname="
                }
            agg_combo=QComboBox()
            agg_combo.addItems(aggs)
            sel_combo=QComboBox()
            sel_combo.addItems(list(sel_dic.keys()))
            cats_combo1=QComboBox()
```

```
cats_combo1.addItems(cats)
cats_combo2=QComboBox()
cats_combo2.addItems(cats)
combo_layout=QHBoxLayout()
combo_layout.addWidget(QLabel("Select "))
combo_layout.addWidget(agg_combo)
of_label=QLabel(" of ")
combo_layout.addWidget(of_label)
combo_layout.addWidget(sel_combo)
combo_layout.addWidget(QLabel(" based on "))
combo_layout.addWidget(cats_combo1)
combo_layout.addWidget(QLabel(" and "))
combo_layout.addWidget(cats_combo2)

def agg_changed(i):
    new_agg=list(aggs.keys())[i]
    if (new_agg=="count"):
        sel_combo.setVisible(False)
        of_label.setVisible(False)
    else:
        sel_combo.setVisible(True)
        of_label.setVisible(True)
    pass
agg_combo.currentIndexChanged.connect(agg_changed)

table=QTableWidget()
selectButton=QPushButton("Select data")

def selectData():
    conn=sqlite3.connect("steam_games.db")
```

```
        cursor=conn.cursor()

        table.clear()

        cat1=cats[cats_combo1.currentIndex()]
        cat2=cats[cats_combo2.currentIndex()]
        cat_list1=cat_lists[cat1]
        cat_list2=cat_lists[cat2]
        cat_where1=cat_where[cat1]
        cat_where2=cat_where[cat2]

        table.setColumnCount(len(cat_list1))
        table.setHorizontalHeaderLabels([i if i!="" else "None" for i in
cat_lists[cat1]])
        table.setRowCount(len(cat_list2))
        table.setVerticalHeaderLabels([i if i!="" else "None" for i in
cat_lists[cat2]])

        what_name1=f"{cat1}name"
        if cat1=="developer" or cat1=="publisher":
            what_name1="producername"
        what_name2=f"{cat2}name"
        if cat2=="developer" or cat2=="publisher":
            what_name2="producername"

        agg=aggs[list(aggs.keys())[agg_combo.currentIndex()]]
        sel=sel_dic[list(sel_dic.keys())[sel_combo.currentIndex()]]
        what_agg=f"{agg}(game.{sel})"
        if sel=="price" or sel=="initialprice":
            what_agg=what_agg+"/100"
```

```
                where=buildQuery(self.selector.return_value())

                from_q1=f"game JOIN game_{cat1} ON
game.appid=game_{cat1}.appid JOIN {cat1} ON
game_{cat1}.{cat1}id={cat1}.{cat1}id"
                if cat2=="developer":
                    from_q=f"game JOIN game_developer ON
game.appid=game_developer.appid JOIN producer ON
game_developer.producerid=producer.producerid"
                if cat2=="publisher":
                    from_q=f"game JOIN game_publisher ON
game.appid=game_publisher.appid JOIN producer ON
game_publisher.producerid=producer.producerid"

                from_q2=f"game JOIN game_{cat2} ON
game.appid=game_{cat2}.appid JOIN {cat2} ON
game_{cat2}.{cat2}id={cat2}.{cat2}id"
                if cat2=="developer":
                    from_q=f"game JOIN game_developer ON
game.appid=game_developer.appid JOIN producer ON
game_developer.producerid=producer.producerid"
                if cat2=="publisher":
                    from_q=f"game JOIN game_publisher ON
game.appid=game_publisher.appid JOIN producer ON
game_publisher.producerid=producer.producerid"

                cat1p=cat1
                if (cat1=="developer" or cat1=="publisher"):
                    cat1p="producer"
                cat2p=cat2
```

