

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка ігрового застосунку для гри пінг-понг»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Олександр ВІНЦИК

(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Олександр ВІНЦИК

Керівник: Максим КУКЛІНСЬКИЙ
канд. техн. наук

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Вінцику Олександрю Анатолійовичу

1. Тема кваліфікаційної роботи: «Розробка ігрового застосунку для гри пінг-понг»
керівник кваліфікаційної роботи канд. техн. наук Максим КУКЛІНСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних
технологій від «20» лютого 2026 року № 51.
2. Строк подання кваліфікаційної роботи «29» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розвитку дрібної моторики, аркадні ігри, принципи побудови ігрової фізики, методи збору та аналізу статистики користувача.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
 1. Огляд та аналіз існуючих технологій тренування дрібної моторики та сучасних ігор.
 2. Засоби розробки та проектування застосунку для тренування дрібної моторики.
 3. Програмна реалізація та опис функціонування застосунку.
 4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма прецедентів.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд алгоритмів ігрової фізики, методів збору статистики та технологій аналізу дій користувача.	08.03-14.03.2026	
4	Проектування застосунку програмного застосунку	15.03-29.03.2026	
5	Програмна реалізація ігрового застосунку та аналітичного застосунку	30.03-23.04.2026	
6	Тестування застосунку	24.04-26.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	04.05-07.05.2026	
8	Розробка демонстраційних матеріалів	08.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти

_____ (підпис)

Олександр ВІНЦИК

Керівник
кваліфікаційної роботи

_____ (підпис)

Максим КУКЛІНСЬКИЙ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 92 сторінок, 43 рисунків, 2 таблиць, 22 використаних джерел.

Метою кваліфікаційної роботи є розробка гри для тренування дрібної моторики та аналізу дій користувача на основі ігрового застосунку типу «Пінг-понг».

Об'єктом дослідження є процес взаємодії користувача з ігровими застосунками для розвитку координації рухів та швидкості реакції.

Предметом дослідження є методи та засоби програмної реалізації ігрових застосунків та аналітичних систем для збору й аналізу статистики дій користувача.

У кваліфікаційній роботі проведено аналіз сучасних аркадних ігрових застосунків, засобів розробки та методів використання ігор для розвитку дрібної моторики. Розглянуто особливості застосування інтерактивних ігрових систем для тренування швидкості реакції, координації рухів та точності керування.

У результаті виконання роботи було розроблено програмний застосунок, який складається з двох окремих модулів.

Ігровий модуль реалізує аркадну гру типу «Пінг-понг» із системою рівнів, керуванням платформою, фізикою руху м'яча, системою підрахунку балів, звуковими ефектами та записом статистики дій користувача.

Аналітичний модуль забезпечує зчитування та обробку статистичних даних із CSV-файлів, аналіз швидкості реакції користувача, аналіз точності керування, кількості корисних та зайвих рухів, а також побудову графіків і таблиць результатів.

Для реалізації програмного застосунку використано мову програмування C++, мову Python, бібліотеки стандартного введення-виведення, засоби роботи з CSV-файлами та інструменти побудови графічної аналітики

Практична цінність роботи полягає у можливості використання розробленого застосунку для тренування дрібної моторики, розвитку швидкості реакції користувачів та аналізу ефективності взаємодії з ігровим застосунком.

КЛЮЧОВІ СЛОВА: C++, PYTHON, ПІНГ-ПОНГ, ІГРОВИЙ ЗАСТОСУНОК,
АНАЛІТИКА, ДРІБНА МОТОРИКА, CSV, ІГРОВИЙ МОДУЛЬ, АНАЛІТИЧНИЙ
МОДУЛ

ЗМІСТ

ВСТУП.....	10
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	12
1.1 Особливості використання ігрових застосунків для розвитку дрібної моторики	12
1.2 Основні відомості про гру «Пінг-понг».....	14
1.3 Опис сучасних ігрових додатків	18
1.4 Аналіз ігрових додатків та їх статистика.....	23
1.5 Аналіз систем аналітики ігрових застосунків	31
1.6 Визначення функціональних та нефункціональних вимог до програмного продукту	35
2. ЗАСОБИ РОЗРОБКИ ТА ПРОЕКТУВАННЯ ПРОГРАМИ	39
2.1 Аналіз мов програмування	39
2.1.1 C++	39
2.1.2 C#.....	40
2.1.3 Java.....	40
2.1.4 Python	41
2.1.5 PHP	42
2.2 Проектування структури програмного модулю	43
2.3 Проектування ігрового модуля	45
2.4 Проектування аналітичного модуля	48
2.5 UML-діаграма та схема взаємодії модулів	52
2.6 Особливості інтерфейсу користувача програмного застосунку.....	56
3 РЕАЛІЗАЦІЯ ПРОЄКТУ	

.....	62
3.1 Налаштування проєкту та середовища розробки	62
3.2 Реалізація ігрового модуля мовою C++	63
3.3 Реалізація аналітичного модуля мовою Python	66
3.4 Механізм збору та збереження статистичних даних	69
ВИСНОВКИ	72
ПЕРЕЛІК ПОСИЛАНЬ	74
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	76
ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ	84

ВСТУП

Актуальність: в сучасних умовах розвитку інформаційних технологій комп'ютерні ігри є не лише засобом розваги, але й ефективним інструментом розвитку когнітивних здібностей, швидкості реакції та координації рухів користувача. Особливої актуальності набуває використання інтерактивних ігрових застосунків для розвитку дрібної моторики рук, що є важливою складовою координації рухів, точності дій та швидкості взаємодії користувача з цифровими системами.

Аркадні ігри завдяки своїй динамічності, простому керуванню та необхідності швидкого реагування дозволяють тренувати уважність, координацію рухів і точність натискань клавіш. Одним із найбільш відомих прикладів таких ігор є «Пінг-понг», принцип роботи якої базується на постійному контролі руху платформи та взаємодії з рухомим об'єктом.

Сучасні програмні засоби дозволяють не лише реалізувати ігровий процес, але й здійснювати аналіз дій користувача під час гри. Використання систем аналітики відкриває можливість оцінювання швидкості реакції, точності рухів, кількості корисних та зайвих дій користувача. Такий підхід дозволяє поєднати ігровий процес із аналізом ефективності взаємодії користувача із програмною системою.

У зв'язку з цим актуальною є розробка ігрового застосунку, який поєднує ігровий застосунок та аналітичну систему для збору, обробки та аналізу статистики дій користувача.

Об'єктом дослідження є процес взаємодії користувача з ігровими застосунками для розвитку дрібної моторики та швидкості реакції.

Предметом дослідження є методи та засоби програмної реалізації аркадних ігрових застосунків і систем аналітики користувацьких дій.

Метою роботи є розробка програмного застосунку для розвитку дрібної моторики та аналізу дій користувача на основі аркадного ігрового застосунку типу «Пінг-понг».

Методи дослідження: під час виконання кваліфікаційної роботи були використані методи аналізу та порівняння сучасних програмних систем, методи об'єктно-орієнтованого програмування, методи проєктування програмного забезпечення, а також методи аналізу статистичних даних користувача.

Наукова новизна роботи полягає у поєднанні консольного ігрового застосунку «Ping-Pong» із модулем аналітики моторних навичок користувача. У межах проєкту реалізовано механізм збору та аналізу статистики дій гравця, що дозволяє оцінювати швидкість реакції, точність керування та ефективність взаємодії користувача з ігровим середовищем.

Особливістю розробки є інтеграція ігрового модуля мовою програмування C++ із аналітичним модулем на Python для автоматизованої обробки даних та побудови графічної статистики, що розширює можливості використання класичних аркадних ігор у навчальних та дослідницьких цілях.

Практична значущість роботи полягає у можливості використання розробленого програмного застосунку для розвитку дрібної моторики, тренування швидкості реакції та аналізу ефективності дій користувача під час взаємодії з ігровим застосунком.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості використання ігрових застосунків для розвитку дрібної моторики

У сучасному суспільстві інформаційні технології дедалі більше інтегруються у повсякденне життя людини. Комп'ютерні системи використовуються у навчанні, професійній діяльності, медицині, наукових дослідженнях та сфері розваг. Разом із розвитком інтерактивних технологій значної популярності набули програмні застосунки, спрямовані не лише на організацію дозвілля, але й на розвиток когнітивних та моторних навичок користувача. Одним із таких напрямів є використання ігрових застосунків для розвитку дрібної моторики рук та швидкості реакції.

Дрібна моторика являє собою систему точних координованих рухів кистей рук і пальців, які виконуються за участю нервової, м'язової та зорової систем людини [11]. Рівень розвитку дрібної моторики безпосередньо впливає на точність рухів, швидкість реакції, уважність та координацію дій користувача під час роботи з технічними пристроями [14]. Особливого значення розвиток моторики набуває в умовах постійної взаємодії людини із цифровими системами, де точність і швидкість введення інформації є важливими складовими ефективної роботи.

У процесі взаємодії з комп'ютерними системами користувач постійно виконує велику кількість дрібних рухів. До таких дій належать натискання клавіш клавіатури, керування комп'ютерною мишею, швидка зміна положення рук, а також координація рухів відповідно до візуальної інформації на екрані. Саме тому сучасні інтерактивні програмні системи можуть використовуватися як інструмент тренування моторних навичок та швидкості реакції.

Особливу роль у розвитку дрібної моторики відіграють аркадні ігрові застосунки. Такі ігри характеризуються високою динамічністю, швидкою зміною ігрових подій та необхідністю постійного контролю дій користувача. Під час гри

користувач змушений оперативно аналізувати ситуацію, прогнозувати траєкторію руху об'єктів та виконувати точні рухи за допомогою клавіатури або інших засобів керування.

Під час використання аркадних ігор активно розвиваються координація рухів, концентрація уваги та швидкість реакції [11-13]. Користувач повинен постійно контролювати рухомі об'єкти та швидко змінювати власні дії відповідно до зміни ігрової ситуації. У результаті формується тісний взаємозв'язок між зоровим сприйняттям інформації та моторною активністю рук.

Особливо ефективними для розвитку моторики є ігри типу «Пінг-понг», у яких користувач керує платформою для відбиття м'яча. Основна складність таких ігор полягає у необхідності швидко реагувати на зміну траєкторії руху м'яча та точно позиціонувати платформу у межах ігрового поля. Навіть незначна помилка у швидкості реакції або координації рухів може призвести до завершення гри.

У процесі гри користувач виконує велику кількість однотипних швидких дій, що позитивно впливає на розвиток моторної пам'яті. Регулярне повторення подібних рухів дозволяє покращити точність натискань клавіш та підвищити швидкість взаємодії з програмною системою. Крім того, постійна концентрація уваги сприяє розвитку когнітивних здібностей користувача.

Сучасні дослідження у сфері інтерактивних технологій показують, що використання ігрових застосунків може позитивно впливати на розвиток реакції та координації рухів [11-15]. Саме тому комп'ютерні ігри все частіше використовуються у навчальних та тренувальних системах. Особливого поширення набули програмні застосунки, які поєднують ігровий процес із системами аналізу дій користувача.

Використання систем аналітики дозволяє не лише організувати ігровий процес, але й виконувати оцінювання ефективності взаємодії користувача із програмою. У сучасних ігрових системах можуть аналізуватися:

- швидкість натискань клавіш;
- точність керування;
- кількість помилкових дій;

- тривалість реакції;
- кількість корисних та зайвих рухів.

Подібний підхід дозволяє перетворити звичайний ігровий застосунок на систему аналізу моторних навичок користувача.

У межах роботи розробляється застосунок, який поєднує ігровий модуль та аналітичну систему. Ігровий модуль реалізує класичну аркадну механіку гри типу «Пінг-понг», а аналітичний модуль забезпечує збір та обробку статистичних даних користувача.

Під час роботи програми система виконує запис статистики дій користувача у CSV-файли [12]. Надалі отримані дані аналізуються аналітичним модулем мовою програмування Python. Результати аналізу можуть відображатися у вигляді таблиць, графіків та статистичних звітів.

Особливістю розробленого програмного застосунку є поєднання інтерактивного ігрового процесу із системою оцінювання ефективності дій користувача. У процесі аналізу враховуються:

- швидкість реакції користувача;
- точність керування платформою;
- кількість натискань клавіш;
- кількість помилкових рухів;
- ефективність проходження ігрових рівнів.

Таким чином, використання інтерактивних ігрових застосунків для розвитку дрібної моторики є перспективним напрямом розвитку сучасних програмних систем. Поєднання динамічного ігрового процесу із системою аналізу статистичних даних дозволяє не лише підвищити зацікавленість користувача, але й оцінити ефективність його взаємодії із програмним забезпеченням.

1.2 Основні відомості про гру «Пінг-понг»

Дипломний проєкт присвячений розробці логічної гри «Пінг-понг», яка була створена японською компанією Taito Corporation та випущена компанією Atari 29 листопада 1972 року.

Ідею створення гри у стилі пінг-понгу запропонував Нолан Бушнелл своєму співробітнику - програмісту Аллан Алькорн. На той момент Алькорн не мав досвіду у розробці ігор, тому проєкт Pong став для нього навчальним. Варто зазначити, що подібна ідея вже була реалізована у системі Magnavox Odyssey, що згодом призвело до судового позову проти Atari.

У період різдвяних свят 1975 року компанія Atari представила домашню версію Pong, яка продавалася через мережу магазинів Sears. Це рішення виявилось комерційно успішним і сприяло появі численних аналогів, зокрема приставки Color TV Game 6 від Nintendo. Згодом гру було адаптовано для різних домашніх і портативних платформ.

Pong стала першою грою, створеною компанією Atari. Після невдалого запуску проєкту Computer Space Нолан Бушнелл прийняв рішення заснувати нову компанію з метою розробки та ліцензування ігор. Першим контрактом стала угода з Bally Manufacturing Corporation щодо створення гоночної гри.

Незабаром після заснування компанії Бушнелл запросив до співпраці Аллана Алькорна, який мав досвід у галузі електротехніки та інформатики. Раніше Бушнелл і Тед Дабні вже працювали разом із ним у компанії Ampex. Незважаючи на відсутність досвіду у створенні відеоігор, Алькорн отримав навчальне завдання для ознайомлення з процесом розробки.

Зокрема, Бушнелл запропонував йому створити просту гру, яка складалася з рухомої точки, двох ракеток та системи підрахунку очок.

У 2011 році Нолан Бушнелл зазначив, що натхнення для створення гри він отримав від більш ранніх версій електронного тенісу, з якими мав змогу ознайомитися. Зокрема, ще у 1964 році під час навчання в коледжі він грав у подібну гру на комп'ютері PDP-1. Водночас Аллан Алькорн стверджував, що основою для створення гри стала розробка для Magnavox Odyssey, яку Бушнелл побачив у травні 1972 року під час відвідування виставки Magnavox Profit Caravan

у місті Берлінгейм, штат Каліфорнія. Незважаючи на те, що якість цієї гри здалася йому недостатньою, Бушнелл доручив реалізацію проєкту Алькорну.

Алькорн уже мав досвід роботи з TTL-мікросхемами, тому спочатку ознайомився з кресленнями Бушнелла, створеними під час розробки Computer Space. Вважаючи їх недостатньо зрозумілими, він вирішив розробляти проєкт, спираючись на власні інженерні знання. У процесі роботи Алькорн замінив деякі базові компоненти на більш доступні за вартістю рішення.

Крім того, він дійшов висновку, що початкова концепція гри є занадто простою, оскільки м'яч рухався за однаковою траєкторією. У вдосконаленій версії Pong ракетки були поділені на сегменти, що дозволило змінювати напрямок руху м'яча залежно від точки контакту. При попаданні в центральну частину ракетки м'яч відбивався під стандартним кутом, ближче до країв - під меншим, а при ударі в крайні сегменти - змінював напрямок у протилежний бік.

Спочатку гра мала постійну швидкість, що, на думку Алькорна, робило її недостатньо цікавою. Тому він додав механізм прискорення м'яча, внаслідок чого його швидкість поступово зростала. Це зробило ігровий процес більш динамічним і стратегічним, наблизивши його за відчуттями до гри в сквош.

Також у грі була особливість, пов'язана з тим, що ракетки не могли досягати верхньої межі екрана. Це було зумовлено технічними обмеженнями під час проєктування, оскільки використовувалися прості мікросхеми. Однак замість усунення цього недоліку Алькорн вирішив залишити його як елемент геймплею, щоб запобігти нескінченній грі між досвідченими гравцями.

Через три місяці після початку розробки Нолан Бушнелл повідомив Аллан Алькорн, що в грі необхідно реалізувати реалістичні звукові ефекти, зокрема звуки натовпу. Тед Дабні висловив побажання, щоб гра реагувала на поразку гравця характерними звуками, такими як свист або шипіння. Проте Алькорн зіткнувся з обмеженнями щодо розміщення електронних компонентів і не мав чіткого розуміння, як реалізувати подібні звуки за допомогою цифрових схем.

Після аналізу генераторів синхронізації він виявив, що їх можна використовувати для створення звукових сигналів різної тональності. Це дозволило реалізувати необхідні звукові ефекти у грі.

Для створення прототипу Алькорн придбав чорно-білий телевізор марки Hitachi за 75 доларів США, встановив його у дерев'яному корпусі висотою приблизно 1,2 метра та змонтував необхідну електронну схему шляхом підключення проводів до відповідних елементів. Створений прототип справив сильне враження на Бушнелла та Дабні, після чого було вирішено перевірити його в реальних умовах.

У серпні 1972 року Бушнелл і Алькорн встановили прототип гри Pong у місцевому барі «Таверна Енді Кепп». Це місце було обрано завдяки хорошим відносинам із менеджером закладу Біллом Гаддісом, якому Atari постачала пінбольні автомати. Прототип розмістили поряд з іншими розважальними пристроями, такими як музичний автомат, пінбольні машини та Computer Space.

Гра одразу отримала позитивні відгуки відвідувачів, і її популярність стрімко зростала протягом наступних днів. Згодом Бушнелл вирушив до Чикаго, щоб представити Pong керівництву компаній Bally та Midway Manufacturing, плануючи використати гру як альтернативу раніше запланованому проєкту.

Під час перевірки прототипу Алькорн виявив, що причиною його несправності був переповнений монетоприймач. Проблема вдалося швидко усунути шляхом заміни відсіку для монет на більш місткий.

Дізнавшись про популярність гри, Нолан Бушнелл дійшов висновку, що компанія Atari зможе отримати більший прибуток, якщо самостійно займатиметься виробництвом, а не передаватиме права за ліцензією. Водночас компанії Bally і Midway вже проявили зацікавленість у випуску гри. З метою збереження ділових відносин Бушнелл повідомив кожній із цих компаній, що інша сторона не виявила належного інтересу до проєкту. У результаті обидві компанії відмовилися від співпраці.

Разом із тим Бушнелл зіткнувся з труднощами у пошуку фінансування для розвитку Pong. Банківські установи сприймали гру як різновид пінболу, який у той

час асоціювався з незаконною діяльністю. Зрештою компанія отримала кредитну лінію від Wells Fargo, що дозволило розширити виробничі приміщення та організувати складальну лінію.

Офіційний анонс гри Pong відбувся 29 листопада 1972 року. Для налагодження виробництва компанія шукала працівників через місцеві служби зайнятості, однак попит значно перевищував можливості виробництва. Перші аркадні автомати виготовлялися повільно - приблизно по десять одиниць на день, причому частина з них не проходила контроль якості. Згодом процес було оптимізовано, що дало змогу значно збільшити обсяги виробництва.

За словами Аллан Алькорн, собівартість одного автомата становила близько 500 доларів США, тоді як продажна ціна складала приблизно 1200 доларів. Уже в 1973 році Atari почала експортувати Pong в інші країни через міжнародних партнерів.

Ігровий процес передбачає використання трьох основних елементів: ігрового поля, платформи та м'яча. Розмір поля становить 40×20 умовних одиниць, а довжина платформи - 6 одиниць.

Pong є спрощеною моделлю настільного тенісу. М'яч рухається по екрану за прямолінійною траєкторією, змінюючи напрямок після зіткнення з межами поля відповідно до кута відбиття.

Суть гри полягає у тому, що гравець переміщує платформу горизонтально, намагаючись не допустити виходу м'яча за межі ігрового поля.

Гра розрахована на одного користувача, який керує платформою за допомогою клавіш «A» та «D». Основна мета - набрати максимальну кількість балів, утримуючи м'яч у межах поля та не допускаючи його пропуску.

1.3 Опис сучасних ігрових додатків

Комп'ютерні ігри є об'єктом дослідження в багатьох сферах сучасної науки. Водночас їх різноманітність і складність потребують комплексного підходу до

вивчення цього явища. Комп'ютерна гра являє собою програмний продукт, призначений для організації ігрового процесу на персональному комп'ютері.

Ігровий процес зазвичай позначається терміном «геймплей». Це поняття охоплює не лише безпосередній перебіг гри, а й потенційні можливості взаємодії користувача з нею. Таким чином, геймплей можна розглядати як сукупність функціональних характеристик гри як програмного продукту.

Оскільки комп'ютерні ігри поєднують у собі графіку, звук та інтерактивність, у науковій літературі також використовується термін «відеогра». Особливо це стосується ігор із розвиненим ігровим світом, які можуть слугувати основою для створення фільмів, книг та інших медіапродуктів. Це дозволяє розглядати комп'ютерні ігри як окремий вид мистецтва.

У 2011 році у США комп'ютерні ігри були офіційно визнані видом мистецтва нарівні з театром і кінематографом. Це стало можливим завдяки значному прогресу у сфері графіки, анімації, звукових і тактильних ефектів, а також розвитку відкритих ігрових світів і варіативності ігрового процесу. Прикладами таких ігор є *Uncharted*, *Heavy Rain* та *Batman: Arkham series*.

Комп'ютерні ігри також активно використовуються у наукових дослідженнях. Окремі їх види дозволяють залучати гравців до дослідницьких процесів. Крім того, завдяки широкій популярності та можливості одночасної участі великої кількості користувачів, виник новий напрям - кіберспорт, що являє собою змагання у відеоіграх.

