НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «*Інтерактивна гра в шахи з елементами доповненої реальності*»

Виконав: студент *IV* курсу, групи *ІС-72*

Гороховський Іван Олегович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *ст.в., к.т.н. Олійник Юрій Олександрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Новінський Валерій Петрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц. каф. ТК, к.т.н., доц. Ткач Михайло Мартинович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2021 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Гороховського Івана Олеговича
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Інтерактивна гра в шахи з елементами доповненої реальності»*
керівник проєкту Олійник Юрій Олександрович, *к.т.н.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*11*” *травня* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту
Технічне завдання
Створити застосунок для інтерактивної гри в шахи з елементами доповненої реальності.Створити інтелектуальних агентів різного рівня складності для гри проти людини. Зробити порівняльну характеристику методів створення доповненої реальності, детекції образів та інтелектуальних агентів.

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд аналогічних програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних, аналіз отриманої бази даних

3. Математичне забезпечення: змістовна та математична постановки задачі, опис методу розв'язання

4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема нейронної мережі сучасного інтелектуального агента

2. Схема структурна розгортання

3. Схема структурна діяльності

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «*7*» *квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*12.02.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*19.02.2021*	
3.	*Постановка та формалізація задачі*	*05.03.2021*	
4.	*Розробка інформаційного забезпечення*	*12.03.2021*	
5.	*Алгоритмізація задачі*	*26.03.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*09.04.2021*	
7.	*Розробка програмного забезпечення*	*23.04.2021*	
8.	*Налагодження програми*	*30.04.2021*	
9.	*Виконання графічних документів*	*16.04.2021*	
10.	*Оформлення пояснювальної записки*	*07.05.2021*	
11.	*Подання ДП на попередній захист*	*14.05.2021*	
12.	*Подання ДП на основний захист*	*21.05.2021*	
13.	*Подання ДП рецензенту*	*31.05.2021*	

Студент Іван ГОРОХОВСЬКИЙ

Керівник Юрій ОЛІЙНИК

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7205.00.000 ПЗ	Пояснювальна записка	129	
3	А4	ДП 7205.01.000 ТЗ	Технічне завдання	10	
4	А3	ДП 7205.02.000 СНМ	Схема нейронної мережі сучасного	1	
			інтелектуального агента	1	
5	А3	ДП 7205.03.000 ССР	Схема структурна розгортання	1	
6	А3	ДП 7205.04.000 ССД	Схема структурна діяльності	1	
7	А3	ДП 7205.05.000 ТКП	Теплові карти позицій шахових фігур	1	
8	А3	ДП 7205.06.000 СКА	Схема роботи класичного агента	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7205.00.000.00		
Розробн.		Гороховський І.О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Олійник Ю. О.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72	
Н/контр.		Новінський В.П.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «Інтерактивна гра в шахи з елементами доповненої реальності»

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 43 рисунків, 12 таблиць, 1 додатків, 36 джерел.

Дипломний проєкт присвячений розробці застосунку для інтерактивної гри в шахи з елементами доповненої реальності.

У розділі інформаційного забезпечення представлені вхідні та вихідні дані цього застосунку, наведена схема бази даних та проведений аналіз даних з відкритих джерел.

Розділ математичного забезпечення присвячений аналізу методів, алгоритмів та підходів для модуля доповненої реальності та для модуля інтелектуальних агентів.

У розділі з програмного забезпечення описуються технічні вимоги до системи, засоби розробки та архітектуру програмного забезпечення.

У технологічному розділі описана інструкція користувача та проведено тестування розробленого сервісу.

ДОПОВНЕНА РЕАЛЬНІСТЬ, КОМП’ЮТЕРНИЙ ЗІР, ШТУЧНИЙ ІНТЕЛЕКТ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ШАХИ.

					ДП 7205.00.000 ПЗ		
		Прізвище	Підпис	Дата			
Розроб.		Гороховський І.О.			Інтерактивна гра в шахи з елементами доповненої реальності	Літ.	Лист 2 / Листів
Перевірив.		Олійник Ю.О.					
						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72	
Н. кон.		Новінський В.П					
Затв.		Павлов О.А.					

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 43 figures, 12 tables, 1 appendices, 36 sources.

The diploma project is devoted to the development of an application for interactive game of chess with elements of augmented reality.

The information support section presents the input and output data of this application, provides a scheme of the database and analyzes data from open sources.

The section of mathematical software is devoted to the analysis of methods, algorithms and approaches for the augmented reality module and for the intelligent agents module.

The software section describes the technical requirements for the system, software development tools and architecture.

The technological section describes the user manual and tests the developed service.

AUGMENTED REALITY, COMPUTER SIGHT, ARTIFICIAL INTELLIGENCE, SUPPORTED TRAINING, CHESS.

