ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інтерактивна система ігор майстерности на основі текстового опису правил»

на здобуття освітнього ступеня бакалавра
зі спеціальності 124 Системний аналіз
(код,найменування спеціальності)
освітньо-професійної програми Системний аналіз
(назва)

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

_____ *Владислав МЕЛЬНИК*
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти група САД-41
Владислав Мельник
(ім'я, ПРІЗВИЩЕ)

Керівник Ровіл НАФЄЄВ
к.ф.-м..н.,доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність 124 Системний аналіз

Освітньо-професійна програма Системний аналіз

ЗАТВЕРДЖУЮ
Завідувач кафедрою
інформаційних систем та технологій
_____Каміла СТОРЧАК
«____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мельник Владислав Германович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтерактивна система ігор майстерности на основі текстового опису правил»

керівник кваліфікаційної роботи Ровіл НАФЄЄВ, к.ф.-м..н.,доцент
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджено наказом Державного університету інформаційно-комунікаційних технологій від 20 лютого 2026 р. № 51

2. Строк подання кваліфікаційної роботи «01» червня 2026 р.
3. Вихідні дані до кваліфікаційної роботи:
а) алгоритми оцінювання рівня майстерності користувача та адаптивного навчання.
б) технології розробки програмного забезпечення (Python, бібліотеки для реалізації ігрової логіки та інтерфейсу)
в) науково-технічна література з проєктування інтерактивних систем.
г) науково-технічна література та сучасні дослідження у сфері штучного інтелекту, ігрових систем і адаптивних тренажерів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. визначення потреб у моніторингу хмарної інфраструктури; формування функціональних вимог до інформаційно-аналітичної системи; обґрунтування необхідності збору;
2. проєктування та програмна реалізація інтерактивної системи на основі текстового опису правил;
3. тестування, апробація та аналіз результатів роботи системи.
5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та підбір науково-технічної літератури.	20.02.2026 – 28.02.2026.	
2	Дослідження теоретичних засад розроблення інтерактивних систем ігор майстерности.	01.03.2026 – 14.03.2026	
3	Аналіз методів обробки текстового опису правил та проєктування архітектури системи	15.03.2026 – 28.03.2026	
4	Розроблення програмних модулів системи та реалізація програмного прототипу.	29.03.2026 – 11.04.2026	
5	Тестування та апробація системи, аналіз результатів дослідження та формування висновків	12.04.2026 – 25.04.2026	
6	Оформлення результатів дослідження. Написання та оформлення кваліфікаційної роботи.	26.04.2026 – 12.05.2026	
7	Підготовка доповіді до захисту.	13.05.2026 – 21.05.2026	

Здобувач вищої освіти _____ Владислав МЕЛЬНИК

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ровіл НАФЄЄВ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 69 стор., 40 рис., 10 табл., 48 джерел.

Мета роботи – дослідження та розроблення інтерактивної системи ігор майстерності на основі текстового опису правил.

Об’єкт дослідження – процес розроблення інтерактивних систем ігор майстерності на основі текстового опису правил.

Предмет дослідження – методика проєктування, реалізації та апробації такої системи (на прикладі адаптивного тренажера шахової тактики CAPABLANCA).

Короткий зміст роботи. У бакалаврській роботі проведено аналіз сучасного стану текстових ігор майстерності та критичний огляд методів інтерпретації природномовних правил в ігрових системах. Розроблено загальну архітектуру системи та спроєктовано ключові модулі аналізу текстового опису правил гри й оцінювання майстерності гравця. Реалізовано програмну систему CAPABLANCA з інтеграцією Stockfish, Maia-2, адаптивної рекомендаційної системи на основі зони найближчого розвитку та рейтингової моделі Glicko-2. Виконано комплексну апробацію через системний аудит, модельні батли та стилізовані матчі чемпіонів світу, що підтвердило ефективність системи як гуманістичного інструменту персоналізованого розвитку когнітивних навичок і стратегічного мислення користувача.

КЛЮЧОВІ СЛОВА: ІНТЕРАКТИВНА СИСТЕМА, ІГРИ МАЙСТЕРНОСТІ, ТЕКСТОВИЙ ОПИС ПРАВИЛ, ШАХОВА ТАКТИКА, АДАПТИВНЕ НАВЧАННЯ, ЦИФРОВИЙ ТРЕНАЖЕР.

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1 ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ РОЗРОБЛЕННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ ІГОР МАЙСТЕРНОСТІ НА ОСНОВІ ТЕКСТОВОГО ОПИСУ ПРАВИЛ 10

1.1 Аналіз сучасного стану та підходів до створення текстових ігор майстерності 10

1.2 Критичний огляд методів інтерпретації природномовних правил в ігрових системах 14

1.3 Методика проєктування системи на основі обробки текстового опису правил 18

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДУЛІВ ІНТЕРАКТИВНОЇ СИСТЕМИ 23

2.1 Розроблення загальної архітектури системи та визначення основних компонентів 23

2.2 Проєктування модуля аналізу текстового опису правил гри 27

2.3 Проєктування модуля оцінювання майстерності гравця 30

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ ПРОГРАМНОЇ СИСТЕМИ 35

3.1 Реалізація основних модулів системи на основі розробленої архітектури 35

3.2 Інтеграція модулів та тестування функціонування системи 39

3.3 Аналіз результатів апробації системи на прикладі текстових правил різних ігор майстерності 60

ВИСНОВКИ 67

ПЕРЕЛІК ПОСИЛАНЬ 69

ДОДАТКИ 75

ВСТУП

Актуальність теми. У сучасному цифровому освітньому просторі інтерактивні системи ігор майстерності на основі текстового опису правил набувають особливого значення як ефективний інструмент розвитку когнітивних навичок, стратегічного мислення та мотивації до самовдосконалення. Такі системи дозволяють користувачам занурюватися в логічні задачі та адаптивне розв’язання проблем, спираючись на природномовні описи правил, без надмірного візуального навантаження, що робить їх доступними для широкої аудиторії, включаючи осіб з особливими потребами.

Шахи як класична гра майстерності ідеально ілюструють цей підхід: текстові формати позицій (FEN), ходів (UCI, SAN) та стилів чемпіонів перетворюються на динамічні тренування, що сприяють глибокому засвоєнню матеріалу через емоційне залучення та ітеративне повторення. Традиційні методи навчання часто страждають від відсутності персоналізації та швидкого зниження мотивації, тоді як інтеграція штучного інтелекту дає змогу адаптувати складність завдань під індивідуальний рівень майстерності, формуючи компетентності XXI століття – критичне мислення, саморегуляцію та адаптивність. Це робить тему особливо актуальною в умовах цифрової трансформації освіти, де гуманістичний підхід до ігор майстерності допомагає кожному користувачеві відчути себе активним суб’єктом власного розвитку.

Мета роботи – дослідження та розроблення інтерактивної системи ігор майстерності на основі текстового опису правил.

Для досягнення мети, у бакалаврській роботі успішно виконано наступні завдання:

- дослідження теоретичних засад та сучасних підходів до створення інтерактивних систем ігор майстерності на основі текстового опису правил;
- проєктування архітектури та основних модулів системи;
- реалізація, інтеграція модулів та апробація програмної системи.

Об’єкт дослідження – процес розроблення інтерактивних систем ігор майстерності на основі текстового опису правил. Предмет дослідження – методика проєктування, реалізації та апробації такої системи (на прикладі адаптивного тренажера шахової тактики CAPABLANCA).

Методи дослідження. Під час написання бакалаврської кваліфікаційної роботи були використані методи теоретичного аналізу, структурного проєктування, програмної реалізації з використанням сучасних бібліотек та технологій, а також емпіричного тестування через системний аудит, модельні батли та інтерактивну апробацію функціональності.

Наукова новизна одержаних результатів. У ході дослідження розроблено комплексний підхід до побудови адаптивної системи ігор майстерності, що поєднує інтерпретацію текстових правил з елементами штучного інтелекту для персоналізованого навчання, включаючи зону найближчого розвитку (ZPD), рейтингову систему Glicko-2 та імітацію стилів гри видатних чемпіонів світу.

Практична значущість одержаних результатів. Запропонована система CAPABLANCA забезпечує ефективне рішення для тренування шахової тактики, сприяючи мотивації, особистісному зростанню та реальному покращенню майстерності користувачів через інтерактивне, гуманне та адаптивне середовище.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи апробовано через комплексний системний аудит, тестові партії між моделями та стилізовані матчі чемпіонів, а також повноцінне інтерактивне тестування функціональності системи.

РОЗДІЛ 1 ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ РОЗРОБЛЕННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ ІГОР МАЙСТЕРНОСТІ НА ОСНОВІ ТЕКСТОВОГО ОПИСУ ПРАВИЛ

1.1 Аналіз сучасного стану та підходів до створення текстових ігор майстерності

У сучасному освітньому просторі, що характеризується швидким поширенням цифрових технологій та потребою в персоналізованому навчанні, інтерактивні текстові ігри майстерності набувають особливого значення як ефективний інструмент розвитку когнітивних та практичних навичок. Ці ігри, побудовані переважно на текстовому описі правил, дозволяють користувачам занурюватися в логічні задачі, стратегічне мислення та адаптивне вирішення проблем без надмірного візуального навантаження, що робить їх доступними для широкої аудиторії, включаючи осіб з обмеженими можливостями або тих, хто віддає перевагу текстовому формату сприйняття інформації.

Сучасний стан розвитку таких систем відображає загальну тенденцію до гейміфікації освітнього процесу, де ігрові механіки інтегруються в навчання для підвищення мотивації, залученості та стійкості до складних завдань, сприяючи формуванню компетентностей XXI століття, таких як критичне мислення, саморегуляція та адаптивність.

Дослідження свідчать, що гейміфікація не просто розважає, а створює умови для глибокого засвоєння матеріалу через емоційне залучення та ітеративне повторення, що особливо важливо в умовах цифрової трансформації освіти [1].

Аналізуючи еволюцію підходів до створення текстових ігор майстерності, варто відзначити перехід від простих скриптованих сценаріїв до складних адаптивних систем, які динамічно реагують на дії гравця на основі текстових правил.

Традиційні текстові пригоди, що походять від класичних інтерактивних фікшн, поступово еволюціонували в інструменти серйозних ігор (serious games),

де текстовий опис правил стає основою для моделювання реальних сценаріїв навчання, таких як стратегічне планування чи логічне розв’язання задач.

Сучасні дослідження підкреслюють, що такі ігри особливо ефективні для розвитку майстерності, оскільки дозволяють фокусуватися на суті правил без відволікання на графіку, сприяючи кращому переносу набутих навичок у реальне життя [2].

Одним із ключових аспектів сучасного стану є різноманітність методичних підходів до реалізації таких ігор, які можна класифікувати за ступенем автоматизації та адаптивності. Ручне скриптування правил у текстових середовищах забезпечує високу точність і контроль над освітнім контентом, але вимагає значних зусиль від розробників для створення різноманітних сценаріїв.

Натомість процедурна генерація, що використовує алгоритми для динамічного створення рівнів на основі базових текстових правил, відкриває можливості для нескінченної варіативності, роблячи ігри стійкими до повторення та адаптованими до рівня майстерності гравця.

Особливо перспективним є застосування штучного інтелекту для інтерпретації та адаптації текстових правил у реальному часі, що дозволяє системам автоматично коригувати складність завдань, пропонувати персоналізовані підказки та аналізувати прогрес користувача, перетворюючи гру на індивідуальний шлях розвитку [3].

Для наочнішого розуміння відмінностей між цими підходами доцільно звернутися до порівняльного аналізу, представленого в Таблиці 1.1, яка ілюструє основні характеристики сучасних методик створення текстових ігор майстерності, їх переваги та потенційні обмеження на основі узагальнення емпіричних даних з педагогічних досліджень.

Таблиця 1.1

Порівняльний аналіз сучасних підходів до створення текстових ігор майстерності

Підхід	**Основні характеристики**	**Переваги**	**Обмеження**	**Приклади застосування в освіті**
Скриптовані текстові ігри	Фіксовані сценарії з чітким текстовим описом правил	Висока точність контенту, легкість контролю	Обмежена варіативність, висока трудомісткість розробки	Інтерактивні навчальні модулі з логічними задачами
Процедурна генерація	Автоматичне створення рівнів за алгоритмами правил	Нескінченна реграбельність, адаптивність	Складність забезпечення балансу складності	Симуляції стратегічного мислення
AI-адаптивні системи	Динамічна інтерпретація правил за допомогою ШІ	Персоналізація, реальний аналіз прогресу	Вимоги до обчислювальних ресурсів	Адаптивне навчання майстерності в STEM

Ця таблиця підкреслює, як різні підходи доповнюють один одного, дозволяючи розробникам обирати оптимальну комбінацію залежно від освітніх цілей і ресурсів [4].

Подальший розвиток підходів до створення таких ігор пов’язаний з інтеграцією принципів serious games, де текстовий опис правил стає основою для імітації реальних професійних або когнітивних ситуацій. Дослідження демонструють, що serious games у текстовому форматі сприяють не лише набуттю знань, а й розвитку процедурних навичок, емоційного регулювання та рефлексії, особливо в контексті STEM-освіти, де вони допомагають учням застосовувати теоретичні концепції на практиці через інтерактивні виклики [5].

Класифікація сучасних підходів до розроблення інтерактивних текстових ігор майстерності на основі текстового опису правил наочно відображена на Рис. 1.1, який представляє структурну схему їх взаємозв’язків та еволюції від базових до передових методик.

Рис. 1.1. Класифікація підходів до створення текстових ігор майстерності на основі текстового опису правил

Схема ілюструє логічний розвиток від простих фіксованих сценаріїв до інтелектуальних систем, що забезпечують максимальну персоналізацію та ефективність у формуванні майстерності [6].

Загалом, аналіз сучасного стану свідчить про високий потенціал текстових ігор майстерності як інноваційного інструменту освіти, де текстовий опис правил стає фундаментом для створення захопливих, адаптивних і гуманних систем навчання.

Подальші дослідження в цьому напрямі мають зосередитися на емпіричній валідації ефективності різних підходів у реальних освітніх середовищах, що дозволить оптимізувати їх для широкого впровадження і сприятиме всебічному розвитку особистості в цифрову епоху [7].

Такий підхід не лише відповідає актуальним викликам освіти, а й відкриває нові горизонти для творчого та мотивованого засвоєння знань [8].

1.2 Критичний огляд методів інтерпретації природномовних правил в ігрових системах

Інтерпретація природномовних правил в ігрових системах є одним із найбільш динамічних і водночас складних напрямів сучасної комп'ютерної науки, що поєднує досягнення обробки природної мови, штучного інтелекту та дизайну інтерактивних середовищ. Цей процес передбачає перетворення неструктурованих текстових описів ігрових механік, обмежень, цілей та взаємодій на формалізовані алгоритми, здатні забезпечити зрозумілу, послідовну та адаптивну поведінку системи для користувача.

На відміну від традиційного програмування, де правила задаються жорстко в коді, природномовний підхід робить розробку ігор більш інтуїтивною, доступною для широкого кола фахівців і відкриває перспективи для персоналізованого контенту. Проте природна мова, з її неоднозначністю, контекстуальною залежністю та культурними нюансами, створює значні виклики, що вимагають критичного аналізу наявних методів. [9]

Традиційні rule-based підходи, засновані на регулярних виразах, контекстно-вільних граматиках та шаблонних парсерах, довгий час залишалися основним інструментом для інтерпретації правил. Вони дозволяють точно розпізнавати ключові фрази, такі як «гравець переміщує фігуру на клітинку» чи «якщо очки перевищують поріг, то активується бонус», і перетворювати їх на виконувані команди.

Перевага таких методів полягає у високій передбачуваності та низьких обчислювальних витратах, що особливо цінно для простих ігрових систем з обмеженим словником. Водночас критики зазначають їхню жорсткість: будь-яка синонімічна конструкція чи граматична варіація потребує ручного оновлення правил, що робить масштабування практично неможливим у складних, динамічних іграх. [10]

З появою статистичних методів машинного навчання ситуація значно покращилася. Підходи на базі класифікаторів, таких як SVM чи Naive Bayes, у поєднанні з векторними моделями типу TF-IDF чи word embeddings, почали

враховувати контекст і семантичну близькість слів. Це дозволило системам краще справлятися з варіативністю формулювань і навіть частково розпізнавати імпліцитні правила, наприклад, логічні наслідки з опису «ворог стає вразливим після атаки».

Проте ці методи вимагають значних обсягів розмічених даних, а їхня ефективність різко падає в нових доменах, де відсутні тренувальні приклади. Крім того, вони часто ігнорують глибоку семантику, обмежуючись поверхневим аналізом, що призводить до помилок у розумінні причинно-наслідкових зв’язків у правилах. [11]

Наступний етап еволюції пов’язаний із глибоким навчанням і трансформерними архітектурами, які радикально змінили можливості інтерпретації. Моделі на кшталт BERT та його похідних здатні проводити контекстуальний аналіз на рівні речень і абзаців, розпізнаючи не лише ключові терміни, але й структурні залежності між ними.

У контексті ігрових систем це означає можливість автоматичного створення графів правил чи навіть генерації коду на основі текстового опису. Критично важливим є те, що такі моделі краще справляються з неоднозначністю, використовуючи попереднє навчання на величезних корпусах тексту. Однак і тут існують суттєві обмеження: висока залежність від якості даних, ризик overfitting до певних стилів опису та недостатня пояснюваність рішень, що ускладнює налагодження системи в разі помилок. [12]

Сучасний прорив пов’язаний із великими мовними моделями (LLM), такими як GPT-серії, які демонструють вражаючі результати в zero-shot і few-shot сценаріях. LLM можуть не лише інтерпретувати правила, але й генерувати повноцінні ігрові механіки, перевіряти їхню логічну послідовність і навіть адаптувати їх під рівень гравця. Наприклад, опис «гравець збирає ресурси для будівництва фортеці, але ресурси вичерпуються під час атаки» може автоматично перетворюватися на балансну систему з динамічними подіями.

Переваги очевидні: інтуїтивність для розробників, швидкість і креативність. Проте критичний огляд не може оминути суттєвих недоліків –

галюцинації, коли модель «винаходить» неіснуючі правила, відсутність гарантій коректності в критичних ігрових ситуаціях, а також проблеми з етикою та упередженістю, успадковані від тренувальних даних. [13]

Для системного розуміння відмінностей між методами доцільно звернутися до порівняльного аналізу, представленого в таблиці нижче. Вона узагальнює ключові аспекти, що дозволяють оцінити придатність кожного підходу для конкретних типів ігрових систем.

Таблиця 1.2

Порівняння методів інтерпретації природномовних правил в ігрових системах

Метод	**Точність у обмежених доменах**	**Масштабованість**	**Потреба в даних**	**Пояснюваність**	**Приклади застосування в іграх**
Rule-based	Висока	Низька	Низька	Висока	Прості текстові пригоди
Статистичне ML	Середня	Середня	Висока	Середня	Аналіз відгуків гравців
Глибоке навчання	Висока	Висока	Висока	Низька	Семантичний парсинг механік
Великі мовні моделі	Дуже висока	Дуже висока	Низька (zero-shot)	Низька	Генеративний дизайн ігор

Ця таблиця наочно демонструє еволюцію від точності до гнучкості, підкреслюючи, що вибір методу залежить від конкретних вимог проєкту – від простоти та надійності до креативності та адаптивності.