Вплив комп'ютерних ігор на сучасне суспільство постійно зростає. Зокрема, спостерігається інтеграція ігрових елементів у неігрові програмні продукти, що отримала назву гейміфікація.

Функціонування комп'ютерних ігор забезпечується за допомогою різних технічних засобів, включаючи персональні комп'ютери та мультимедійні пристрої. Окрім цього, існують ігрові платформи різних типів: стаціонарні комп'ютери, ноутбуки, планшети, а також спеціалізовані ігрові консолі, такі як PlayStation, Xbox 360, Nintendo Wii та Sony PSP. Також до цієї категорії належать мобільні пристрої, зокрема кишенькові комп'ютери, смартфони та комунікатори.

Окремо варто зазначити ігри, що реалізуються за допомогою ігрових автоматів та подібних пристроїв, які також є складовою частиною індустрії ігор. Варто зазначити, що комп'ютерні ігри пройшли тривалий шлях розвитку. З моменту своєї появи у 1940-х роках вони поступово стали важливою складовою сфери дозвілля. Спочатку орієнтовані переважно на дитячу аудиторію, з часом ігри еволюціонували разом із користувачами, набуваючи більш складних і «дорослих» характеристик.

У зв'язку з широким розповсюдженням комп'ютерних ігор перед людиною постають питання морального вибору та свободи дій у віртуальному середовищі.

Особливістю ігрового процесу є можливість одночасного існування в різних віртуальних ролях. Наприклад, користувач може виступати супергероєм, що рятує світ, згодом - гонщиком, який бере участь у перегонах, а ще через деякий час - доглядати за віртуальним персонажем. Перехід між такими ролями відбувається практично миттєво, що супроводжується зміною моделей поведінки та соціальних ролей.

Таким чином, гравець змушений швидко адаптуватися до нових умов, «приміряючи» різні образи та ролі, фактично створюючи альтернативні варіанти власної ідентичності.

Важливим аспектом є також можливість збереження прогресу в грі. Це дозволяє користувачеві повертатися до попередніх етапів після невдачі та повторювати спроби. Такий механізм можна розглядати як своєрідний «порятунк», оскільки він дає змогу відновити стан гри до моменту поразки.

Функція збереження результатів з'явилася не одразу. У консольних вона почала активно використовуватися з появою таких пристроїв, як Sega Mega Drive та Super Nintendo Entertainment System, а також приблизно в той самий період була впроваджена і на персональних комп'ютерах.

У процесі пошуку оптимального способу проходження гри користувач часто змушений багаторазово «програвати» та повторювати спроби, інколи свідомо обираючи ризиковані дії. Для досягнення результату необхідно постійно балансувати між можливістю поразки та шансом на успіх. У деяких випадках

гравців навіть цікавить сам процес поразки, оскільки він супроводжується новими візуальними ефектами або анімацією.

Життя у віртуальному середовищі відрізняється від реального тим, що після завершення гри її можна розпочати заново. Це надає користувачу можливість багаторазово проживати різні сценарії, змінюючи свої дії та рішення. Наприклад, у ситуаціях вибору гравець може перевірити різні варіанти розвитку подій, повертаючись до попередніх етапів і повторюючи проходження.

Подібна особливість створює унікальні умови, коли користувач може експериментувати з різними моделями поведінки, змінюючи хід подій відповідно до власних уподобань. Водночас існують і певні негативні аспекти. Наприклад, відомий феномен тамагочі, коли діти, звикаючи до умовної «безсмертності» віртуального вихованця, переносили це сприйняття на реальний світ.

Розвиток комп'ютерних ігор супроводжується регулярною зміною ігрових концепцій приблизно кожні кілька років. З'являються нові напрями, удосконалюються вже існуючі, що сприяє формуванню нових жанрів та значному розширенню різноманіття ігор.

Для систематизації такого різноманіття використовується класифікація за різними ознаками. Зокрема:

- за жанром - залежно від ігрових завдань та способів взаємодії;
- за кількістю гравців - одиночні, багатокористувацькі та онлайн-ігри;
- за способом подання інформації - залежно від візуального представлення ігрового процесу.

Розглянемо один із найбільш розвинених на сьогодні жанрів - аркадні ігри. Сам термін «аркада» виник ще в період популярності ігрових автоматів і спочатку використовувався для позначення ігор із простою механікою. З часом цим поняттям почали називати ігри, у яких основний акцент зроблено на динамічних діях, а геймплей характеризується відносною простотою та нереалістичністю.

Термін «аркада» застосовується як для визначення жанру, так і для позначення рівня складності гри. Основною метою таких ігор зазвичай є

проходження рівнів, збирання бонусів та виконання завдань, часто в умовах обмеженого часу. Важливою характеристикою аркад є їхня легкість у засвоєнні.

Класичні аркадні ігри мають такі особливості:

- простий, але динамічний ігровий процес;
- спрощене графічне оформлення;
- велика кількість рівнів або їх умовна нескінченність;
- система нарахування балів за виконання різних дій;
- коротка тривалість проходження;
- наявність кількох спроб (життів) для повторного проходження.

У більшості випадків гравець має повний огляд ігрового поля, що дозволяє приймати рішення на основі всієї доступної інформації. За виконання певних дій або досягнення цілей нараховуються бали. Тривалість однієї гри для середнього користувача зазвичай становить кілька хвилин, тоді як досвідчені гравці можуть утримувати гру значно довше.

Через цю особливість у багатьох аркадних іграх присутні таблиці рекордів, де гравці можуть вводити свої ініціали та порівнювати результати з іншими, що створює додатковий елемент змагання.

Аркадні ігри відзначаються тим, що їх легко освоїти, однак досягнення високого рівня майстерності є складним завданням. При цьому поразка гравця зазвичай є наслідком його власних дій.

У таких іграх відсутня необхідність у складних комбінаціях клавіш або тривалому навчанні. Рідко використовуються додаткові механіки, такі як система здоров'я, щити чи бонуси сили. Це пояснюється прагненням розробників зробити ігри максимально доступними для широкої аудиторії.

Водночас простота ігрового процесу не означає його примітивність - аркадні ігри можуть бути добре продуманими, збалансованими та якісно реалізованими.

Аркадні ігри, як правило, не мають складної сюжетної лінії, і ця тенденція зберігається й сьогодні. Для таких ігор важливо, щоб користувач міг швидко зрозуміти суть того, що відбувається на екрані - незалежно від тематики, будь то наукова фантастика, військові дії чи спортивні змагання. Багато аркадних ігор

мають власний унікальний стиль і атмосферу, як, наприклад, Pac-Man або Q*bert. Розробники класичних аркад не приділяли значної уваги глибокому опрацюванню сюжету, оскільки головним було забезпечення зрозумілого та динамічного ігрового процесу.

Серед найвідоміших ігор цього жанру можна виділити Pac-Man, Donkey Kong, Tetris та Breakout. Вони є типовими представниками класичних аркад.

Аркадні ігри характеризуються простим, але захопливим геймплеєм, де важливу роль відіграють яскраве візуальне оформлення та система нарахування балів. Вони користуються популярністю як серед гравців, так і серед розробників завдяки широкій цільовій аудиторії та здатності швидко зацікавити користувача.

Аркади можна розглядати як окремий жанр, а також як групу піджанрів, до яких належать:

- аркадні гонки - з простим і нереалістичним керуванням;
- класичні аркади - ігри, що виникли на ігрових автоматах;
- бійки - ігри, де гравець керує персонажем і виконує різні бойові прийоми;
- платформери - ігри, у яких потрібно проходити рівні, пересуваючись між платформами;
- скролери - ігри з рухомим екраном та постійною появою противників;
- тир - імітація стрілецьких тренажерів.

Штучний інтелект в аркадних іграх зазвичай реалізується у вигляді простих алгоритмів поведінки, які не адаптуються до дій гравця. Незважаючи на це, налаштування такого ШІ є складним завданням, оскільки розробникам необхідно знайти баланс між цікавістю гри для користувача та її комерційною ефективністю.

1.4 Аналіз ігрових додатків та їх статистика

З розвитком ігрової індустрії та зростанням популярності аркадних ігор цей жанр почав домінувати на ринку. У результаті почала з'являтися велика кількість схожих за змістом і механікою ігор, що призвело до значного збільшення кількості аналогів.

Одним із перших, хто популяризував аркадні ігри серед широкої аудиторії, був Нолан Бушнелл - засновник компанії Atari. У 1977 році він представив один із найуспішніших своїх проєктів - «Chuck E. Cheese's Pizza Time Theatre». У період з початку 1980-х до середини 1990-х років аркадні ігри займали провідні позиції на ринку відеоігор.

Серед найпопулярніших аркадних ігор, які стали лідерами за обсягами продажів і прибутками, можна виділити:

- Pong (1972) - аркадна гра, створена компанією Atari та випущена 29 листопада 1972 року, гру показано на рисунку 1.1. Розробником гри був Allan Alcorn під керівництвом Nolan Bushnell. Гра є симуляцією настільного тенісу, у якій гравці керують платформами та відбивають м'яч. Pong стала однією з перших комерційно успішних відеоігор і започаткувала розвиток аркадної індустрії.

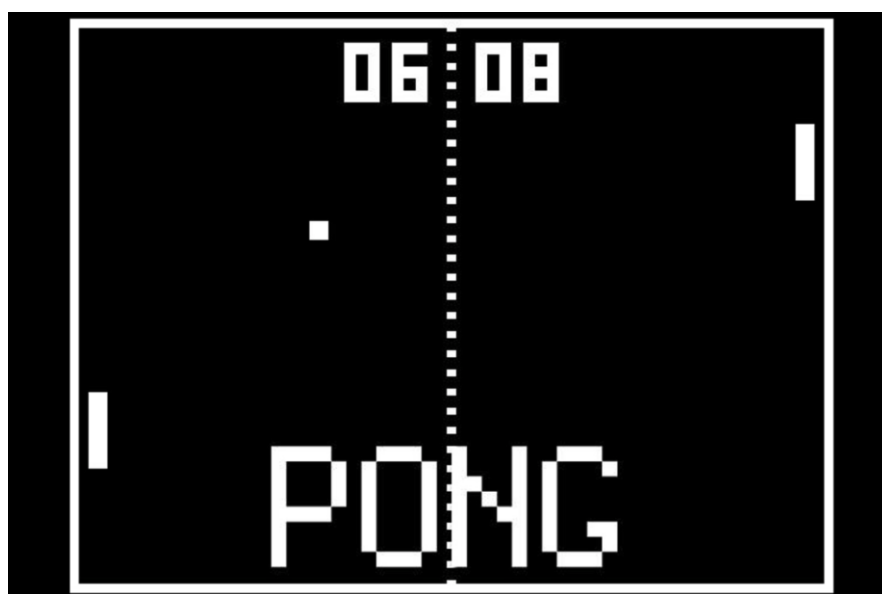


Рис. 1.1 Pong (1972)

- Breakout (1976) - аркадна гра, випущена компанією Atari у 1976 році, гра продемонстрована на рисунку 1.2. У розробці гри брали участь Steve Jobs та Steve Wozniak. Основною метою гри є руйнування блоків за допомогою м'яча, який відбивається платформою. Гра стала популярною завдяки простому керуванню та динамічному ігровому процесу.

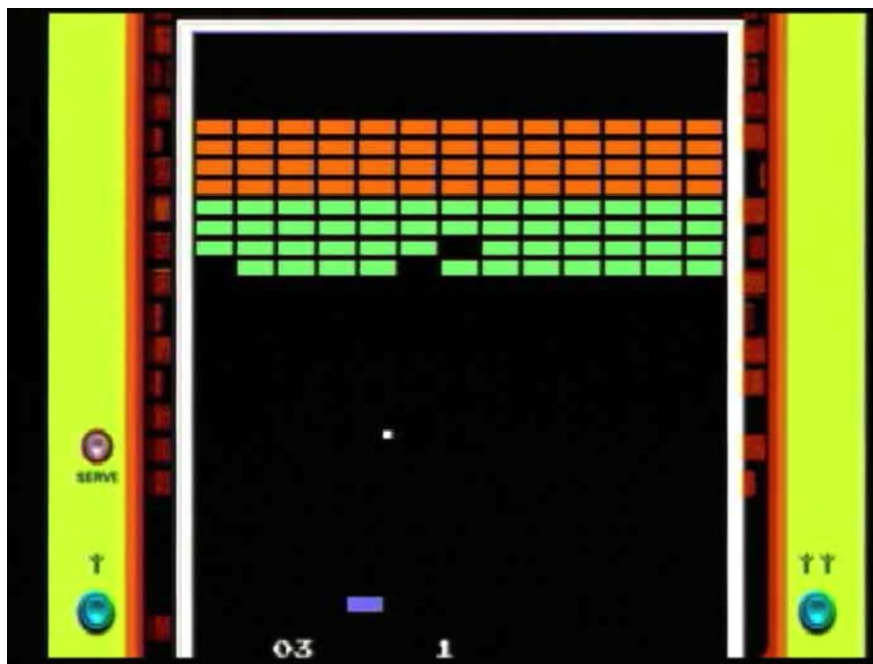


Рис. 1.2 Breakout(1976)

- Pac-Man (1980). Гра витримала понад 30 перевидань і стала доступною на різноманітних платформах - від Android до ZX Spectrum (2015). Pac-Man став одним із перших ігрових персонажів, який вийшов за межі ігрової індустрії та набув популярності в інших сферах, зокрема в іграшках, коміксах і мультфільмах. Уже через кілька років було продано близько 400 000 ігрових автоматів, що принесло прибуток приблизно 2,5 мільярда доларів США. Приклад інтерфейсу гри показано на рисунку 1.3.

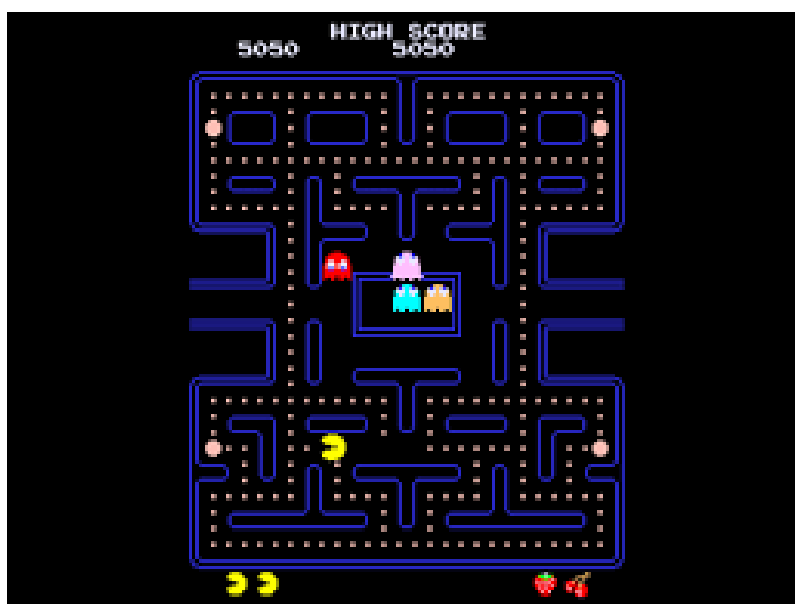


Рис. 1.3 Pac-Man (1980)

- Street Fighter II: Championship Edition (1992). Ця бойова гра привернула значну кількість гравців і суттєво вплинула на розвиток аркадної індустрії протягом наступного десятиліття, показано на рисунку 1.4. Гра отримала численні оновлення та вдосконалення: персонажі стали більш опрацьованими, а геймплей - більш стратегічним і орієнтованим на досвідчених користувачів. Вона неодноразово перевидавалася і була доступна на різних ігрових платформах. У США було продано понад 200 000 автоматів, які менш ніж за п'ять років принесли близько 2,5 мільярда доларів США.



Рис. 1.4 Street Fighter II: Championship Edition (1992)

- Donkey Kong (1981). Гра стала інноваційною для свого часу, оскільки була одним із перших представників жанру платформерів. Гравці виконують різні дії: бігають, стрибають, піднімаються сходами та уникають перешкод. Donkey Kong здобула великий успіх - лише за перший рік було продано понад 120 000 автоматів на суму близько 280 мільйонів доларів США. Приклад гри показано на рисунку 1.5.



Рис. 1.5 Donkey Kong (1981)

- Asteroids (1979). Ця гра стала однією з перших аркад, гра показана на рисунку 1.6, у якій була реалізована можливість введення імені гравця та ведення підрахунку балів. Це дало змогу встановлювати особисті рекорди та організовувати змагання між користувачами. Додатковою перевагою було використання векторної графіки та особливої системи керування, що імітувала рух у стані невагомості. Загальний обсяг продажів перевищив 100 000 автоматів, а до 1991 року дохід наблизився до 800 мільйонів доларів США.



Рис. 1.6 Asteroids (1979)

- Defender (1981). Однією з ключових особливостей цієї гри була широка ігрова область, яка виходила за межі видимого екрана. Це дозволяло гравцям переміщатися по горизонталі без обмежень, показано на рисунку 1.7. Станом на 2002 рік у США було виготовлено понад 60 000 автоматів, що принесли приблизно 1 мільярд доларів доходу.



Рис. 1.7 Defender (1981)

- Space Invaders (1978) - гра, що стала значним проривом у розвитку індустрії. На той час обчислювальні можливості комп'ютерів були обмеженими, однак розробникам вдалося створити інноваційний продукт. Ідея гри базувалася на сюжетах таких відомих творів, як «Війна світів» та «Зоряні війни». Space Invaders і сьогодні вважається однією з найпопулярніших аркадних ігор у світі. Протягом перших чотирьох років було продано понад 360 000 ігрових автоматів, що принесло дохід у розмірі близько 2,7 мільярда доларів США. Гру показано на рисунку 1.8.

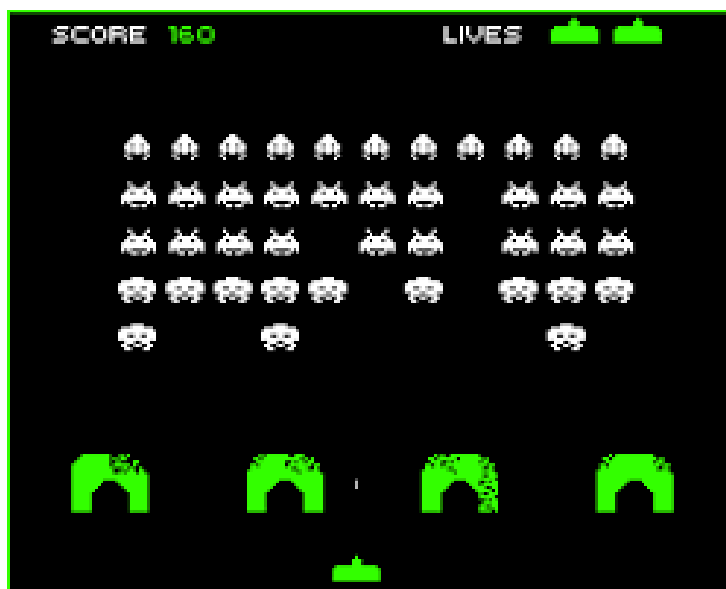


Рис. 1.8 Space invaders (1978)

- Galaxian (1979). Розробники максимально використали технічні можливості того часу, створивши повнокольорову гру з анімованими спрайтами, різноманітними атаками та динамічним зоряним фоном. Також у грі була реалізована якісна фоновіа музика, а вороги мали індивідуальні особливості. Galaxian вдалося уникнути типових проблем того періоду, таких як мерехтіння спрайтів і зниження продуктивності, що зробило її зразком для ігор «золотого віку» аркад. На рисунку 1.9 показано саму гру.



Рис. 1.9 Galaxian (1979)

На сьогоднішній день ігровий ринок представлений великою кількістю різноманітних проєктів. Він насичений не лише аналогами існуючих ігор, а й значною кількістю різних жанрів, що відображено на рисунку 1.10.

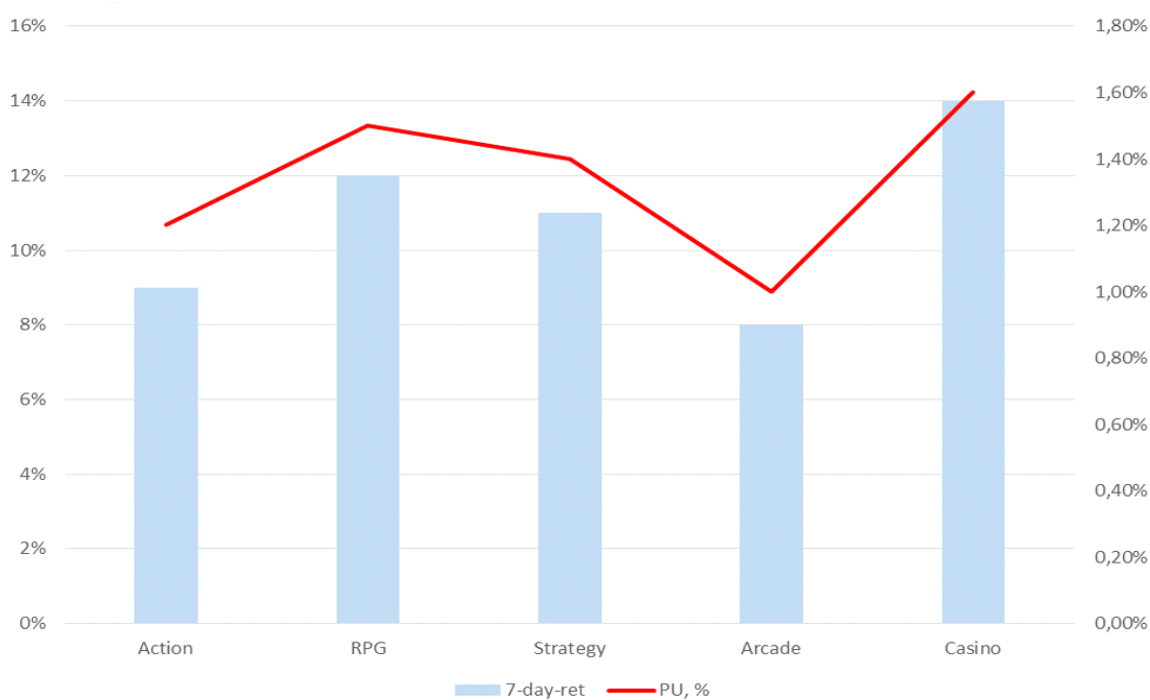


Рис. 1.10 Переваги ігор по жанрам

Діаграма відображає залежність між ключовими показниками: з лівого боку показано частку користувачів, що характеризує «семиденний retention», тобто кількість гравців, які продовжують користуватися грою на сьомий день. Червона лінія, що відповідає правій шкалі, відображає частку користувачів, які здійснюють оплату. Для аркадних ігор характерні відносно низькі показники як утримання користувачів, так і монетизації. Однак це не означає, що аркади є неефективним або непотрібним жанром, оскільки вони можуть залучати велику кількість гравців і приносити стабільний дохід.