ЗМІСТ

ВСТУП

Даний дипломний проєкт присвячений розробці і порівнянню сучасних методів створення доповненої реальності, методів розпізнавання об’єктів, а також підходи для створення інтелектуальних агентів на прикладі всесвітньо-відомої, але не менш захоплюючої гри в шахи.

У сучасному світі стрімко розвиваються цифрові технології. Комп’ютери, за законом Мура [1], стають все більш потужними і різноманітними. Наприклад сьогоднішні смартфони є потужнішими за суперкомп’ютери минулого. Цей процес почав розвиватись ще більш спритно з появою глобальної мережі інтернет. Через появу інтернету людство щоденно накопичує і обробляє все більші об’єми інформації. Наразі стають популярними та загальновживаними методи пов’язані зі штучним інтелектом, аналізом даних та машинним навчанням. Більше того ці технології стрімко змінюють індустрію та стають стандартами для підприємств. І виконують ті функції які досі вважались неможливими.

Проєкт також ставить своєю метою порівняння класичних та сучасних методів обробки інформації з відкритим програмним кодом. Бо на сьогоднішній день, більшість подібних розробок мають закриту комерційну ліцензію і через це унеможливлюють вільне користування даними технологіями а також ставлять під сумніви результати свої експериментів та тестів.

Для перевірки цікавлячих напрямів та технологій було обрана загально-відома гра в шахи. Ця гра захоплює розуми людства вже більше як 5 тисяч років. Але за всю багату та різноманітну історію шахів, ця гра досі не має жодних безумовно виграшних стратегій, а також, враховуючи сьогоднішні обчислювальні потужності, не може бути вирішена комбінаторно чи так званим методом brute force [2] (метод «грубої сили»). Але у той же самий час ця гра є детермінованою та не занадто складною для гри як людини так і комп’ютера.

Практичне значення одержаних результатів. Створений застосунок і проведений порівняльний аналіз алгоритмів створення доповненої реальності і створення інтелектуальних агентів для гри в шахи.

Публікації. Результати роботи були опубліковані у матеріалах Всеукраїнської науково-практичній конференції молодих вчених та студентів «Інформаційні системи та технології управління»(ІСТУ-2021).

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

1.1.1 Опис процесу діяльності

В сучасному світі через стрімкий розвиток технологій інтерактивні ігри стають все більш популярними. В багатьох іграх використовуються інтелектуальні агенти як суперники людини. Деякі ці агенти мають простий початковий кол або звід правил, а деякі вже використовують складні сучасні технології для того, щоб виконувати дії які не були зазделегідь прописані в поведінці агента, також такі агенти можуть прогресувати з часов коли вони набирають досвіду гри. Крім того, через розвиток смартфонів, на сьогодняшній день у великої кількості людей є швидкий доступ до камери, що робить можливим використовувати застосунки які засновані на доповненій реальності. Доповнена реальність – це вже не просто технологія яка використовується в дуже вузьких спеціальностях, а технологія яка швидко наберає обертів і починає використовуватись у багатьох сферах людської діяльності. В цьому проєкті були поєднані ці два сучасні тренди. І щоб застосунок був в цей же час простим і зрозумілим, була вибрана всесвітньо відома гра в шахи.

Центральним процесом діяльності є гра в шахи. А головними акторами процесу - гравці. Програмний продукт створює графічний інтерфейс за допомогою Open GL Window, Open CV Window або за допомогою інтерактивних зошитів Jupyter Notebook. Програма отримує зображення з камери або відеопотоку та відображає гравцю картинку з елементами доповненої реальності. Крім того програмний продукт передбачає взаємодію через класичні інтерфейси без використання камери або відео, даний режим роботи дозволяє зручно грати проти штучного інтелекту.

Рисунок 1.1 – Схема роботи застосунку

1.1.2 Опис функціональної моделі

Основним актором системи є користувач. В користувача є можливість обрати режим гри, варіант рендерінгу (двомірна або тривимірна картинка) і обрати інтелектуального агента.

Функціональні вимоги та їх пріоритетність наведені в таблиці 1.1

Таблиця 1.1 – функціональні вимоги

Варіант використання	**Функціональна вимога**	**Пріоритет**
Отримання відеопотоку в реальному часі	Користувач може підключити камеру для взаємодії з застосунком в процесі реального часу	Високий
Отримання відеопотоку з запису	Користувач може підключити заздалегідь записане відео для взаємодії з застосунком в процесі реального часу	Низький
Можливість гри для одного гравця	Користувач має можливість грати з комп’ютерним інтелектуальним агентом	Високий
Можливість гри для двох гравців	Користувач має можливість грати з іншим користувачем, при цьому за бажанням обидва гравці можуть отримати підказки та аналітику гри	Високий

Продовження таблиці 1.1

Взаємодія з базою знань та можливість продовження зацікавленої гри	Користувач має можливість ознайомитись зі списком вже зіграних матчів користувачів різного рівня, та продовжити будь-який матч з довільного моменту.	Середній
Розпізнавання фігур	Користувач має можливість розпізнавання фігур, цей use-case може бути використаний коли гравець хоче перенести вже розпочатий матч із реальної дошки в цифровий формат і наприклад дограти гру пізніше або отримати підказки в деякій “скрутній позиції” гри	Низький

Кінець таблиці 1.1

1.2 Огляд наявних аналогів

Перед початком роботи над дипломним проектом було здійснено пошук аналогічних доданків зі схожою функціональністю.