```
        if (cat2=="developer" or cat2=="publisher"):
            cat2p="producer"
        q1=f"SELECT game_{cat1}.appid AS appid, {cat1p}.{cat1p}id AS {cat1p}id, {cat1p}.{cat1p}name AS {cat1p}name FROM game_{cat1} JOIN {cat1p} ON game_{cat1}.{cat1p}id={cat1p}.{cat1p}id"
        q2=f"SELECT game_{cat2}.appid AS appid, {cat2p}.{cat2p}id AS {cat2p}id, {cat2p}.{cat2p}name AS {cat2p}name FROM game_{cat2} JOIN {cat2p} ON game_{cat2}.{cat2p}id={cat2p}.{cat2p}id"
        query=f'SELECT {what_agg}, q1.{cat1p}name a, q2.{cat2p}name b, MAX(game.{sel}) FROM ({q1}) q1 JOIN ({q2}) q2 ON q1.appid=q2.appid JOIN game ON q1.appid=game.appid WHERE {where} GROUP BY a, b'
        print(query)
        cursor.execute(query)
        print("AAA")
        max_v=0
        count=0
        while(True):
            result=cursor.fetchone()
            if (result is None):
                break
            if (agg!="COUNT"):
                if (result[0]>max_v):
                    max_v=result[0]
            else:
                max_v=max_v+result[0]
            count=count+1
            i_index=cat_list1.index(result[1])
            j_index=cat_list2.index(result[2])
            item=QTableWidgetItem(str(result[0]))
            table.setItem(i_index, j_index, item)
```

```
                    if (agg!="COUNT"):
                        item.setBackground(QColor.fromHsv( min((result[0]/(result[3] if
result[3]!=0 else 0.001))*(255), 300), (125), (125) ))
                        print(result)
                print("m", max_v)
                for i in range(table.rowCount()):
                    for j in range(table.columnCount()):
                        item=table.item(i, j)
                        if item is None or item.text()=="":
                            table.setItem(i, j, QTableWidgetItem(""))
                            item=table.item(i, j)
                            item.setBackground(QColor.fromHsv( (125), (0), (59) ))
                        else:
                            t=item.text()
                            f=float(t)
                            print(item.background().color().hsvHue())
                            if agg!="COUNT":

item.setBackground(QColor.fromHsv( (item.background().color().hsvHue()),
min((f/max_v)*(170)+80, 240), (125) ))
                                else:
                                    print( (f/(max_v/count))*170 )
                                    item.setBackground(QColor.fromHsv( (125),
min(( f/(max_v/count))*(140), 240), (125) ))
                                if (item.background().color().hsvSaturation()>170) or
(item.background().color().hsvHue()>170):
                                    item.setForeground(QColor.fromHsv( (0), (0), (255) ))
                        pass
                conn.close()
                pass
```

```
            selectButton.clicked.connect(selectData)
            combo_layout.setAlignment(Qt.AlignLeft)
            self.content.addLayout(combo_layout)
            self.content.addWidget(selectButton)
            self.content.addWidget(table)
        else:
            for i in reversed(range(self.content.count())):
                deleteItem(i)
            self.argument_type="None"
        pass

    def type_changed(self, old_value, new_value):
        self.type_label.setText(new_value)
        self.render_content()

    def combo_index_changed(self, index):
        self.visualiser_type=visualiser_types[index]
        pass
    @property
    def visualiser_type(self):
        return self._visualiser_type
    @visualiser_type.setter
    def visualiser_type(self, new_value):
        old_value=self._visualiser_type
        self._visualiser_type=new_value
        self.type_changed(old_value, new_value)
app=QApplication()
window=MainWindow()
app.exec()
```

**Додаток В. Завантаження даних**

Завантаження набору даних за допомогою Steam Spy API і збереження набору даних у форматі JSON:

```
import requests
import json

session=requests.Session()
io=0
data={}
t=True
while(t):
    req=session.get("https://steamspy.com/api.php?request=all&page="+str(io))
    if req.status_code!=200:
        print("Failed")
        t=False
        break
    else:
        print(io)
        try:
            data_i=req.json()
            data=data | data_i
            io=io+1
        except:
            t=False
            break
            print("e")
```

```
        appdetails={}

        for i in data:
            appid=i
            print(appid)
            print(list(data.keys()).index(i), "/", len(data.keys()))
            #print(data.index(i)+"/"+data.count())

req=session.get("https://steamspy.com/api.php?request=appdetails&appid="+str(appi
d))
            try:
                data_i=req.json()
                appdetails[appid]=data_i
                print(appdetails[appid]["genre"])
            except:
                print("ERROR!")

        with open("appdetails.json", 'w') as f:
            f.write(json.dumps(appdetails, indent=4))
            pass
```

**Додаток Г. Формування бази даних**

Створення на основі даних з файлу JSON бази даних у СКБД SQLite:

```
import sqlite3
import json

conn=sqlite3.connect("steam_games.db")
cursor=conn.cursor()

with open("appdetails.json", 'r') as file:
    data=json.load(file)

cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND name="game"')
if (cursor.fetchall()==[]):
    cursor.execute('''CREATE TABLE game(
        appid INTEGER PRIMARY KEY,
        name TEXT,
        score_rank INTEGER,
        positive INTEGER,
        negative INTEGER,
        userscore INTEGER,
        owners_lower INTEGER,
        owners_upper INTEGER,
        average_forever NUMERIC,
        average_2weeks NUMERIC,
        median_forever NUMERIC,
        median_2weeks NUMERIC,
        price NUMERIC,
        initialprice NUMERIC,
```