Аналіз сучасних аркадних ігор показує, що на сьогодні існує низка проблем, пов'язаних із їх розробкою та просуванням, а також із тим, що вони не завжди повною мірою задовольняють потреби користувачів через обмежену складність та якість реалізації. Серед основних проблем можна виділити:

- небажання великих компаній із висококваліфікованими спеціалістами та потужними технічними ресурсами займатися невеликими проектами;
- зростання інтересу користувачів до 3D-ігор та онлайн-ігор;
- велика кількість аналогів і жанрів, що створює високу конкуренцію;
- складність формування власної аудиторії через недостатню рекламу та просування;
- обмежені можливості малих компаній, які не мають достатнього досвіду або фінансування для розробки аркадних ігор.

Для кращого розуміння особливостей аркадного жанру доцільно порівняти його із симуляторами, які є його протилежністю. У симуляторах основна увага приділяється реалістичності керування та фізики, навіть якщо це ускладнює ігровий процес. Головним критерієм є достовірність.

Наприклад, ігрова серія Forza для консолі Xbox поділяється на дві гілки: Forza Motorsport та Forza Horizon. Перша з них є симулятором і забезпечує максимально реалістичний досвід керування автомобілем, тоді як друга належить до аркадного жанру і орієнтована на розважальний, динамічний і менш реалістичний ігровий процес.

При цьому аркадні ігри не обов'язково є простими. Наприклад, Super Meat Boy, незважаючи на інтуїтивне керування, відзначається високим рівнем складності навіть для досвідчених гравців.

Отже, аркадні ігри спрямовані передусім на забезпечення задоволення від ігрового процесу без необхідності глибокого вивчення складних механік чи систем керування.

1.5 Аналіз систем аналітики ігрових застосунків

У сучасних програмних системах значну увагу приділяють збору, обробці та аналізу статистичних даних користувача. Особливо актуальним це стало у сфері розробки ігрових застосунків, де аналітичні системи дозволяють оцінювати ефективність взаємодії користувача із програмним забезпеченням, аналізувати

поведінку гравця та визначати рівень складності ігрового процесу. Використання аналітики забезпечує можливість не лише вдосконалення програмного продукту, але й оцінювання реакції, точності дій та координації рухів користувача.

Перші системи аналітики в ігрових застосунках використовувалися переважно для збору статистики про кількість запусків гри, тривалість ігрових сесій та результати проходження рівнів. З розвитком сучасних технологій можливості аналітичних систем значно розширилися. Сьогодні такі системи здатні виконувати детальний аналіз дій користувача, оцінювати швидкість реакції, аналізувати ефективність керування та формувати статистичні звіти у режимі реального часу.

Особливо активно системи аналітики використовуються у багатокористувацьких онлайн-іграх та кіберспортивних проєктах. У таких програмних продуктах аналітичні модулі дозволяють аналізувати поведінку користувачів, оцінювати баланс гри та визначати найбільш складні елементи ігрового процесу. Крім того, зібрана статистика використовується для вдосконалення механік гри та оптимізації взаємодії користувача із системою.

У більшості сучасних ігрових систем аналітика реалізується шляхом запису дій користувача у спеціальні файли або бази даних. Надалі отримані дані обробляються аналітичними модулями, які виконують формування статистики, побудову графіків та аналіз результатів взаємодії користувача із програмою.

Серед основних функцій сучасних систем аналітики можна виділити:

- збір статистичних даних;
- аналіз дій користувача;
- аналіз швидкості реакції;
- побудову графіків;
- формування звітів;
- оцінювання продуктивності системи;
- аналіз ефективності проходження рівнів.

Особливе значення системи аналітики мають у програмних продуктах, спрямованих на розвиток когнітивних здібностей та моторних навичок

користувача. У таких системах аналітичні модулі дозволяють оцінювати точність взаємодії користувача із програмою та визначати ефективність тренування.

У межах даної кваліфікаційної роботи аналітичний модуль є окремою складовою програми та взаємодіє з ігровим модулем через систему запису статистичних даних. Під час роботи гри інформація про дії користувача автоматично записується у CSV-файли, після чого виконується її подальший аналіз.

Для реалізації систем аналітики найчастіше використовуються мови програмування Python, JavaScript та Java. Однією з найбільш популярних мов програмування для реалізації аналітичних систем є Python, оскільки вона забезпечує просту роботу зі статистичними даними, підтримує велику кількість бібліотек для аналізу інформації та дозволяє ефективно реалізовувати системи візуалізації результатів.

Основними перевагами використання Python для реалізації аналітичних систем є:

- простота синтаксису;
- ефективна робота з CSV-файлами;
- підтримка побудови графіків;
- можливість швидкої обробки статистики;
- велика кількість бібліотек для аналізу даних.

У розробленому застосунку аналітичний модуль реалізований мовою програмування Python та виконує:

- зчитування статистичних даних;
- аналіз швидкості реакції;
- аналіз точності керування;
- аналіз натискань клавіш;
- аналіз кількості корисних та зайвих рухів;
- побудову графіків та таблиць результатів.

Для збереження статистичних даних використовуються CSV-файли, які забезпечують простий та ефективний спосіб запису інформації. CSV-файли мають

текстовий формат, що дозволяє легко виконувати їх обробку засобами Python та інших мов програмування.

У межах програмного застосунку використовуються такі файли статистики:

- game_sessions.csv;
- game_actions.csv.

Файл game_sessions.csv використовується для збереження загальної інформації про ігрові сесії, включаючи тривалість гри, кількість набраних балів та результат проходження рівня. Файл game_actions.csv призначений для запису інформації про дії користувача, зокрема натискання клавіш, швидкість реакції та кількість рухів платформи.

Після зчитування даних аналітичний модуль виконує їх обробку та формує результати аналізу у вигляді таблиць і графіків. Використання графічної аналітики дозволяє наочно відображати результати взаємодії користувача із програмною системою та оцінювати ефективність тренування дрібної моторики.

Для порівняння можливостей сучасних систем аналітики було сформовано таблицю 1.1.

Таблиця 1.1

Порівняння сучасних систем аналітики

Функціональність	Google Analytics	Unity Analytics	Розроблена система
Аналіз дій користувача	Так	Так	Так
Аналіз швидкості реакції	Ні	Частково	Так
Аналіз моторики	Ні	Ні	Так
Побудова графіків	Так	Так	Так
Робота з CSV	Ні	Ні	Так
Аналіз натискань клавіш	Ні	Частково	Так

Як видно з таблиці 1.1, більшість сучасних систем аналітики орієнтовані переважно на загальний аналіз поведінки користувача та продуктивності програмного забезпечення. Водночас у розробленій програмі основна увага

приділяється аналізу дій користувача, швидкості реакції та розвитку дрібної моторики.

Таким чином, системи аналітики є важливою складовою сучасних ігрових застосунків та забезпечують можливість збору й аналізу статистичних даних користувача. Поєднання ігрового процесу із системою аналітики дозволяє створити гру, яка може використовуватися не лише для розваги, але й для оцінювання ефективності взаємодії користувача із програмною системою та розвитку моторних навичок.

1.6 Визначення функціональних та нефункціональних вимог до програмного продукту

Одним із найважливіших етапів розробки програмного забезпечення є формування функціональних та нефункціональних вимог до програмного продукту. Правильне визначення вимог дозволяє сформувати структуру майбутньої системи, визначити її основні можливості, забезпечити стабільність роботи програмного забезпечення та організувати взаємодію між окремими модулями системи.

В даній роботі розробляється програмний продукт, який складається з двох основних модулів: ігрового та аналітичного. Ігровий модуль реалізує аркадну гру типу «Пінг-понг», а аналітичний модуль забезпечує збір, обробку та аналіз статистичних даних користувача.

Основною метою ігрового застосунку є створення інтерактивного ігрового застосунку, який може використовуватися для розвитку дрібної моторики, швидкості реакції та координації рухів користувача. Крім того, система повинна забезпечувати можливість аналізу дій користувача та формування статистичних результатів.

Під час формування вимог до програмного продукту необхідно враховувати:

- функціональні можливості системи;
- вимоги до продуктивності;

- стабільність роботи програми;
- зручність інтерфейсу;
- швидкість обробки даних;
- ефективність взаємодії між модулями.

Функціональні вимоги визначають основні можливості ігрового застосунку та описують функції, які повинна виконувати система під час роботи.

Ігровий модуль повинен забезпечувати:

- запуск та завершення гри;
- відображення ігрового поля;
- рух м'яча;
- керування платформою;
- систему зіткнень;
- систему рівнів;
- систему підрахунку балів;
- звукові ефекти;
- можливість паузи гри;
- запис статистики дій користувача;
- збереження результатів у CSV-файли.

Користувач повинен мати можливість керувати платформою за допомогою клавіатури. Для цього у програмі реалізовано систему обробки натискань клавіш, яка забезпечує швидку реакцію гри на дії користувача. Особливістю розробленої гри є реалізація системи збору статистики дій користувача. Під час роботи програми система автоматично записує:

- кількість натискань клавіш;
- кількість рухів платформи;
- швидкість реакції;
- тривалість ігрової сесії;
- результати проходження рівнів;
- кількість корисних та зайвих рухів.

Аналітичний модуль повинен забезпечувати:

- зчитування CSV-файлів;
- аналіз статистичних даних;
- побудову графіків;
- формування таблиць результатів;
- відображення статистики користувача;
- аналіз ефективності взаємодії користувача із системою.

Крім того, система повинна забезпечувати можливість оцінювання ефективності розвитку дрібної моторики користувача шляхом аналізу швидкості та точності виконання дій під час гри.

Нефункціональні вимоги визначають характеристики якості програмного забезпечення, зокрема стабільність роботи, швидкодію, надійність та зручність використання. Гра повинна забезпечувати стабільну роботу під час виконання ігрового процесу та обробки статистичних даних. Важливою вимогою є висока швидкість обробки подій, оскільки затримки у роботі програми можуть негативно впливати на ігровий процес та точність аналізу дій користувача.

Для забезпечення високої продуктивності ігровий модуль реалізовано мовою програмування C++, яка дозволяє ефективно обробляти події у режимі реального часу. Використання мови Python для реалізації аналітичного модуля забезпечує швидку обробку статистичних даних та зручну роботу з CSV-файлами.

До основних нефункціональних вимог належать:

- стабільність роботи системи;
- висока швидкість обробки подій;
- мінімальне навантаження на систему;
- точність обробки статистичних даних;
- зручність інтерфейсу користувача;
- простота використання;
- можливість подальшого розширення функціоналу.

Особливу увагу під час розробки гри необхідно приділити точності роботи аналітичної системи. Помилки під час запису або обробки статистики можуть призвести до некоректного аналізу результатів взаємодії користувача із системою.

Для коректної роботи застосунку комп'ютерна система повинна відповідати певним технічним вимогам.

Мінімальні системні вимоги:

- операційна система Windows 10 або новіша;
- процесор із тактовою частотою не менше 1.6 ГГц;
- оперативна пам'ять не менше 2 ГБ;
- наявність клавіатури;
- не менше 100 МБ вільного місця на диску.

Рекомендовані системні вимоги:

- операційна система Windows 11;
- процесор Intel Core i5 або аналогічний;
- оперативна пам'ять 8 ГБ і більше;
- SSD-накопичувач;
- підтримка сучасних бібліотек Python.

Інтерфейс користувача повинен бути простим, зрозумілим. Основна увага під час розробки інтерфейсу приділяється швидкості взаємодії користувача із системою та мінімізації кількості зайвих елементів.

Ігровий інтерфейс повинен містити:

- ігрове поле;
- платформу;
- м'яч;
- інформацію про рахунок;
- інформацію про рівень;
- повідомлення про завершення гри.

Аналітичний модуль повинен забезпечувати:

- відображення таблиць статистики;
- побудову графіків;
- перегляд результатів ігрових сесій;
- відображення статистики натискань клавіш;
- відображення показників ефективності користувача.

2. ЗАСОБИ РОЗРОБКИ ТА ПРОЕКТУВАННЯ ПРОГРАМИ

2.1 Аналіз мов програмування

Мова програмування є основним інструментом створення програмного забезпечення. Від вибору мови залежить швидкодія програми, ефективність використання пам'яті, складність реалізації функціональних можливостей та продуктивність роботи програмного продукту.

На сучасному етапі розвитку інформаційних технологій існує велика кількість мов програмування, кожна з яких орієнтована на певну сферу використання. Одні мови краще підходять для створення вебзастосунків, інші - для роботи з базами даних, аналізу великих обсягів інформації або створення високопродуктивних програм реального часу.

Для реалізації консольної гри «Пінг-понг» було проаналізовано декілька популярних мов програмування, серед яких C++, C#, Java, Python та PHP.

2.1.1 C++

C++ є однією з найпоширеніших мов програмування системного рівня. Дана мова широко використовується для створення операційних систем, драйверів, комп'ютерних ігор та високопродуктивних програмних застосунків [1],[4]. Основною особливістю C++ є висока швидкість виконання програм та можливість безпосередньої роботи з оперативною пам'яттю [1].

Мова підтримує процедурне, об'єктно-орієнтоване та узагальнене програмування, що дозволяє створювати складні програмні системи та ефективно організовувати структуру програмного коду [4]. Завдяки компіляції у машинний код програми мовою C++ працюють швидко та використовують мінімальну кількість системних ресурсів [1], [18].

Однією з головних переваг C++ є висока продуктивність. Саме тому дана мова активно використовується під час створення комп'ютерних ігор, графічних

рушіїв та програм реального часу [1], [4]. Для реалізації консольної гри «Пінг-понг» це є важливою перевагою, оскільки ігровий процес потребує швидкого оновлення координат об'єктів та оперативної обробки дій користувача [18].

C++ також дозволяє працювати зі стандартними бібліотеками Windows та ефективно реалізовувати консольні застосунки без використання додаткових графічних рушіїв [18].

До недоліків мови можна віднести складніший синтаксис та необхідність самостійного контролю використання пам'яті, що підвищує складність розробки програмного забезпечення [4].

2.1.2 C#

C# є сучасною об'єктно-орієнтованою мовою програмування, розробленою компанією Microsoft для платформи .NET. Дана мова широко використовується для створення десктопних програм, вебзастосунків та комп'ютерних ігор.

Однією з основних переваг C# є простота розробки та зручність роботи з графічними інтерфейсами. Мова має автоматичне керування пам'яттю, що значно спрощує процес програмування та зменшує кількість помилок під час роботи з ресурсами системи.

C# активно використовується у сфері розробки ігор завдяки підтримці ігрового рушія Unity. Проте більшість програм мовою C# працює через платформу .NET, що створює додаткове навантаження на систему та знижує швидкодію порівняно з C++.

Для реалізації невеликих графічних програм C# є зручним рішенням, однак для консольних застосунків реального часу мова поступається C++ у швидкості роботи та ефективності використання системних ресурсів.

2.1.3 Java

Java є платформонезалежною мовою програмування, яка широко використовується для створення корпоративних систем, серверних застосунків та мобільних програм.

Основною перевагою Java є можливість запуску програм на різних операційних системах без зміни програмного коду. Це забезпечується використанням віртуальної машини JVM, яка виконує програму незалежно від типу операційної системи.

Java активно використовується для створення Android-застосунків, серверних систем та програм обробки великих обсягів даних. Мова також підтримує багатопотоковість та має велику кількість бібліотек для роботи з мережевими технологіями.

Недоліком Java є нижча швидкість роботи порівняно з C++, оскільки виконання програми відбувається через віртуальну машину. Крім того, Java рідше використовується для створення консольних ігор через складнішу взаємодію із системними ресурсами.

2.1.4 Python

Python є високорівневою мовою програмування, яка відзначається простим синтаксисом та швидкістю розробки програмного забезпечення [2], [6]. Дана мова широко використовується у сфері веброзробки, автоматизації процесів, аналізу даних та штучного інтелекту [2], [3].

Python активно застосовується під час роботи з Big Data та машинним навчанням завдяки великій кількості спеціалізованих бібліотек, серед яких NumPy, Pandas, TensorFlow та Scikit-learn [3], [9], [21].

Основною перевагою Python є простота читання програмного коду та швидкість створення програмних застосунків [2], [6]. Завдяки цьому мова є популярною серед початківців та активно використовується у навчальних проєктах [6].

Проте через інтерпретований принцип виконання Python поступається C++ за швидкістю роботи та ефективністю використання пам'яті [2]. Для реалізації програм реального часу та високопродуктивних ігрових застосунків Python використовується значно рідше [3].

2.1.5 PHP

PHP є серверною мовою програмування, яка використовується переважно для створення вебзастосунків та динамічних вебсайтів. Дана мова широко застосовується для роботи з базами даних, серверної обробки інформації та створення систем керування контентом.

Основною перевагою PHP є простота інтеграції з вебтехнологіями та підтримка більшості серверних платформ. Мова має велику кількість бібліотек та фреймворків, серед яких Laravel та Symfony.

PHP орієнтований насамперед на серверну обробку даних та не призначений для створення високопродуктивних консольних або ігрових застосунків. Саме тому використання PHP для реалізації аркадної гри є недоцільним.

Для більш наочного порівняння розглянутих мов програмування було сформовано таблицю їх основних характеристик.

Таблиця 2.1

Порівняльний аналіз мов програмування

Мова	Основне призначення	Швидкодія	Робота з пам'яттю	Основна сфера використання
C++	Системне та ігрове ПЗ	Висока	Прямий доступ	Ігри, системні програми
C#	Прикладне ПЗ	Висока	Автоматична	Desktop, Unity
Java	Корпоративні системи	Середня	Автоматична	Android, сервери
Python	Big Data та AI	Низька	Автоматична	Аналіз даних, AI
PHP	Веброзробка	Середня	Автоматична	Вебсайти, сервери

Проведений аналіз показує, що різні мови програмування мають різне призначення та сферу використання. Для реалізації консольної гри «Пінг-понг» найбільш доцільним є використання мови програмування C++, оскільки вона

забезпечує високу швидкість роботи, ефективне використання системних ресурсів та зручну реалізацію програм реального часу.

2.2 Проєктування структури програмного модулю

Проєктування структури є одним із ключових етапів розробки програмного забезпечення, оскільки саме на цьому етапі визначаються основні складові системи, їх призначення, способи взаємодії між компонентами та структура даних, які використовуються у процесі роботи застосунку [5]. Для даної кваліфікаційної роботи важливою особливістю є те, що програмний продукт складається не лише з ігрової частини, але й з окремого аналітичного модуля, який забезпечує обробку результатів взаємодії користувача з грою.

Розроблений програмний застосунок можна поділити на два основні модулі: ігровий модуль, реалізований мовою програмування C++, та аналітичний модуль, реалізований мовою Python. Ігровий модуль відповідає за реалізацію логіки гри типу «Пінг-понг», обробку дій користувача, рух м'яча, керування платформою, систему рівнів, підрахунок результатів та запис статистичних даних. Аналітичний модуль призначений для зчитування CSV-файлів, обробки статистики, обчислення показників ефективності дій користувача, побудови графіків та відображення таблиць результатів.

Загальну структуру програмного застосунку доцільно подати у вигляді схеми, на якій відображено взаємодію користувача, ігрового модуля, файлів статистики та аналітичного модуля. Вона показана на рисунку 2.1.



Рис. 2.1 Загальна структура програмного застосунку

Основним елементом системи є ігровий модуль. Під час запуску гри користувач взаємодіє із застосунком за допомогою клавіатури. Система обробляє натискання клавіш, змінює положення платформи, оновлює координати м'яча та виконує перевірку зіткнень. У процесі гри також здійснюється підрахунок балів і фіксація дій користувача. Для подальшого аналізу ці дані записуються у файли статистики.

Файли статистики виконують роль проміжного сховища між ігровим та аналітичним модулями. Такий підхід дозволяє розділити логіку гри та логіку аналізу даних. Ігровий модуль не виконує складних аналітичних обчислень, а лише зберігає необхідну інформацію у структурованому вигляді. Аналітичний модуль, у свою чергу, зчитує ці дані та виконує їх подальшу обробку.

У межах програмного застосунку використовуються два основні CSV-файли: `game_sessions.csv` та `game_actions.csv`. Файл `game_sessions.csv` призначений для збереження загальної інформації про ігрові сесії, зокрема тривалості гри, кількості набраних балів, рівня складності та результату завершення гри. Файл `game_actions.csv` містить інформацію про окремі дії користувача, зокрема натискання клавіш, час виконання дії, напрямок руху платформи та інші показники, необхідні для аналізу точності керування.

Аналітичний модуль працює з уже збереженими даними. Він зчитує CSV-файли, виконує обчислення статистичних показників, аналізує кількість корисних і зайвих рухів, швидкість реакції користувача та загальну ефективність взаємодії з грою. Результати аналізу відображаються у вигляді таблиць та графіків, що дає змогу наочно оцінити поведінку користувача під час гри.

Структура проєкту в цілому повинна містити окремі файли для ігрової логіки, налаштувань, логування та аналітики. Такий підхід забезпечує зручність підтримки програмного коду та дозволяє надалі розширювати функціональність застосунку. Наприклад, до ігрового модуля можна додати нові рівні або режими гри, а до аналітичного модуля - нові графіки, додаткові показники або експорт результатів. Схема взаємодії показана на рисунку 2.2.