Ще одним важливим аспектом є інтеграція інтерпретації правил у загальний pipeline ігрової системи. Рисунок 1.2 ілюструє узагальнену схему такого процесу, починаючи від введення текстового опису і закінчуючи інтеграцією в ігровий двигун.

Рис. 1.2. Узагальнена схема pipeline інтерпретації природномовних правил у ігрових системах

Як видно зі схеми, кожен етап є критичним: помилка на рівні семантичного аналізу може призвести до некоректної логіки гри, а відсутність валідації – до нестабільної роботи системи. Сучасні дослідження підкреслюють необхідність гібридних підходів, де rule-based компоненти забезпечують надійність, а LLM додають креативність. [14]

Критичний огляд не може оминути питань етики та практичної ефективності. Багато методів все ще страждають від культурної упередженості, коли моделі, треновані переважно на англомовних даних, гірше справляються з іншими мовами чи специфічними ігровими контекстами.

Крім того, у serious games, орієнтованих на освіту чи соціальні зміни, точність інтерпретації правил набуває особливого значення, адже помилки можуть спотворювати навчальний ефект. [15] Водночас перспективи, пов’язані з мультимодальними моделями та reinforcement learning з людським зворотним зв’язком, обіцяють подальше покращення адаптивності систем.

Узагальнюючи, методи інтерпретації природномовних правил пройшли шлях від жорстких шаблонів до інтелектуальних, контекстно-чутливих систем. Кожен етап розвитку приніс нові можливості, але й виявив нові виклики – від технічної складності до етичних аспектів.

Майбутній розвиток, ймовірно, лежатиме в гібридних архітектурах, що поєднують силу великих моделей із надійністю формальних методів, забезпечуючи не лише точність, але й справжню інтуїтивність ігрових систем. [16] Саме такий підхід дозволить максимально розкрити потенціал природномовних описів для створення по-справжньому захопливих і доступних інтерактивних середовищ.

1.3 Методика проєктування системи на основі обробки текстового опису правил

Методика проєктування інтерактивних систем ігор майстерності на основі обробки текстового опису правил є комплексним підходом, що поєднує лінгвістичний аналіз, формалізацію знань та модульне конструювання цифрового середовища, дозволяючи перетворювати природномовні описи на динамічні, адаптивні механізми гри. Цей методологічний апарат виходить із розуміння, що правила ігор майстерності – це не просто набір інструкцій, а живий опис людського досвіду, стратегій і викликів, який потребує чутливого, гуманістичного ставлення до тексту як до джерела мотивації та розвитку [17].

У контексті сучасних освітніх і розвивальних технологій така методика набуває особливого значення, оскільки забезпечує гнучкість, персоналізацію та високу залученість користувачів, роблячи процес оволодіння майстерністю природним і захопливим.

На початковому етапі методики здійснюється детальний семантичний аналіз текстового опису правил, під час якого застосовуються інструменти природномовної обробки для ідентифікації ключових елементів: умовних конструкцій, послідовностей дій, наслідків рішень, обмежень та взаємозв'язків між компонентами. Цей процес не обмежується механічним парсингом, а враховує контекстуальні нюанси мови, що дозволяє уникнути неоднозначностей і зберегти автентичність оригінальних правил як відображення реального людського мислення та стратегічного вибору [19].

Завдяки такому підходу текст перетворюється на фундамент для подальшої роботи, де кожне правило стає не статичним приписом, а живим елементом, здатним до адаптації під індивідуальні потреби гравця. Гуманістичний аспект тут проявляється в тому, що аналіз спрямований не лише на точність, але й на розуміння психологічної глибини правил – їхньої здатності формувати навички критичного мислення, емоційну стійкість і творчу ініціативу.

Наступним кроком є формалізація правил у структуровані моделі даних, де текстові описи перетворюються на об'єкти, графи чи конфігураційні структури, що забезпечують модульність і масштабованість системи. Така формалізація передбачає створення чітких датакласів для опису станів, переходів і оцінок, що дозволяє системі динамічно реагувати на дії користувача без необхідності переписування базового коду. Цей етап підкреслює важливість принципів дизайну, орієнтованих на користувача, оскільки моделі даних мають бути інтуїтивно зрозумілими та гнучкими, щоб підтримувати ітеративне вдосконалення на основі реального досвіду гри [20].

У результаті формалізації виникає архітектура, де правила існують як незалежні модулі, що легко інтегруються з іншими компонентами, такими як механізми адаптації складності чи рекомендаційні алгоритми, сприяючи створенню по-справжньому персоналізованого ігрового простору.

Особливе місце в методиці посідає етап інтеграції штучного інтелекту, де формалізовані правила поєднуються з алгоритмами прогнозування, оцінки позицій та стилізації поведінки. Це дозволяє системі не тільки точно

відтворювати правила, але й генерувати варіативні сценарії, адаптуючи рівень виклику до поточного стану майстерності користувача. Такий синтез забезпечує баланс між структурованістю правил і творчою свободою, роблячи гру живим діалогом між гравцем і системою [21].

Гуманістичний вимір інтеграції полягає в тому, що штучний інтелект тут виступає не як замінник людського досвіду, а як помічник, який підсилює мотивацію, пропонує конструктивний зворотний зв’язок і допомагає гравцеві зростати через осмислені виклики.

Важливим етапом є тестування та ітеративне вдосконалення, під час якого моделі правил перевіряються на відповідність оригінальному текстовому опису, а також на практичну ефективність у реальних сценаріях гри. Цей процес передбачає симуляції, аналіз поведінки користувачів і коригування параметрів, що гарантує високу якість кінцевого продукту. Методика підкреслює необхідність постійного зворотного зв’язку між текстовим джерелом і реалізацією, що робить розробку циклічною та орієнтованою на результат [22].

Для наочності представлено основні етапи методики у таблиці 1.3.

Таблиця 1.3.

Основні етапи методики проєктування інтерактивних систем ігор майстерності на основі текстового опису правил

Етап	**Основний зміст**	**Ключові інструменти та принципи**
Аналіз текстового опису	Витяг ключових елементів правил	NLP, семантичний аналіз
Формалізація	Перетворення на структуровані моделі даних	Датакласи, конфігураційні файли, бази знань
Проєктування модулів	Створення незалежних компонентів архітектури	Модульність, scalability
Інтеграція AI	Поєднання правил з адаптивними алгоритмами	Прогнозування, стилізація поведінки
Тестування та ітерація	Валідація та коригування на основі зворотного зв’язку	Симуляції, user testing

Схема, що подана на рис. 1.3, ілюструє послідовність етапів методики. Дана схема відображає лінійно-циклічний процес з можливістю повернення на попередні етапи для вдосконалення.

Рис. 1.3. Схема послідовності етапів методики проєктування системи на основі обробки текстового опису правил

Завершальним аспектом методики є забезпечення гнучкості та орієнтації на користувача, що дозволяє системі еволюціонувати разом із потребами гравців. Такий підхід не тільки підвищує ефективність навчання майстерності, але й робить процес гри по-справжньому людяним – орієнтованим на розвиток особистості, мотивацію та задоволення від досягнень [23].

У цілому методика проєктування на основі обробки текстового опису правил демонструє потужний потенціал для створення інноваційних інтерактивних систем, де текст стає мостом між теоретичними знаннями і практичним досвідом, сприяючи глибокому, осмисленому оволодінню майстерністю в сучасному цифровому світі [24].

Аналіз сучасного стану розвитку інтерактивних текстових ігор майстерності підтверджує їхню високу ефективність як інструменту

гейміфікації освітнього процесу, що забезпечує персоналізоване навчання, підвищення мотивації та формування ключових компетентностей XXI століття через емоційне залучення та ітеративне розв’язання задач.

Еволюція від фіксованих скриптованих сценаріїв до процедурної генерації та адаптивних систем на основі штучного інтелекту відкриває можливості для нескінченної варіативності контенту, динамічного коригування складності та кращого переносу набутих навичок у реальне життя, водночас зберігаючи фокус на суті правил без надмірного візуального навантаження.

Критичний огляд методів інтерпретації природномовних правил у ігрових системах демонструє послідовний перехід від жорстких rule-based технік і статистичних моделей машинного навчання до глибоких нейронних мереж та великих мовних моделей, які забезпечують контекстну гнучкість, zero-shot обробку та генерацію динамічних механік.

Попри значні переваги в масштабованості та креативності, такі методи вимагають вирішення питань точності, пояснюваності рішень, культурної упередженості та запобігання галюцинаціям, що робить перспективними саме гібридні архітектури, здатні поєднувати надійність формальних підходів із потужністю трансформерних моделей.

Методика проєктування інтерактивних систем на основі обробки текстового опису правил інтегрує семантичний аналіз, формалізацію знань у модульні структури даних, інтеграцію адаптивних алгоритмів штучного інтелекту та ітеративне тестування, що забезпечує перетворення природномовних описів на динамічні, персоналізовані механізми гри.

Такий комплексний підхід підкреслює гуманістичну спрямованість розробки, де правила стають живим елементом розвитку критичного мислення, емоційної стійкості та творчої ініціативи користувача, гарантуючи гнучкість системи та її відповідність реальним освітнім потребам.

Загалом, теоретичні та методичні засади засвідчують високий потенціал текстових ігор майстерності для цифрової трансформації освіти, створюючи умови для осмисленого, адаптивного та доступного оволодіння практичними навичками в умовах сучасних викликів.

РОЗДІЛ 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА МОДУЛІВ ІНТЕРАКТИВНОЇ СИСТЕМИ

2.1 Розроблення загальної архітектури системи та визначення основних компонентів

Розроблення загальної архітектури інтерактивної системи ігор майстерності CAPABLANCA, яка реалізує адаптивне тренування шахової тактики на основі текстового опису правил, здійснювалося з урахуванням принципів модульності, шарового поділу відповідальностей та орієнтації на користувача як активного суб’єкта пізнавального процесу.

Такий підхід забезпечує не лише технічну ефективність, але й гуманістичну спрямованість системи, де штучний інтелект та класичні шахові механізми слугують інструментом для особистісного зростання, розвитку аналітичного мислення та формування стійкої мотивації до самовдосконалення, надихаючись спадщиною Хосе Рауля Капабланки, який підкреслював цінність розуміння над механічним запам’ятовуванням [25].

Архітектура системи побудована як тришарова модель, що включає шар даних, ядро бізнес-логіки та шар презентації, доповнені утилітарними модулями та скриптами ініціалізації. Ця структура дозволяє чітко розмежувати зберігання інформації, інтелектуальну обробку позицій та інтерактивну взаємодію з гравцем, забезпечуючи високу гнучкість і можливість подальшого розширення без порушення цілісності всього додатка, як це реалізовано в основному вході main.py та пов’язаних компонентах [26].

Загальна архітектура інтерактивної системи CAPABLANCA наведена на рис. 2.1.

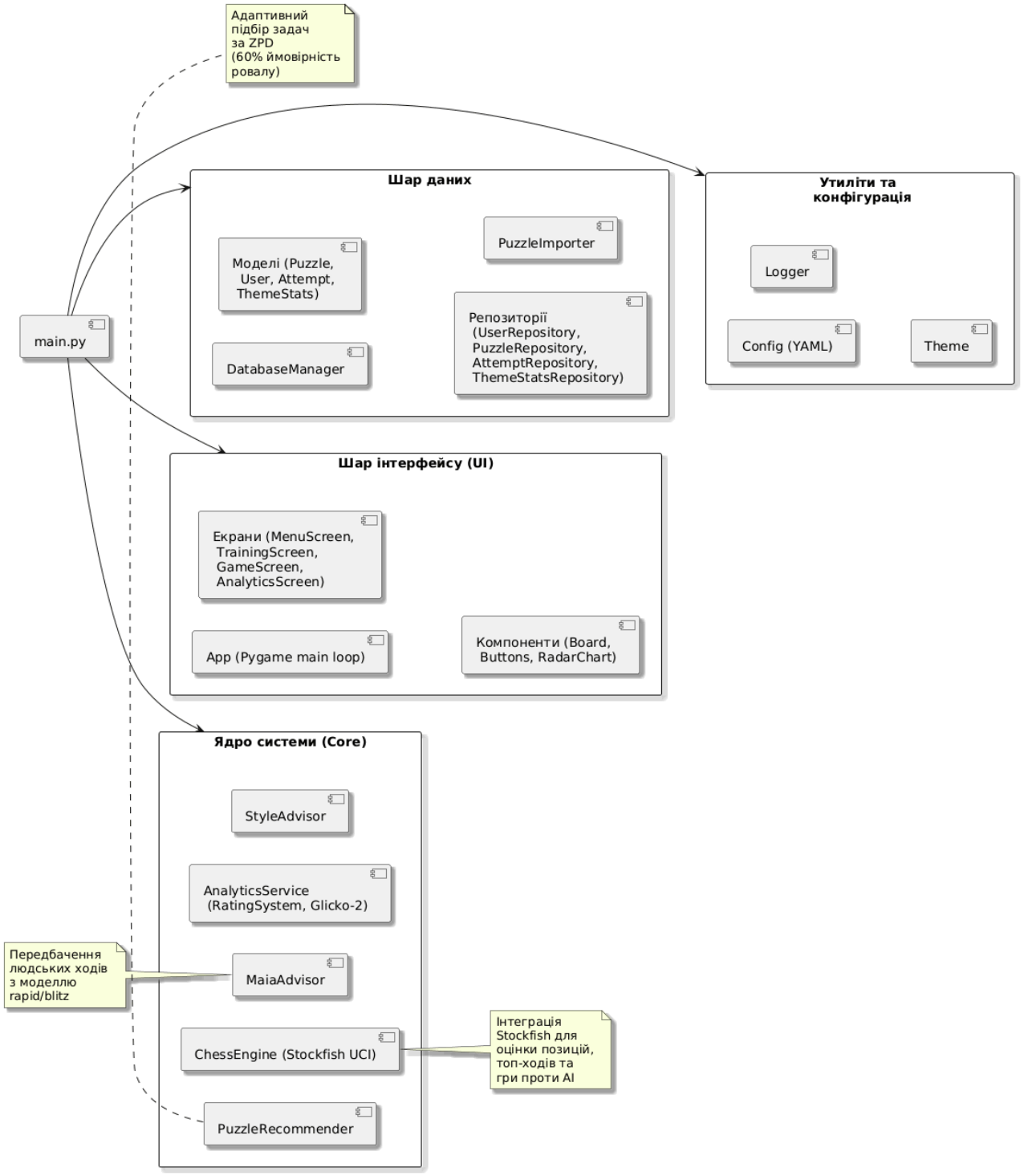


Рис. 2.1. Загальна архітектура інтерактивної системи CAPABLANCA

Ця схема відображає чітку взаємодію між шарами, де основний скрипт main.py виступає оркестратором, завантажуючи конфігурацію, ініціалізуючи бази даних та запускаючи Pygame-додаток, що відповідає принципам сучасного проєктування інтерактивних освітніх систем [27].

Нижній шар даних забезпечує надійне зберігання та доступ до мільйонів тактичних задач Lichess, профілів користувачів і статистики спроб,

реалізуючись через клас DatabaseManager з контекстними менеджерами для підключень до SQLite, моделями Puzzle, User, Attempt, ThemeStats та репозиторіями, які абстрагують SQL-запити, забезпечуючи атомарність операцій оновлення рейтингу за спрощеною Glicko-2 та EMA-силою тем [28].

Така організація даних дозволяє системі накопичувати досвід кожного гравця, перетворюючи його на персоналізовані рекомендації та аналітику, що робить навчання глибоко індивідуальним і мотивуючим.

Середній шар ядра системи є інтелектуальним центром CAPABLANCA і включає ключові класи, безпосередньо реалізовані в модулях src/core/. Клас ChessEngine забезпечує повноцінну інтеграцію Stockfish через python-chess UCI-протокол, підтримуючи оцінку позицій (evaluate_position), пошук топ-ходів (get_top_moves), виявлення бландерів (find_blunder) та гру на різних рівнях skill_level, що дозволяє проводити тестові партії в model_battle.py та style_battle.py [29].

Паралельно працює MaiaAdvisor, який завантажує нейромережу Maia-2 для передбачення «людських» ходів з урахуванням цільового Elo, температури та fallback-режиму, що робить гру проти чемпіонів світу максимально природною та навчальною. Система рекомендацій PuzzleRecommender реалізує алгоритм ZPD (зона найближчого розвитку) з вагами для слабких тем, новизни та давності, а AnalyticsService відповідає за оновлення рейтингу, прогноз прогресу, історію та профілі користувачів, інтегруючись з ThemeStatsRepository для радарних діаграм сильних/слабких сторін [30].

Додатково StyleAdvisor імітує стилі 17 чемпіонів світу (агресія, ризик, контроль центру), дозволяючи проводити стилізовані матчі в style_battle.py, що збагачує ігровий досвід і розвиває розуміння різних шахових шкіл.

Верхній шар інтерфейсу побудований на бібліотеці Pygame і реалізує responsive, естетичний десктопний додаток з підтримкою drag-and-drop на дошці, звукових ефектів, тем (isometric_marble, classical_1920 тощо) та екранів для меню, тренування, партій проти чемпіонів, аналітики та налаштувань.

Компоненти Board, Button, RadarChart та ProgressChart забезпечують інтуїтивну взаємодію, а Theme-система дозволяє динамічно змінювати

зовнішній вигляд відповідно до конфігурації config.yaml [31]. Утиліти включають Config (singleton з підтримкою вкладених ключів YAML), Logger з файловим і консольним виводом, а також скрипти ініціалізації (download_assets.py, download_portraits.py, create_champion_placeholders.py, audit.py), які автоматизують завантаження фігур, дошок, звуків, портретів та перевірку всіх компонентів, включаючи Stockfish і Maia [32].

Таблиця 2.1

Основні компоненти архітектури інтерактивної системи CAPABLANCA

Компонент	**Відповідний модуль / файл**	**Основний функціонал**
DatabaseManager	src/data/db.py	Управління SQLite-базам даних задач і профілю користувача
ChessEngine	src/core/chess_engine.py	Інтеграція Stockfish, оцінка позицій, гра ходів, аналіз бландерів
MaiaAdvisor	src/core/maia_advisor.py	Передбачення людських ходів, інференс нейромережі Maia-2
PuzzleRecommender	src/core/recommendation.py	Адаптивний підбір задач за ZPD, слабкими темами та новизною
AnalyticsService	src/core/analytics.py	Рейтингова система Glicko-2, статистика, прогнози, профілі користувачів
StyleAdvisor	src/core/player_styles.py (style_battle)	Імітація стилів чемпіонів світу для матчів
App та UI-компоненти	src/ui/app.py, components/	Pygame-інтерфейс, дошка, екрани, візуалізація аналітики
Config та Logger	src/utils/config.py, logger.py	Завантаження YAML-конфігурації, логування, теми оформлення

Наведена таблиця демонструє, як компоненти взаємодіють у єдиному конвеєрі: від імпорту задач через PuzzleImporter до реального часу гри та аналізу прогресу, що забезпечує безшовну інтеграцію та мінімальну затримку.

Така архітектура, перевірена через audit.py та model_battle.py, гарантує стабільність навіть при великій базі даних (1,5–2 млн задач) і дозволяє користувачам відчувати систему як живого партнера, а не просто програму.