На сьогоднішній день у відкритому доступі немає прямих аналогів даного застосунку, оскільки зазвичай інші програми виконують лише частину функціоналу програми.

Таблиця 1.2 – проаналізовані аналоги

Назва	**Опис**
AR-Chess	Застосунок який створює доповнену реальність для гри в шахи, використовує спеціальний джойстик для керування.

Продовження таблиці 1.2

Augmented Reality Chess App	Застосунок для смартфонів із закритим вихідним кодом який створяює доповнену реальність для гри в шахи.
chess-alpha-zero	Застосунок який створює нейронну мережу для гри в шахи на основі навчання з підкріпленням.
chess-recognition-single	Застосунок який використовує згорткові нейронні мережі для розпізнавання шахових фігур.

Кінець таблиці 1.2

Розглянемо запропоновані аналоги більш детально.

Застосунок AR-Chess

Даний застосунок створює шахову дошку на місці спеціальної фішки яка ідентифікує ігрову поверхню. А також дозволяє грати проти іншого гравця (на цьому ж самому пристрої). Для гри застосовується інша спеціальна фішка-указник яка керує процесом через часові затримки. Написаний на мові програмування C++.

Характеристики застосунку:

- використовує протокол універсального шахового інтерфейсу (UCI) для зв'язку з шаховим двигуном (наразі використовується Stockfish [3], але можна використовувати будь-який механізм UCI);
- має 3 різні режими гри: «Людина проти людини», «Людина проти комп’ютера» та «Комп’ютер проти комп’ютера»;
- дозволяє рухатись вперед і назад у ході шахової фігури гравця;
- впроваджує та застосовує всі шахові правила;
- показані можливі ходи для шахової фігури, яку гравець хоче перемістити;
- гравець не може робити недійсні ходи;

– використовує ARToolkit для виявлення та відстеження маркерів;
– вся взаємодія з користувачем здійснюється за допомогою двох маркерів (1 для дошки та 1 для курсору);
– використовує OpenSceneGraph для 3D-рендерінгу.

Група застосунків під мобільні платформи Augmented Reality Chess App.

Ці застосунки ближче за всі до комерційного застосування, вони дозволяють грати як проти інших гравців так і проти компьютера (проти класичних алгоритмів), але ці застосунки мають деякі проблеми і недоліки при детекції поверхні і “сповзаючі” фігури.

Для функціонування використовуються мови програмування Swift/Java/Kotlin/C# використовуючи закрите програмне забезпечення ARFoundation [4] та Unity які базуються на пакеті розробки програмного забезпечення ARCore / ARKit для Android / iOS відповідно [5].