Рис 2.2 – Схема взаємодії модулів системи

Таким чином, структура програмного застосунку побудована за модульним принципом. Ігровий модуль забезпечує взаємодію користувача з грою та формування статистики, CSV-файли виконують роль сховища даних, а аналітичний модуль забезпечує обробку та візуалізацію результатів. Такий підхід дозволяє розділити функціональні частини системи, спростити її супровід і забезпечити можливість подальшого розвитку програмного продукту.

2.3 Проектування ігрового модуля

Однією з основних складових програмного застосунку є ігровий модуль, реалізований мовою програмування C++ [1], [3]. Даний модуль забезпечує взаємодію користувача з грою, керування платформою, рух м'яча, систему рівнів, підрахунок балів та запис статистичних даних. Основною задачею під час проектування ігрового модуля було забезпечення стабільної роботи гри у режимі реального часу та можливості подальшого розширення функціональності застосунку.

Під час розробки було використано модульний підхід, за якого окремі елементи гри реалізуються у вигляді класів та структур даних [2]. Такий підхід дозволяє спростити підтримку програмного коду, підвищити читабельність програми та забезпечити можливість повторного використання окремих компонентів системи [5].

Основним класом ігрового модуля є клас Game, який відповідає за запуск гри, виконання головного циклу програми, оновлення стану ігрового поля та взаємодію між окремими компонентами системи [2]. Саме у межах цього класу виконується

обробка натискань клавіш, рух м'яча, перевірка зіткнень та підрахунок результатів користувача.

Для реалізації системи налаштувань використовується клас `Settings`, який містить параметри гри, зокрема швидкість руху м'яча, розміри ігрового поля, параметри рівнів та інші конфігураційні значення. Використання окремого класу налаштувань дозволяє змінювати параметри застосунку без необхідності внесення змін до основної логіки гри.

Для запису статистики користувача використовується клас `Logger`. Даний клас відповідає за формування CSV-файлів та запис інформації про ігрові сесії, натискання клавіш і результати користувача [12]. Завдяки цьому аналітичний модуль отримує необхідні дані для подальшої обробки.

Клас `SoundManager` використовується для реалізації звукових ефектів гри. Він забезпечує відтворення звуків зіткнення м'яча зі стінками або платформою, а також звуків завершення гри чи переходу між рівнями [17].

Для визначення напрямку руху м'яча використовується перелік `enum Direction`, який містить можливі варіанти переміщення об'єкта у межах ігрового поля [2]. Такий підхід спрощує реалізацію логіки руху м'яча та перевірки зіткнень.

Крім того, у програмі використовується структура `Level`, яка містить інформацію про рівень складності гри, швидкість руху м'яча та кількість балів, необхідних для переходу на наступний рівень.

Основою роботи гри є головний цикл програми [2]. Після запуску застосунку система виконує ініціалізацію змінних, завантаження налаштувань та створення основних об'єктів гри. Далі запускається цикл, у межах якого постійно виконуються функції обробки клавіш, оновлення координат об'єктів та перевірки зіткнень.

Під час кожного оновлення гри система аналізує положення м'яча та платформи, перевіряє можливі зіткнення зі стінками або іншими об'єктами та змінює напрямок руху м'яча відповідно до результатів перевірки. У випадку пропуску м'яча користувачем гра завершується, а результати сесії записуються до CSV-файлу [12].

Для реалізації керування платформою використовується система обробки клавіш клавіатури. Користувач має можливість переміщувати платформу вліво або вправо, реагуючи на рух м'яча. У процесі гри система також аналізує швидкість натискань клавіш та записує відповідну інформацію до файлів статистики, алгоритм представлений на рисунку 2.4 [13].



Рис. 2.4 Алгоритм роботи ігрового циклу

Таким чином, ігровий модуль забезпечує реалізацію основних механік аркадної гри типу «Пінг-понг», підтримує систему рівнів, виконує обробку дій користувача та забезпечує збір статистичних даних для подальшого аналізу в аналітичному модулі системи [11].

2.4 Проєктування аналітичного модуля

Однією з головних особливостей розробленого програмного застосунку є наявність окремого аналітичного модуля, призначеного для обробки статистичних даних користувача. На відміну від класичних аркадних ігор, у яких результати взаємодії користувача обмежуються лише кількістю набраних балів, у межах даного проєкту реалізовано систему аналізу дій користувача, що дозволяє оцінювати швидкість реакції, точність керування та ефективність взаємодії з грою [11], [14].

Аналітичний модуль реалізований мовою програмування Python та функціонує незалежно від ігрової частини застосунку [10]. Такий підхід дозволяє розділити логіку гри та логіку аналізу статистичних даних, що значно спрощує структуру програмного забезпечення та забезпечує можливість подальшого розширення функціональності системи [5].

Основним файлом аналітичного модуля є `analytics_app.py`. Даний модуль відповідає за:

- зчитування CSV-файлів;
- аналіз статистичних даних;
- обчислення показників ефективності;
- побудову графіків;
- формування таблиць результатів;
- візуалізацію статистики користувача.

Структура аналітичного модуля показана на рисунку 2.5.



Рис. 2.5 Структура аналітичного модуля

Для реалізації графічного інтерфейсу аналітичного модуля використовується бібліотека `tkinter`. Дана бібліотека входить до стандартного набору Python та дозволяє створювати графічні застосунки без використання сторонніх фреймворків [10]. Використання `tkinter` забезпечує можливість створення вікон, таблиць, кнопок та інших елементів інтерфейсу, необхідних для відображення результатів аналізу. Аналітичний модуль працює з файлами `game_session.csv` та `game_action.csv`, які формуються у процесі роботи гри [12].

Файл `game_sessions.csv` містить загальну інформацію про ігрові сесії користувача, зокрема дату запуску гри, тривалість сесії, кількість набраних балів та результат завершення гри.

Файл `game_actions.csv` містить інформацію про окремі дії користувача:

- натискання клавіш;
- напрямок руху платформи;
- час виконання дії;
- швидкість реакції;
- кількість рухів користувача.

Після зчитування CSV-файлів аналітичний модуль виконує обробку статистичних даних [12]. У межах аналізу система визначає:

- середню швидкість реакції користувача;
- кількість натискань клавіш;
- кількість корисних рухів;
- кількість зайвих рухів;
- точність керування платформою;
- ефективність проходження рівнів.

Особливу увагу приділено аналізу корисних та зайвих рухів користувача [11]. Корисними рухами вважаються дії, які сприяють правильному переміщенню платформи для відбиття м'яча. Зайвими рухами є непотрібні натискання клавіш або переміщення платформи, які не впливають на успішне проходження гри.

Крім того, система виконує оцінювання точності рухів користувача [14]. Для цього аналізується відповідність положення платформи траєкторії руху м'яча та швидкість реакції користувача під час зміни напрямку руху об'єкта. На рисунку 2.6 представлений сам алгоритм.

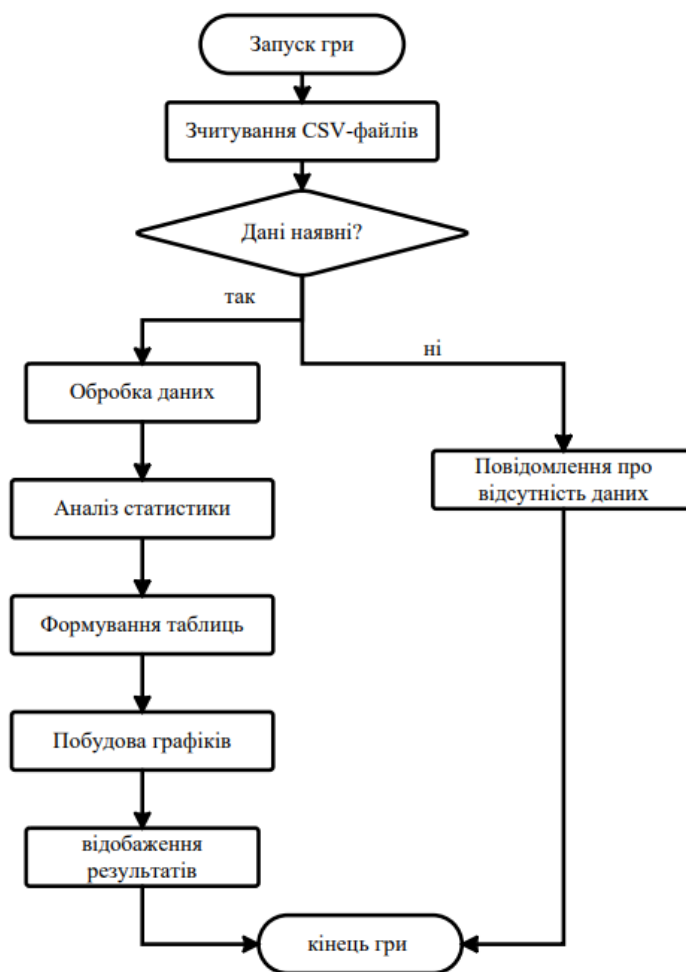


Рис. 2.6 Алгоритм роботи аналітичного модуля

Для візуалізації результатів аналізу використовуються графіки та таблиці [10]. Графіки дозволяють наочно відобразити зміни показників користувача під час гри, зокрема зміну швидкості реакції, ефективність керування платформою та кількість помилкових дій. На рисунку 2.7 показано схему взаємодії компонентів аналітичного модуля.

У межах аналітичного модуля реалізовано функції:

- read_csv() - зчитування статистичних даних;
- refresh_data() - оновлення інформації;
- build_charts() - побудова графіків;
- show_sessions_table() - відображення таблиці ігрових сесій;
- show_actions_table() - відображення таблиці дій користувача.

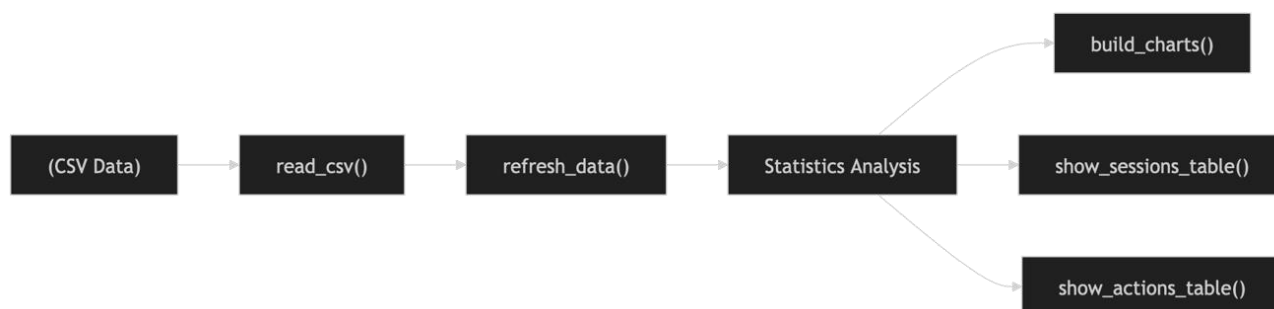


Рис. 2.7 – Схема взаємодії компонентів аналітичного модуля

Таким чином, аналітичний модуль забезпечує збір, обробку та візуалізацію статистичних даних користувача. Використання Python, CSV-файлів та бібліотеки tkinter дозволяє реалізувати окрему систему аналізу дій користувача та забезпечити оцінювання ефективності взаємодії з ігровим застосунком.

2.5 UML-діаграма та схема взаємодії модулів

Для кращого розуміння структури програмного застосунку доцільно використати UML-діаграми та схеми взаємодії компонентів [18]. Вони дозволяють показати не лише окремі частини системи, але й те, як саме ці частини пов'язані між собою. У межах даного проєкту це особливо важливо, оскільки застосунок складається з ігрового модуля, модуля аналітики та CSV-файлів, які використовуються для обміну даними.

Першою важливою схемою є діаграма прецедентів ігрового застосунку представлена на рисунку 2.8 [18]. Вона показує, які дії може виконувати користувач під час взаємодії з грою. До основних дій належать запуск гри, вибір рівня складності, керування платформою, зміна кута платформи, увімкнення або вимкнення звуку, пауза, завершення гри та збереження статистики.

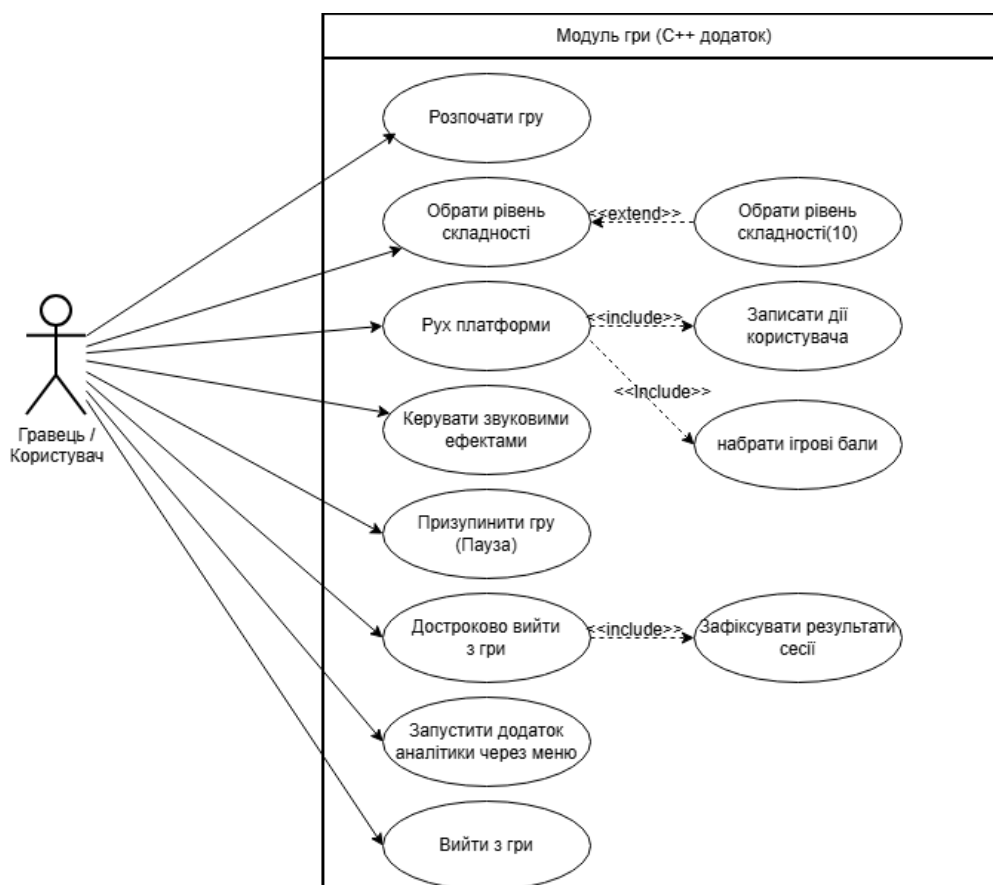


Рис. 2.8 Діаграма прецедентів ігрового застосунку

Дана діаграма дозволяє зрозуміти, що користувач взаємодіє не лише з ігровим процесом, але й із додатковими функціями застосунку [18]. Окремо варто зазначити, що під час керування платформою, набору балів та завершення гри відбувається збереження статистики, яка надалі використовується аналітичним модулем [12].

Другою важливою схемою є діаграма прецедентів модуля візуалізації представлена на рисунку 2.9. Вона описує взаємодію користувача вже не з грою, а з аналітичною частиною застосунку [18]. У цьому модулі користувач може запустити аналітику, зчитати CSV-файли, переглянути таблицю ігрових сесій, переглянути таблицю дій користувача, оновити статистику та побудувати графіки.



Рис. 2.9 Діаграма прецедентів модуля візуалізації

На відміну від ігрового модуля, модуль візуалізації працює з уже збереженими даними. Він не впливає безпосередньо на ігровий процес, а використовується для аналізу результатів після завершення гри [10]. Такий поділ є зручним, оскільки дозволяє не перевантажувати ігровий модуль додатковими обчисленнями.

Для відображення структури програмного коду було створено UML-діаграму класів. Вона показує основні класи, які використовуються у програмному застосунку, та зв'язки між ними. Основним класом є Game, який відповідає за запуск гри, відображення ігрового поля, обробку вводу користувача, оновлення стану гри та підрахунок балів [2]. Клас Settings відповідає за налаштування, Logger - за запис статистики, а SoundManager - за звуковий супровід гри [17]. Діаграма показана на рисунку 2.10.

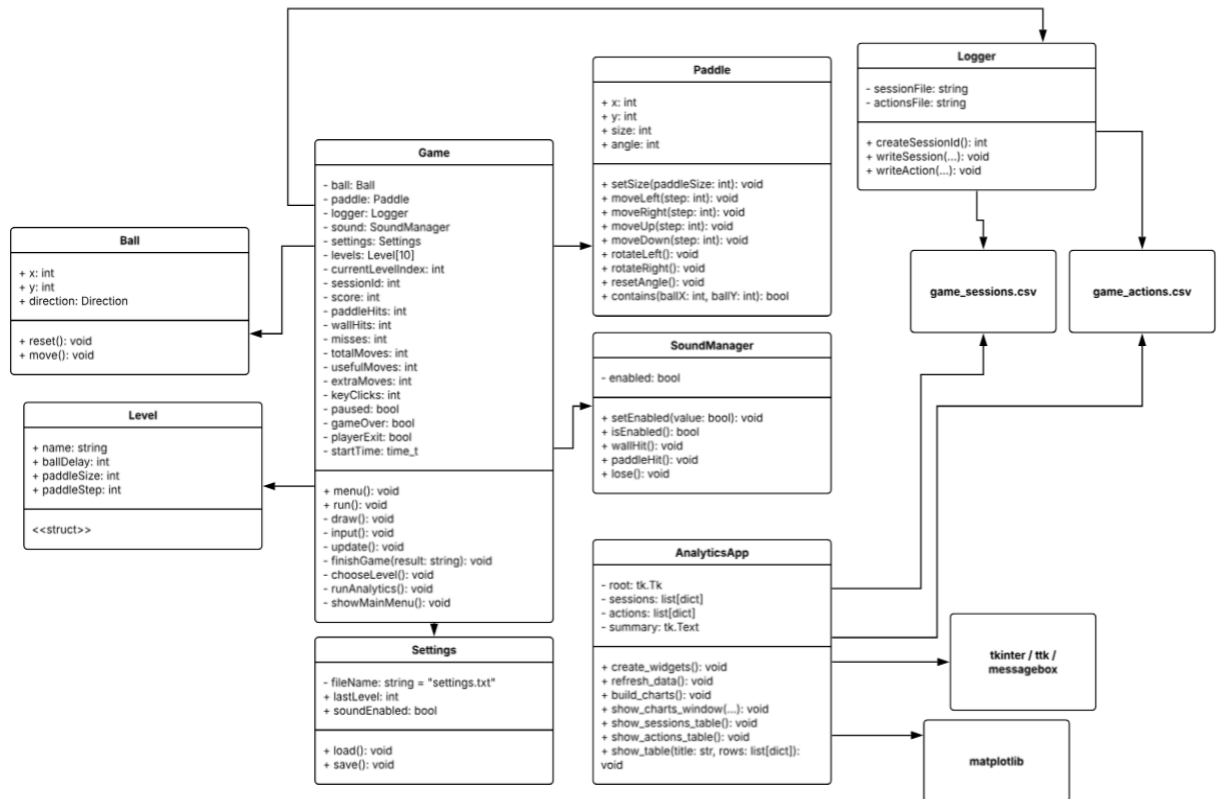


Рис. 2.10 UML-діаграма класів програмного застосунку

UML-діаграма класів показує, що клас Game взаємодіє з іншими допоміжними класами [18]. Такий підхід дозволяє не змішувати всю логіку в одному місці, а розділити її на окремі частини. Це робить програмний код більш зрозумілим і зручним для подальшого редагування.

Окремо було спроектовано схему взаємодії модулів системи. Вона демонструє, як саме дані передаються між ігровим та аналітичним модулями. Під час гри користувач взаємодіє з ігровим модулем, після чого результати записуються у CSV-файли [12]. Потім аналітичний модуль зчитує ці файли та формує таблиці й графіки. Схема показана на рисунку 2.11.

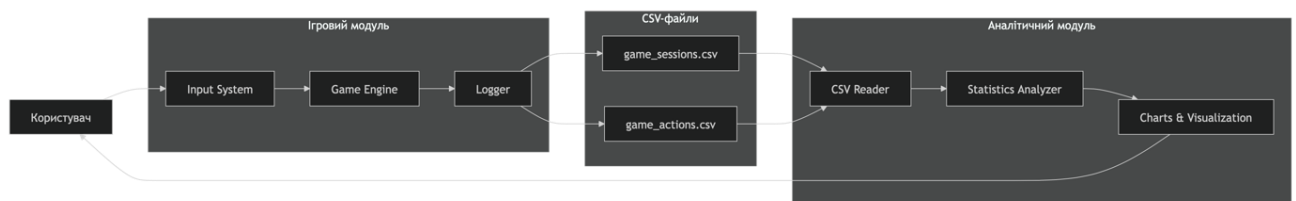


Рис. 2.11 Схема взаємодії модулів системи

Таким чином, використання UML-діаграм та схем взаємодії дозволило більш чітко описати архітектуру програмного застосунку. Діаграма прецедентів ігрового застосунку показує можливості користувача під час гри, діаграма прецедентів модуля візуалізації описує роботу з аналітикою, UML-діаграма класів демонструє структуру коду, а схема взаємодії модулів пояснює обмін даними між грою та системою аналізу.

2.6 Особливості інтерфейсу користувача програмного застосунку

Інтерфейс користувача розробленого гри відіграє ключову роль у забезпеченні зручної та ефективної взаємодії з системою. Оскільки гра складається з двох окремих модулів - ігрового та аналітичного - інтерфейс реалізовано у двох різних формах: консольній та графічно-аналітичній. Консольний інтерфейс використовується в ігровому модулі, де основна увага приділяється швидкості реакції, точності керування та мінімізації відволікаючих елементів. Графічно-аналітичний інтерфейс застосовується в модулі статистики, де важливо забезпечити наочність, структурованість та зручність сприйняття результатів аналізу.