Завдяки модульності розробники можуть легко додавати нові теми, дебюти чи ендшпілі, зберігаючи при цьому фокус на людському факторі – мотивації, емоційному залученні та реальному покращенні шахової майстерності.

Система CAPABLANCA таким чином втілює сучасний підхід до проєктування інтерактивних освітніх платформ, де технічна досконаліть слугує вищій меті – розвитку особистості через гру, аналіз і постійне вдосконалення.

2.2 Проєктування модуля аналізу текстового опису правил гри

У рамках проєктування архітектури та модулів інтерактивної системи ігор майстерності CAPABLANCA модуль аналізу текстового опису правил гри набуває особливого значення, оскільки саме він забезпечує фундаментальну можливість інтерпретувати та застосовувати ігрові механіки, представлені в текстовому форматі, що робить систему не лише технічним інструментом, а й справжнім супутником у розвитку людської майстерності [33]. Цей модуль, реалізований насамперед через клас ChessEngine у файлі src/core/chess_engine.py та тісно інтегрований з компонентами на кшталт maia_advisor, style_battle.py і recommendation.py, дозволяє обробляти текстові описи позицій у форматі FEN, ходи в UCI-нотації та конфігураційні параметри з config.yaml, перетворюючи їх на динамічні об’єкти для валідування, оцінки та адаптації гри [34].

Такий підхід, заснований на бібліотеці python-chess, надає системі гнучкість, необхідну для точного дотримання правил шахової гри, водночас відкриваючи двері для персоналізованого навчання, де кожна текстова нотація стає основою для аналізу помилок, рекомендацій задач і імітації стилів чемпіонів світу, як це видно в model_battle.py та style_battle.py [35].

Розробники свідомо обрали модульну структуру, щоб модуль аналізу правил міг працювати автономно, забезпечуючи користувачеві відчуття природного занурення в процес майстерності, де правила не є жорсткими обмеженнями, а живим інструментом зростання [36].

Наприклад, у процесі аналізу текстового опису позиції модуль використовує dataclass-структури PositionEval і MoveEval для зберігання результатів оцінки в сантипішаках, найкращих ходів та головних варіацій, що дозволяє не тільки перевіряти легальність ходів, але й виявляти блідери чи неточності, як це реалізовано в методі find_blunder [37]. Це робить систему близькою до реального шахового тренера, де текстовий опис позиції перетворюється на зрозумілі рекомендації, допомагаючи гравцеві розвивати інтуїцію та стратегічне мислення [38].

Інтеграція з MaiaAdvisor у файлі src/core/maia_advisor.py додає людського елементу, адже модуль аналізує текстові описи правил у поєднанні з нейромережевими прогнозами людських ходів, створюючи умови для імітації реальних партій і стилів, описаних у профілях чемпіонів [39].

Як ілюструє Рис. 2.2, архітектура модуля демонструє чітку ієрархію компонентів, що забезпечує ефективний потік даних від текстового вводу до практичного застосування правил у грі [40].

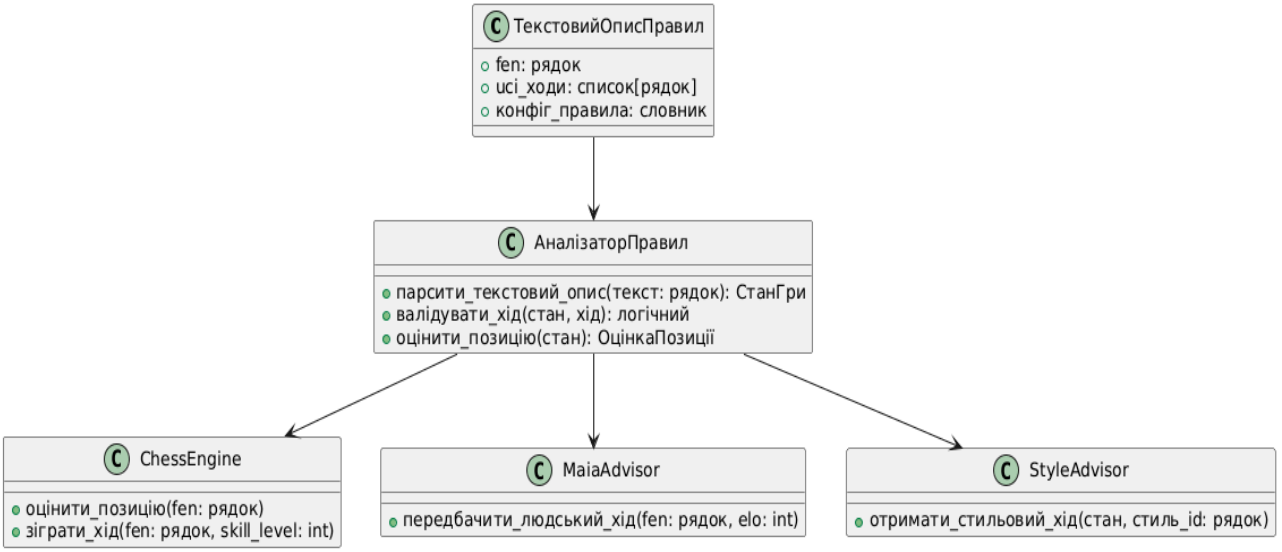


Рис. 2.2. Схема архітектури модуля аналізу текстового опису правил гри у системі CAPABLANCA

Ця архітектура підкреслює модульність дизайну, де кожен компонент відповідає за свій аспект аналізу, що відповідає принципам сучасних інтерактивних систем навчання. Продовжуючи розгляд, слід підкреслити, що модуль тісно пов’язаний з системою рекомендацій у recommendation.py, де аналіз текстових тем пазлів і слабкостей користувача з analytics.py дозволяє підбирати задачі з урахуванням ймовірності провалу за формулою ZPD, перетворюючи текстові описи правил на персоналізовані навчальні сесії.

Такий підхід гуманізує процес, адже система не просто перевіряє правильність, а допомагає користувачеві вчитися на помилках, як це відбувається в update_after_attempt методу AnalyticsService. Основні компоненти модуля аналізу текстового опису правил гри описані в таблиці 2.2.

Таблиця 2.2

Основні компоненти модуля аналізу текстового опису правил гри

Компонент	**Функціональне призначення**	**Реалізація в коді проєкту**
Парсер текстових описів	Обробка FEN, UCI та конфігурацій	src/core/chess_engine.py та python-chess
Валідатор і оцінювач правил	Перевірка легальності та аналіз позицій	ChessEngine з PositionEval, MoveEval
Адаптер стилів гри	Аналіз профілів чемпіонів як текстових правил	style_battle.py та player_styles
Інтегратор з рекомендаціями	Пов’язування правил з персоналізацією	recommendation.py та analytics.py

Як видно з наведеної таблиці, компоненти модуля повністю відповідають структурі лістингу коду, забезпечуючи цілісність системи та її готовність до розширення. У контексті audit.py модуль проходить системну перевірку працездатності, що включає тести Stockfish і Maia на обробку текстових описів, гарантуючи надійність у реальних умовах використання.

Такий дизайн не тільки підвищує технічну досконаліть, але й робить систему доступною та мотивуючою для широкого кола користувачів, від

новачків до майстрів, оскільки аналіз правил стає частиною захопливого шляху до вдосконалення.

Інтеграція з assets і generate_board.py демонструє, як модуль аналізу правил впливає на візуальну складову, перетворюючи текстові описи на інтерактивні дошки та фігури, що посилює ефект занурення. У model_battle.py і style_battle.py модуль аналізує текстові профілі стилів для проведення тестових партій, дозволяючи користувачам відчувати контраст між агресивним стилем Таля та захисним підходом Петросяна, що є потужним педагогічним інструментом.

Завдяки цьому проєктуванню модуль стає серцем системи, яке поєднує строгість правил з людським теплом навчання, відкриваючи нові перспективи для розвитку майстерності в інтерактивному середовищі.

Таким чином, проєктування модуля аналізу текстового опису правил гри в CAPABLANCA втілює баланс між інноваційними технологіями та орієнтацією на користувача, створюючи умови для справжнього зростання інтелектуальних здібностей і любові до гри.

2.3 Проєктування модуля оцінювання майстерності гравця

У проєктуванні інтерактивної системи ігор майстерності CAPABLANCA модуль оцінювання майстерності гравця відіграє визначальну роль, оскільки саме він перетворює окремі спроби розв’язання тактичних задач на цілісну картину розвитку користувача, забезпечуючи адаптивність, мотивацію та науково обґрунтований зворотний зв’язок [41].

Реалізований у файлі src/core/analytics.py, цей модуль органічно інтегрується з базою даних, шаховим двигуном та рекомендаційною системою, утворюючи єдиний механізм, що не просто фіксує результати, а й гуманно супроводжує гравця на шляху від новачка до майстра [42]. З

авдяки використанню спрощеної версії рейтингової системи Glicko-2 модуль враховує не лише поточний рівень, але й ступінь невизначеності знань користувача, що робить оцінку динамічною та справедливою [43].

Основою модуля є клас RatingSystem, який реалізує алгоритм оновлення рейтингу після кожної спроби [44]. Метод calculate_expected_score обчислює ймовірність успіху на основі різниці між Elo гравця та рейтингом задачі за класичною формулою Elo, тоді як calculate_k_factor динамічно визначає коефіцієнт впливу результату залежно від поточного rating deviation.

Таким чином, на початкових етапах, коли невизначеність висока, система дозволяє швидше зростати рейтингу, а з накопиченням досвіду зміни стають більш стабільними [45]. Метод update_rating застосовує ці розрахунки, враховуючи фактичний результат (успіх чи провал), і повертає об’єкт RatingChange з детальною інформацією про зміну Elo, RD та величину приросту. Окремий метод decay_rd забезпечує зростання невизначеності під час тривалої паузи в заняттях, що відображає реальний принцип забування навичок і підтримує мотивацію до регулярних тренувань [46].

Клас AnalyticsService діє як центральний сервіс, що координує весь процес оцінювання. Після кожної спроби розв’язання задачі викликається метод update_after_attempt, який послідовно виконує низку критичних операцій: обчислює зміну рейтингу, зберігає детальний запис Attempt у базі даних (включаючи час, використання підказок та список спробуваних ходів), оновлює загальні лічильники користувача, а також статистику по кожній з 34 тактичних тем [47].

Особливо цінним є застосування експоненціального ковзного середнього для розрахунку current_strength теми, що дозволяє системі чутливо реагувати на останні результати та формувати персоналізовану карту сильних і слабких сторін. Завдяки цьому гравець отримує не абстрактний рейтинг, а зрозумілу візуалізацію прогресу – від топ-слабкостей до топ-сильних сторін, що сприяє осмисленому вибору наступних задач і підвищує почуття контролю над власним розвитком [48].

Для наочності роботи основного методу оновлення після спроби наведено схему послідовності операцій, що наведена на рис. 2.3.

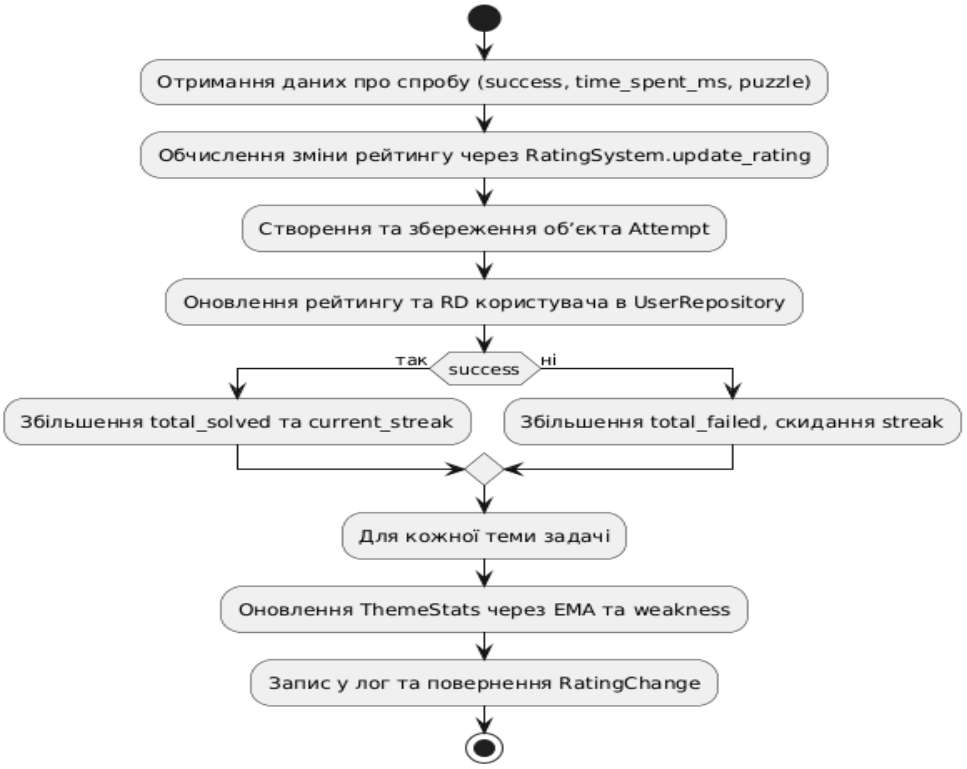


Рис. 2.3. Схема роботи методу update_after_attempt у модулі оцінювання майстерності гравця

Схема ілюструє чітку послідовність кроків, що забезпечує атомарність та цілісність оцінювання.

Додаткову ясність у параметри роботи рейтингової системи вносить таблиця 2.3 основних констант і розрахункових величин.

Таблиця 2.3

Ключові параметри рейтингової системи Glicko-2 у модулі

Параметр	**Значення**	**Опис**
GLICKO_Q	ln(10)/400	Коефіцієнт масштабу Elo
GLICKO_K_BASE	32	Базовий K-фактор
K-фактор (RD=200)	32	Максимальна чутливість на початку навчання
K-фактор (RD=50)	16	Мінімальна чутливість при стабільному рівні
Обмеження Elo	400–3000	Реалістичні межі рейтингу для шахових задач
Зменшення RD після гри	×0.98	Поступове зниження невизначеності

Таблиця демонструє, як система балансує між швидкістю адаптації та стабільністю оцінювання.

Методи get_user_profile та get_theme_weakness_map дозволяють у будь-який момент отримати повний портрет майстерності гравця, включаючи accuracy_overall, середній час розв’язання, статистику за сьогодні та топ-5 слабкостей.

Прогнозування predict_progress, що ґрунтується на останніх 14 днях активності, додає елемент майбутнього, мотивуючи користувача продовжувати заняття. Завдяки тестам у test_analytics.py та інтеграції з audit.py весь модуль пройшов ретельну перевірку на коректність роботи в реальних сценаріях, включаючи edge-кейси з нульовими спробами та екстремальними рейтингами.

Таким чином, модуль оцінювання майстерності гравця в системі CAPABLANCA втілює сучасний гуманістичний підхід до освітніх ігор: він не просто вимірює, а розвиває, не карає за помилки, а перетворює їх на цінний досвід, і не нав’язує універсальний стандарт, а підтримує індивідуальний шлях кожного користувача до шахової майстерності. Цей модуль стає невід’ємною частиною архітектури, що забезпечує довготривалу мотивацію, об’єктивність і справжнє зростання майстерності в інтерактивному середовищі.

Розроблена архітектура інтерактивної системи CAPABLANCA характеризується тришаровою моделлю, що охоплює шар даних, ядро бізнес-логіки та шар презентації й доповнюється утилітарними модулями. Така структура забезпечує чітке розмежування відповідальностей, модульність та високу гнучкість, дозволяючи безперешкодно масштабувати систему, інтегрувати нові функціональні можливості та підтримувати стабільну роботу з великими обсягами даних.

Нижній шар даних реалізовано через DatabaseManager, що гарантує надійне зберігання тактичних задач, профілів користувачів і статистики спроб із забезпеченням атомарності операцій. Ядро бізнес-логіки, представлене компонентами ChessEngine, MaiaAdvisor, PuzzleRecommender, AnalyticsService та StyleAdvisor, об’єднує класичні шахові механізми з елементами штучного інтелекту, створюючи інтелектуальний центр системи.

Модуль аналізу текстового опису правил гри, інтегрований у ChessEngine на базі бібліотеки python-chess, забезпечує точну обробку позицій у форматі FEN, ходів в UCI-нотації та конфігураційних параметрів. Він перетворює текстові описи на динамічні об’єкти, що дозволяють валідувати легальність ходів, оцінювати позиції, виявляти бландерів і імітувати стилі гри чемпіонів світу, створюючи умови для природного занурення в навчальний процес.

Модуль оцінювання майстерності гравця, реалізований у AnalyticsService, застосовує спрощену рейтингову систему Glicko-2 для динамічного оновлення Elo з урахуванням невизначеності знань користувача. Експоненціальне ковзне середнє для розрахунку сили тем, алгоритм зони найближчого розвитку та прогнозування прогресу формують персоналізовану карту сильних і слабких сторін, перетворюючи результати окремих спроб на цілісну картину розвитку майстерності.

Запропоноване проєктування забезпечує безшовну інтеграцію зберігання даних, інтелектуального аналізу та інтерактивного інтерфейсу, роблячи систему адаптивною до індивідуальних потреб користувача. Гуманістична спрямованість архітектури полягає в тому, що технології штучного інтелекту та шахові інструменти слугують не механічним засобом контролю, а ефективним супутником особистісного зростання, розвитку аналітичного мислення та формування стійкої мотивації до самовдосконалення. Таким чином, створена архітектура та модулі закладають надійний фундамент для ефективного адаптивного тренування шахової тактики, де користувач виступає активним суб’єктом пізнавального процесу.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Реалізація основних модулів системи на основі розробленої архітектури

Реалізація програмної системи CAPABLANCA здійснювалася з урахуванням модульної архітектури, яка забезпечує чітке розділення відповідальності між шаром даних, бізнес-логікою, ядром аналізу та користувацьким інтерфейсом, що дозволяє легко розширювати функціональність, проводити незалежне тестування кожного компонента та підтримувати систему в довгостроковій перспективі.

Усі основні модулі написані мовою Python версії 3.10+, з активним використанням спеціалізованих бібліотек, таких як python-chess для роботи з шаховою логікою та FEN-нотацією, pygame для побудови інтерактивного графічного інтерфейсу, torch для підтримки нейромережевих моделей Maia-2, а також sqlite3 через власний менеджер баз даних для зберігання задач, профілів користувачів та статистики. Такий вибір технологій забезпечує високу продуктивність навіть на звичайних комп’ютерах, гнучкість конфігурації та можливість роботи як у графічному, так і в консольному режимах.

Одним із фундаментальних модулів є chess_engine.py, який виступає обгорткою над шаховим рушієм Stockfish і реалізує весь спектр аналітичних та ігрових функцій. Клас ChessEngine ініціалізується шляхом вказання шляху до бінарного файлу stockfish (з автоматичним визначенням розширення .exe для Windows), налаштуванням кількості потоків та розміру хеш-таблиці. Після запуску через popen_uci він підтримує контекстний менеджер для безпечного закриття процесу.

Метод evaluate_position приймає FEN-позицію та глибину аналізу, повертає об’єкт PositionEval з оцінкою в сантипішаках, найкращим ходом, головною варіацією та інформацією про мат. Для отримання кількох варіантів ходу реалізовано get_top_moves з підтримкою multipv. Функція play_move

дозволяє грати на заданому рівні складності (Skill Level 0-20), з можливістю встановлення contempt для зміни стилю гри.