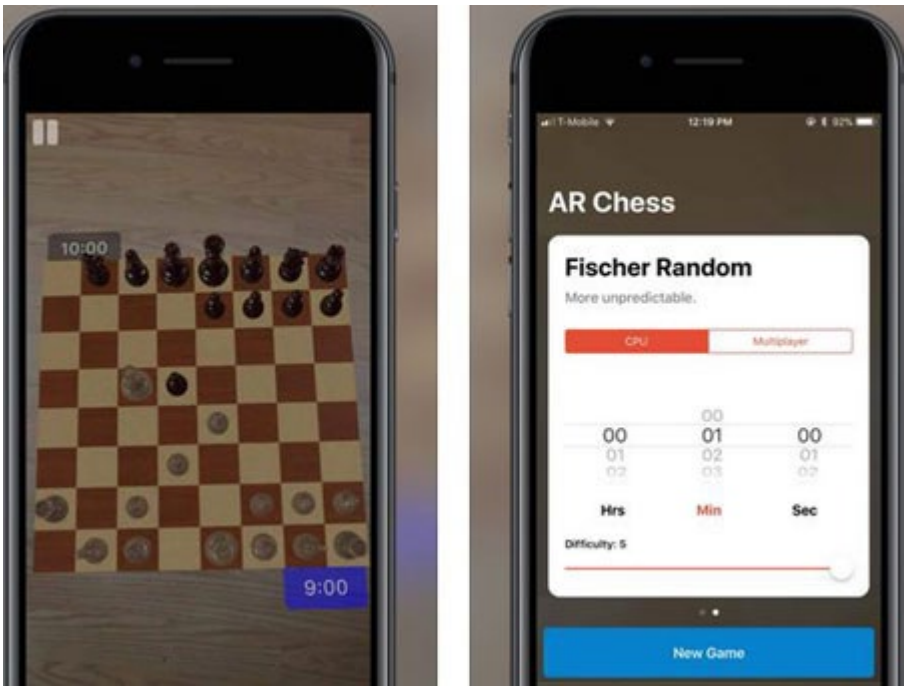


Рисунок 1.2 – графічний інтерфейс застосунку Augmented Reality Chess App

Застосунок chess-alpha-zero

Цей програмний продукт використовує найсучасніші методи для навчання автономних агентів, підходи в застосунку схожі на методи які використовувались компанією Google для створення власного агента який перемагав в кращих гравців і інших агентів сучасності.

Цей проект базується на таких основних ресурсах:

– публікація DeepMind [6] від 19 жовтня: Освоєння гри Go без людського знання та втручання;
– великий розвиток реверсивних ідей DeepMind;
– DeepMind щойно випустив нову версію AlphaGo Zero (названа наразі AlphaZero), де вони опановують шахи з нуля [7];
– AlphaZero перевершив Stockfish всього за 4 години (кроки 300 тис.) Що є неперевершеним результатом для штучного інтелекту в шахах на сьогодняшній день.

Застосунок chess-recognition-single

Даний алгоритм дозволяє детектувати 1 шахову фігуру на зображені використовуючи згорткові нейронні мережі [8]. Цей підхід має високу точність але його не ефективно використовувати для детекції великої кількості фігур, а тим паче відслідковувати їх положення у просторі.

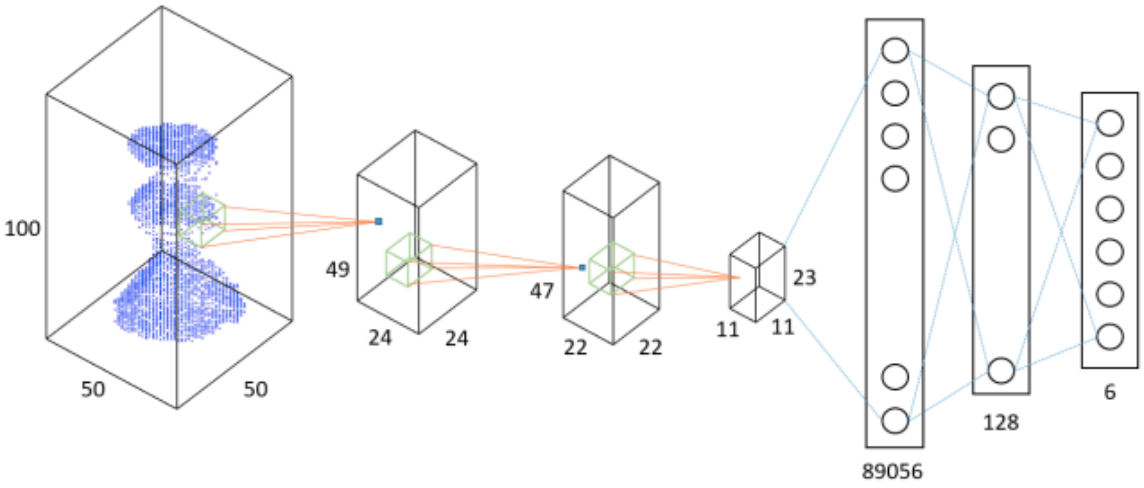


Рисунок 1.3 – архітектура нейронної мережі для детекції фігур

Пропонований спосіб інтегрований у розроблену роботизовану-систему яка спроможна грати в шахи проти людей. Робота має стерео 3D-камеру. Мета проєкту – це забезпечити роботу візуальним інтелектом, щоб він міг ідентифікувати шахові фігури на шаховій дошці з використанням інформації про глибину, зафіксованої 3D-камерою. Автори будують згорткову нейронну мережу для вирішення цієї проблеми розпізнавання 3D-об’єктів.

Хоча навчання нейронних мереж для розпізнавання 3D-об’єктів стає популярним в наші дні, збирається достатня кількість навчальних даних залишається дуже трудомістким завданням. Автори демонструють, що набагато зручніше та ефективніше генерувати необхідні навчальні дані з 3D-моделей САПР. Нейромережа навчена використання відображених даних добре працює на реальних вхідних даних протягом тестування. Крім того, експериментальні результати показують, що використовуючи навчальні дані, отримані з моделей САПР значно підвищують точність розпізнавання. Коли проводиться оцінка точності продукту на реальних даних, зафіксованих 3D-камерою, даний метод досягає точності 90,3%.