Після запуску програми користувач потрапляє до головного меню, яке містить основні функції системи: запуск ігрового процесу, перегляд статистики та вихід із програми. Меню реалізоване у вигляді текстового інтерфейсу, що дозволяє швидко орієнтуватися та обирати потрібні дії без зайвих графічних елементів. Такий підхід забезпечує високу швидкість роботи та сумісність із будь-яким обладнанням. На рисунку 2.12 наведено головного меню, що демонструє структуру та вигляд початкового екрана програми:

Після завершення гри користувач має можливість переглянути статистику, яка формується на основі даних, зібраних під час ігрової сесії. У цьому режимі відображаються такі показники зображення на рисунку 2.14, як кількість успішних відбиттів, кількість помилок, середній час реакції, кількість натискань клавіш та інші параметри, що характеризують ефективність взаємодії користувача з ігровим модулем. Інтерфейс статистики також реалізовано у консольному форматі, що забезпечує швидке та зручне ознайомлення з результатами.

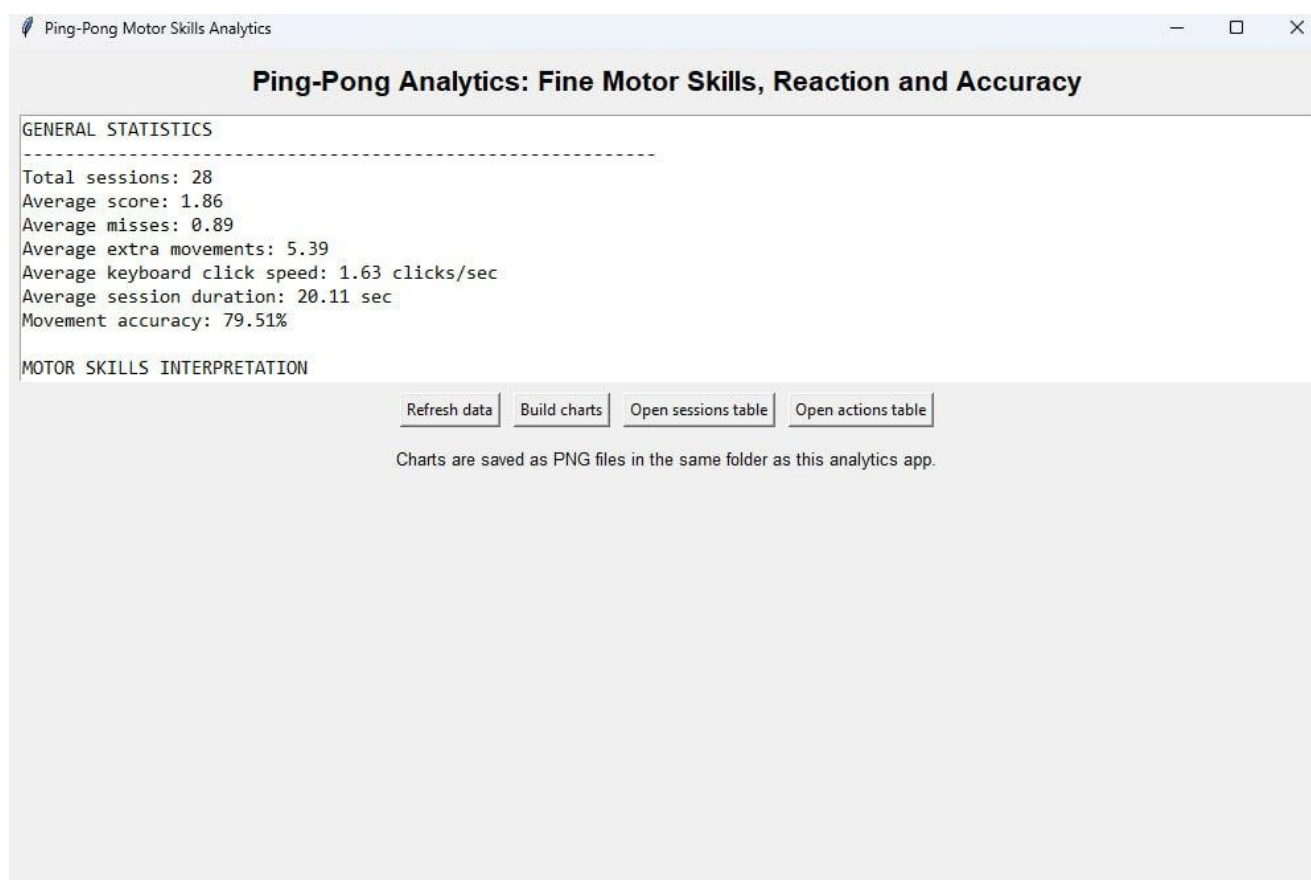


Рис. 2.14 Меню перегляду статистичних даних.

Аналітичний модуль, реалізований мовою Python, забезпечує побудову графіків та діаграм, які дозволяють наочно оцінити динаміку реакції користувача, частоту подій, ефективність керування платформою та інші параметри. Графічні звіти створюються за допомогою бібліотеки matplotlib рисунок 2.15-2.18, що дозволяє формувати лінійні графіки, гістограми та інші типи візуалізацій.



Рис. 2.15 Графік динаміки набраних балів за сесіями

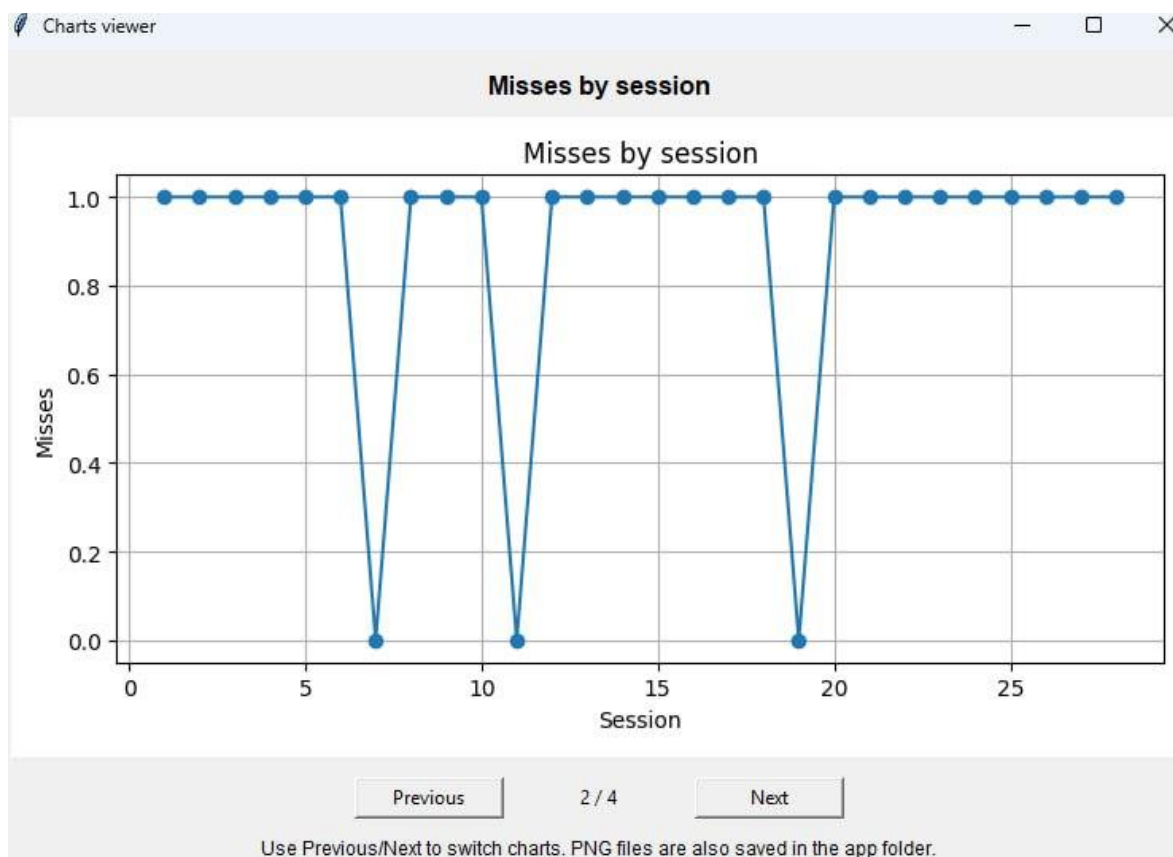


Рис. 2.16 Графік кількості пропущених м'ячів

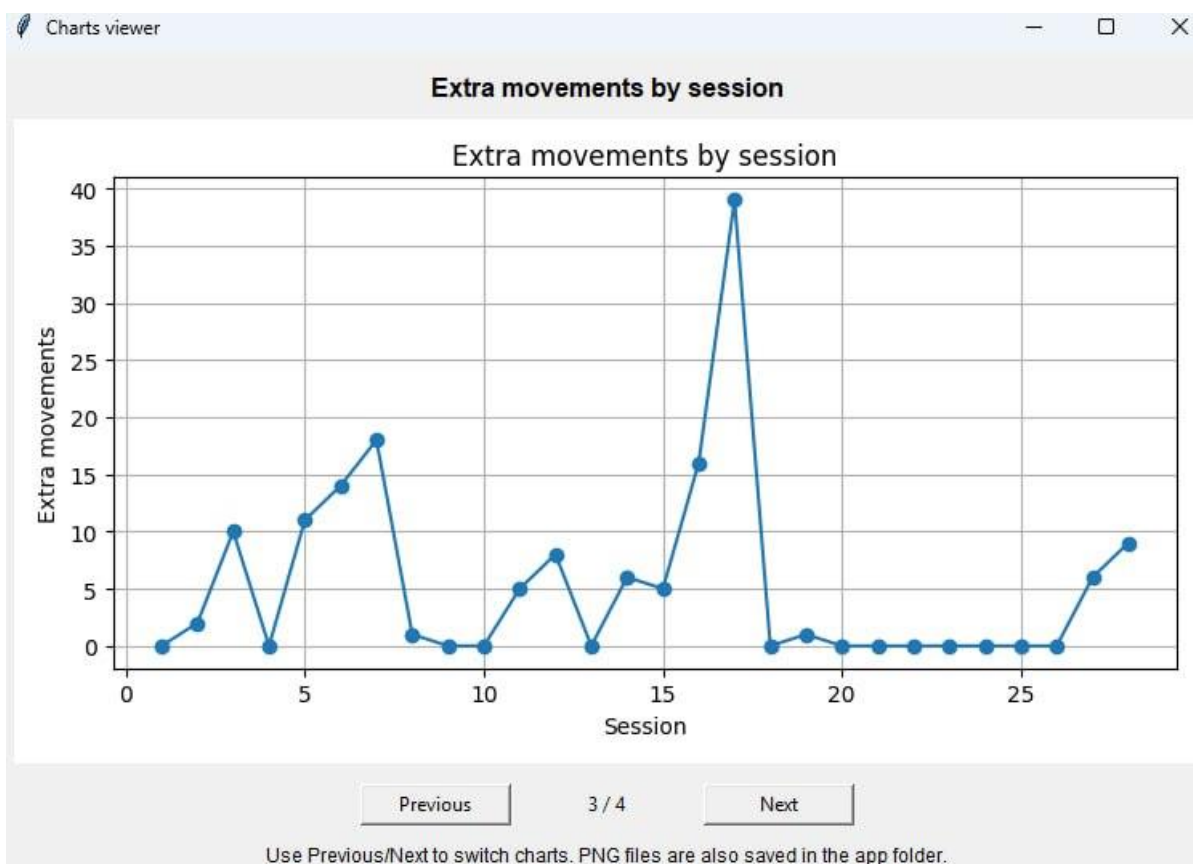


Рис. 2.17 Графік кількості зайвих рухів

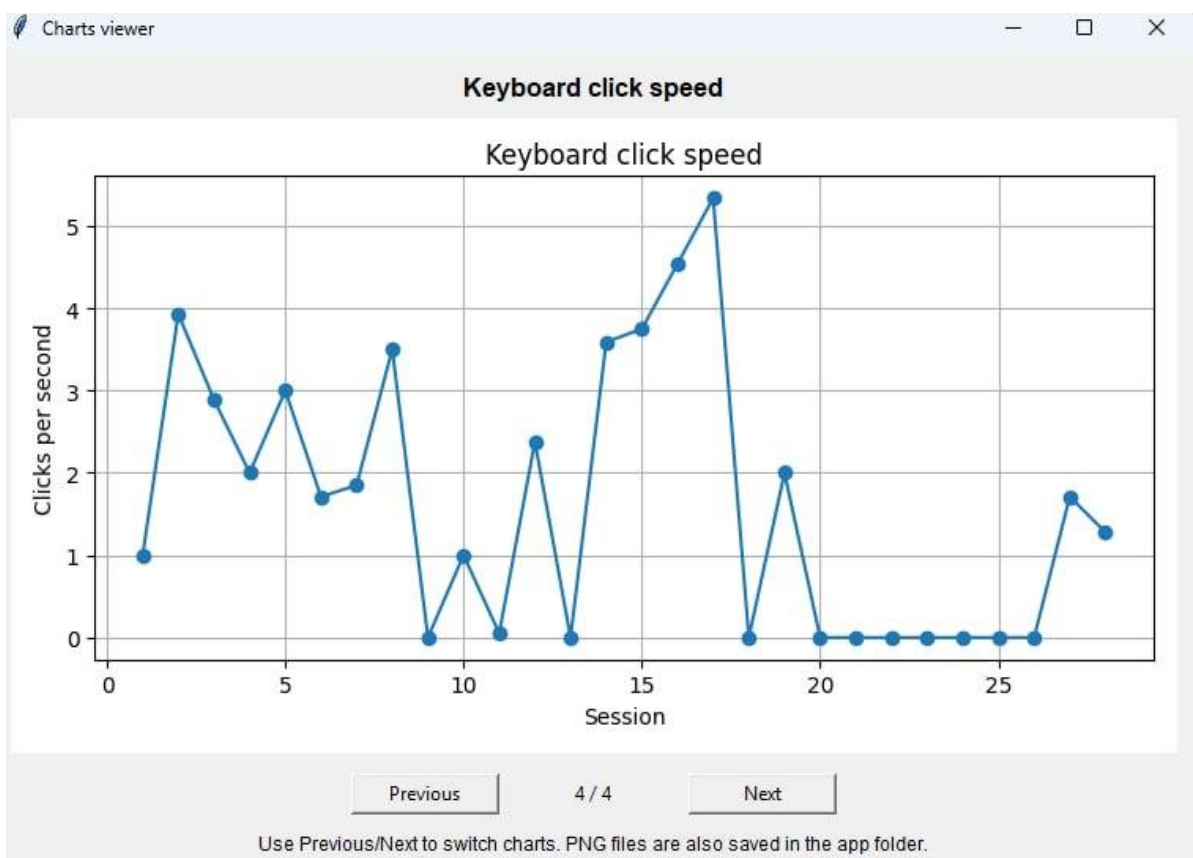


Рис. 2.18 Графік швидкості натискання клавіш

Окрім графічних звітів, аналітичний модуль формує таблиці представлені на рисунку 2.19-2.20, що містять числові показники, такі як середній, мінімальний та максимальний час реакції, кількість подій кожного типу, загальна кількість рухів та інші метрики. Табличне представлення дозволяє швидко оцінити результати та порівняти їх між різними ігровими сесіями.

session_id	date	level	score	paddle_hits	wall_hits	misses	total_moves	useful_moves	extra_moves	key_clicks	clicks_per_second	duration_seconds	result
1779373749	2026-05-21 17:29:11	Level 1	0	0	0	1	2	2	0	2	1.00	2	Loss
1779373751	2026-05-21 17:29:25	Level 1	2	2	3	1	55	53	2	55	3.93	14	Loss
1779373820	2026-05-21 17:30:29	Level 1	1	1	2	1	26	16	10	26	2.89	9	Loss
1779373832	2026-05-21 17:30:34	Level 1	0	0	0	1	4	4	0	4	2.00	2	Loss
1779373872	2026-05-21 17:31:33	Level 1	3	3	6	1	63	52	11	63	3.00	21	Loss
1779373933	2026-05-21 17:32:50	Level 1	2	2	4	1	29	15	14	29	1.71	17	Loss
1779374135	2026-05-21 17:36:01	Level 1	6	6	6	0	48	30	18	48	1.85	26	Exit
1779374221	2026-05-21 17:37:03	Level 1	0	0	0	1	7	6	1	7	3.50	2	Loss
1779374340	2026-05-21 17:39:02	Level 1	0	0	0	1	0	0	0	0	0.00	2	Loss
1779374344	2026-05-21 17:39:06	Level 1	0	0	0	1	2	2	0	2	1.00	2	Loss
1779374347	2026-05-21 17:43:08	Level 1	1	1	2	0	11	6	5	11	0.05	241	Exit
1779374660	2026-05-21 17:44:33	Level 1	2	2	4	1	31	23	8	31	2.38	13	Loss
1779374694	2026-05-21 17:49:56	Level 1	0	0	0	1	0	0	0	0	0.00	2	Loss
1779374996	2026-05-21 17:50:10	Level 1	3	3	5	1	79	73	6	79	3.59	22	Loss
1779375174	2026-05-21 17:53:02	Level 1	2	2	2	1	30	25	5	30	3.75	8	Loss
1779375183	2026-05-21 17:53:16	Level 1	2	2	4	1	59	43	16	59	4.54	13	Loss
1779375206	2026-05-21 17:53:55	Level 7	11	11	15	1	155	116	39	155	5.34	29	Loss
1779393839	2026-05-21 23:04:00	Level 7	0	0	0	1	0	0	0	0	0.00	1	Loss
1779393844	2026-05-21 23:04:07	Level 7	1	1	2	0	6	5	1	6	2.00	3	Exit
1779614688	2026-05-24 12:24:59	Level 4	0	0	0	1	0	0	0	0	0.00	1	Loss
1779614733	2026-05-24 12:25:34	Level 4	0	0	0	1	0	0	0	0	0.00	1	Loss
1779614740	2026-05-24 12:25:41	Level 4	0	0	0	1	0	0	0	0	0.00	1	Loss
1779614744	2026-05-24 12:25:45	Level 4	0	0	0	1	0	0	0	0	0.00	1	Loss
1779614752	2026-05-24 12:26:34	Level 4	0	0	0	1	0	0	0	0	0.00	42	Loss
1779614823	2026-05-24 12:27:05	Level 4	0	0	0	1	0	0	0	0	0.00	2	Loss
1779614829	2026-05-24 12:27:11	Level 4	0	0	0	1	0	0	0	0	0.00	2	Loss
1779614833	2026-05-24 12:28:05	Level 4	10	10	20	1	89	83	6	89	1.71	52	Loss
1779614898	2026-05-24 12:28:50	Level 4	6	6	12	1	41	32	9	41	1.28	32	Loss

Рис. 2.19 Загальна статистика ігрових сесій

session_id	second	action	paddle_x	paddle_y	ball_x	ball_y	paddle_angle	useful_move
1779373751	10	move_left	24	11	23	0	0	1
1779373751	23	move_left	23	11	23	1	0	1
1779373751	10	move_down	22	11	22	2	0	0
1779373751	10	reset_angle	22	12	21	3	0	1
1779373751	11	rotate_right_45	22	12	15	9	0	1
1779373751	12	rotate_right_45	22	13	12	12	45	1
1779373751	12	rotate_left_45	22	13	9	15	45	1
1779373751	13	reset_angle	22	13	7	17	-45	1
1779373820	0	move_right	18	20	28	14	0	1
1779373820	1	move_right	19	20	29	15	0	1

Рис. 2.20 Журнал дій користувача

Таким чином, інтерфейс ігрового модуля поєднує простоту та функціональність, забезпечуючи зручну взаємодію з ігровим модулем і наочне представлення результатів аналізу. Консольний інтерфейс сприяє розвитку моторних навичок, а графічно-аналітичний - дозволяє детально оцінити ефективність дій користувача та зробити висновки щодо його прогресу

3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Налаштування проєкту та середовища розробки

Розробка гри для тренування дрібної моторики та аналізу дій користувача потребує попереднього налаштування середовища розробки, вибору відповідних інструментів та підготовки структури проєкту [5]. На цьому етапі визначаються технології, засоби компіляції, бібліотеки, формати даних та загальна організація програмного коду, що забезпечує подальшу коректну роботу системи.

Так як гра складається з двох окремих модулів - ігрового модуля мовою C++ та аналітичного модуля мовою Python, налаштування середовища розробки здійснювалося окремо для кожної частини, з урахуванням їхніх функціональних особливостей [10].

Для реалізації ігрового модуля було обрано мову програмування C++, оскільки вона забезпечує високу продуктивність, низький рівень затримок та прямий контроль над обчисленнями, що є критично важливим для інтерактивних застосунків реального часу. Розробка здійснювалася у середовищі Microsoft Visual Studio, яке надає зручні засоби компіляції, налагодження та роботи з консольними застосунками [3]. Для коректного відображення ігрового поля та керування курсором використовувалися стандартні бібліотеки `<iostream>`, `<conio.h>`, `<windows.h>` та інші допоміжні модулі.

Структура C++-проєкту була організована таким чином, щоб забезпечити чітке розділення логіки гри на окремі компоненти [5]. Усі класи (Ball, Paddle, Game) були винесені у відповідні файли, що спрощує супровід та розширення функціоналу. Окремо було створено модуль для запису статистики у CSV-файли, який забезпечує взаємодію з аналітичним модулем [12]. Для реалізації аналітичної частини було обрано мову Python, яка є стандартом де-факто для роботи з даними, статистикою та візуалізацією. Налаштування середовища виконувалося у Python 3.14 із використанням бібліотек:

- pandas - для зчитування та обробки CSV-файлів;
- numpy - для математичних операцій;
- matplotlib - для побудови графіків;
- tkinter - для створення простого графічного інтерфейсу перегляду діаграм.