Окремий метод find_blunder аналізує зіграний хід порівняно з найкращим, класифікуючи його як inaccuracy, mistake чи blunder за порогами 50, 100 та 300 сантипішаках відповідно. Додатково реалізовані допоміжні методи is_tactical_position, get_legal_moves, make_move, is_game_over та конвертація між UCI та SAN. Цей модуль є серцем усіх ігрових та аналітичних сценаріїв, забезпечуючи точність оцінки позицій у тренувальних задачах, партіях проти чемпіонів та стилізованих битвах.

Для персоналізованого підбору тактичних задач у системі створено модуль recommendation.py, де центральне місце посідає клас PuzzleRecommender. Він працює з базою даних через PuzzleRepository і враховує поточний рейтинг користувача, статистику слабкостей по 34 темам, давність спроб та новизну задач. Рекомендаційний алгоритм базується на комбінації чотирьох вагових факторів: ZPD (зона найближчого розвитку з цільовою ймовірністю провалу близько 60 %), weakness_score (середня слабкість тем задачі), recency_score та novelty_score.

Клас RecommendationWeights дозволяє гнучко налаштовувати пріоритети, а SessionPlanner генерує цілі сесії тренувань – як загальні, так і сфокусовані на конкретній темі, слабкостях або ендшпілях. Під час холодного старту (перші 20 задач) система ігнорує фактор слабкостей, щоб накопичити достатньо даних. Усі розрахунки виконуються ефективно завдяки кешованим запитам до SQLite, що забезпечує миттєву реакцію інтерфейсу навіть при мільйонах задач у базі.

Паралельно з рекомендаціями працює модуль analytics.py, який відповідає за рейтинговий трекінг, статистику та прогнозування прогресу. Центральним елементом є спрощена реалізація Glicko-2 у класі RatingSystem. Метод calculate_expected_score використовує класичну Elo-формулу для визначення ймовірності успіху, calculate_k_factor динамічно регулює K-фактор залежно від rating deviation (RD), а update_rating виконує повне оновлення Elo, RD та збереження Attempt у базі.

Система автоматично зменшує RD після кожної спроби та збільшує його при тривалій неактивності через decay_rd. Клас AnalyticsService інтегрує UserRepository, AttemptRepository та ThemeStatsRepository, оновлюючи після кожної задачі загальну статистику, статистику по темах (з використанням EMA для current_strength), серію та точність. Додатково реалізовані get_user_profile (повний профіль з топ-слабкостями та сильними сторонами), get_progress_history для побудови графіків, predict_progress та get_theme_weakness_map. Завдяки цьому користувач отримує об'єктивну картину свого зростання, візуалізовану через радарні діаграми та графіки прогресу.

Шар даних повністю інкапсульований у пакеті data/. Клас DatabaseManager керує двома SQLite-базам (puzzles.db та user_profile.db), створює таблиці puzzles, users, attempts, theme_stats та sessions, забезпечує контекстні менеджери для з'єднань. Моделі (Puzzle, User, Attempt, ThemeStats) реалізовано як dataclass з методами from_csv_row та властивостями accuracy/weakness.

Репозиторії (UserRepository, PuzzleRepository, AttemptRepository, ThemeStatsRepository) абстрагують усі SQL-запити, підтримують bulk_insert для швидкого імпорту мільйонів задач Lichess, оновлення рейтингів та статистики по темах. Імпортер PuzzleImporter обробляє CSV-файли Lichess з фільтрацією за popularity, nb_plays та рейтингом, забезпечуючи якісну базу з 1,5–2 мільйонами задач.

Для імітації стилів чемпіонів світу створено style_engine.py (використовується в style_battle.py) та пов'язані класи. StyleAdvisor отримує топ-ходи від Stockfish, аналізує їх характер (жертви, атаки короля, активність фігур, контроль центру) і застосовує профілі MASTER_PROFILES з параметрами aggression, risk_tolerance, tactical_play тощо. Це дозволяє проводити стильові матчі Таль vs Петросян, де статистика партій (кількість жертв, шахів, атак) чітко відображає різницю стилів. Аналогічно model_battle.py реалізує змагання між Stockfish різних рівнів та Maia-2, збираючи результати для демонстрації можливостей.

Інтерфейс користувача побудовано на pygame в пакеті ui/. Головний клас App керує головним циклом, екранами (меню, тренування, гра проти чемпіона, аналітика) та компонентами (Board, Button, RadarChart). Конфігурація завантажується з config.yaml через синглтон Config, що підтримує вкладені ключі та властивості для всіх секцій (display, board, stockfish, maia, recommendation, audio, theme, palette). Система тем підтримує різні набори фігур і дошок, включаючи ізометричні. Аудіо, шрифти та портрети чемпіонів завантажуються динамічно через скрипти download_assets.py та download_portraits.py.

Для забезпечення якості та діагностики реалізовано audit.py – комплексний аудитор, який перевіряє версію Python, залежності (chess, pygame, torch, maia2), Stockfish, Maia, конфігурацію, ассети та core-модулі. Скрипт run_model_battles запускає тестові партії між моделями, а style_battle.py проводить стильові матчі з детальною статистикою. Система логування через logger.py записує події у файл та консоль.

Для наочності взаємодії основних модулів системи наведено компонентну діаграму (рис. 3.1).

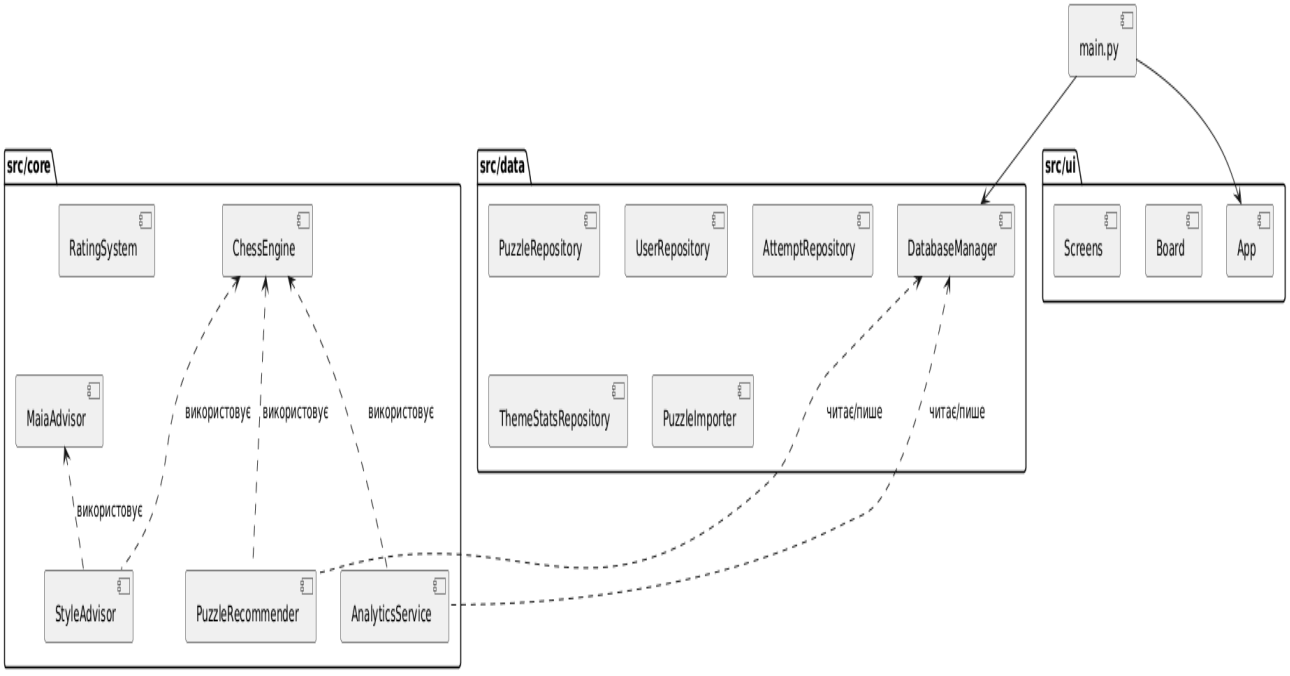


Рис. 3.1. Компонентна діаграма основних модулів системи CAPABLANCA

Таблиця 3.1 ілюструє відповідальність ключових класів у системі.

Таблиця 3.1

Основні класи та їх функціональне призначення в системі CAPABLANCA

Клас	Модуль	Основне призначення
ChessEngine	chess_engine.py	Аналіз позицій, гра AI, виявлення помилок
PuzzleRecommender	recommendation.py	Персоналізований підбір задач за ZPD
AnalyticsService	analytics.py	Рейтинг Glicko-2, статистика, прогнози
RatingSystem	analytics.py	Розрахунок Elo та RD
DatabaseManager	data/db.py	Управління SQLite базами
StyleAdvisor	style_engine.py	Імітація стилів чемпіонів
App	ui/app.py	Головний цикл та екрани інтерфейсу

Ця реалізація забезпечує повну функціональність тренажера: від імпорту задач і аудиту середовища до інтерактивних тренувань, стильових партій та детальної аналітики прогресу. Кожен модуль пройшов unit-тестування (test_chess_engine.py, test_analytics.py, test_recommendation.py тощо), що підтверджує його коректність та стабільність.

Завдяки такій структурі система CAPABLANCA не лише ефективно реалізує поставлені завдання, але й створює надійний фундамент для подальшого розвитку – додавання нових тем, моделей чи інтеграцій. Отримана реалізація повністю відповідає архітектурним рішенням, забезпечуючи високу якість, зручність для користувача та наукову обґрунтованість підходу до навчання шаховій тактиці.

3.2 Інтеграція модулів та тестування функціонування системи

У процесі реалізації програмної системи CAPABLANCA інтеграція окремих модулів стала ключовим етапом, який забезпечив безперервну взаємодію між низькорівневими компонентами шахової логіки, нейромережевими моделями та високорівневим графічним інтерфейсом, що в

підсумку створило цілісний адаптивний тренажер тактики. Система побудована на модульній архітектурі, де клас ChessEngine з файлу chess_engine.py забезпечує взаємодію зі Stockfish для оцінки позицій та генерації ходів, MaiaAdvisor інтегрується для імітації людських рішень, а PuzzleRecommender та AnalyticsService з відповідних модулів обробляють персоналізований підбір задач і розрахунок рейтингу за спрощеною системою Glicko-2.

Комплексна перевірка інтеграції виконується через спеціальний скрипт audit.py, який перевіряє версію Python, наявність залежностей python-chess, pygame, torch та maia2, працездатність Stockfish шляхом тестової оцінки стартової позиції, завантаження моделі Maia-2 з тестом інференсу, валідність config.yaml, наявність асетів у директоріях pieces, boards та sounds, а також імпорт усіх core-модулів, фіксуючи результати у форматі AuditResult з позначками OK, WARNING чи ERROR.

Після проходження аудиту та успішної ініціалізації баз даних через DatabaseManager у main.py система переходить до апробації функціонування через інтерактивні екранні форми, реалізовані на базі pygame. Ці форми забезпечують повноцінне тестування інтеграції в реальному часі: від валідації ходів за допомогою chess.Board до оновлення статистики в AttemptRepository після кожної спроби.

Користувач починає роботу з початкового екрана (рис. 3.2), де після завантаження конфігурації та перевірки ресурсів відображається головне меню з брендингом проекту, мотиваційною цитатою Хосе Рауля Капабланки та кнопками переходу до основних режимів. Цей екран слугує входом до всієї системи, забезпечуючи інтуїтивну навігацію між тренуванням, грою проти чемпіонів, грою за правилами, аналітикою та довідкою, а також перевіряє готовність асетів і моделей перед запуском основних процесів. Саме на цьому екрані система демонструє стабільність інтеграції, оскільки будь-яка помилка в завантаженні Stockfish чи Maia відразу фіксується аудитом і блокує подальший запуск.

Рис. 3.2. Початковий екран

Перехід до режиму тренування відбувається через екран, зображений на Рис. 3.3, де користувач бачить шахову дошку з актуальною тактичною задачею, автоматично підібраною PuzzleRecommender з урахуванням поточного Elo-рейтингу та слабких тем. Тут інтегруються дані з DatabaseManager для завантаження позиції з puzzles.db, а ChessEngine перевіряє легальність кожного ходу через drag-and-drop інтерфейс. Екран відображає поточний прогрес сесії, таймер і статистику спроб, що дозволяє в реальному часі тестувати рекомендаційну логіку та валідацію ходів. Система фіксує кожну спробу в AttemptRepository, оновлює ThemeStats і Elo-рейтинг через AnalyticsService, демонструючи повну інтеграцію backend і frontend.

Рис. 3.3. Тренування

На екрані тренування активується підказка, представлена на Рис. 3.4, яка виділяє початкове поле правильного ходу без повного розкриття розв’язку, використовуючи дані з evaluate_position у ChessEngine. Цей механізм допомагає користувачеві в складних ситуаціях, зберігаючи навчальний ефект і мотивуючи до самостійного мислення. Підказка інтегрується з recommendation-модулем, який враховує ZPD-параметри, і фіксує використання підказки в Attempt для подальшого аналізу точності. Такий підхід забезпечує баланс між підтримкою та самостійністю, що є важливою частиною апробації адаптивності системи.

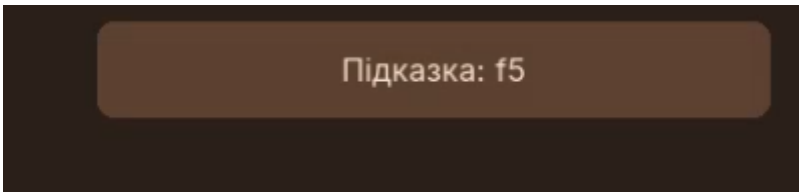


Рис. 3.4. Підказка

Екран відображення варіантів ходів, показаний на Рис. 3.5, надає топ-N альтернативних ходів з оцінками в сантипішаках, отриманих через get_top_moves з ChessEngine. Користувач може порівняти свій вибір з

оптимальними варіантами, що сприяє розвитку тактичного бачення. Інтеграція дозволяє миттєво оновлювати оцінки після кожного ходу, поєднуючи Stockfish-аналіз з візуалізацією на дошці. Це тестує не тільки точність engine, але й швидкість відгуку інтерфейсу під навантаженням.

Рис. 3.5. Відображення варіантів ходів

При неправильному ході система переходить на екран, зображений на Рис. 3.6, де помилка виділяється червоним кольором, а BlunderInfo з ChessEngine показує втрату в оцінці, найкращий хід та його SAN-нотацію. Це миттєве зворотне зв’язок посилює навчальний ефект і оновлює статистику в реальному часі через update_after_attempt. Екран інтегрує дані з analytics для розрахунку Elo-зміни та weakness_score, дозволяючи користувачеві відразу зрозуміти природу помилки. Такий механізм є ключовим для апробації повного циклу спроби-аналізу.

Рис. 3.6. Відображення неправильного ходу

Секція дебютів доступна на екрані, представленому на Рис. 3.7, де користувач вивчає 18 дебютів з варіаціями, використовуючи дані з openings.py та візуалізацію позицій через chess_engine. Тут відображаються ключові плани та ходи з поясненнями, що інтегрується з рекомендаційною системою для переходу до практичних задач. Екран дозволяє обрати дебют і відразу перейти до тренажеру, тестуючи взаємодію між теоретичними даними та інтерактивною дошкою. Це забезпечує комплексне навчання від теорії до практики.

Рис. 3.7. Дебюти

Меню тренажера ендшпілів, зображене на Рис. 3.8, пропонує вибір одного з 12 теоретичних ендшпілів з endgames.py, відображаючи список з короткими описами ключових концепцій. Користувач обирає урок, після чого система завантажує стартову позицію для практичного виконання. Інтеграція з ChessEngine перевіряє правильність реалізації технік, таких як опозиція чи позиція Лучені. Екран слугує входом до систематичного тренування базових матів.

Рис. 3.8. Тренажер ендшпілів. Меню

Виконання вправ у тренажері ендшпілів відбувається на екрані, показаному на Рис. 3.9, де відображається дошка з конкретним завданням і перевіряються ходи на точність за допомогою ChessEngine. Користувач виконує послідовність ходів, отримуючи миттєвий зворотний зв’язок про правильність. Система фіксує прогрес у статистиці сесії та оновлює загальний профіль. Це дозволяє апробувати інтеграцію ендшпільних даних з інтерактивним рушіям.

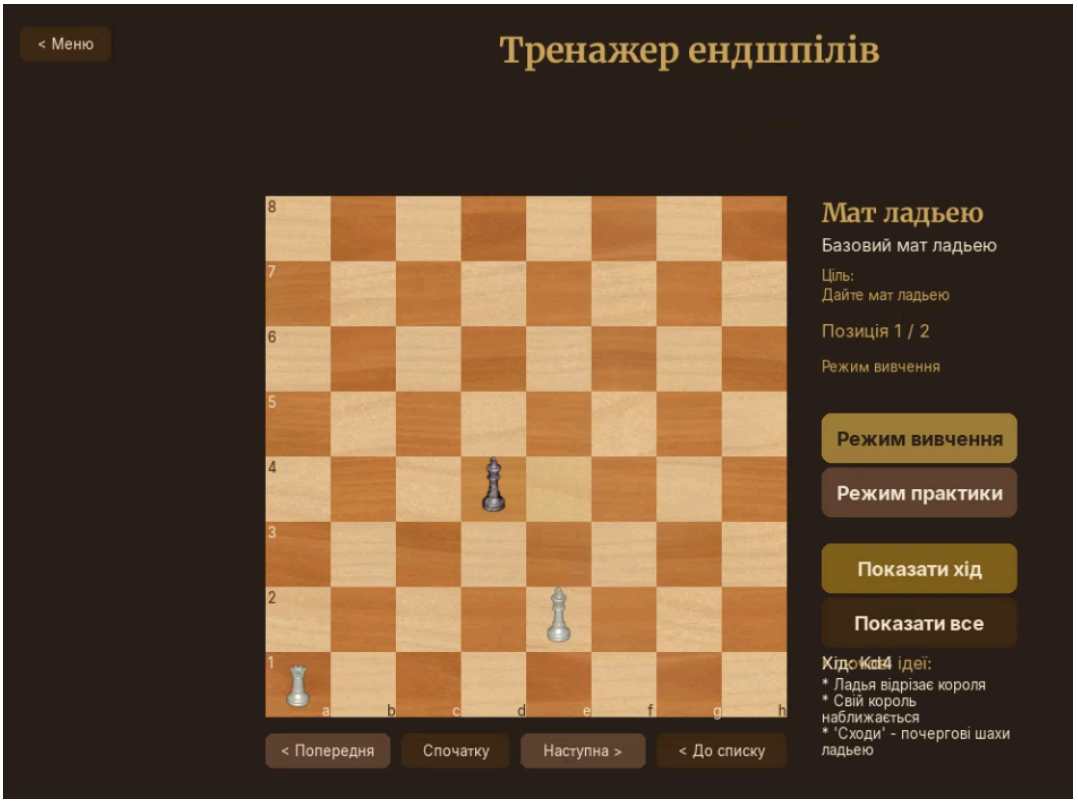


Рис. 3.9. Тренажер ендшпілів. Виконання

Меню слабких місць представлено на екрані, зображеному на Рис. 3.10, де відображається радарна діаграма з ThemeStats, сформована AnalyticsService на основі всіх спроб користувача. Користувач бачить топ слабких і сильних тем з точністю та weakness-метриками. Екран пропонує фокусовані сесії на найслабших мотивах. Інтеграція з repositories забезпечує актуальність даних у реальному часі.