```
            discount NUMERIC,
            ccu NUMERIC
            )''')

        cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="producer"')
        if (cursor.fetchall()==[]):
            cursor.execute('''CREATE TABLE producer(
                producerid INTEGER PRIMARY KEY,
                producername TEXT
            )''')

        cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="language"')
        if (cursor.fetchall()==[]):
            cursor.execute('''CREATE TABLE language(
                languageid INTEGER PRIMARY KEY,
                languagename TEXT
            )''')

        cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="genre"')
        if (cursor.fetchall()==[]):
            cursor.execute('''CREATE TABLE genre(
                genreid INTEGER PRIMARY KEY,
                genrename TEXT
            )''')

        cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="tag"')
```

```
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE tag(
            tagid INTEGER PRIMARY KEY,
            tagname TEXT
        )''')

    cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="game_developer"')
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE game_developer(
            appid INTEGER,
            producerid INTEGER,
            PRIMARY KEY (appid, producerid)
        )''')

    cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="game_publisher"')
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE game_publisher(
            appid INTEGER,
            producerid INTEGER,
            PRIMARY KEY (appid, producerid)
        )''')

    cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="game_language"')
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE game_language(
            appid INTEGER,
            languageid INTEGER,
```

```
            PRIMARY KEY (appid, languageid)
        )''')

    cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="game_genre"')
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE game_genre(
            appid INTEGER,
            genreid INTEGER,
            PRIMARY KEY (appid, genreid)
        )''')

    cursor.execute(f'SELECT name FROM sqlite_master WHERE type="table" AND
name="game_tag"')
    if (cursor.fetchall()==[]):
        cursor.execute('''CREATE TABLE game_tag(
            appid INTEGER,
            tagid INTEGER,
            votes INTEGER,
            PRIMARY KEY (appid, tagid)
        )''')

    for i in data:
        print(str(list(data.keys()).index(i))+"/"+str(len(data.keys())))
        entry=data[i]
        print(f'''({entry["appid"]},
        {entry["name"]},
        {entry["score_rank"]},
        {entry["positive"]},
        {entry["negative"]},
```

```
{entry["userscore"]},
{int(entry["owners"].split("..")[0].replace(',', ''))},
{entry["owners"].split("..")[1]},
{entry["average_forever"]},
{entry["average_2weeks"]},
{entry["median_forever"]},
{entry["median_2weeks"]},
{entry["price"]},
{entry["initialprice"]},
{entry["discount"]},
{entry["ccu"]})'')
cursor.execute(f''INSERT INTO game
(appid, name, score_rank, positive, negative, userscore, owners_lower,
owners_upper, average_forever, average_2weeks, median_forever, median_2weeks,
price, initialprice, discount, ccu)
VALUES
({entry["appid"]},
"{entry["name"].replace('"', '"')}",
{"NULL" if (entry["score_rank"])=="" else (entry["score_rank"])},
{entry["positive"]},
{entry["negative"]},
{entry["userscore"]},
{int(entry["owners"].split("..")[0].replace(',', ''))},
{int(entry["owners"].split("..")[1].replace(',', ''))},
{entry["average_forever"]},
{entry["average_2weeks"]},
{entry["median_forever"]},
{entry["median_2weeks"]},
{"NULL" if (entry["price"]) is None else (entry["price"])},
{"NULL" if (entry["initialprice"]) is None else (entry["initialprice"])},
```

```
        {"NULL" if (entry["discount"]) is None else (entry["discount"])},
        {entry["ccu"]})
        ''')
        appid=entry["appid"]
        if entry["languages"] is not None:
            languages=entry["languages"].split(', ')
            language_string=""
            for language in languages:
                cursor.execute(f"SELECT EXISTS (SELECT 1 FROM language WHERE
languagename='{language}')")
                isthere=cursor.fetchone()[0]
                if isthere==0:
                    new_id=0
                    cursor.execute(f"SELECT MAX(languageid) FROM language")
                    max_id=cursor.fetchone()[0]
                    if (max_id is not None):
                        new_id=max_id+1
                    cursor.execute(f"INSERT INTO language (languageid, languagename)
VALUES ({new_id}, '{language}')")