Усі залежності були встановлені через менеджер пакетів `pip`, що забезпечує простоту розгортання проєкту на будь-якому комп'ютері. Структура Python-модуля передбачає окремі файли для обробки даних, побудови графіків та відображення результатів у графічному інтерфейсі. Для забезпечення коректної взаємодії між модулями було визначено єдиний формат обміну даними - CSV-файли, які зберігаються у спеціальній директорії проєкту [12]. Ігровий модуль автоматично створює та оновлює ці файли після кожної сесії, а аналітичний модуль зчитує їх для подальшої обробки. Це дозволяє запускати модулі незалежно один від одного, а також повторно аналізувати дані без необхідності повторного проходження гри.

Таким чином, налаштування середовища розробки включало вибір відповідних інструментів, підготовку структури проєкту, встановлення необхідних бібліотек та визначення формату обміну даними. Це створило основу для подальшої реалізації ігрової логіки, механізмів збору статистики та аналітичних інструментів, що розглядаються у наступних підрозділах.

3.2 Реалізація ігрового модуля мовою C++

Ігровий модуль реалізує основну логіку роботи тренажера: рух м'яча, керування платформою, обробку зіткнень, підрахунок очок та фіксацію дій користувача. Уся механіка побудована на консольному відображенні ігрового поля та циклічному оновленні стану об'єктів. Основними елементами ігрової логіки є класи **Ball**, **Paddle** та **Game**. Клас **Ball** відповідає за координати м'яча та зміну напрямку руху. Клас **Paddle** реалізує платформу користувача, включаючи її нахил та вертикальні переміщення. Клас **Game** об'єднує всі компоненти, забезпечує

головний цикл гри, обробку клавіш та взаємодію з модулем логування показано на рисунку 3.1 [12].

```

197 class Ball {
198 public:
199     int x;
200     int y;
201     Direction direction;
202
203     Ball() {
204         reset();
205     }
206
207     void reset() {
208         x = WIDTH / 2;
209         y = HEIGHT / 2;
210         direction = RIGHT_DOWN;
211     }
212
213     void move() {
214         switch (direction) {
215             case LEFT_DOWN:
216                 x--;
217                 y++;
218                 break;
219             case RIGHT_DOWN:
220                 x++;
221                 y++;
222                 break;
223             case LEFT_UP:
224                 x--;
225                 y--;
226                 break;
227             case RIGHT_UP:
228                 x++;
229                 y--;
230                 break;
231             default:
232                 break;
233         }
234     }
235 };

```

Рис. 3.1 Рух м'яча

Рух платформи реалізовано з урахуванням її розміру показано на рисунку 3.2, кута нахилу та обмежень ігрового поля. Платформа може рухатися вліво, вправо, вгору та вниз, а також нахилитися на $\pm 45^\circ$, що впливає на траєкторію м'яча [16].

```

237 class Paddle {
238 public:
239     int x;
240     int y;
241     int size;
242     int angle;
243
244     Paddle(int paddleSize = 12) {
245         size = paddleSize;
246         x = WIDTH / 2 - size / 2;
247         y = HEIGHT - 2;
248         angle = 0;
249     }
250
251     void setSize(int paddleSize) {
252         size = paddleSize;
253         x = WIDTH / 2 - size / 2;
254         y = HEIGHT - 2;
255         angle = 0;
256     }
257
258     void moveLeft(int step) {
259         x -= step;
260         if (x < 0) x = 0;
261     }

```

Рис. 3.2 Рух платформи

Головний цикл гри відповідає за оновлення стану об'єктів, обробку зіткнень, фіксацію очок та завершення гри у разі пропуску м'яча продемонстровано на рисунку 3.3 [2]. У цьому циклі також викликається логування дій користувача [12].

```

792 void run() {
793     clearConsoleFull();
794     hideCursor();
795
796     sessionId = logger.createSessionId();
797     score = 0;
798     paddleHits = 0;
799     wallHits = 0;
800     misses = 0;
801     totalMoves = 0;
802     usefulMoves = 0;
803     extraMoves = 0;
804     keyClicks = 0;
805     gameOver = false;
806     playerExit = false;
807     paused = false;
808
809     ball.reset();
810     paddle.setSize(levels[currentLevelIndex].paddleSize);
811     startTime = time(nullptr);
812
813     while (!gameOver) {
814         draw();
815         input();
816         update();
817         Sleep(levels[currentLevelIndex].ballDelay);
818     }
819
820     if (playerExit) {
821         finishGame("Exit");
822     }
823     else {
824         finishGame("Loss");
825     }
826 }
827 };

```

Рис. 3.3 Головний цикл гри

Обробка зіткнень включає зміну напрямку м'яча при ударі об стінки або платформу, а також фіксацію пропуску м'яча показано на рисунку 3.4 [16].

```

659         if (paddle.contains(ball.x, ball.y)) {
660             score++;
661             paddleHits++;
662             sound.paddleHit();
663
664             if (paddle.angle < 0) {
665                 ball.direction = LEFT_UP;
666             }
667             else if (paddle.angle > 0) {
668                 ball.direction = RIGHT_UP;
669             }
670             else {
671                 if (ball.x < paddle.x + paddle.size / 2) {
672                     ball.direction = LEFT_UP;
673                 }
674                 else {
675                     ball.direction = RIGHT_UP;
676                 }
677             }

```

Рис. 3.4 Обробка зіткнень

Таким чином, ігровий модуль забезпечує повний цикл взаємодії користувача з тренажером, включаючи рух об'єктів, обробку подій та підрахунок результатів.

3.3 Реалізація аналітичного модуля мовою Python

Аналітичний модуль призначений для обробки статистичних даних, які формує ігровий модуль, та для побудови графіків, що відображають динаміку моторики користувача. Модуль реалізовано мовою Python із використанням стандартних бібліотек csv, tkinter та matplotlib [10]. Він зчитує дані з файлів game_sessions.csv та game_actions.csv, обчислює ключові показники та формує візуалізації [12].

Після запуску програма завантажує дані, обчислює середні значення та формує текстовий звіт. На рисунку 3.5 наведено фрагмент коду, який відповідає за зчитування CSV-файлів:

```

21 def read_csv(path):
22     if not path.exists():
23         return []
24     with path.open("r", encoding="utf-8", newline="") as file:
25         return list(csv.DictReader(file))

```

Рис. 3.5 Зчитування даних із CSV

Після завантаження даних виконується їх обробка: обчислення середніх значень, точності рухів, швидкості натискань клавіш та інших показників. Результати виводяться у текстове поле інтерфейсу наведено на рисунку 3.6.

```

75 scores = [safe_int(row.get("score")) for row in self.sessions]
76 misses = [safe_int(row.get("misses")) for row in self.sessions]
77 extra_moves = [safe_int(row.get("extra_moves")) for row in self.sessions]
78 key_clicks = [safe_int(row.get("key_clicks")) for row in self.sessions]
79 click_speed = [safe_float(row.get("clicks_per_second")) for row in self.sessions]
80 durations = [safe_int(row.get("duration_seconds")) for row in self.sessions]
81 useful_moves = [safe_int(row.get("useful_moves")) for row in self.sessions]
82 total_moves = [safe_int(row.get("total_moves")) for row in self.sessions]
83
84 total_sessions = len(self.sessions)
85 avg_score = sum(scores) / total_sessions
86 avg_misses = sum(misses) / total_sessions
87 avg_extra = sum(extra_moves) / total_sessions
88 avg_click_speed = sum(click_speed) / total_sessions
89 avg_duration = sum(durations) / total_sessions

```

Рис. 3.6 Обробка статистики

Для побудови графіків використовується бібліотека matplotlib [10]. Модуль створює чотири графіки: прогрес очок, кількість пропущених м'ячів, зайві рухи та швидкість натискань клавіш [11]. Графіки зберігаються у PNG-файли та можуть бути переглянуті у вбудованому вікні, рисунок 3.7.

```

143 charts = [
144     ("score_progress.png", "Score progress", "Session", "Score", scores),
145     ("misses_chart.png", "Misses by session", "Session", "Misses", misses),
146     ("extra_moves_chart.png", "Extra movements by session", "Session", "Extra movements", extra_moves),
147     ("click_speed_chart.png", "Keyboard click speed", "Session", "Clicks per second", click_speed),
148 ]
149
150 for filename, title, xlabel, ylabel, values in charts:
151     plt.figure(figsize=(6, 3.5))
152     plt.plot(x, values, marker="o")
153     plt.title(title)
154     plt.xlabel(xlabel)
155     plt.ylabel(ylabel)
156     plt.grid(True)
157     plt.tight_layout()
158     plt.savefig(filename, dpi=120, bbox_inches="tight")
159     plt.close()

```

Рис. 3.7 Побудова графіків

Для перегляду графіків у програмі використовується окреме вікно з кнопками «Previous» та «Next». Воно дозволяє перемикатися між графіками без повторного запуску побудови. Фрагмент коду показано на рисунку 3.8.

```

187 figure = Figure(figsize=(6.4, 3.6), dpi=100)
188 axis = figure.add_subplot(111)
189 canvas = FigureCanvasTkAgg(figure, master=chart_frame)
190 canvas.get_tk_widget().pack(fill="both", expand=True)

```

Рис. 3.8 Вікно перегляду графіків

Також модуль дозволяє переглядати таблиці сесій та дій у вигляді інтерактивних таблиць Treeview, рисунок 3.9, що спрощує аналіз окремих подій.

```

254 tree = ttk.Treeview(frame, columns=columns, show="headings")
255 tree.pack(side="left", (parameter) text: str ie)
256
257 for col in columns:
258     tree.heading(col, text=col)
259     tree.column(col, width=120)
260
261 for row in rows:
262     tree.insert("", tk.END, values=[row.get(col, "") for col in columns])
263
264 scrollbar = ttk.Scrollbar(frame, orient="vertical", command=tree.yview)
265 tree.configure(yscrollcommand=scrollbar.set)
266 scrollbar.pack(side="right", fill="y")

```

Рис. 3.9 Відображення таблиці

Аналітичний модуль забезпечує повний цикл обробки даних: від зчитування та аналізу до побудови графіків та відображення таблиць, що дозволяє користувачу оцінити свою динаміку та якість моторних навичок.

3.4 Механізм збору та збереження статистичних даних

Збір статистики є ключовою частиною гри, оскільки саме ці дані використовуються аналітичним модулем для побудови графіків та оцінки моторики користувача. Уся логіка фіксації подій реалізована в класі `Logger`, який відповідає за створення CSV-файлів `game_sessions.csv` та `game_actions.csv` [12].

Під час гри фіксуються два типи даних:

- підсумкова статистика сесії (рівень, очки, кількість ударів, пропуски, рухи, швидкість натискань тощо);
- покадрові дії користувача (натискання клавіш, координати платформи, координати м'яча, кут нахилу, корисність руху).

Файли створюються автоматично, а заголовки додаються лише при першому запуску [12]. Приклад зображено на рисунку 3.10.

```

88 void writeSession(
89     int sessionId,
90     const string& level,
91     int score,
92     int paddleHits,
93     int wallHits,
94     int misses,
95     int totalMoves,
96     int usefulMoves,
97     int extraMoves,
98     int keyClicks,
99     double clicksPerSecond,
100    int duration,
101    const string& result
102 ) {
103     bool needHeader = isFileEmpty(sessionFile);
104     ofstream file(sessionFile, ios::app);
105
106     if (!file.is_open()) return;
107
108     if (needHeader) {
109         file << "session_id,date,level,score,paddle_hits,wall_hits,misses,total_moves,useful_moves,extra_moves,key_clicks,clicks_per_second,duration_seconds,result\n";
110     }
111
112     file << sessionId << ","
113         << getCurrentDateTime() << ","
114         << level << ","
115         << score << ","
116         << paddleHits << ","
117         << wallHits << ","
118         << misses << ","
119         << totalMoves << ","
120         << usefulMoves << ","
121         << extraMoves << ","
122         << keyClicks << ","
123         << fixed << setprecision(2) << clicksPerSecond << ","
124         << duration << ","
125         << result << "\n";
126
127     file.close();
128 }

```

Рис. 3.10 Запис підсумкової статистики

Окремо фіксуються усі дії користувача: рухи платформи, повороти, натискання клавіш, а також координати м'яча в момент дії, приклад показано на рисунку 3.11. Це дозволяє виконувати детальний аналіз поведінки гравця.

```

130 void writeAction(
131     int sessionId,
132     int secondFromStart,
133     const string& action,
134     int paddleX,
135     int paddleY,
136     int ballX,
137     int ballY,
138     int paddleAngle,
139     bool usefulMove
140 ) {
141     bool needHeader = isFileEmpty(actionsFile);
142     ofstream file(actionsFile, ios::app);
143
144     if (!file.is_open()) return;
145
146     if (needHeader) {
147         file << "session_id,second,action,paddle_x,paddle_y,ball_x,ball_y,paddle_angle,useful_move\n";
148     }
149
150     file << sessionId << ","
151         << secondFromStart << ","
152         << action << ","
153         << paddleX << ","
154         << paddleY << ","
155         << ballX << ","
156         << ballY << ","
157         << paddleAngle << ","
158         << (usefulMove ? 1 : 0) << "\n";
159
160     file.close();
161 }
162 };

```

Рис. 3.11 Запис дій користувача

Виклик логування інтегровано безпосередньо в обробку клавіш [12]. Для кожного натискання визначається, чи був рух корисним, після чого подія записується у файл, рисунок 3.12.

```

446 void logKeyAction(char key, bool usefulMove) {
447     int sec = static_cast<int>(time(nullptr) - startTime);
448     logger.writeAction(
449         sessionId,
450         sec,
451         keyToAction(key),
452         paddle.x,
453         paddle.y,
454         ball.x,
455         ball.y,
456         paddle.angle,
457         usefulMove
458     );
459 }

```

Рис. 3.12 Логування натискань клавіш

Після завершення гри викликається метод `finishGame()`, який обчислює тривалість сесії, швидкість натискань клавіш та записує підсумкові дані, рисунок 3.13 [12], [14].

```

688 void finishGame(const string& result) {
689     int duration = static_cast<int>(time(nullptr) - startTime);
690     double clicksPerSecond = 0.0;
691
692     if (duration > 0) {
693         clicksPerSecond = static_cast<double>(keyClicks) / duration;
694     }
695
696     logger.writeSession(
697         sessionId,
698         levels[currentLevelIndex].name,
699         score,
700         paddleHits,
701         wallHits,
702         misses,
703         totalMoves,
704         usefulMoves,
705         extraMoves,
706         keyClicks,
707         clicksPerSecond,
708         duration,
709         result
710     );
711
712     if (result == "Loss") {
713         sound.lose();
714     }

```

Рис. 3.13 Завершення гри та збереження статистики

Таким чином, механізм збору статистики забезпечує повний запис усіх подій і параметрів ігрового процесу показано на рисунку 3.13, що дозволяє аналітичному модулю виконувати глибокий аналіз моторики, точності та реакції користувача [11], [14].

ВИСНОВКИ

У кваліфікаційній роботі було реалізовано повний цикл створення інтерактивного тренажера для розвитку дрібної моторики, точності рухів та швидкості реакції користувача. Розроблена гра складається з двох автономних, але взаємопов'язаних модулів: ігрового застосунку на C++ та аналітичного модуля на Python. Така архітектура дала змогу поєднати високу продуктивність консольної гри в реальному часі з гнучкими можливостями статистичного аналізу.

У процесі розробки було створено повноцінну ігрову логіку: рух м'яча, адаптивну платформу з можливістю нахилу, систему рівнів складності, обробку зіткнень, підрахунок очок та фіксацію всіх дій користувача. Реалізований механізм логування формує два структуровані CSV-файли, що містять як покадрові дії, так і підсумкові параметри кожної сесії. Це забезпечує високу точність збору даних та дозволяє виконувати глибокий аналіз моторики.

Аналітичний модуль Python обробляє зібрані дані, обчислює ключові метрики (точність рухів, кількість зайвих дій, швидкість натискань клавіш, динаміку результатів) та будує графіки, що відображають прогрес користувача. Інтерактивний інтерфейс на основі tkinter дозволяє переглядати статистику, графіки та таблиці безпосередньо в застосунку, що робить систему зручною для практичного використання.

Досягнуті результати підтверджують працездатність розробленого тренажера як інструмента для оцінки та розвитку моторних навичок. Система стабільно працює на широкому спектрі комп'ютерів, не потребує зовнішніх залежностей, а модульна архітектура дозволяє масштабувати проєкт у напрямках розширення ігрової механіки, додавання нових метрик або інтеграції машинного навчання для персоналізованих рекомендацій.

Робота пройшла апробацію. За її результатами було підготовлено та опубліковано наступні тези доповідей:

1. Вінчик О.А., Куклінський М.В. Визначення вимог гри пінг-понг. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в

програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.381 - 383.

2. Вінчик О.А., Куклінський М.В. Забезпечення безпеки інформації в ігровому модулі «Пінг-понг». Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2026, С.145 -146.

ПЕРЕЛІК ВИКПОСИЛАНЬ

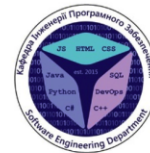
1. Страуструп Б. Програмування. Принципи та практика з використанням C++. – Київ : Вільямс, 2020. – 912 с.
2. Lutz M. Learning Python. – 5th ed. – Sebastopol : O'Reilly Media, 2013. – 1648 p. – ISBN 978-1449355739.
3. McKinney W. Python for Data Analysis. – 3rd ed. – Sebastopol : O'Reilly Media, 2022. – 550 p. – ISBN 978-1098104030.
4. Герберт Ш., Саттер Г. Ефективне використання C++. – Київ : Кондор, 2019. – 352 с.
5. VanderPlas J., Grus J. Data Science Handbook. – 2nd ed. – Boston : O'Reilly, 2021. – 620 p.
6. Downey A., Elkner J., Meyers C. How to Think Like a Computer Scientist: Learning with Python. – Needham : Green Tea Press, 2015. – 300 p.
7. Алгоритми: побудова та аналіз / Т. Кормен та ін. – Київ : Вільямс, 2018. – 1312 с.
8. Data Visualization Handbook / J. Knafllic et al. – New York : Wiley, 2020. – 420 p.
9. Python 3.10 Documentation / за ред. G. Rossum. – London : Python Software Foundation, 2021. – 380 p.
10. Computer Graphics Handbook / ed. by A. Keller. – Berlin : Springer, 2019. – 512 p.
11. Zhang Q., Lu J., Zhang G. Reaction Time and Motor Skills in Interactive Systems. *Journal of Human–Computer Interaction*. – 2022. – Vol. 18, No. 2. – P. 44–59.
12. Patel D., Chauhan U. Data Logging and Behavioral Tracking in Real-Time Systems. *International Journal of Computing and Digital Systems*. – 2023. – Vol. 13, No. 1. – P. 851–868. – URL: <https://doi.org/10.12785/ijcds/130168>
13. Cherednichenko O.Y., Ivashchenko O.V., Gontar Y.M. Методи аналізу поведінкових даних у тренажерних системах. *Вісник ХТУ «ХПІ»*. – 2021. – №1320 (44). – С. 57–66.

14. Henderson R., Hollingworth A. Human Reaction Time Models. *Cognitive Systems Review*. – 2020. – Vol. 12. – P. 101–118.
15. Урба С. І. Методи аналізу поведінкових даних у цифрових системах : дис. ... д-ра техн. наук : 05.13.06. – Львів, 2020. – 480 с.
16. Про захист інформації в інформаційно-телекомунікаційних системах : Закон України від 05.07.1994 р. № 80/94-ВР. – URL: <https://zakon.rada.gov.ua/laws/show/80/94-вр> (дата звернення: 24.05.2026).
17. Про авторське право і суміжні права : Закон України від 23.12.1993 р. № 3792-XII. – URL: <https://zakon.rada.gov.ua/laws/show/3792-12> (zakon.rada.gov.ua in Bing) (дата звернення: 24.05.2026).
18. Microsoft. C++ Language Reference. – URL: <https://learn.microsoft.com/cpp/> (дата звернення: 24.05.2026).
19. Python Software Foundation. Python 3.10 Documentation. – URL: <https://docs.python.org/3.10/> (дата звернення: 24.05.2026).
20. Matplotlib Documentation. – URL: <https://matplotlib.org/stable/> (дата звернення: 24.05.2026).
21. Pandas Documentation. – URL: <https://pandas.pydata.org/docs/> (дата звернення: 24.05.2026).
22. Tkinter Reference Manual. – URL: <https://docs.python.org/3/library/tkinter.html> (дата звернення: 24.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка ігрового застосунку для гри пінг-понг

Виконав студент 4 курсу
групи ПД-42

Вінчик Олександр Анатолійович

Керівник роботи: доцент, канд. технічних наук

Куклінський Максим Володимирович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності розвитку дрібної моторики та аналізу дій користувачів за допомогою інтерактивного консольного ігрового застосунку «Пінг-понг»

Об'єкт дослідження: процес взаємодії користувача з ігровими застосунками для розвитку координації рухів та швидкості реакції.

Предмет дослідження: методи та засоби програмної реалізації ігрових застосунків та аналітичних систем для збору й аналізу статистики дій користувача.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати особливості використання ігрових застосунків для розвитку дрібної моторики.
2. Дослідити та проаналізувати існуючі ігри типу «Пінг-Понг» для визначення можливостей та обмежень.
3. Сформувати функціональні та нефункціональні вимоги.
4. Обґрунтувати вибір мови програмування та середовища розробки.
5. Виконати проєктування програмного застосунку.
6. Реалізувати ігровий модуль.
7. Розробити механізм збору статистичних даних.
8. Створити аналітичний модуль.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Aim Lab	Kovaak's	osu!	Typing Trainers	Reflex PC	Eye-Hand PC	Застосунок Пінг-понг
Тренування дрібної моторики	+	+	+	+	+	+	+
Тренування швидкості реакції	+	+	+	+	+	+	+
Аналітика дій користувача	+	+	–	+	–	–	+
Збір статистики (CSV)	–	–	–	–	–	–	+
Графічна аналітика	+	+	–	+	–	–	+
Оцінка корисних/зайвих рухів	–	–	–	–	–	–	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги до ігрового застосунку

1. Наявність головного меню.
2. Програма повинна відображати ігрове поле, м'яч, платформу та поточний рахунок користувача.
3. Програма повинна забезпечувати керування платформою за допомогою клавіатури.
4. Програма повинна автоматично переміщувати м'яч по ігровому полю.
5. Програма повинна обробляти зіткнення м'яча з межами ігрового поля та платформою.
6. Підрахунок балів та завершення раунду.
7. Програма повинна завершувати гру у разі пропуску.
8. Зміна швидкості м'яча залежно від рівня складності.
9. Відтворення звукових ефектів.
10. Збереження статистики після кожної ігрової сесії.