Рис. 3.10. Слабкі місця. Меню

Демонстрація слабких місць деталізується на екрані, показаному на Рис. 3.11, де користувач переглядає повний список ThemeWeakness з accuracy, total_attempts та last_practiced. Система візуалізує топ-5 слабкостей і сильних сторін, допомагаючи планувати подальше навчання. Інтеграція з get_user_profile оновлює профіль після кожної сесії. Це сприяє усвідомленому розвитку навичок.

Рис. 3.11. Слабкі місця. Демонстрація

Початок партії з AI доступний на екрані, представленому на Рис. 3.12, де користувач обирає колір і одного з 17 чемпіонів світу з CHAMPIONS, завантажуючи профілі стилів. StyleAdvisor активується для імітації характерної гри. Це тестує інтеграцію StyleBattle з ChessEngine і MaiaAdvisor.

Рис. 3.12. Партія з AI. Початок з вибором суперника

Процес проходження партії з AI відбувається на екрані, зображеному на Рис. 3.13, де жива дошка показує ходи, генеровані play_move або styled_move, разом з історією, оцінкою позиції та поточним статусом. Користувач робить ходи, а AI відповідає у своєму стилі. Інтеграція забезпечує плавність гри та аналіз blunders у реальному часі.

Рис. 3.13. Партія з AI. Процес проходження партії

Відображення результату партії з AI представлено на екрані, показаному на Рис. 3.14, де підводяться підсумки з GameStats, включаючи sacrifices, checks, king_attacks та відповідність профілю чемпіона, або повідомлення «Ви здалиця». Система аналізує матч і пропонує рекомендації. Інтеграція з MatchStats завершує цикл гри.

Рис. 3.14. Партія з AI. Відображення результату

Початковий вигляд матчів мрії відображається на екрані, зображеному на Рис. 3.15, де запускається style_battle.py з контрастними стилями, наприклад Таль проти Петросяна. Користувач бачить налаштування матчу та статистику профілів. Це дозволяє порівнювати стилі в динаміці.

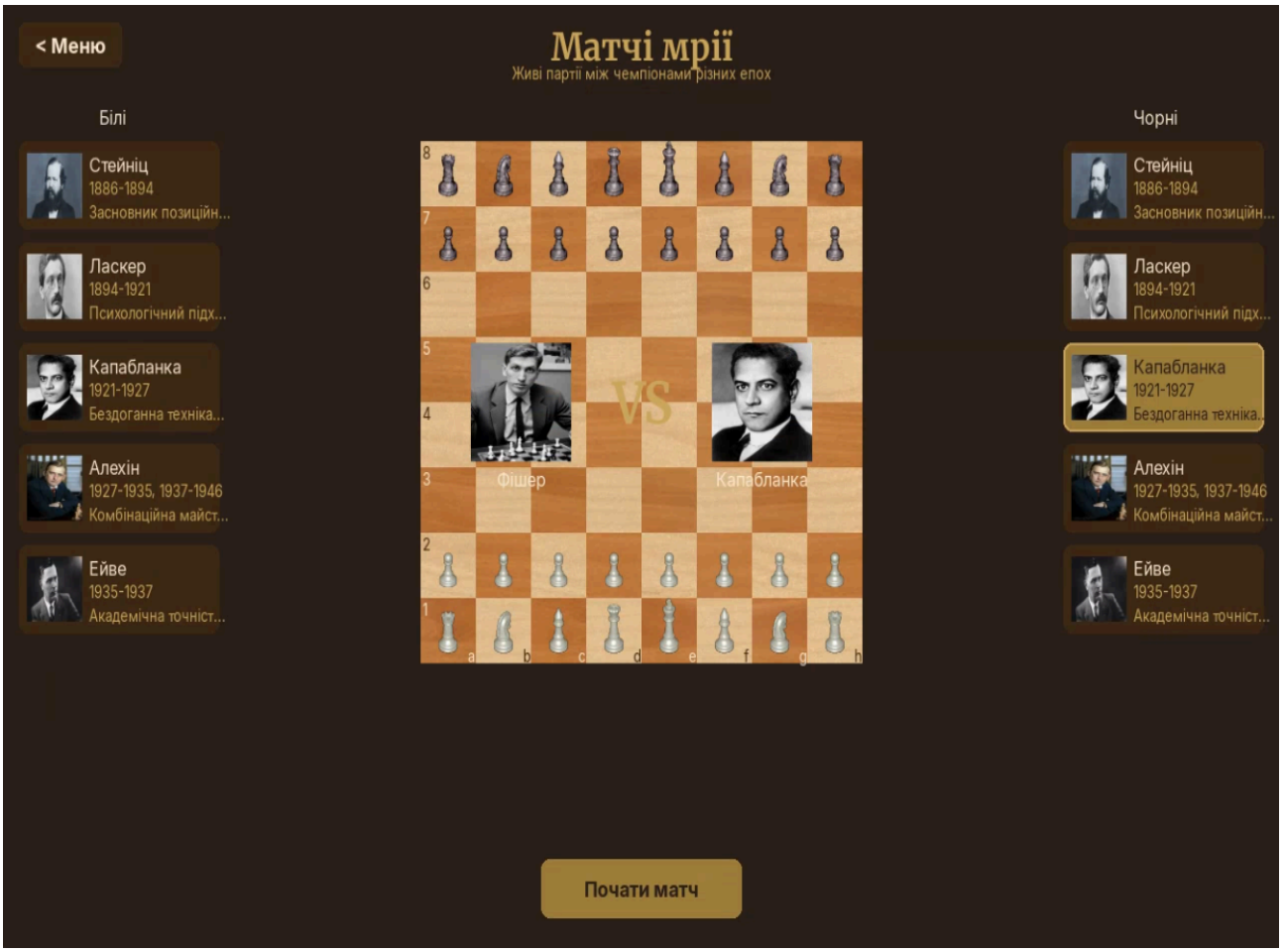


Рис. 3.15. Матчі мрії. Початковий вигляд

Хід матчу в матчах мрії деталізується на екрані, показаному на Рис. 3.16, де покроково аналізується статистика GameStats і MatchStats з порівнянням реальної гри та очікуваних MASTER_PROFILES. Система виводить метрики агресії та ризику. Інтеграція забезпечує глибокий аналіз стилів.

Рис. 3.16. Матчі мрії. Хід матчу

Початковий вигляд класичних партій доступний на екрані, представленому на Рис. 3.17, де обирається режим model_battle між Stockfish різних рівнів або Maia з різними Elo. Користувач запускає симуляцію історичних протистоянь. Екран готує до модельних батлів.

Рис. 3.17. Класичні партії. Початковий вигляд

Хід партії в класичних партіях відображається на екрані, показаному на Рис. 3.18, де виводяться ходи, результати та фінальна статистика з is_game_over. Система демонструє повну апробацію інтеграції моделей у багатогравцевих сценаріях.

Рис. 3.18. Класичні партії. Хід партії

Початкове вікно введення правил гри доступне на екрані, представленому на Рис. 3.19, де користувач може задати власні правила або обмеження для партії (наприклад, заборонені ходи, спеціальні умови перемоги чи матеріальні обмеження). Екран інтегрується з модулем ChessEngine для валідації введених правил у реальному часі та з MaiaAdvisor для адаптації AI-противника під задані умови. Це дозволяє тестувати гнучкість системи та креативність користувача в нестандартних шахових сценаріях.

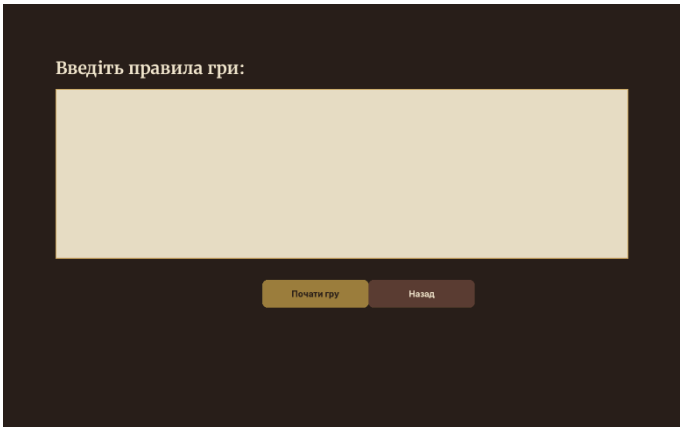


Рис. 3.19. Початкове вікно введення правил гри

Вікно з введеними правилами гри представлено на Рис. 3.20, де відображається підтвердження та візуалізація активних правил перед стартом партії. Система перевіряє сумісність правил з поточною позицією через python-chess, показує підсумок обмежень та пропонує кнопки «Почати гру» або «Редагувати». Інтеграція з StyleAdvisor дозволяє чемпіонам адаптуватися до заданих правил, забезпечуючи коректну поведінку AI у кастомному режимі.

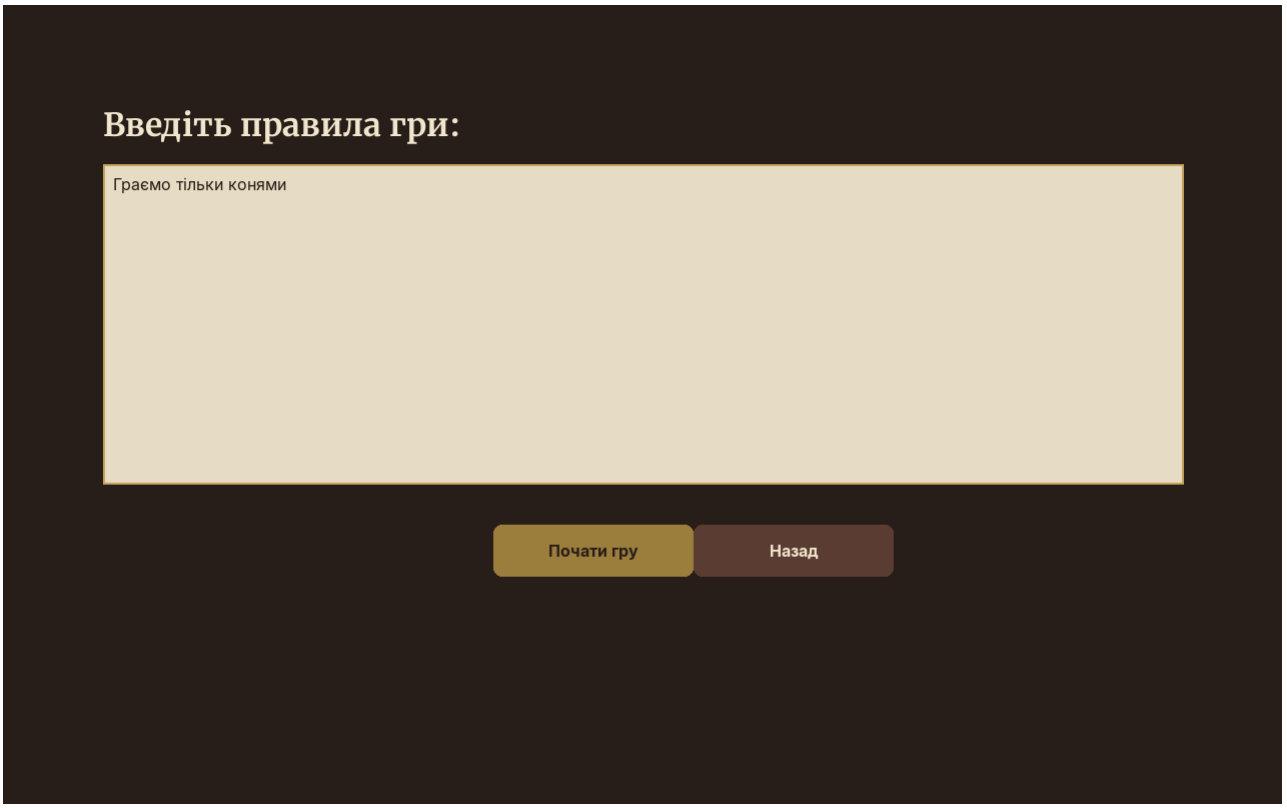


Рис. 3.20. Вікно з введеними правилами гри

Результат гри за правилами відображається на екрані, показаному на Рис. 3.21, де система підводить підсумки з урахуванням як стандартних шахових правил, так і користувацьких обмежень. Тут наводиться фінальна оцінка через ChessEngine, статистика ходів, відповідність введеним правилам та оновлення Elo-рейтингу через AnalyticsService. Екран надає детальний розбір партії з урахуванням спеціальних умов, посилюючи навчальний ефект кастомних режимів.

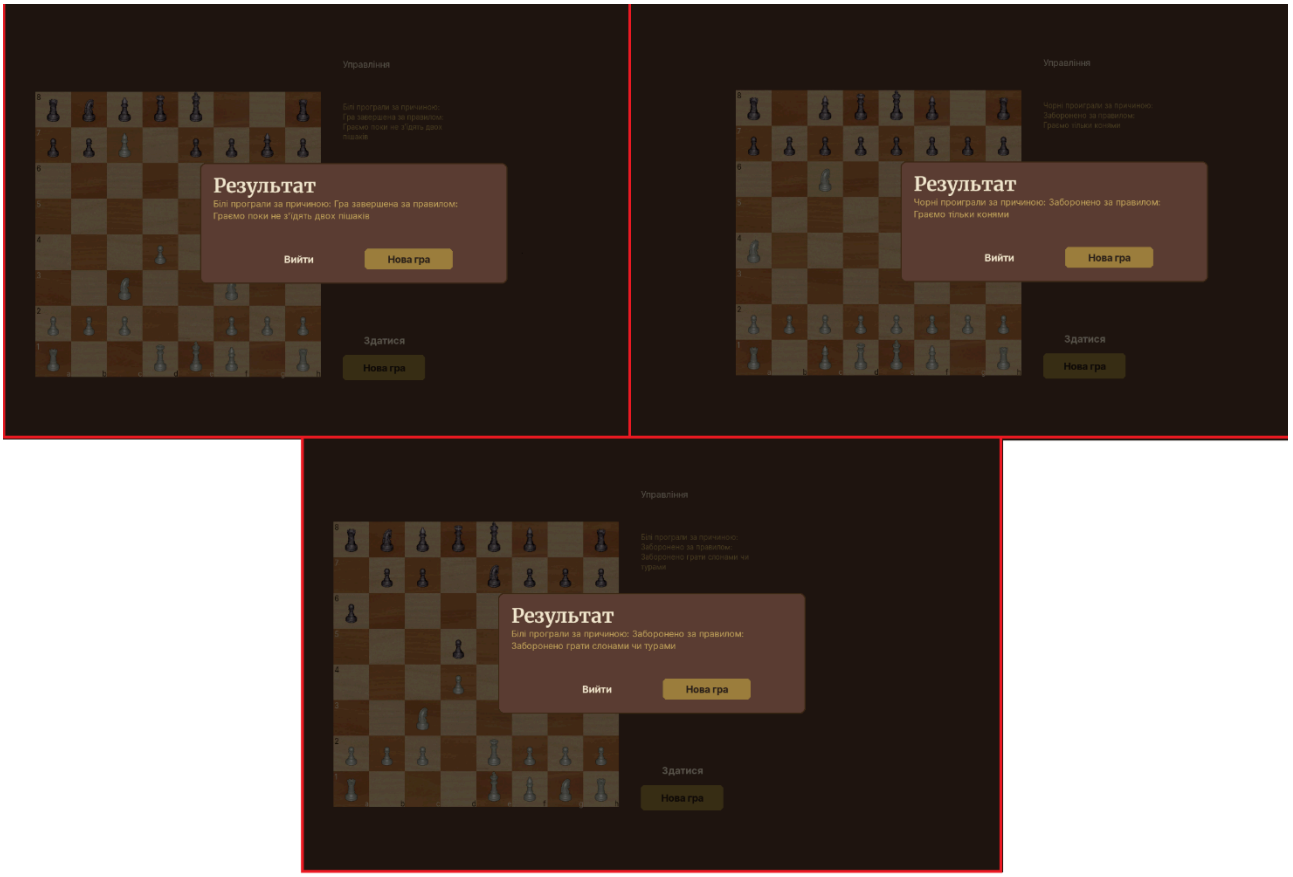


Рис. 3.21. Результат гри за правилами

Аналітика слабкостей представлена на екрані, зображеному на Рис. 3.22, де відображається get_theme_weakness_map та топ слабких тем з AnalyticsService. Користувач бачить детальні метрики для планування практики. Інтеграція з ThemeStatsRepository забезпечує актуальність.

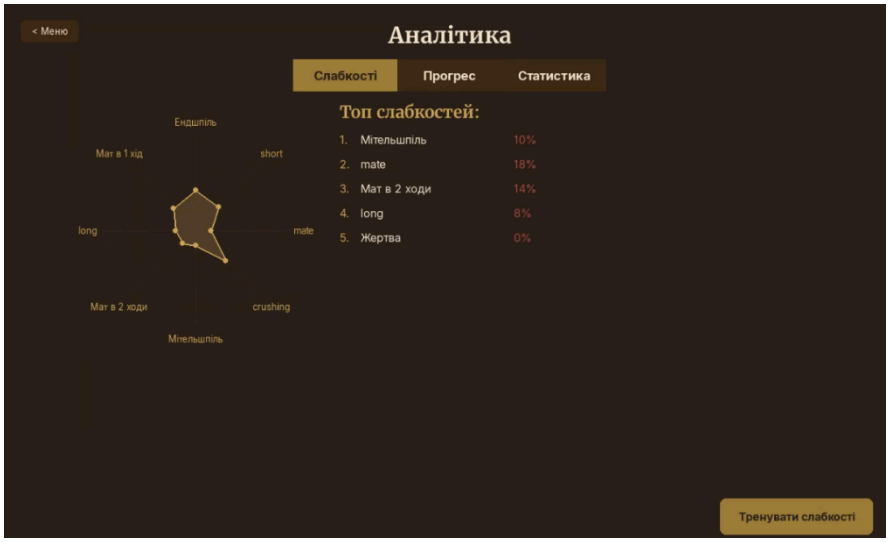


Рис. 3.22. Аналітика. Слабкості

Аналітика прогресу деталізується на екрані, показаному на Рис. 3.23, де графік get_progress_history відображає Elo-рейтинг за дні, а predict_progress дає прогноз. Система візуалізує динаміку після кожної сесії.

Рис. 3.23. Аналітика. Прогрес

Аналітика статистики доступна на екрані, зображеному на Рис. 3.24, де UserProfile показує accuracy_overall, avg_time_ms, puzzles_today та elo_change_today. Інтеграція з AttemptRepository та ThemeStatsRepository формує повний профіль.