Рисунок 1.4 – схема перевірки результатів розпізнавання фігур

Із таблиці 1.3 видно, що аналоги з відкритим кодом, які представлені нині на ринку, не реалізують весь функціонал, які запропоновані в даному дипломному проекті. Крім того в проекті зроблен порівняльний аналіз сучасних технологій і підходів.

Таблиця 1.3 – порівняння аналогів

Функція/Аналог	AR-Chess	Augmented Reality Chess App	Chess-alpha-zero	chess-recognition-single
Детекція об’єктів				+++
Агент для автономної гри		++	+++	
Відображення з доповненою реальністю	+++	++		

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням роботи є забезпечення обробка відеопотоку в реальному часі з додаванням в цей потік елементів доповненої реальності і , за потребою, додавання інтелектуальних агентів для збільшення інтерактивних можливостей гравця в шахи.

1.3.2 Цілі та задачі розробки

Мета: збільшення інтерактивних можливостей гравця в шахи за рахунок додавання елементів доповненої реальності та штучного інтелекту для гри в

шахи, порівняння сучасних та класичних підходів, отримання висновків та рекомендацій для використання.

Для досягнення цілі необхідно:

а) Провести аналіз:
   1) Сучасних алгоритмів детекції об’єктів;
   2) Методів створення доповненої реальності;
   3) Підходів у навчанні автономних інтелектуальних агентів (класичних та сучасних);

б) Розробити модуль доповненої реальності;

в) Розробити модуль штучного інтелекту для допомоги гри в шахи.

Висновок до розділу

В цьому розділі було детально проаналізовано предметне середовище, сформульовані задачі, визначені функціональні вимоги та проаналізовані різницю, недоліки та переваги аналогічних застосунків. В предметному середовищі були визначені та описані процеси діяльності та наведено структурна схема. При пошуку аналогів дипломного проекту були знайдені аналогічні застосунки, але жоден з них повністю не повторює і не порівнює весь функціонал та всі запропоновані методи, також важливим фактором є те, що більшість аналогічних застосунків мають комерційну ліцензію та закритий вихідний код. Крім цього, була сформована постановка задачі і призначення застосунку, мету та задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Програмний продукт може бути умовно поділений на три частини і в залежності від того, який пункт обираємо, то маємо різні вхідні дані.

Для модуля доповненою реальністю вхідними даними можуть виступати:

а) Відеопотік з камери у реальному часі (встроєна або зовнішня камера);

б) Відеопотік збережений заздалегідь у файл (AVI, MP4, MOV);

в) Фотографія збережена у файл (PNG, JPG, RAW).

Обробка єдиного кадру (фотографії) має менше сенсу для роботи з модулем доповненої реальності застосунку, але також має право на існування і може бути оброблена застосунком так само як і окремий кадр відеопотоку.

Для частини з розпізнаванням образів підходить або збережена в файл фотографія або заздалегідь розділеними по фреймам відеопотік. Для частини зі штучним інтелектом є декілька форматів вхідних даних, деякі з яких є шахматними стандартами, а деякі - зручними репрезентаціями зі сторони програмування.

- FEN;
- PGN;
- EPD;
- Chess 960;
- Python Chess Object.

Ці формати можуть зберігатись як в базі даних (в цьому застосунку використовується реляційна база даних PostgreSQL [9]), так і в текстовому форматі (csv, tsv, txt, json) або в бінарному форматі (Apache Parquet [10], Python Pickle). Слід особливу увагу звернути саме на бінарні формати адже Apache Parquet дозволяє зберігати дані більш стисло і пришвидшує пошук по даним,

а Python Pickle дозволяє отримати готовий об’єкт який вже ініціалізований та відразу готовий до роботи. Трохи детальніше про ці формати.

Apache Parquet - це бінарний, стовпчик-орієнтований формат зберігання даних, який дозволяє використовувати переваги стисненого та ефективного колоночного-орієнтованого представлення інформації зокрема швидкий пошук і фільтрацію, а також архівацію даних. Parquet дозволяє задавати схеми стиснення на рівні стовпців і додавати нові кодування в міру їх появи. Перевагами зберігання в даному форматі:

– Економія дискового простору;
– Висока швидкість;
– Можливість реалізації власних схем та кодування даних;
– Підтримка багатьох сучасних мов програмування та фреймворків для обробки даних;
– Простота і зручність роботи з файлами.