Функціональні вимоги до аналітичного модулю

1. Зчитування CSV-файлів, створених ігровим модулем.
2. Аналіз показників:
 - час реакції;
 - точність керування;
 - кількість корисних/зайвих рухів;
 - кількість натискань клавіш;
 - динаміка руху платформи.
3. Побудова графіків:
 - прогрес результату;
 - кількість пропусків за сесіями;
 - зайві рухи за сесіями;
 - швидкість натискання клавіш.
4. Формування таблиць статистичних показників
5. Генерація підсумкових звітів
6. Відображення результатів у зручному форматі для користувача

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Нефункціональні вимоги до ігрового застосунку

1. Вимоги до інтерфейсу та керування:
 - керування платформою здійснюється клавішами W/A/S/D, Q/E, R;
 - затримка реакції на натискання клавіш не більше 50 мс;
 - меню повинно бути простим, текстовим, доступним у консолі та містити не менше 5 пунктів.
2. Вимоги до ігрової логіки:
 - рух платформи повинен бути плавним при FPS ≥ 20 ;
 - система нарахування балів збільшує значення на 1 при кожному відбитті м'яча;
 - м'яч повинен коректно змінювати напрямок при зіткненні зі стінами та платформою;
 - нахил платформи ($\pm 45^\circ$) повинен впливати на траєкторію м'яча.
3. Вимоги до формату даних:
 - логи зберігаються у форматі CSV (UTF-8);
 - файл game_actions.csv містить 9 структурованих полів;
 - файл game_sessions.csv містить 14 полів (ідентифікатор, час, статистика);
 - файли повинні відкриватися у Excel, Google Sheets, Python pandas.
4. Вимоги до продуктивності:
 - програма повинна працювати стабільно при швидкості м'яча до 30 мс затримки між кадрами;
 - час оновлення кадру не повинен перевищувати 50 мс.
5. Вимоги до надійності:
 - при аварійному завершенні CSV-файли не повинні пошкоджуватися;
 - програма не повинна зависати при швидких повторних натисканнях клавіш.
6. Вимоги до апаратного забезпечення:
 - мінімальна конфігурація:
 - CPU ≥ 1.6 GHz;
 - RAM ≥ 2 GB;
 - HDD ≥ 100 MB.
 - не потребує GPU або спеціального обладнання.
7. Вимоги до операційної системи:
 - підтримка Windows 7 / 8 / 10 / 11;
 - не потребує додаткових драйверів або SDK.

Нефункціональні вимоги до аналітичного застосунку

1. Вимоги до інтерфейсу:
 - інтерфейс побудований на Tkinter, відображатися при роздільній здатності від 1366×768;
 - вікна таблиць та графіків повинні масштабуватися без артефактів.
2. Вимоги до обробки даних:
 - програма повинна читати CSV-файли, створені ігровим модулем;
 - обробка даних повинна виконуватися менше ніж за 0.5 секунди при обсязі до 10 000 записів.
3. Вимоги до побудови графіків:
 - побудова кожного графіка повинна виконуватися менше ніж за 5 секунд;
 - графіки зберігаються у форматі PNG.
4. Вимоги до сумісності:
 - працює на Windows та Linux при наявності Python ≥ 3.14 ;
 - потребує лише стандартних бібліотек + matplotlib.
5. Вимоги до надійності:
 - програма повинна працювати при відсутності файлів CSV (виводити повідомлення);
 - не повинна аварійно завершуватися при порожніх або частково пошкоджених CSV.

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



Visual Studio



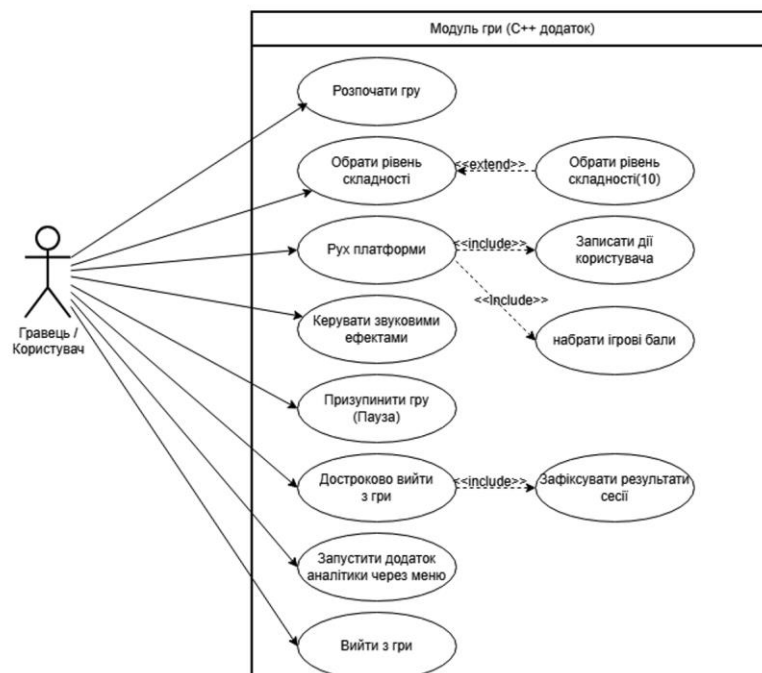
C++



Python

7

ДІАГРАМИ ПРЕЦЕДЕНТІВ



8

ДІАГРАМИ ПРЕЦЕДЕНТІВ



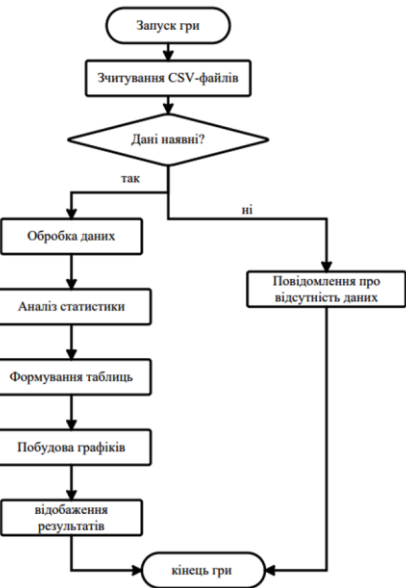
9

ОПИС ПРОГРАМИ: АЛГОРИТМИ РОБОТИ ІГРОВОГО МОДУЛЯ

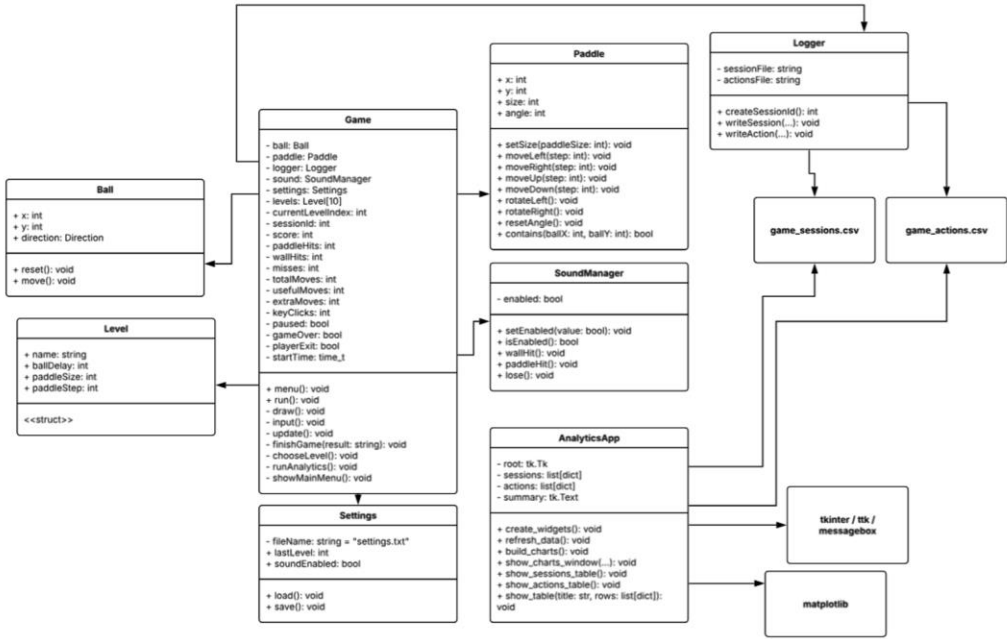


10

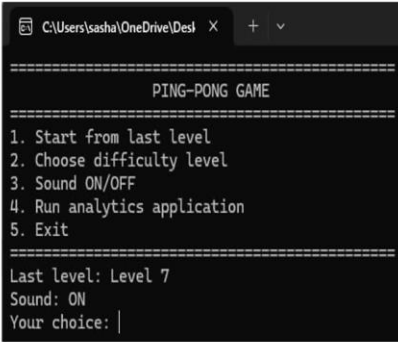
ОПИС ПРОГРАМИ: АЛГОРИТМИ РОБОТИ
АНАЛІТИЧНОГО МОДУЛЯ



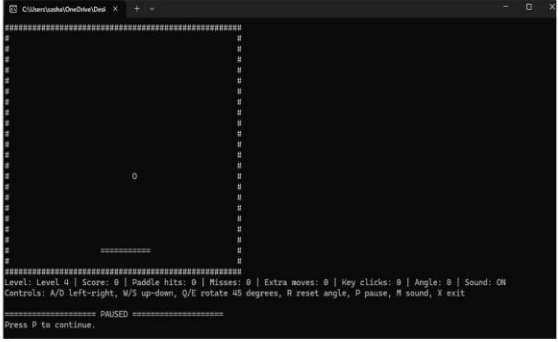
ДІАГРАМА КЛАСІВ



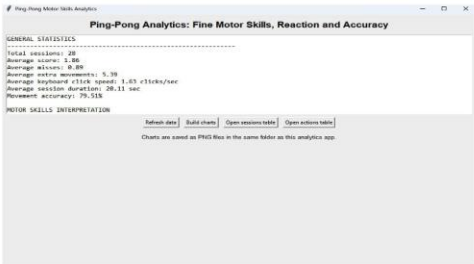
ЕКРАННІ ФОРМИ



Головне меню застосунку



геймплей

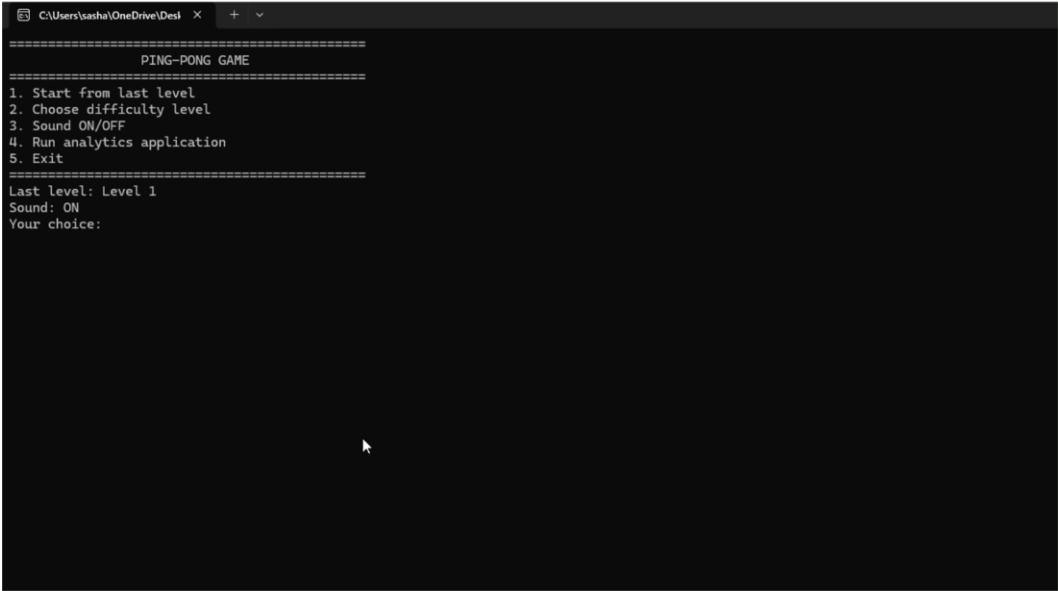


Головне меню аналітичної частини



Приклад діаграми статистики

ЕКРАННІ ФОРМИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Вінник О.А., Куклінський М.В. Визначення вимог гри пінг-понг. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.381 -383.

2. Вінник О.А., Куклінський М.В. Забезпечення безпеки інформації в ігровому -модулі «Пінг-понг». Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. С.145 -146.

15

ВИСНОВКИ

1. Проаналізовано особливості використання ігрових застосунків для розвитку дрібної моторики, визначено їх вплив на швидкість реакції, координацію рухів та точність дій користувача.
2. Досліджено існуючі ігри типу «Пінг-Понг», проаналізовано їх механіку, функціональні можливості та обмеження, що дозволило сформулювати вимоги до власного застосунку.
3. Сформовано функціональні та нефункціональні вимоги до програмного комплексу, що визначили структуру, поведінку та технічні характеристики системи.
4. Обґрунтовано вибір мови програмування C++ та середовища Visual Studio для реалізації ігрового модуля, а також Python для створення аналітичного модуля.
5. Виконано проєктування програмного застосунку.
6. Реалізовано ігровий модуль, який забезпечує роботу аркадної гри «Пінг-Понг», керування платформою, фізику руху м'яча, систему балів та консольний інтерфейс.
7. Розроблено механізм збору статистичних даних, що включає логування натискань клавіш, часу реакції, координат платформи та інших параметрів у форматі CSV.
8. Створено аналітичний модуль, який виконує обробку CSV-файлів, аналізує показники моторики, будує графіки, формує таблиці та генерує підсумкові звіти.
9. Проведено тестування застосунку.

16

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

Game.Cpp

```

#include <iostream>
#include <conio.h>
#include <windows.h>
#include <fstream>
#include <string>
#include <ctime>
#include <iomanip>
#include <sstream>
#include <cctype>
#include <limits>

using namespace std;

const int WIDTH = 50;
const int HEIGHT = 22;
const int LEVEL_COUNT = 10;

enum Direction {
    STOP = 0,
    LEFT_DOWN,
    RIGHT_DOWN,
    LEFT_UP,
    RIGHT_UP
};

struct Level {
    string name;
    int ballDelay;
    int paddleSize;
    int paddleStep;
};

class Settings {
private:
    string fileName = "settings.txt";

public:
    int lastLevel = 1;
    bool soundEnabled = true;

    void load() {
        ifstream file(fileName);

        if (file.is_open()) {
            file >> lastLevel;
            file >> soundEnabled;
            file.close();
        }

        if (lastLevel < 1 || lastLevel > LEVEL_COUNT) {
            lastLevel = 1;
        }
    }

    void save() {
        ofstream file(fileName);
        file << lastLevel << endl;
        file << soundEnabled << endl;
        file.close();
    }
};

}
};

class Logger {
private:
    string sessionFile = "game_sessions.csv";
    string actionsFile = "game_actions.csv";

    string getCurrentDateTime() {
        time_t now = time(nullptr);
        tm localTime;
        localtime_s(&localTime, &now);

        stringstream ss;
        ss << put_time(&localTime, "%Y-%m-%d %H:%M:%S");
        return ss.str();
    }

    bool isEmpty(const string& fileName) {
        ifstream file(fileName);
        return !file.is_open() || file.peek() == ifstream::traits_type::eof();
    }

public:
    int createSessionId() {
        return static_cast<int>(time(nullptr));
    }

    void writeSession(
        int sessionId,
        const string& level,
        int score,
        int paddleHits,
        int wallHits,
        int misses,
        int totalMoves,
        int usefulMoves,
        int extraMoves,
        int keyClicks,
        double clicksPerSecond,
        int duration,
        const string& result
    ) {
        bool needHeader = isEmpty(sessionFile);
        ofstream file(sessionFile, ios::app);

        if (!file.is_open()) return;

        if (needHeader) {
            file <<
                "session_id,date,level,score,paddle_hits,wall_hits,misses,total_moves,useful_moves,extra_moves,key_clicks,clicks_per_second,duration_seconds,result\n";
        }

        file << sessionId << ","
            << getCurrentDateTime() << ","
            << level << ","
            << score << ","
            << paddleHits << ","
            << wallHits << ","

```

```

        << misses << ", "
        << totalMoves << ", "
        << usefulMoves << ", "
        << extraMoves << ", "
        << keyClicks << ", "
        << fixed << setprecision(2) << clicksPerSecond << ", "
        << duration << ", "
        << result << "\n";

    file.close();
}

void writeAction(
    int sessionId,
    int secondFromStart,
    const string& action,
    int paddleX,
    int paddleY,
    int ballX,
    int ballY,
    int paddleAngle,
    bool usefulMove
) {
    bool needHeader = isEmpty(actionsFile);
    ofstream file(actionsFile, ios::app);

    if (!file.is_open()) return;

    if (needHeader) {
        file <<
"session_id,second,action,paddle_x,paddle_y,ball_x,ball_y,paddle_angle,useful_move\n";
    }

    file << sessionId << ", "
        << secondFromStart << ", "
        << action << ", "
        << paddleX << ", "
        << paddleY << ", "
        << ballX << ", "
        << ballY << ", "
        << paddleAngle << ", "
        << (usefulMove ? 1 : 0) << "\n";

    file.close();
}

};

class SoundManager {
private:
    bool enabled;

public:
    SoundManager(bool isEnabled = true) {
        enabled = isEnabled;
    }

    void setEnabled(bool value) {
        enabled = value;
    }

    bool isEnabled() const {
        return enabled;
    }

    void wallHit() {
        if (enabled) Beep(700, 35);
    }

    void paddleHit() {
        if (enabled) Beep(1000, 45);
    }

    void lose() {
        if (enabled) {
            Beep(350, 130);
            Beep(250, 180);
        }
    }
};

class Ball {
public:
    int x;
    int y;
    Direction direction;

    Ball() {
        reset();
    }

    void reset() {
        x = WIDTH / 2;
        y = HEIGHT / 2;
        direction = RIGHT_DOWN;
    }

    void move() {
        switch (direction) {
            case LEFT_DOWN:
                x--;
                y++;
                break;
            case RIGHT_DOWN:
                x++;
                y++;
                break;
            case LEFT_UP:
                x--;
                y--;
                break;
            case RIGHT_UP:
                x++;
                y--;
                break;
            default:
                break;
        }
    }
};

class Paddle {
public:
    int x;
    int y;
    int size;
    int angle;

    Paddle(int paddleSize = 12) {
        size = paddleSize;
        x = WIDTH / 2 - size / 2;
        y = HEIGHT - 2;
        angle = 0;
    }

    void setSize(int paddleSize) {
        size = paddleSize;
        x = WIDTH / 2 - size / 2;
    }
}

```

```

    y = HEIGHT - 2;
    angle = 0;
}

void moveLeft(int step) {
    x -= step;
    if (x < 0) x = 0;
}

void moveRight(int step) {
    x += step;
    if (x + size >= WIDTH) x = WIDTH - size;
}

int getPlatformYAt(int drawX) const {
    if (drawX < x || drawX >= x + size) return -1;

    if (angle == 0) return y;

    // Visual slope like on the screenshot: not 45 terminal
    cells, // but a gentle diagonal. Height depends on paddle size,
    so // each level keeps its own paddle length.
    int maxRise = size / 3;
    if (maxRise < 1) maxRise = 1;

    int offset = drawX - x;
    int rise = (offset * maxRise + (size - 1) / 2) / (size - 1);

    if (angle > 0) {
        // right side is higher: =====/
        return y - rise;
    }
    else {
        // left side is higher: \=====
        return y - (maxRise - rise);
    }
}

void keepInsideField() {
    int maxRise = (angle == 0) ? 0 : size / 3;
    if (maxRise < 1 && angle != 0) maxRise = 1;

    if (y - maxRise < 0) y = maxRise;
    if (y > HEIGHT - 1) y = HEIGHT - 1;
    if (x < 0) x = 0;
    if (x + size >= WIDTH) x = WIDTH - size;
}

void moveUp(int step) {
    y -= step;
    if (y < HEIGHT / 2) y = HEIGHT / 2;
    keepInsideField();
}

void moveDown(int step) {
    y += step;
    keepInsideField();
}

void rotateLeft() {
    angle = -45;
    keepInsideField();
}

void rotateRight() {
    angle = 45;
    keepInsideField();
}

}

void resetAngle() {
    angle = 0;
    keepInsideField();
}

bool contains(int ballX, int ballY) {
    int platformY = getPlatformYAt(ballX);
    return platformY != -1 && ballY == platformY;
}

bool isPlatformPoint(int drawX, int drawY) {
    int platformY = getPlatformYAt(drawX);
    return platformY != -1 && drawY == platformY;
}

char getPlatformSymbol() const {
    return '=';
}
};

class Game {
private:
    Ball ball;
    Paddle paddle;
    Logger logger;
    SoundManager sound;
    Settings settings;

    Level levels[LEVEL_COUNT] = {
        {"Level 1", 130, 14, 1},
        {"Level 2", 115, 13, 1},
        {"Level 3", 100, 12, 1},
        {"Level 4", 90, 11, 2},
        {"Level 5", 80, 10, 2},
        {"Level 6", 70, 9, 2},
        {"Level 7", 60, 8, 3},
        {"Level 8", 50, 7, 3},
        {"Level 9", 40, 6, 3},
        {"Level 10", 30, 5, 4}
    };

    int currentLevelIndex;
    int sessionId;
    int score;
    int paddleHits;
    int wallHits;
    int misses;
    int totalMoves;
    int usefulMoves;
    int extraMoves;
    int keyClicks;
    bool gameOver;
    bool playerExit;
    bool paused;
    time_t startTime;

    void hideCursor() {
        HANDLE consoleHandle =
        GetStdHandle(STD_OUTPUT_HANDLE);
        CONSOLE_CURSOR_INFO info;
        info.dwSize = 100;
        info.bVisible = FALSE;
        SetConsoleCursorInfo(consoleHandle, &info);
    }

    void setCursorPosition(int x, int y) {
        COORD position;