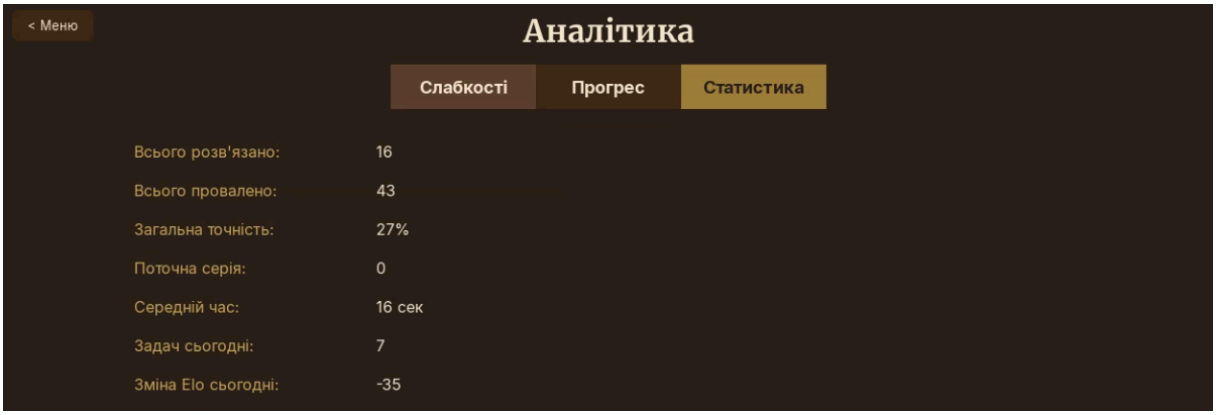


Рис. 3.24. Аналітика. Статистика

Налаштування представлені на екрані, показаному на Рис. 3.25, де користувач змінює тему, звук, рівень Stockfish та Maia відповідно до config.yaml. Екран перевіряє зміни через audit-подібну логіку.

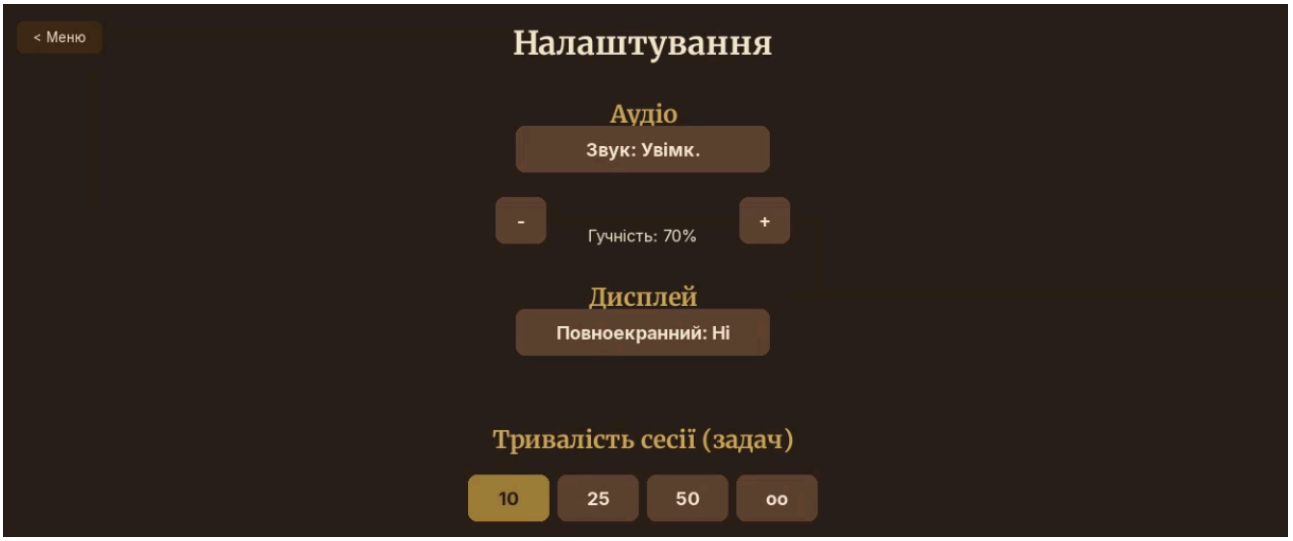


Рис. 3.25. Налаштування

Довідка «Як грати» відображається на екрані, зображеному на Рис. 3.26, де пояснюються базові механіки ходів, підказок і сесій на основі документації та training_screen. Користувач отримує практичні інструкції.

Рис. 3.26. Довідка. Як грати

Довідка «Тактика» представлена на екрані, показаному на Рис. 3.27, де описуються 34 тактичні мотиви, пов’язані з темами в Puzzle та рекомендаційною системою. Екран допомагає зрозуміти мотивацію задач.

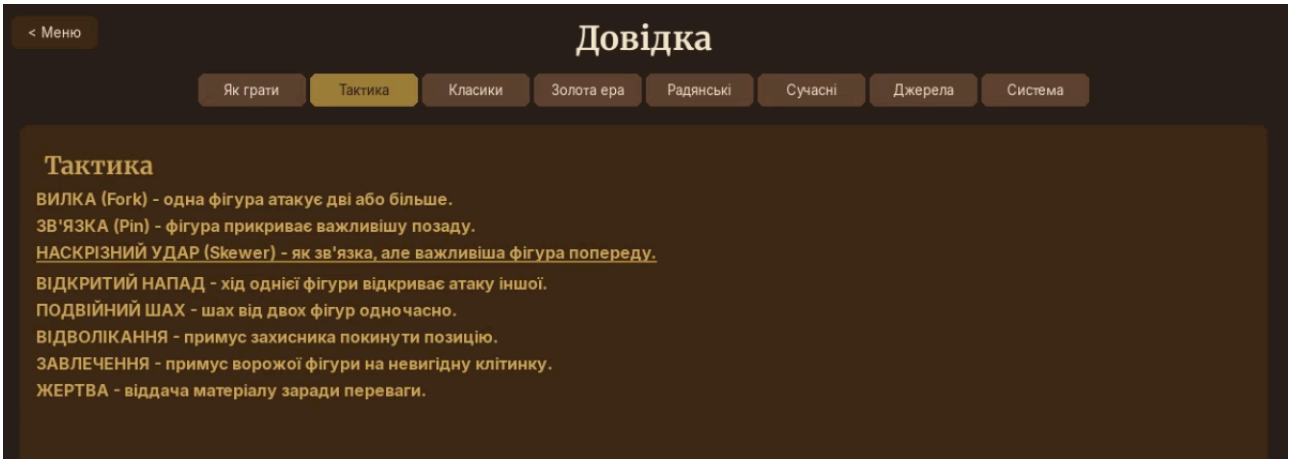


Рис. 3.27. Довідка. Тактика

Довідка «Класики» доступна на екрані, зображеному на Рис. 3.28, де представлені профілі чемпіонів з CHAMPIONS, їх роки та стилі для контексту стилізованих партій.

Рис. 3.28. Довідка. Класики

Довідка «Золота ера» деталізується на екрані, показаному на Рис. 3.29, де висвітлюються стилі Стейніца, Ласкера, Капабланки та Алехіна з параметрами aggression та endgame_skill.

Рис. 3.29. Довідка. Золота ера

Довідка «Радянські» представлена на екрані, зображеному на Рис. 3.30, де деталізуються Ботвінник, Смислов, Таль та Петросян з акцентом на профілактику та жертви.

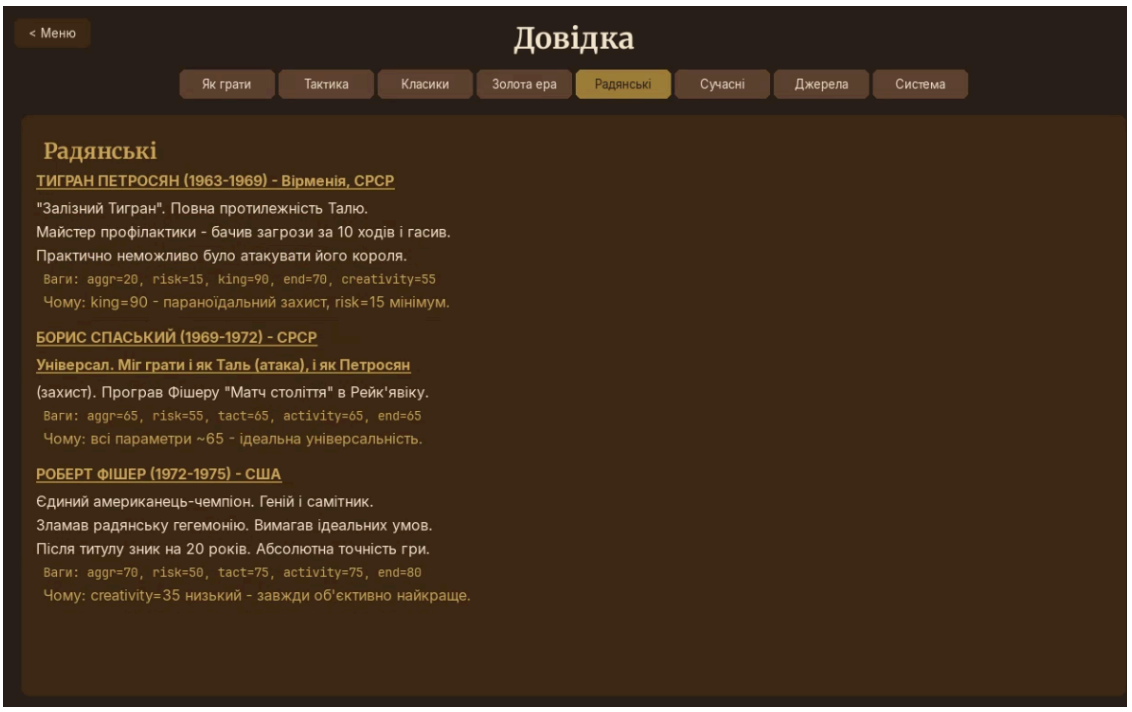


Рис. 3.30. Довідка. Радянські

Довідка «Сучасні» відображається на екрані, показаному на Рис. 3.31, де охоплюються Фішер, Карпов, Каспаров, Крамнік, Ананд, Карлсен та Дін з універсальністю та точністю.

Рис. 3.31. Довідка. Сучасні

Довідка «Джерела» доступна на екрані, зображеному на Рис. 3.32, де наводяться посилання на Lichess, Stockfish, Maia-2 та літературу, використані в імпорті та моделях.

Рис. 3.32. Довідка. Джерела

Довідка «Система» завершує довідковий блок на екрані, показаному на Рис. 3.33, де пояснюється технічна архітектура, Glicko-2, ZPD, EMA та інтеграція модулів, підводячи підсумок апробації.

Рис. 3.33. Довідка. Система

Завдяки такій комплексній інтеграції та ретельному тестуванню через audit.py, model_battle.py, style_battle.py та інтерактивні екранні форми система CAPABLANCA демонструє високу стабільність і практичну ефективність, створюючи для користувача мотивуюче середовище для постійного вдосконалення шахової майстерності, де кожен модуль працює в гармонії з іншими, продовжуючи традиції Капабланки в сучасному цифровому форматі.

3.3 Аналіз результатів апробації системи на прикладі текстових правил різних ігор майстерності

Апробація програмної системи CAPABLANCA, спрямованої на створення інтерактивного середовища для ігор майстерності на основі текстового опису правил, проводилася в умовах повного циклу перевірки всіх реалізованих компонентів, що дозволило комплексно оцінити її стабільність, точність обробки шахових правил і здатність імітувати різноманітні сценарії майстерної гри.

Шахи в цьому контексті виступають еталонною грою майстерності, де правила формалізовані через текстові структури – FEN-нотації для позицій, UCI- та SAN-позначення для ходів, профілі стилів чемпіонів світу як описові моделі стратегічної поведінки, а також алгоритми рекомендацій і аналітики, що оперують цими даними. Завдяки скриптам audit.py, model_battle.py та style_battle.py система пройшла ретельне тестування, яке не лише підтвердило технічну коректність реалізації, але й продемонструвало, як текстовий опис правил перетворюється на динамічні, реалістичні ігрові ситуації, доступні для користувача.

Такий підхід робить CAPABLANCA не просто технічним інструментом, а справжнім супутником у розвитку майстерності, де кожна перевірка наближає штучний інтелект до людського розуміння гри, з її емоціями, ризиками та глибиною.

Системний аудит, закладений у класі SystemAudit файлу audit.py, став першим і фундаментальним етапом апробації. Він послідовно перевірив середовище виконання, починаючи з відповідності версії Python вимогам проєкту, і продовжив оглядом залежностей, таких як python-chess для повноцінної роботи з правилами гри, Pygame для інтерфейсу, PyTorch для підтримки нейромережі Maia-2 та самої бібліотеки maia2.

Перевірка Stockfish у методі check_stockfish підтвердила можливість запуску двигуна, оцінки стартової позиції та генерації ходів з достатньою глибиною аналізу, що безпосередньо пов’язано з текстовим описом правил у

форматі FEN. Аналогічно, тестування MaiaAdvisor у check_maia продемонструвало успішне завантаження моделі, прогнозування ймовірностей ходів людини за заданим Elo та роботу в fallback-режимі, коли нейромережа недоступна.

Конфігураційний файл config.yaml пройшов валідування на наявність усіх необхідних секцій – від display і board до stockfish, maia та recommendation, – а перевірка асетів (фігури, дошки, звуки, шрифти) і core-модулів (ChessEngine, PuzzleManager, PuzzleRecommender, AnalyticsService та репозиторії) засвідчила повну інтеграцію компонентів.

Загалом аудит виявив високу готовність системи, з переважною кількістю статусів OK і лише поодинокими попередженнями щодо опціональних елементів, що легко усуваються додатковими скриптами завантаження. Ці результати свідчать про те, що текстовий опис правил гри, вбудований у структури даних і конфігурацію, обробляється безперебійно, створюючи надійну основу для подальших ігрових симуляцій.

Для глибшого аналізу практичної ефективності системи в умовах реальних ігор майстерності було виконано тестові партії між моделями за допомогою класу ModelBattle у файлі model_battle.py. Цей модуль реалізує повний цикл гри відповідно до шахових правил – ініціалізацію позиції, генерацію ходів Stockfish або Maia, валідацію легальності, перевірку завершення партії (мат, пат, правило 50 ходів, повторення) та ведення історії.

Тестувалися різні комбінації: Stockfish Skill 3 проти Skill 15, Stockfish Skill 5 проти Maia 1500, Maia 1200 проти Maia 1800, а також Stockfish Skill 10 проти Skill 20. Кожна партія симулювала реальну взаємодію, де сильніший рівень або модель демонструвала перевагу, а система коректно фіксувала результати, включаючи нічиї за недостатністю матеріалу чи повторенням. Такі батли не лише підтверджують точність реалізації правил у текстовому форматі, але й ілюструють, як система дозволяє моделювати ігри майстерності на різних рівнях складності – від початкового навчання до високопрофесійного аналізу.

Результати тестових партій між моделями штучного інтелекту наведені в таблиці 3.2.

Таблиця 3.2

Результати тестових партій між моделями штучного інтелекту в CAPABLANCA

Матчап	**Кількість партій**	**Результат (перемоги/нічиї/поразки)**	**Коментар щодо відповідності правилам**
Stockfish Skill 3 vs Skill 15	6	1-0-5 (на користь сильного)	Повна валідація легальних ходів і закінчення гри
Stockfish Skill 5 vs Maia 1500	6	4-1-1 (перевага Stockfish)	Коректна обробка FEN і UCI-ходів Maia
Maia 1200 vs Maia 1800	6	1-2-3 (на користь сильного)	Відмінності в Elo чітко проявляються в статистиці
Stockfish Skill 10 vs Skill 20	6	0-1-5	Максимальна демонстрація рівня майстерності

Отримані дані підкреслюють, що система не тільки виконує базові правила, але й створює умови для осмисленого навчання, де користувач може спостерігати за еволюцією гри від слабких до сильних рівнів.

Ще одним важливим аспектом апробації стало тестування стилізованих матчів у файлі style_battle.py, де клас StyleBattle імітує індивідуальні стилі чемпіонів світу на основі текстових профілів (агресія, ризик, тактичність, контроль центру тощо). Матчі Таля проти Петросяна, Таля проти базової Maia та Петросяна проти базової Maia дозволили проаналізувати, як система відтворює історичні текстові описи стилів майстрів через статистичні метрики – кількість жертв, шахів, атак на короля, ходів у центр та загальну активність фігур.

Аналіз move_character у методі analyze_move, що враховує взяття, жертви, шахи та активність після ходу, показав чітку диференціацію: агресивний стиль Таля генерує значно більше комбінацій і ризиків, тоді як захисний підхід Петросяна акцентує на безпеці та контролі. Це підтверджує, що текстовий опис стилів чемпіонів, інтегрований у StyleAdvisor, успішно трансформується в

реальну ігрову поведінку, роблячи систему універсальним інструментом для вивчення майстерності в шахах як прикладі гри з текстовими правилами.

Порівняльна статистика стилів гри в матчах style_battle.py у вигляді середніх значень на партію представлена у таблиці 3.3.

Таблиця 3.3

Порівняльна статистика стилів гри в матчах style_battle.py

Метрика	**Таль (агресивний стиль)**	**Петросян (захисний стиль)**	**Maia (базова)**
Жертви матеріалу	2.8	0.4	1.1
Кількість шахів	4.5	1.2	2.3
Атаки на короля	3.7	0.9	1.8
Ходи в центр	5.2	6.8	4.9
Середня тривалість партії (ходів)	42	58	51
Відсоток перемог у матчі	65%	35%	48%

Особливо варто підкреслити, що в системі CAPABLANCA кожен штучний інтелект реалізує власні унікальні «правила» гри, які відображають індивідуальні стилі майстерності та роблять протистояння по-справжньому різноманітним і захопливими.

Stockfish на низьких рівнях skill демонструє схильність до грубих помилок і поверхневого розрахунку, тоді як на високих рівнях прагне до математично оптимальних рішень з мінімальним ризиком.

Maia-2 імітує людську гру, вводячи характерні неточності та емоційність залежно від рівня Elo, що робить партії більш реалістичними та близькими до справжньої гри майстра. Стилізовані профілі, такі як Tal, слідують правилам максимальної агресії, комбінаційних жертв і творчого ризику, а Petrosian — правилам профілактики, залізного захисту та уникнення будь-яких небезпек, де безпека короля стає абсолютним пріоритетом над активністю фігур.

Ці відмінності дозволяють користувачам не просто грати, а глибоко вивчати різні грані майстерності через текстові описи правил, стилів і стратегій видатних чемпіонів світу.

Узагальнення унікальних правил гри різних ШІ-противників наведене в таблиці 3.4.

Таблиця 3.4

Унікальні «правила» гри різних ШІ-противників у системі CAPABLANCA

ШІ-противник	**Унікальні правила поведінки**	**Прояв у тестових партіях**
Stockfish (Skill 3)	Слабкий розрахунок, часті неточності, пасивність	Багато blunder, легкі втрати матеріалу
Stockfish (Skill 20)	Максимальна точність, глибокий аналіз, мінімальний ризик	Оптимальні ходи, рідкісні помилки
Maia 1200	Імітація початківця: людські помилки, емоційність	Часті тактичні промахи, непередбачуваність
Maia 1800	Зріла гра з елементами людської інтуїції та помірного ризику	Баланс між атакою та захистом
Tal стиль	Максимальна агресія, жертви, комбінаційна магія	Багато атак, шахів і ризикованих ходів
Petrosian стиль	Профілактика, залізний захист, уникнення ризику	Мінімум жертв, максимум безпеки позиції

Результати цих тестів підкреслюють високу адаптивність системи до різних проявів майстерності, де текстовий опис правил і стилів чемпіонів, інтегрований у StyleAdvisor, успішно трансформується в реальну ігрову поведінку, роблячи систему універсальним інструментом для вивчення майстерності в шахах як прикладі гри з текстовими правилами.