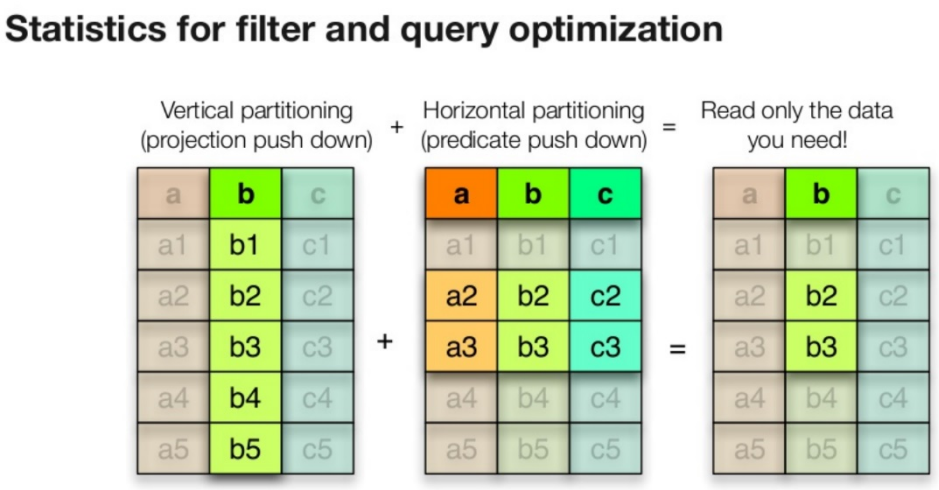


Рисунок 2.1 – візуалізації переваг зберігання даних у Apache Parquet

Якщо розглядати продукт в цілому, то стан дошки зберігається в модулі доповненої реальності, передається інтелектуальному агенту для отримання підказки і повертається назад до модуля доповненої реальності. Python Pickle використовується для зберігання об’єктів Python. Нам не потрібно створювати та змінювати один й той самий об’єкт знову і знову при стартуванні програми. Спочатку створимо об'єкт один раз, а потім збережемо його на диску (pickling),

а пізніше, завантажуємо цей об'єкт з диска (unpickling) без необхідності створювати об'єкт знову. Цей процес дозволяє значно пришвидшити процес ініціалізації деяких об’єктів.

Також, Python Pickle часто використовується в машинному навчанні (яке ми використовуємо для тренування інтелектуальних агентів і для тренування детекторів об’єктів). Модель машинного навчання навчається на дуже великому наборі даних, а навчання моделі займає значну кількість часу. Отже, якщо нам доведеться тренувати одну і ту ж модель, то на це буде витрачатись значна кількість часу. Щоб уникнути повторної зайвої праці, формат Python Pickle стає дуже зручним. Нам потрібно лише один раз навчити свою модель, яку потім можна зберегти на локальний диск, а коли нам потрібно протестувати нашу модель, ми можемо просто завантажити її з диска, не тренуючи її знову.

Оскільки цей формат характерний лише для Python, він не має сумісність з іншими мовами програмування.

2.2 Вихідні дані

Аналогічно до вхідних даних, вихідні дані кожного з модулів застосунку різні.

Для частини з доповненою реальністю вихідними даними є оброблений вхідний відеопотік, на якому нанесені елементи доповненої реальності. Якщо на вхідному відео не будуть розташовані зазначені ключові елементи для детекції поверхні рендерінгу картинки, то вхідний відеопотік буде співпадати з вихідним.

Для частини з розпізнаванням образів, в залежності від режиму використання (для єдиного об’єкту або для множини об’єктів) класифікатор визначить мітку класу фігури та (за потребою) виділить об’єкт на зображені.

Для частини інтелектуальним агентом вихідними даними є рекомендована дія для конкретної ігрової ситуації яка прийшла на вхід агенту.

2.3 Опис структури бази даних

Більша частина даних зберігається на боці інтелектуального агента, але як і пунктах вище розглянемо кожний компонент окремо.

Для модуля доповненої реальності зберігаються наступні елементи:

– 3D моделі шахматних фігур;

– позначки (картинки) шахматних фігур для використання програми у форматі 2D-рендерінгу (цей режим менш ефектний, але зменшує навантаження на апаратну частину комп’ютеру);

– конфігурація з коефіцієнтами калібрації камери, ці дані дуже важливі. Більш детально про них буде розказано пізніше.