```

```

    position.X = static_cast<SHORT>(x);
    position.Y = static_cast<SHORT>(y);

SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), position);
}

void clearConsoleFull() {
    HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    DWORD count;
    DWORD cellCount;
    COORD homeCoords = { 0, 0 };

    if (hConsole == INVALID_HANDLE_VALUE) return;
    if (!GetConsoleScreenBufferInfo(hConsole, &csbi))
return;

    cellCount = csbi.dwSize.X * csbi.dwSize.Y;

    FillConsoleOutputCharacter(hConsole, ' ', cellCount,
homeCoords, &count);
    FillConsoleOutputAttribute(hConsole, csbi.wAttributes,
cellCount, homeCoords, &count);
    SetConsoleCursorPosition(hConsole, homeCoords);
}

void clearScreen() {
    setCursorPosition(0, 0);
}

void pauseMessage() {
    cout << endl << "Press any key to continue...";
    _getch();
}

bool isUsefulMovement(char key) {
    int paddleCenter = paddle.x + paddle.size / 2;

    if (key == 'a') return ball.x < paddleCenter;
    if (key == 'd') return ball.x > paddleCenter;
    if (key == 'w') return ball.y < paddle.y;
    if (key == 's') return ball.y > paddle.y;
    if (key == 'q' || key == 'e' || key == 'r') return true;

    return false;
}

string keyToAction(char key) {
    if (key == 'a') return "move_left";
    if (key == 'd') return "move_right";
    if (key == 'w') return "move_up";
    if (key == 's') return "move_down";
    if (key == 'q') return "rotate_left_45";
    if (key == 'e') return "rotate_right_45";
    if (key == 'r') return "reset_angle";
    if (key == 'm') return "sound_toggle";
    if (key == 'p') return "pause";
    if (key == 'x') return "exit";
    return "other";
}

void logKeyAction(char key, bool usefulMove) {
    int sec = static_cast<int>(time(nullptr) - startTime);
    logger.writeAction(
        sessionId,
        sec,
        keyToAction(key),

        paddle.x,
        paddle.y,
        ball.x,
        ball.y,
        paddle.angle,
        usefulMove
    );
}

void showMainMenu() {
    clearConsoleFull();

    cout <<
    "===== " <<
endl;
    cout << "          PING-PONG GAME          " << endl;
    cout <<
    "===== " <<
endl;
    cout << "1. Start from last level" << endl;
    cout << "2. Choose difficulty level" << endl;
    cout << "3. Sound ON/OFF" << endl;
    cout << "4. Run analytics application" << endl;
    cout << "5. Exit" << endl;
    cout <<
    "===== " <<
endl;
    cout << "Last level: " << levels[settings.lastLevel - 1].name << endl;
    cout << "Sound: " << (settings.soundEnabled ? "ON" : "OFF") <<
endl;
    cout << "Your choice: ";
}

void chooseLevel() {
    clearConsoleFull();

    cout << "Choose difficulty level:" << endl;
    cout << "-----" << endl;

    for (int i = 0; i < LEVEL_COUNT; i++) {
        cout << i + 1 << ". " << levels[i].name
        << " | ball delay: " << levels[i].ballDelay << " ms"
        << " | paddle size: " << levels[i].paddleSize
        << " | paddle speed: " << levels[i].paddleStep << endl;
    }

    cout << "-----" << endl;
    cout << "Enter level number from 1 to 10: ";

    int choice;
    cin >> choice;

    if (cin.fail()) {
        cin.clear();
        cin.ignore((numeric_limits<streamsize>::max)(), '\n');
        return;
    }

    if (choice >= 1 && choice <= LEVEL_COUNT) {
        settings.lastLevel = choice;
        settings.save();
    }

    cin.ignore((numeric_limits<streamsize>::max)(), '\n');
}

void runAnalytics() {
    clearConsoleFull();

```

```

cout << "Running analytics application..." << endl;
cout << "If it does not start, run manually:" << endl;
cout << "python analytics_app.py" << endl << endl;

int result = system("python analytics_app.py");

if (result != 0) {
    cout << endl << "Trying alternative command: py
analytics_app.py" << endl;
    system("py analytics_app.py");
}

pauseMessage();
}

void draw() {
    clearScreen();

    for (int i = 0; i < WIDTH + 2; i++) cout << "#";
    cout << endl;

    for (int y = 0; y < HEIGHT; y++) {
        cout << "#";

        for (int x = 0; x < WIDTH; x++) {
            if (x == ball.x && y == ball.y) {
                cout << "O";
            }
            else if (paddle.isPlatformPoint(x, y)) {
                cout << paddle.getPlatformSymbol();
            }
            else {
                cout << " ";
            }
        }

        cout << "#" << endl;
    }

    for (int i = 0; i < WIDTH + 2; i++) cout << "#";
    cout << endl;

    cout << "Level: " << levels[currentLevelIndex].name
    << " | Score: " << score
    << " | Paddle hits: " << paddleHits
    << " | Misses: " << misses
    << " | Extra moves: " << extraMoves
    << " | Key clicks: " << keyClicks
    << " | Angle: " << paddle.angle
    << " | Sound: " << (sound.isEnabled() ? "ON" :
"OFF") << endl;

    cout << "Controls: A/D left-right, W/S up-down, Q/E
rotate 45 degrees, R reset angle, P pause, M sound, X exit" <<
endl;

    if (paused) {
        cout << endl << "=====
PAUSED =====" << endl;
        cout << "Press P to continue." << endl;
    }
}

void input() {
    if (_kbhit()) {
        char key = _getch();
        key = static_cast<char>(tolower(static_cast<unsigned
char>(key)));

        int step = levels[currentLevelIndex].paddleStep;
        bool usefulMove = isUsefulMovement(key);

        if (key == 'a' || key == 'd' || key == 'w' || key == 's' || key == 'q' ||
key == 'e' || key == 'r') {
            keyClicks++;
            totalMoves++;

            if (usefulMove) usefulMoves++;
            else extraMoves++;
        }

        logKeyAction(key, usefulMove);

        switch (key) {
            case 'a':
                if (!paused) paddle.moveLeft(step);
                break;
            case 'd':
                if (!paused) paddle.moveRight(step);
                break;
            case 'w':
                if (!paused) paddle.moveUp(step);
                break;
            case 's':
                if (!paused) paddle.moveDown(step);
                break;
            case 'q':
                if (!paused) paddle.rotateLeft();
                break;
            case 'e':
                if (!paused) paddle.rotateRight();
                break;
            case 'r':
                if (!paused) paddle.resetAngle();
                break;
            case 'p':
                paused = !paused;
                break;
            case 'm':
                settings.soundEnabled = !settings.soundEnabled;
                sound.setEnabled(settings.soundEnabled);
                settings.save();
                break;
            case 'x':
                playerExit = true;
                gameOver = true;
                break;
            default:
                break;
        }
    }
}

void update() {
    if (paused) return;

    ball.move();

    if (ball.x <= 0) {
        ball.x = 0;
        if (ball.direction == LEFT_DOWN) ball.direction =
RIGHT_DOWN;
        else if (ball.direction == LEFT_UP) ball.direction = RIGHT_UP;
        wallHits++;
        sound.wallHit();
    }

    if (ball.x >= WIDTH - 1) {

```

```

        ball.x = WIDTH - 1;
        if (ball.direction == RIGHT_DOWN) ball.direction =
LEFT_DOWN;
        else if (ball.direction == RIGHT_UP) ball.direction =
LEFT_UP;
        wallHits++;
        sound.wallHit();
    }

    if (ball.y <= 0) {
        ball.y = 0;
        if (ball.direction == LEFT_UP) ball.direction =
LEFT_DOWN;
        else if (ball.direction == RIGHT_UP) ball.direction =
RIGHT_DOWN;
        wallHits++;
        sound.wallHit();
    }

    if (paddle.contains(ball.x, ball.y)) {
        score++;
        paddleHits++;
        sound.paddleHit();

        if (paddle.angle < 0) {
            ball.direction = LEFT_UP;
        }
        else if (paddle.angle > 0) {
            ball.direction = RIGHT_UP;
        }
        else {
            if (ball.x < paddle.x + paddle.size / 2) {
                ball.direction = LEFT_UP;
            }
            else {
                ball.direction = RIGHT_UP;
            }
        }

        ball.y--;
    }

    if (ball.y > HEIGHT - 1) {
        misses++;
        gameOver = true;
    }
}

void finishGame(const string& result) {
    int duration = static_cast<int>(time(nullptr) - startTime);
    double clicksPerSecond = 0.0;

    if (duration > 0) {
        clicksPerSecond = static_cast<double>(keyClicks) /
duration;
    }

    logger.writeSession(
        sessionId,
        levels[currentLevelIndex].name,
        score,
        paddleHits,
        wallHits,
        misses,
        totalMoves,
        usefulMoves,
        extraMoves,
        keyClicks,
        clicksPerSecond,

        duration,
        result
    );

    if (result == "Loss") {
        sound.lose();
    }

    clearConsoleFull();

    cout <<
"===== " <<
endl;
    cout << "          GAME OVER          " << endl;
    cout <<
"===== " <<
endl;
    cout << "Result: " << result << endl;
    cout << "Level: " << levels[currentLevelIndex].name << endl;
    cout << "Score: " << score << endl;
    cout << "Paddle hits: " << paddleHits << endl;
    cout << "Wall hits: " << wallHits << endl;
    cout << "Misses: " << misses << endl;
    cout << "Total moves: " << totalMoves << endl;
    cout << "Useful moves: " << usefulMoves << endl;
    cout << "Extra moves: " << extraMoves << endl;
    cout << "Keyboard clicks: " << keyClicks << endl;
    cout << "Click speed: " << fixed << setprecision(2) <<
clicksPerSecond << " clicks/sec" << endl;
    cout << "Duration: " << duration << " sec." << endl;
    cout << "Session log saved to game_sessions.csv" << endl;
    cout << "Action log saved to game_actions.csv" << endl;
    pauseMessage();
}

public:
    Game() {
        settings.load();
        sound.setEnabled(settings.soundEnabled);
        currentLevelIndex = settings.lastLevel - 1;
        score = 0;
        paddleHits = 0;
        wallHits = 0;
        misses = 0;
        totalMoves = 0;
        usefulMoves = 0;
        extraMoves = 0;
        keyClicks = 0;
        gameOver = false;
        playerExit = false;
        paused = false;
        sessionId = 0;
    }

    void menu() {
        bool exitProgram = false;

        while (!exitProgram) {
            showMainMenu();

            char choice = _getch();

            switch (choice) {
                case '1':
                    currentLevelIndex = settings.lastLevel - 1;
                    run();
                    break;
                case '2':
                    chooseLevel();

```

```

        currentLevelIndex = settings.lastLevel - 1;
        run();
        break;
    case '3':
        settings.soundEnabled = !settings.soundEnabled;
        sound.setEnabled(settings.soundEnabled);
        settings.save();
        break;
    case '4':
        runAnalytics();
        break;
    case '5':
        exitProgram = true;
        break;
    default:
        break;
    }
}

void run() {
    clearConsoleFull();
    hideCursor();

    sessionId = logger.createSessionId();
    score = 0;
    paddleHits = 0;
    wallHits = 0;
    misses = 0;
    totalMoves = 0;
    usefulMoves = 0;
    extraMoves = 0;
    keyClicks = 0;
    gameOver = false;
    playerExit = false;
    paused = false;

    ball.reset();
    paddle.setSize(levels[currentLevelIndex].paddleSize);
    startTime = time(nullptr);

    while (!gameOver) {
        draw();
        input();
        update();
        Sleep(levels[currentLevelIndex].ballDelay);
    }

    if (playerExit) {
        finishGame("Exit");
    }
    else {
        finishGame("Loss");
    }
}

};

int main() {
    Game game;
    game.menu();
    return 0;
}

```

Analytics.Py

```

import csv
from pathlib import Path
import tkinter as tk
from tkinter import ttk, messagebox

```

```

SESSIONS_FILE = Path("game_sessions.csv")
ACTIONS_FILE = Path("game_actions.csv")

def safe_int(value, default=0):
    try:
        return int(value)
    except Exception:
        return default

def safe_float(value, default=0.0):
    try:
        return float(value)
    except Exception:
        return default

def read_csv(path):
    if not path.exists():
        return []
    with path.open("r", encoding="utf-8", newline="") as file:
        return list(csv.DictReader(file))

class AnalyticsApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Ping-Pong Motor Skills Analytics")
        self.root.geometry("1000x650")

        self.sessions = []
        self.actions = []

        self.create_widgets()
        self.refresh_data()

    def create_widgets(self):
        title = tk.Label(
            self.root,
            text="Ping-Pong Analytics: Fine Motor Skills, Reaction and Accuracy",
            font=("Arial", 16, "bold")
        )
        title.pack(pady=10)

        self.summary = tk.Text(self.root, height=11, font=("Consolas", 11))
        self.summary.pack(fill="x", padx=10)

        button_frame = tk.Frame(self.root)
        button_frame.pack(pady=8)

        tk.Button(button_frame, text="Refresh data",
            command=self.refresh_data).grid(row=0, column=0, padx=5)
        tk.Button(button_frame, text="Build charts",
            command=self.build_charts).grid(row=0, column=1, padx=5)
        tk.Button(button_frame, text="Open sessions table",
            command=self.show_sessions_table).grid(row=0, column=2, padx=5)
        tk.Button(button_frame, text="Open actions table",
            command=self.show_actions_table).grid(row=0, column=3, padx=5)

        self.chart_info = tk.Label(
            self.root,
            text="Charts are saved as PNG files in the same folder as this analytics app.",
            font=("Arial", 10)
        )
        self.chart_info.pack(pady=5)

    def refresh_data(self):
        self.sessions = read_csv(SESSIONS_FILE)
        self.actions = read_csv(ACTIONS_FILE)
        self.summary.delete("1.0", tk.END)

```

```

    if not self.sessions:
        self.summary.insert(tk.END, "No game_sessions.csv
found or file is empty.\n")
        self.summary.insert(tk.END, "Play at least one game
session first.\n")
        return

    scores = [safe_int(row.get("score")) for row in
self.sessions]
    misses = [safe_int(row.get("misses")) for row in
self.sessions]
    extra_moves = [safe_int(row.get("extra_moves")) for row
in self.sessions]
    key_clicks = [safe_int(row.get("key_clicks")) for row in
self.sessions]
    click_speed = [safe_float(row.get("clicks_per_second"))
for row in self.sessions]
    durations = [safe_int(row.get("duration_seconds")) for
row in self.sessions]
    useful_moves = [safe_int(row.get("useful_moves")) for
row in self.sessions]
    total_moves = [safe_int(row.get("total_moves")) for row
in self.sessions]

    total_sessions = len(self.sessions)
    avg_score = sum(scores) / total_sessions
    avg_misses = sum(misses) / total_sessions
    avg_extra = sum(extra_moves) / total_sessions
    avg_click_speed = sum(click_speed) / total_sessions
    avg_duration = sum(durations) / total_sessions

    movement_accuracy = 0
    if sum(total_moves) > 0:
        movement_accuracy = sum(useful_moves) /
sum(total_moves) * 100

    self.summary.insert(tk.END, "GENERAL
STATISTICS\n")
    self.summary.insert(tk.END, "-----
-----\n")
    self.summary.insert(tk.END, f"Total sessions:
{total_sessions}\n")
    self.summary.insert(tk.END, f"Average score:
{avg_score:.2f}\n")
    self.summary.insert(tk.END, f"Average misses:
{avg_misses:.2f}\n")
    self.summary.insert(tk.END, f"Average extra
movements: {avg_extra:.2f}\n")
    self.summary.insert(tk.END, f"Average keyboard click
speed: {avg_click_speed:.2f} clicks/sec\n")
    self.summary.insert(tk.END, f"Average session duration:
{avg_duration:.2f} sec\n")
    self.summary.insert(tk.END, f"Movement accuracy:
{movement_accuracy:.2f}%\n\n")

    self.summary.insert(tk.END, "MOTOR SKILLS
INTERPRETATION\n")
    self.summary.insert(tk.END, "-----
-----\n")

    if total_sessions >= 2 and scores[-1] > scores[0]:
        self.summary.insert(tk.END, "Score increased
compared with the first session.\n")
    else:
        self.summary.insert(tk.END, "Score growth is not
clearly detected yet.\n")

    if total_sessions >= 2 and extra_moves[-1] < extra_moves[0]:
        self.summary.insert(tk.END, "Extra movements decreased. This
may indicate better fine motor control.\n")
    else:
        self.summary.insert(tk.END, "Extra movements are still high or
stable.\n")

    if total_sessions >= 2 and click_speed[-1] > click_speed[0]:
        self.summary.insert(tk.END, "Keyboard click speed increased.
This may indicate faster motor reaction.\n")
    else:
        self.summary.insert(tk.END, "Keyboard click speed did not
increase significantly.\n")

def build_charts(self):
    if not self.sessions:
        messagebox.showerror("Error", "No session data found.")
        return

    try:
        import matplotlib.pyplot as plt
    except Exception:
        messagebox.showerror(
            "Matplotlib is missing",
            "Install matplotlib first:\n\npip install matplotlib"
        )
        return

    x = list(range(1, len(self.sessions) + 1))
    scores = [safe_int(row.get("score")) for row in self.sessions]
    misses = [safe_int(row.get("misses")) for row in self.sessions]
    extra_moves = [safe_int(row.get("extra_moves")) for row in
self.sessions]
    click_speed = [safe_float(row.get("clicks_per_second")) for row in
self.sessions]

    charts = [
        ("score_progress.png", "Score progress", "Session", "Score",
scores),
        ("misses_chart.png", "Misses by session", "Session", "Misses",
misses),
        ("extra_moves_chart.png", "Extra movements by session",
"Session", "Extra movements", extra_moves),
        ("click_speed_chart.png", "Keyboard click speed", "Session",
"Clicks per second", click_speed),
    ]

    for filename, title, xlabel, ylabel, values in charts:
        plt.figure(figsize=(6, 3.5))
        plt.plot(x, values, marker="o")
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(filename, dpi=120, bbox_inches="tight")
        plt.close()

    self.show_charts_window(charts, x)

def show_charts_window(self, charts, x):
    try:
        from matplotlib.figure import Figure
        from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
    except Exception:
        messagebox.showinfo(
            "Charts created",

```

```

        "Charts saved as PNG files, but they cannot be
        shown inside the app."
    )
    return

    win = tk.Toplevel(self.root)
    win.title("Charts viewer")
    win.geometry("760x520")
    win.minsize(620, 430)

    current_chart = tk.IntVar(value=0)

    title_label = tk.Label(win, font=("Arial", 12, "bold"))
    title_label.pack(pady=(10, 2))

    chart_frame = tk.Frame(win)
    chart_frame.pack(fill="both", expand=True, padx=10,
pady=5)

    figure = Figure(figsize=(6.4, 3.6), dpi=100)
    axis = figure.add_subplot(111)
    canvas = FigureCanvasTkAgg(figure,
master=chart_frame)
    canvas.get_tk_widget().pack(fill="both", expand=True)

    nav_frame = tk.Frame(win)
    nav_frame.pack(pady=8)

    counter_label = tk.Label(nav_frame, width=12)
    counter_label.grid(row=0, column=1, padx=10)

    def draw_chart():
        index = current_chart.get()
        filename, title, xlabel, ylabel, values = charts[index]

        axis.clear()
        axis.plot(x, values, marker="o")
        axis.set_title(title)
        axis.set_xlabel(xlabel)
        axis.set_ylabel(ylabel)
        axis.grid(True)
        figure.tight_layout()

        title_label.config(text=title)
        counter_label.config(text=f"{index + 1} /
{len(charts)}")
        canvas.draw()

    def previous_chart():
        current_chart.set((current_chart.get() - 1) %
len(charts))
        draw_chart()

    def next_chart():
        current_chart.set((current_chart.get() + 1) %
len(charts))
        draw_chart()

    tk.Button(nav_frame, text="Previous", width=12,
command=previous_chart).grid(row=0, column=0, padx=5)
    tk.Button(nav_frame, text="Next", width=12,
command=next_chart).grid(row=0, column=2, padx=5)

    hint = tk.Label(
        win,
        text="Use Previous/Next to switch charts. PNG files
are also saved in the app folder.",
        font=("Arial", 9)
    )

```

```

        hint.pack(pady=(0, 8))

    draw_chart()

    def show_sessions_table(self):
        self.show_table("Sessions", self.sessions)

    def show_actions_table(self):
        self.show_table("Actions", self.actions)

    def show_table(self, title, rows):
        if not rows:
            messagebox.showinfo(title, "No data found.")
            return

        win = tk.Toplevel(self.root)
        win.title(title)
        win.geometry("1100x500")

        columns = list(rows[0].keys())

        frame = tk.Frame(win)
        frame.pack(fill="both", expand=True)

        tree = ttk.Treeview(frame, columns=columns, show="headings")
        tree.pack(side="left", fill="both", expand=True)

        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=120)

        for row in rows:
            tree.insert("", tk.END, values=[row.get(col, "") for col in
columns])

        scrollbar = ttk.Scrollbar(frame, orient="vertical",
command=tree.yview)
        tree.configure(yscrollcommand=scrollbar.set)
        scrollbar.pack(side="right", fill="y")

    def main():
        root = tk.Tk()
        AnalyticsApp(root)
        root.mainloop()

if __name__ == "__main__":
    main()

```