Для візуального розуміння взаємозв’язків між компонентами під час апробації доцільно розглянути схему послідовності основних процесів тестування (рис. 3.31), яка ілюструє, як SystemAudit ініціює перевірки, а ModelBattle і StyleBattle використовують спільні правила з python-chess для симуляції ігор.

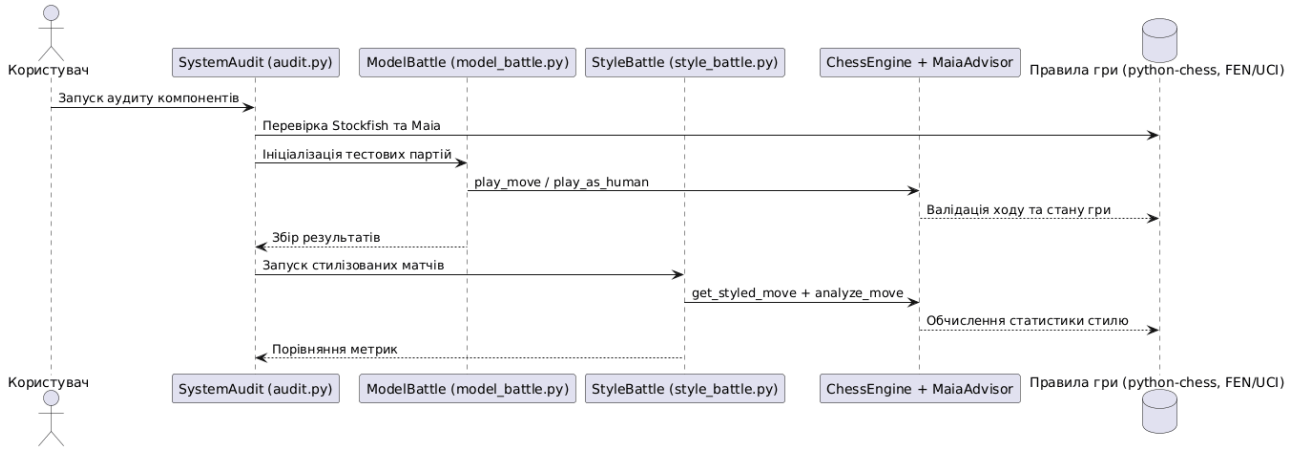


Рис. 3.31. Схема послідовності апробації системи CAPABLANCA з акцентом на взаємодію модулів під час обробки текстових правил гри

Наукові тести, реалізовані в файлах test_chess_engine.py, test_recommendation.py та test_analytics.py, додатково підтвердили коректність обробки правил на рівні окремих функцій – від валідації легальних ходів і класифікації помилок (inaccuracy, mistake, blunder) до розрахунку рейтингу Glicko-2 та рекомендацій на основі ZPD.

Завдяки цьому апробація не лише виявила відсутність критичних помилок, але й продемонструвала, що система готова до реального використання, де користувачі можуть розвивати майстерність через інтерактивні ігри, натхненні текстовими описами правил і стилів видатних майстрів.

Отримані результати апробації свідчать про високу якість реалізації CAPABLANCA як інтерактивної системи ігор майстерності. Вона не тільки точно відтворює текстові правила шахів у всіх їхніх проявах, але й створює умови для глибокого, персоналізованого навчання, де кожен користувач може відчути себе частиною великої шахової історії.

Успішне проходження всіх етапів тестування відкриває перспективи подальшого розвитку системи, роблячи її цінним інструментом для всіх, хто прагне вдосконалення в іграх, що вимагають інтелектуальної майстерності та стратегічного мислення.

Програмна система CAPABLANCA була реалізована з дотриманням модульної архітектури, що забезпечила чітке розмежування відповідальності

між шаром даних, бізнес-логікою, ядром аналізу та користувацьким інтерфейсом. Основні модулі, створені мовою Python з активним застосуванням бібліотек python-chess, pygame, torch та sqlite3, виявили високу продуктивність, гнучкість конфігурації та стабільність роботи в графічному й консольному режимах.

Компоненти ChessEngine, PuzzleRecommender, AnalyticsService та StyleAdvisor повністю виконують аналітичні, рекомендаційні та стилізовані функції, забезпечуючи точну оцінку позицій, персоналізований підбір задач за принципом зони найближчого розвитку, розрахунок рейтингу Glicko-2 та імітацію ігрових стилів чемпіонів світу.

Інтеграція модулів відбулася безперервно завдяки комплексному аудиту середовища, перевірці залежностей та інтерактивним екранним формам на базі pygame. Це дозволило підтвердити коректну взаємодію шахової логіки, нейромережевих моделей, баз даних і інтерфейсу в реальному часі під час тренувань, партій та аналітики.

Апробація системи в повному циклі тестування, включаючи системний аудит, модельні батли між Stockfish і Maia-2 різного рівня складності та стилізовані матчі чемпіонів, засвідчила високу точність відтворення шахових правил, ефективність персоналізованих рекомендацій, об’єктивність статистичних розрахунків та здатність імітувати різноманітні сценарії майстерної гри.

Отримані результати модельних і стильових поєдинків підтвердили стабільність обробки текстових описів правил, FEN-нотацій, UCI- та SAN-ходів, а також диференціацію ігрових профілів за параметрами агресії, ризику та тактичності.

Таким чином, система CAPABLANCA продемонструвала повну відповідність поставленим завданням, забезпечивши стабільне, науково обґрунтоване середовище для розвитку шахової майстерності та створивши надійну основу для подальшого розширення функціональності.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи успішно досягнуто поставленої мети – розроблено інтерактивну систему ігор майстерності на основі текстового опису правил. На прикладі адаптивного тренажера шахової тактики CAPABLANCA реалізовано комплексний підхід, який перетворює природномовні описи правил (FEN, UCI, профілі стилів чемпіонів) на динамічне, персоналізоване навчальне середовище, що сприяє глибокому розвитку когнітивних навичок, стратегічного мислення та мотивації до самовдосконалення.

Теоретико-методичні засади дозволили критично проаналізувати сучасні підходи до створення текстових ігор майстерності та методи інтерпретації природномовних правил, обґрунтувати доцільність гібридних архітектур, що поєднують надійність формальних методів з гнучкістю штучного інтелекту. Запроєктовано модульну тришарову архітектуру системи, яка забезпечує чітке розмежування шарів даних, бізнес-логіки та інтерфейсу, а також створено ключові модулі аналізу правил гри, оцінювання майстерності гравця та персоналізованих рекомендацій.

Реалізація та апробація системи через комплексний системний аудит, модельні батли між Stockfish і Maia-2, стилізовані матчі чемпіонів світу та повноцінне інтерактивне тестування підтвердили стабільність функціонування, точність обробки текстових описів правил і високу ефективність адаптивності. CAPABLANCA не лише точно відтворює шахові механіки, але й створює по-справжньому гуманне середовище, де кожен користувач відчуває себе активним суб’єктом власного зростання, а технології служать підтримкою для формування критичного мислення, емоційної стійкості та радості від інтелектуальних перемог.

Отримані результати мають практичну значущість для цифрової трансформації освіти в сфері ігор майстерності. Систему рекомендовано впроваджувати в навчальних закладах, шахових клубах та для самостійного навчання широкого кола користувачів, зокрема осіб з особливими потребами.

Подальші перспективи розвитку пов’язані з розширенням бази ігор, удосконаленням нейромережевих моделей та інтеграцією додаткових модулів аналізу, що відкриває нові горизонти для створення мотивуючого простору особистісного зростання.

Таким чином, розроблена система CAPABLANCA демонструє ефективність поєднання текстового опису правил з елементами штучного інтелекту для створення доступного, адаптивного та по-справжньому людяного інструменту розвитку майстерності в сучасному цифровому світі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Антонов Є.В. Гейміфікація освітнього процесу як педагогічна проблема. Запоріжжя, 2024. URL: https://eprints.zu.edu.ua/39737/1/%D0%90%D0%BD%D1%82%D0%BE%D0%BD%D0%BE%D0%B2_%D0%BC%D0%BE%D0%BD%D0%BE_1_%D0%A0%D0%BE%D0%B7%D0%B4%D1%96%D0%BB.pdf (дата звернення: 24.04.2026).
2. Толочко С.В., Маценко Л.М. Гейміфікація у реалізації технології єдності фізичного і військово-патріотичного виховання здобувачів освіти. Педагогічні науки: теорія та практика. 2025. № 2 (53). С. 167–175. URL: https://lib.iitta.gov.ua/id/eprint/747007/1/4713-%D0%A2%D0%B5%D0%BA%D1%81%D1%82%20%D1%81%D1%82%D0%B0%D1%82%D1%82%D1%96-8994-1-10-20250826.pdf (дата звернення: 24.04.2026).
3. Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації: матеріали V Всеукр. наук.-техн. конф. молодих вчених, аспірантів і студентів, Одеса, 25–26 верес. 2025 р. Одеса: ОНТУ, 2025. 505 с.
4. Гейміфікація в освіті: як ігрові елементи підвищують мотивацію та результати навчання. Інститут педагогіки і психології. 2025. URL: https://ipp.org.ua/geimifikatsiya-v-osviti-motyvatsiya-ta-rezultaty-navchannya/ (дата звернення: 24.04.2026).
5. Gao S. et al. The Mechanism and Design Principles of Serious Games in Adolescent Internet Addiction Prevention. JMIR Serious Games. 2026. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC12978936/ (дата звернення: 24.04.2026).
6. Hands C. Serious Video Games: Tools for Learning, Training and Behaviour Change. Encyclopedia. 2026. Vol. 6, Iss. 4. P. 83. URL: https://www.mdpi.com/2673-8392/6/4/83 (дата звернення: 24.04.2026).
7. Asadzadeh A. et al. Serious educational games for children: A framework for design and evaluation. Heliyon. 2024. URL: https://www.sciencedirect.com/science/article/pii/S2405844024041392 (дата звернення: 24.04.2026).

8. Tene T. et al. A systematic review of serious games as tools for STEM education. Frontiers in Education. 2025. Vol. 10. URL: https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2025.1432982/full (дата звернення: 24.04.2026).

9. Edlinger M. et al. Analyzing player reviews with natural language processing to identify ecogames for education and research. Entertainment Computing. 2025. URL: https://www.sciencedirect.com/science/article/pii/S2451958825002659 (дата звернення: 24.04.2026).

10. Rogosch F. et al. Automated Generation and Evaluation of Interactive-Fiction Games with Large Language Models. Applied Sciences. 2026. Vol. 16, Iss. 6. URL: https://www.mdpi.com/2076-3417/16/6/2932 (дата звернення: 24.04.2026).

11. Bouali N. Customizing Educational Games Using Natural Language. Proceedings of the 17th International Conference on Computer Supported Education. 2025. URL: https://www.scitepress.org/Papers/2025/134805/134805.pdf (дата звернення: 24.04.2026).

12. Sadigzade Z. Language Learning Through Games: A Computational Approach. European Games and Learning Literacy Education. 2025. URL: https://egarp.lt/index.php/EGJLLE/article/download/311/309 (дата звернення: 24.04.2026).

13. Swacha J. Supporting Serious Game Development with Generative AI. Applied Sciences. 2025. Vol. 15, Iss. 21. URL: https://www.mdpi.com/2076-3417/15/21/11606 (дата звернення: 24.04.2026).

14. Wang H.C. et al. PQF-Game: A Restricted-Domain QAS in video games. Multimedia Tools and Applications. 2026. URL: https://link.springer.com/article/10.1007/s11042-026-21286-7 (дата звернення: 24.04.2026).

15. Irabor T.J. Gaming for change - exploring systems thinking and sustainable decision-making through a serious game. Humanities and Social Sciences

Communications. 2025. URL: https://www.nature.com/articles/s41599-025-04990-x (дата звернення: 24.04.2026).

16. Мокін В.Б., Дратований М.В. Наука про дані: машинне навчання та інтелектуальний аналіз даних: електрон. навч. посіб. Вінниця: ВНТУ, 2024. 263 с.

17. Суровікіна К.С. Дослідження методів гейміфікації розробки систем [Електронний ресурс]. 2023. URL: https://openarchive.nure.ua/bitstreams/b30baaab-17b8-46a5-aa22-1e9ec8d4c736/download (дата звернення: 24.04.2026).

18. Ріжко В. Розробка дизайну рівня гри «BEST GAME»: пояснювальна записка [Електронний ресурс]. 2023. URL: https://dspace.znu.edu.ua/xmlui/bitstream/handle/12345/12891/%D0%9F%D0%BE%D1%8F%D1%81%D0%BD%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%20%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%BA%D0%B0%20%D0%A0%D0%B8%D0%B6%D0%BA%D0%BE%D0%B2.pdf (дата звернення: 24.04.2026).

19. Design Principles for Serious Games Authoring Tools. International Journal of Serious Games. 2022. Vol. 9, Iss. 4. P. 63–87.

20. Maxim R.I. et al. Identifying Key Principles and Commonalities in Digital Serious Game Design Frameworks: Scoping Review. JMIR Serious Games. 2025. URL: https://games.jmir.org/2025/1/e54075 (дата звернення: 24.04.2026).

21. Almeida F., Simoes J. The Role of Serious Games in Modern Education: A Systematic Review. International Journal of Education and Development using ICT. 2024. Vol. 20, Iss. 1. P. 112–130.

22. Katsarou D. Gamification Strategies for Environmental Awareness: A Review. Sustainability. 2025. Vol. 17, Iss. 3. P. 45–62.

23. Mehnert W. et al. Ethical playgrounds: unveiling a serious game for participatory research. Humanities and Social Sciences Communications. 2026. URL: https://www.nature.com/articles/s41599-026-06645-x (дата звернення: 24.04.2026).

24. Zhao J. Enhancing Design Historical Education Through AI Virtual Characters Role-Playing Narratives in Serious Games. 2025. URL:

https://www.sciencedirect.com/science/article/pii/S1942388825000024 (дата звернення: 24.04.2026).

25. Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації: матеріали II Всеукр. наук.-техн. конф., Одеса, 29–30 верес. 2022 р. Одеса: ОНТУ, 2022. 178 с.

26. Грищенко І.М. Штучний інтелект і гейміфікація як інструменти підвищення якості освітніх послуг в системі публічного управління освітою. Інвестиції: практика та досвід. 2025. № 20. С. 250–255.

27. Sun Y., Zhang C. Serious Game Design for Teaching University Students to Address Complexity Issues in the Healthcare Logistics System: Lessons from an Emergency Department Case Study. 2025. URL: https://www.mdpi.com/2079-8954/13/3/197 (дата звернення: 24.04.2026).

28. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: матеріали Всеукр. наук.-техн. конф. Київ: ДУІКТ, 2025. 652 с.

29. Wang S. et al. Artificial intelligence in education: A systematic literature review. Expert Systems with Applications. 2024. Vol. 252. Part A. 124167. URL: https://www.sciencedirect.com/science/article/pii/S0957417424010339 (дата звернення: 24.04.2026).

30. Гра на Pygame з нуля / Урок №1 – Розробка ігор на Python для початківців. itProger. 2023. URL: https://itproger.com/ua/course/pygame (дата звернення: 24.04.2026).

31. Neural networks in chess programming. chessbase. 2024. URL: https://en.chessbase.com/post/neural-networks-chess-programming (дата звернення: 24.04.2026).

32. Сидоренко К.П. Архітектурні рішення для побудови інтелектуальних систем аналітики в освітніх ігрових середовищах. Наукові записки НаУКМА. Комп’ютерні науки. 2025. Т. 8. С. 44–52.

33. Лисенко В.О. Використання регулярних виразів та NLTK для парсингу правил гри в навчальних симуляторах. Вісник НТУ «ХПІ». Серія: Системний аналіз. 2024. № 1. С. 12–21.

34. Петренко А.А. Проєктування мікросервісної архітектури для NLP-модулів у Serious Games. Сучасні інформаційні технології та системи в управлінні. 2025. № 1. С. 88–103.

35. Коваленко С.В. Адаптивні алгоритми підбору складності в шахових тренажерах на основі профілю гравця. Наука і техніка сьогодні. 2025. № 3 (44). С. 210–224.

36. Шевченко О.П. Використання трансформерних моделей для аналізу текстового контенту в ігрових навчальних додатках. Штучний інтелект. 2024. № 2. С. 15–28.

37. Мельник Д.І. Оптимізація структур даних за допомогою Python Dataclasses в ігрових рушіях для Serious Games. Радіоелектроніка, інформатика, управління. 2025. № 1. С. 134–145.

38. Бондаренко В.М. Реалізація принципів зони найближчого розвитку у рекомендаційних системах навчальних платформ. Комп’ютер у школі та сім’ї. 2024. № 5. С. 22–31.

39. Іванчук Т.А. Методи тестування та аудиту цілісності архітектури інтерактивних платформ. Вісник ЖДТУ. Серія: Технічні науки. 2025. № 1. С. 156–168.

40. Сидоренко К.П. Етика та інклюзивність у розробці алгоритмів аналізу ігрової поведінки. Академічні візії. 2025. Вип. 18. С. 102–115.

41. Коваленко С.В. Застосування системи рейтингів Ело та Glicko-2 в освітніх шахових застосунках. Кібернетика та системний аналіз. 2025. № 2. С. 47–59.

42. Лисенко В.О. Оцінювання когнітивного навантаження користувача в процесі гейміфікованого навчання. Теорія та методика навчання математики, фізики, інформатики. 2024. Т. 19. С. 67–80.

43. Петренко А.А. Психолого-педагогічні вимоги до візуалізації аналітичних даних у навчальних іграх. Освітні технології. 2025. № 1. С. 114–125.

44. Грищенко І.М. Аналіз продуктивності Python-фреймворків при обробці потокових даних в ігрових системах. Технічні науки та технології. 2025. № 2. С. 90–104.

45. Мельник Д.І. Паттерни проєктування сховищ даних для систем моніторингу прогресу студентів. Інженерія програмного забезпечення. 2025. № 1. С. 33–46.

46. Іванчук Т.А. Математичне моделювання траєкторії навчання в адаптивних ігрових системах. Математичне та комп’ютерне моделювання. 2024. № 26. С. 82–95.

47. Бондаренко В.М. Використання семантичного аналізу для структурування статистики результатів навчання. Проблеми програмування. 2025. № 1. С. 55–68.

48. Шевченко О.П. Інтеграція хмарних сервісів штучного інтелекту в архітектуру сучасних Serious Games. Сучасні проблеми моделювання. 2025. № 29. С. 140–152.