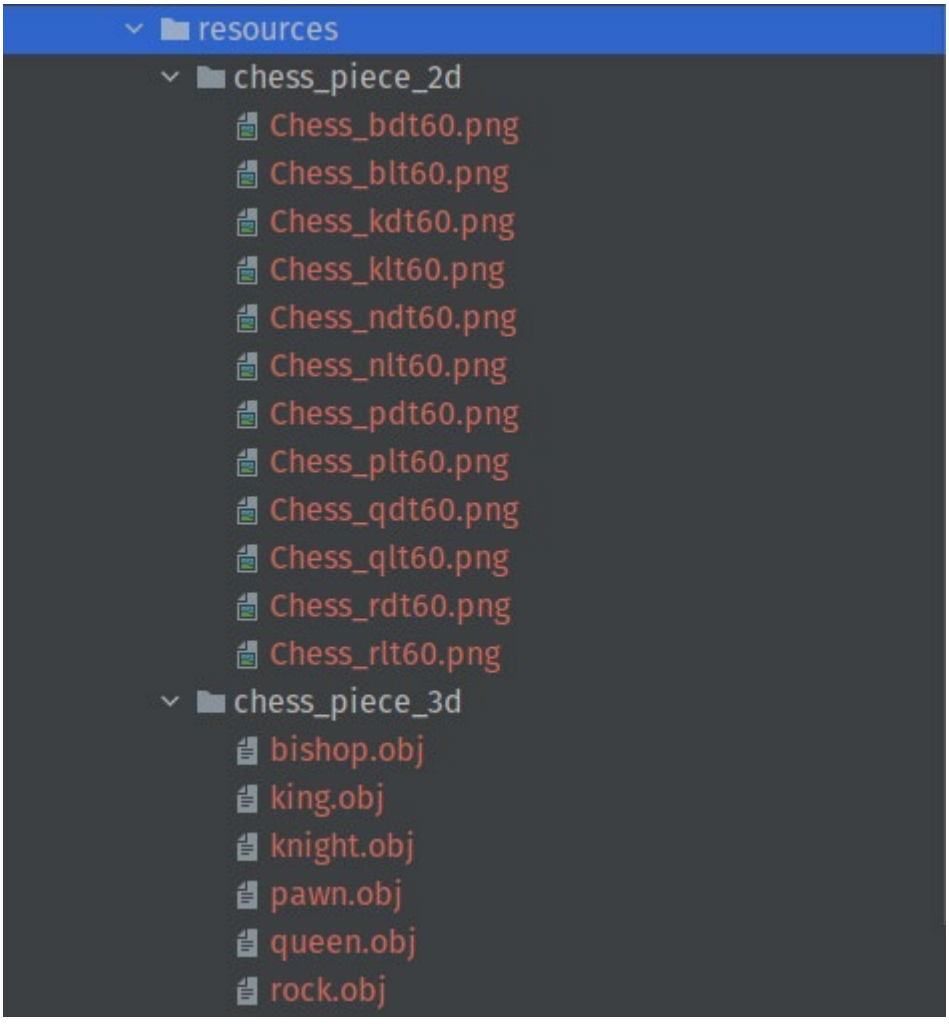


Рисунок 2.2 – список моделей та картинок шахових фігур модуля доповненої реальності

Рисунок 2.3 – приклад файлу з параметрами камери

Для модуля розпізнавання образів, для кожної з моделей будуть зберігатись параметри архітектури моделі, а також в моделях на основі нейронних мереж будуть зберігатись ваги цих моделей [11].

На боці інтелектуального агента зберігається 2 вида даних. По-перше, це ваги та конфігурація для агентів які основані на нейронних мережах. По-друге, це база знань з іграми гравців різного рівня на яких навчались деякі інтелектуальні агенти. Оскільки застосунок потенційно може бути перероблений в повноцінний онлайн сервіс, то база даних має бути OLTP через кількість кількість подій і потенційне навантаження на сервер. Після OLTP бази, в подальшому можно буде додати аналітичне OLAP data warehouse [12]. Крім того, дані які потрібно зберігати системі мають постійну структуру і зв'язки. Через це, я вирішив обрати реляційну базу даних з відкритим вихідним кодом PostgreSQL. Для зручності та ізольованості розвертання бази, була використана технологія легковажних віртуальних машин (контейнерів) Docker.

Docker - програмне забезпечення яке дозволяє автоматизувати управління і розгортання застосунків в середовищах з підтримкою

віртуалізації та контейнеризації. Docker дозволяє пакувати застосунок з усіма зовнішніми залежностями та змінними середовища в так званий “контейнер”, який вже потім розгортається на довільній Unix-подібній системі (наприклад Linux). Але завдяки оптимізації, ізоляції на рівні файлової системи, мережі та ресурсів, спеціальній технології легковажної контерізації, Docker дозволяє одночасно розгорнути навіть на звичайному комп’ютері багато різних контейнерів і ефективно поділяти фізичні ресурси операційної системи між процесорами. Ця технологія на сьогоднішній день є стандартом індустрії і навіть існує ціла архітектура яка дозволяє масштабувати ресурси не тільки вертикально а й горизонтально (створення кластеру з комп’ютерів), називається така технологія оркестрації – kubernetes [13] і вона активно розвивається найпередовішими технологічними компаніями світу і використовується в усіх значних public cloud (Amazon Web Services, Google Cloud, Microsoft Azure) [14].