                cursor.execute(f"SELECT languageid FROM language WHERE
languagename='{language}'")
                language_id=cursor.fetchone()[0]
                cursor.execute(f"INSERT INTO game_language (appid, languageid)
VALUES ({appid}, '{language_id}')")
                language_string=language_string+language+"|"
            print(language_string)

        developers=entry["developer"].split(', ')
        developer_string=""
```

```
        for developer in developers:
            developer=developer.replace("'", "''")
            cursor.execute(f'SELECT EXISTS (SELECT 1 FROM producer WHERE
producername="{developer}")')
            isthere=cursor.fetchone()[0]
            if isthere==0:
                new_id=0
                cursor.execute(f"SELECT MAX(producerid) FROM producer")
                max_id=cursor.fetchone()[0]
                if (max_id is not None):
                    new_id=max_id+1
                cursor.execute(f'INSERT INTO producer (producerid, producername)
VALUES ({new_id}, "{developer}")')

            cursor.execute(f'SELECT producerid FROM producer WHERE
producername="{developer}"')
            producer_id=cursor.fetchone()[0]
            try:
                cursor.execute(f"INSERT INTO game_developer (appid, producerid)
VALUES ({appid}, '{producer_id}')")
                developer_string=developer_string+developer+"|"
            except:
                print("WEIRD?")
        print(developer_string)

        publishers=entry["publisher"].split(', ')
        publisher_string=""
        for publisher in publishers:
            publisher=publisher.replace("'", "''")
```

```
            cursor.execute(f'SELECT EXISTS (SELECT 1 FROM producer WHERE
producername="{publisher}")')
            isthere=cursor.fetchone()[0]
            if isthere==0:
                new_id=0
                cursor.execute(f"SELECT MAX(producerid) FROM producer")
                max_id=cursor.fetchone()[0]
                if (max_id is not None):
                    new_id=max_id+1
                cursor.execute(f'INSERT INTO producer (producerid, producername)
VALUES ({new_id}, "{publisher}")')

            cursor.execute(f'SELECT producerid FROM producer WHERE
producername="{publisher}"')
            producer_id=cursor.fetchone()[0]
            try:
                cursor.execute(f"INSERT INTO game_publisher (appid, producerid)
VALUES ({appid}, '{producer_id}')")
                publisher_string=publisher_string+developer+"|"
            except:
                print("WEIRD?")
        print(publisher_string)

        genres=entry["genre"].split(', ')
        genre_string=""
        for genre in genres:
            genre=genre.replace("", "")
            cursor.execute(f'SELECT EXISTS (SELECT 1 FROM genre WHERE
genrename="{genre}")')
            isthere=cursor.fetchone()[0]
```

```
            if isthere==0:
                print("THERE IS NOT")
                new_id=0
                cursor.execute(f"SELECT MAX(genreid) FROM genre")
                max_id=cursor.fetchone()[0]
                if (max_id is not None):
                    new_id=max_id+1
                cursor.execute(f'INSERT INTO genre (genreid, genrename) VALUES
({new_id}, "{genre}")')

            cursor.execute(f'SELECT genreid FROM genre WHERE
genrename="{genre}"')
            genre_id=cursor.fetchone()[0]
            try:
                cursor.execute(f"INSERT INTO game_genre (appid, genreid) VALUES
({appid}, '{genre_id}')")
                genre_string=genre_string+genre+"|"
            except:
                print("WEIRD?")
        print(genre_string)

        tags=entry["tags"]
        tag_string=""
        for tag in tags:
            tag=tag.replace("", """)
            cursor.execute(f'SELECT EXISTS (SELECT 1 FROM tag WHERE
tagname="{tag}")')
            isthere=cursor.fetchone()[0]
            if isthere==0:
                new_id=0
```

```
                cursor.execute(f"SELECT MAX(tagid) FROM tag")
                max_id=cursor.fetchone()[0]
                if (max_id is not None):
                    new_id=max_id+1
                cursor.execute(f'INSERT INTO tag (tagid, tagname) VALUES ({new_id},
"{tag}")')

                cursor.execute(f'SELECT tagid FROM tag WHERE tagname="{tag}"')
                tag_id=cursor.fetchone()[0]
                try:
                    cursor.execute(f"INSERT INTO game_tag (appid, tagid, votes) VALUES
({appid}, '{tag_id}', {tags[tag]})")
                    tag_string=tag_string+tag+"|"
                except:
                    print("WEIRD?")
            print(tag_string)

            conn.commit()
            conn.close()
```