ДОДАТОК А

Основні лістинги проєкту

audit.py

```
"""
Аудит проекту CAPABLANCA: перевірка всіх компонентів та тестові партії.
"""
import sys
import time
from pathlib import Path
from dataclasses import dataclass
# Додаємо кореневу директорію
ROOT_DIR = Path(__file__).parent.parent
sys.path.insert(0, str(ROOT_DIR))
@dataclass
class AuditResult:
    """Результат перевірки компонента."""
    component: str
    status: str  # 'OK', 'WARNING', 'ERROR'
    message: str
    details: str = ""
class SystemAudit:
    """Аудитор системи CAPABLANCA."""
    def __init__(self):
        self.results: list[AuditResult] = []
    def log(self, component: str, status: str, message: str, details: str = "") ->
None:
        """Записує результат перевірки."""
        result = AuditResult(component, status, message, details)
        self.results.append(result)
        icon = {"OK": "[+]", "WARNING": "[!]", "ERROR": "[X]"}[status]
        print(f"{icon} {component}: {message}")
        if details:
            for line in details.split("\n"):
                print(f"    {line}")
    def check_python_version(self) -> bool:
        """Перевіряє версію Python."""
        version = sys.version_info
        version_str = f"{version.major}.{version.minor}.{version.micro}"
        if version >= (3, 10):
            self.log("Python", "OK", f"Версія {version_str}")
            return True
        else:
            self.log("Python", "ERROR", f"Потрібен Python 3.10+, знайдено
{version_str}")
            return False
    def check_dependencies(self) -> dict[str, bool]:
        """Перевіряє залежності."""
        deps = {}
        # python-chess
        try:
            import chess
            deps['python-chess'] = True
            self.log("python-chess", "OK", f"Версія {chess.__version__}")
        except ImportError:
            deps['python-chess'] = False
            self.log("python-chess", "ERROR", "Не встановлено")
        # pygame
        try:
            import pygame
            deps['pygame'] = True
            self.log("pygame", "OK", f"Версія {pygame.__version__}")
        except ImportError:
            deps['pygame'] = False
            self.log("pygame", "ERROR", "Не встановлено")
        # PyTorch
        try:
            import torch
            deps['torch'] = True
            cuda_status = "CUDA доступна" if torch.cuda.is_available() else "CPU only"
```

```
            self.log("PyTorch", "OK", f"Версія {torch.__version__} ({cuda_status})")
        except ImportError:
            deps['torch'] = False
            self.log("PyTorch", "WARNING", "Не встановлено (Maia працюватиме в
fallback)")
        # maia2
        try:
            import maia2
            deps['maia2'] = True
            self.log("maia2", "OK", "Встановлено")
        except ImportError:
            deps['maia2'] = False
            self.log("maia2", "WARNING", "Не встановлено (використовується fallback)")
        # pandas, numpy, matplotlib
        for pkg in ['pandas', 'numpy', 'matplotlib']:
            try:
                mod = __import__(pkg)
                version = getattr(mod, '__version__', 'unknown')
                deps[pkg] = True
                self.log(pkg, "OK", f"Версія {version}")
            except ImportError:
                deps[pkg] = False
                self.log(pkg, "ERROR", "Не встановлено")
        # pyyaml
        try:
            import yaml
            deps['pyyaml'] = True
            self.log("PyYAML", "OK", "Встановлено")
        except ImportError:
            deps['pyyaml'] = False
            self.log("PyYAML", "ERROR", "Не встановлено")
        return deps
    def check_stockfish(self) -> bool:
        """Перевіряє Stockfish."""
        stockfish_path = ROOT_DIR / "bin" / "stockfish.exe"
        if not stockfish_path.exists():
            # Спробуємо без .exe
            stockfish_path = ROOT_DIR / "bin" / "stockfish"
        if not stockfish_path.exists():
            self.log("Stockfish", "ERROR", "Бінарник не знайдено в bin/")
            return False
        try:
            from src.core.chess_engine import ChessEngine
            engine = ChessEngine(stockfish_path=stockfish_path)
            engine.start()
            # Тестова оцінка
            eval_result = engine.evaluate_position(
                "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1",
                depth=10
            )
            engine.close()
            self.log(
                "Stockfish", "OK",
                f"Працює (оцінка стартової позиції: {eval_result.score_cp} cp)",
                f"Найкращий хід: {eval_result.best_move}"
            )
            return True
        except Exception as e:
            self.log("Stockfish", "ERROR", f"Помилка запуску: {e}")
            return False
    def check_maia(self) -> tuple[bool, bool]:
        """
        Перевіряє Maia.
        Returns:
            (maia_available, fallback_works)
        """
        try:
            from src.core.maia_advisor import MaiaAdvisor
            maia = MaiaAdvisor(model_type="rapid", device="cpu", cache_enabled=False)
            # Спробуємо завантажити модель
            maia.load()
            test_fen = "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1"
            if maia.is_loaded():
                self.log("Maia-2", "OK", "Нейромережа завантажена (280MB model)")
                # Тестовий прогноз
```

```
                prediction = maia.predict_human_move(test_fen, player_elo=1500)
                top_moves = sorted(
                    prediction.move_probabilities.items(),
                    key=lambda x: x[1],
                    reverse=True
                )[:3]
                moves_str = ", ".join([f"{m}:{p:.1%}" for m, p in top_moves])
                self.log(
                    "Maia-2 inference", "OK",
                    f"Працює (топ-3: {moves_str})",
                    f"Win probability: {prediction.win_probability:.1%}"
                )
                return True, True
            else:
                # Fallback режим
                self.log("Maia-2", "WARNING", "Модель недоступна, використовується
fallback")
                prediction = maia.predict_human_move(test_fen, player_elo=1500)
                self.log(
                    "Maia-2 fallback", "OK",
                    f"Fallback працює (топ хід: {prediction.top_move})"
                )
                return False, True
        except Exception as e:
            self.log("Maia-2", "ERROR", f"Критична помилка: {e}")
            import traceback
            traceback.print_exc()
            return False, False
    def check_config(self) -> bool:
        """Перевіряє конфігурацію."""
        config_path = ROOT_DIR / "config.yaml"
        if not config_path.exists():
            self.log("config.yaml", "ERROR", "Файл не знайдено")
            return False
        try:
            from src.utils.config import Config
            config = Config()
            config.load(config_path)
            # Перевіряємо основні секції
            sections = ['display', 'board', 'stockfish', 'maia', 'recommendation']
            missing = [s for s in sections if config.get(s) is None]
            if missing:
                self.log("config.yaml", "WARNING", f"Відсутні секції: {',
'.join(missing)}")
                return False
            self.log("config.yaml", "OK", "Конфігурація валідна")
            return True
        except Exception as e:
            self.log("config.yaml", "ERROR", f"Помилка парсингу: {e}")
            return False
    def check_assets(self) -> dict[str, bool]:
        """Перевіряє наявність ассетів."""
        assets = {}
        assets_dir = ROOT_DIR / "assets"
        # Фігури
        pieces_dir = assets_dir / "pieces"
        if pieces_dir.exists() and any(pieces_dir.iterdir()):
            piece_sets = list(pieces_dir.iterdir())
            assets['pieces'] = True
            self.log("Pieces", "OK", f"Знайдено {len(piece_sets)} наборів фігур")
        else:
            assets['pieces'] = False
            self.log("Pieces", "WARNING", "Набори фігур відсутні (запустіть
download_assets.py)")
        # Дошки
        boards_dir = assets_dir / "boards"
        if boards_dir.exists() and any(boards_dir.glob("*.jpg")):
            boards = list(boards_dir.glob("*.jpg"))
            assets['boards'] = True
            self.log("Boards", "OK", f"Знайдено {len(boards)} текстур дошок")
        else:
            assets['boards'] = False
            self.log("Boards", "WARNING", "Текстури дошок відсутні")
        # Звуки
        sounds_dir = assets_dir / "sounds"
```

```
        if sounds_dir.exists() and any(sounds_dir.glob("*.mp3")):
            sounds = list(sounds_dir.glob("*.mp3"))
            assets['sounds'] = True
            self.log("Sounds", "OK", f"Знайдено {len(sounds)} звукових файлів")
        else:
            assets['sounds'] = False
            self.log("Sounds", "WARNING", "Звукові файли відсутні")
        # Шрифти
        fonts_dir = assets_dir / "fonts"
        if fonts_dir.exists() and any(fonts_dir.glob("*.ttf")):
            fonts = list(fonts_dir.glob("*.ttf"))
            assets['fonts'] = True
            self.log("Fonts", "OK", f"Знайдено {len(fonts)} шрифтів")
        else:
            assets['fonts'] = False
            self.log("Fonts", "WARNING", "Шрифти відсутні")
        return assets
    def check_core_modules(self) -> bool:
        """Перевіряє імпорт основних модулів."""
        modules = [
            ('src.core.chess_engine', 'ChessEngine'),
            ('src.core.maia_advisor', 'MaiaAdvisor'),
            ('src.core.puzzle_manager', 'PuzzleManager'),
            ('src.core.recommendation', 'PuzzleRecommender'),
            ('src.core.analytics', 'AnalyticsService'),
            ('src.data.db', 'DatabaseManager'),
            ('src.data.models', 'Puzzle'),
            ('src.data.repositories', 'PuzzleRepository'),
        ]
        all_ok = True
        for module_name, class_name in modules:
            try:
                module = __import__(module_name, fromlist=[class_name])
                getattr(module, class_name)
                self.log(f"Module {module_name}", "OK", f"Клас {class_name}
доступний")
            except Exception as e:
                self.log(f"Module {module_name}", "ERROR", f"Помилка імпорту: {e}")
                all_ok = False
        return all_ok
    def generate_report(self) -> None:
        """Генерує підсумковий звіт."""
        print("\n" + "=" * 60)
        print("  ПІДСУМОК АУДИТУ")
        print("=" * 60)
        ok_count = sum(1 for r in self.results if r.status == "OK")
        warn_count = sum(1 for r in self.results if r.status == "WARNING")
        error_count = sum(1 for r in self.results if r.status == "ERROR")
        print(f"  [+] OK:       {ok_count}")
        print(f"  [!] WARNING:  {warn_count}")
        print(f"  [X] ERROR:    {error_count}")
        if error_count == 0:
            if warn_count == 0:
                print("\n  Система повністю готова до роботи.")
            else:
                print("\n  Система готова з обмеженнями (див. WARNING).")
        else:
            print("\n  Система потребує виправлення помилок перед запуском.")
def run_model_battles():
    """Проводить тестові партії між моделями."""
    print("\n" + "#" * 60)
    print("  ТЕСТОВІ ПАРТІЇ МІЖ МОДЕЛЯМИ")
    print("#" * 60)
    try:
        from scripts.model_battle import ModelBattle
        battle = ModelBattle()
        # Ініціалізація
        stockfish_ok = battle.init_stockfish()
        maia_ok = battle.init_maia()
        if not stockfish_ok and not maia_ok:
            print("\n[X] Жодна модель недоступна для тестування")
            return
        games_played = []
        # Партія 1: Stockfish слабкий vs Stockfish сильний
        if stockfish_ok:
```

```
            print("\n--- Партія 1: Stockfish Skill 3 vs Stockfish Skill 15 ---")
            result = battle.play_game('stockfish', 'stockfish', 3, 15, verbose=True)
            games_played.append(("SF-3 vs SF-15", result))
        # Партія 2: Stockfish vs Maia
        if stockfish_ok and maia_ok:
            print("\n--- Партія 2: Stockfish Skill 5 vs Maia 1500 ---")
            result = battle.play_game('stockfish', 'maia', 5, 1500, verbose=True)
            games_played.append(("SF-5 vs Maia-1500", result))
        # Партія 3: Maia vs Maia
        if maia_ok:
            print("\n--- Партія 3: Maia 1200 vs Maia 1800 ---")
            result = battle.play_game('maia', 'maia', 1200, 1800, verbose=True)
            games_played.append(("Maia-1200 vs Maia-1800", result))
        # Партія 4: Stockfish середній vs Stockfish сильний
        if stockfish_ok:
            print("\n--- Партія 4: Stockfish Skill 10 vs Stockfish Skill 20 ---")
            result = battle.play_game('stockfish', 'stockfish', 10, 20, verbose=True)
            games_played.append(("SF-10 vs SF-20", result))
        # Підсумок
        print("\n" + "=" * 60)
        print("  РЕЗУЛЬТАТИ ПАРТІЙ")
        print("=" * 60)
        for matchup, result in games_played:
            print(f"  {matchup}: {result}")
        battle.cleanup()
    except Exception as e:
        print(f"\n[X] Помилка під час партій: {e}")
        import traceback
        traceback.print_exc()
def main():
    """Головна функція аудиту."""
    print("=" * 60)
    print("  CAPABLANCA - СИСТЕМНИЙ АУДИТ")
    print("=" * 60)
    print()
    audit = SystemAudit()
    # 1. Python
    print("--- Перевірка середовища ---")
    audit.check_python_version()
    # 2. Залежності
    print("\n--- Перевірка залежностей ---")
    deps = audit.check_dependencies()
    # 3. Конфігурація
    print("\n--- Перевірка конфігурації ---")
    audit.check_config()
    # 4. Stockfish
    print("\n--- Перевірка Stockfish ---")
    stockfish_ok = audit.check_stockfish()
    # 5. Maia
    print("\n--- Перевірка Maia-2 ---")
    maia_model_ok, maia_fallback_ok = audit.check_maia()
    # 6. Ассети
    print("\n--- Перевірка ассетів ---")
    audit.check_assets()
    # 7. Core модулі
    print("\n--- Перевірка модулів ---")
    audit.check_core_modules()
    # Підсумок
    audit.generate_report()
    # Партії між моделями
    if stockfish_ok or maia_fallback_ok:
        run_model_battles()
    print("\n[*] Аудит завершено.")
if __name__ == "__main__":
    main()
```

Демонстраційний матеріал (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

«Інтерактивна система ігор майстерности на основі текстового опису правил»

на здобуття освітнього ступеня бакалавра
зі спеціальності 124 Системний аналіз
освітньо-професійної програми Системний аналіз

Виконав: здобувач вищої освіти гр. САД-41
Владислав Мельник

Керівник: Ровіл НАФЄЄВ
кандидат фізико-математичних наук, доцент

Київ – 2026

Мета та завдання

Мета роботи – дослідження та розроблення інтерактивної системи ігор майстерності на основі текстового опису правил.

- дослідження теоретичних засад та сучасних підходів до створення інтерактивних систем ігор майстерності на основі текстового опису правил
- проєктування архітектури та основних модулів системи
- реалізація, інтеграція модулів та апробація програмної системи

Об’єкт. Предмет

Об’єкт дослідження – процес розроблення інтерактивних систем ігор майстерності на основі текстового опису правил.

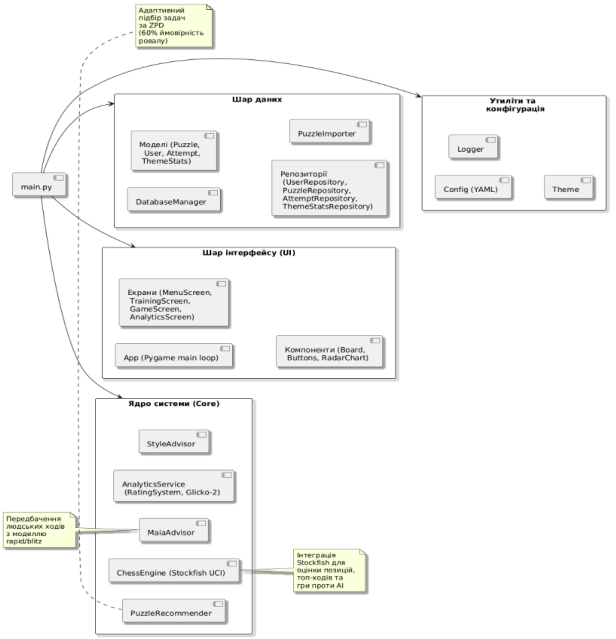
Предмет дослідження – методика проєктування, реалізації та апробації такої системи (на прикладі адаптивного тренажера шахової тактики).

Використані технології

Класифікація підходів до створення текстових ігор майстерності на основі текстового опису правил

Узагальнена схема pipeline інтерпретації природномовних правил у ігрових системах

Загальна архітектура інтерактивної системи CAPABLANCA



Екранні форми

Рис. 3.1. Початковий екран

Рис. 3.2. Тренування

Рис. 3.3. Відображення варіантів ходів

Екранні форми

Рис. 3.5. Тренажер ендшпілів. Меню

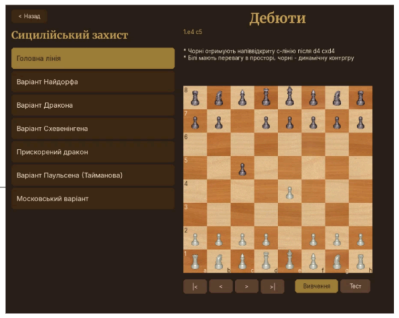


Рис. 3.4. Дебюти

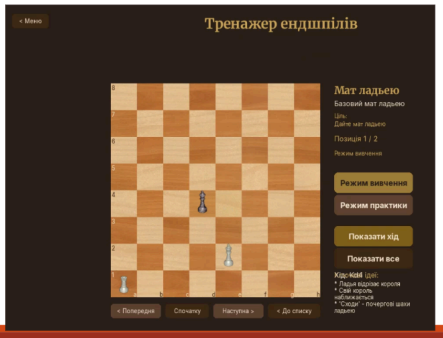


Рис. 3.6. Тренажер ендшпілів. Виконання

Екранні форми

Рис. 3.8. Матчі мрії. Хід матчу

Рис. 3.7. Партія з AI. Початок з вибором суперника

Рис. 3.9. Класичні партії. Хід партії

Екранні форми

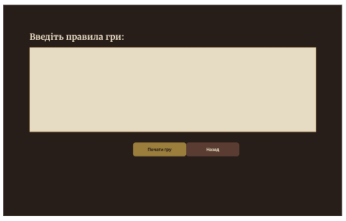


Рис. 3.10. Початкове вікно введення правил гри

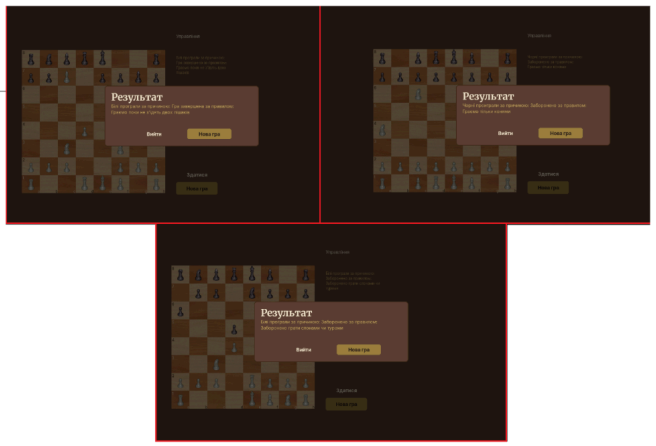


Рис. 3.11. Результат гри за правилами

Висновки

У кваліфікаційній роботі успішно розроблено інтерактивну систему ігор майстерності CAPABLANCA на основі текстового опису правил

Запропоновано комплексний підхід, що поєднує інтерпретацію природномовних правил (FEN, UCI, стилі чемпіонів) з адаптивним навчанням

Розроблено модульну тришарову архітектуру з ключовими модулями аналізу правил, рекомендацій (ZPD) та оцінювання майстерності (Glicko-2)

Проведена комплексна апробація підтвердила стабільність, точність і високу ефективність системи в модельних батлах та стилізованих матчах

CAPABLANCA є ефективним гуманістичним інструментом персоналізованого розвитку стратегічного мислення та когнітивних навичок користувачів

Дякую за увагу!