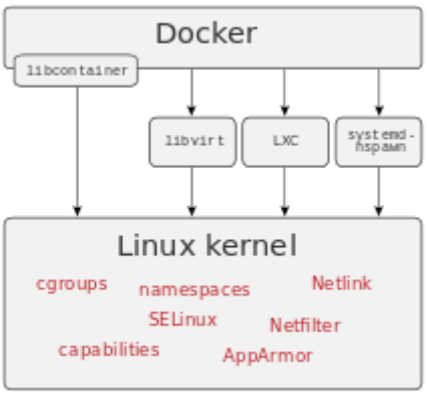


Рисунок 2.4 – основні компоненти технології контейнизації Docker

PostgreSQL - об'єктно-реляційна система керування баз даних з відкритим вихідним кодом, ця система базується на мові SQL та розширює її, крім того вона поєднує її з багатьма нестандартними розширеними функціями, які дозволяють зберігати і масштабувати складні і наватажені системи легко та безпечно.

PostgreSQL має гарну репутацію завдяки своїй надійній архітектурі, цілісності даних, широкому набору функцій, розширюваності та відданості

програмістський спільноти та має відкритий вихідний код, який дозволяє постійно розвивати програмне забезпечення спільнотою, а також притримуватись продуктивних та інноваційних рішень розвитку продукту. Дана система працює на більшості сучасних операційних систем, сама база сумісна з ACID [15] концепцією та має потужні користувацькі розширення, такі як популярний додаток геопросторових баз даних PostGIS.

Тепер, коли обґрунтований вибір бази даних та подальші кроки до масштабування і розширення цієї бази, розберемось зі схемою бази. Для сумісності бази даних дипломного застосунка з іншими базами, було обрано наступний денормалізований формат бази даних. Ця база денормалізована через концепції Big Data рішень і сумісність з No SQL розподіленими базами даних [16]. Таку схем можливо було б представити в нормалізованому форматі, але таблиця стрімко змінюється і через вимоги до консистентності та швидкості доступу до бази, нормалізувати її не потрібно.

Рисунок 2.5 – денормалізована схема бази даних

Якщо згрупувати поля бази даних, то їх умовно можливо поділити на наступні компоненти:

– інформація про гру, рахунок в партії, редакція правил та тип поєдинку;
– інформація про гравців (ім’я, рейтинг, відсоток перемог, колір шахів, тощо);
– інформація про поточний стан гри (позиція на дошці, кількість ключових фігур, чий хід, інформація про час на хід).

Рисунки 2.6, 2.7 та 2.8 містять приклади зберігаємих даних.

game_id	type	result	white_player	black_player	white_elo	black_elo	time_control	num_ply	termination
5 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
6 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
7 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
8 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
9 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
10 YZ7jXSLk	Blitz	0-1	Daniel-mate	Peredo64	1154	996	300+3	26	Normal
11 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
12 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
13 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
14 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
15 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
16 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
17 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
18 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
19 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal
20 nztBrDvH	Blitz	1-0	NAPTOWNDRKNITE	parisboy	1915	1565	180+0	47	Normal

Рисунок 2.6 – приклад бази даних з інформацією про матч та гравців

Рисунок 2.7 – приклад бази даних з порівняльною характеристикої гравців

Рисунок 2.8 – приклад бази даних з інформацією про поточну партію

2.4 Аналіз отриманої бази даних

Для більшого розуміння природи даних та отримання деяких закономірностей в даних, було проведено так званий EDA (дослідницьких аналіз даних) [17].

З рисунку 2.9 видно, що через стрімкий розвиток технологій зокрема інтернету та мобільних пристроїв (смартфони та ноутбуки) люди мають тенденцію грати в шахи все більше. Як видно на графіку кількість матчів росте експоненційно. Цей факт ще раз підтверджує актуальність і сучасність даного дослідження.

Рисунок 2.9 – залежності року від кількості зіграних партій

Наступний графік (рисунок 2.10) демонструє розподіл по різним режимам гри в шахи. І несподівано виявляється, що бліц-шахи є найбільш популярними.

Рисунок 2.10 – залежності режиму гри від кількості зіграних партій

Рисунок 2.11 показує розподіл по завершенням матчів. І не дивлячись на те, що вважається що гра в шахи збалансована по кольору, білі виграють набагато частіше. Це видно навіть неозброєним оком, але це також підтверджує статистичний тест Стьюдента з 95% рівнем значущості.

Рисунок 2.11 – гістограма завершень гри

Наступний графік показує розподіл рейтингових очок серед гравців. Це допоможе пізніше при навчанні та оцінках інтелектуальних агентів.

Рисунок 2.12 – гістограма рейтингу гравців

Далі йде графік який показує розподіл різниць в рейтингах гравців. Для нас цей графік важливий через те, що ми бачимо що зазвичай, між собою грають суперники близького рангу, отже система зможе рівноцінно аналізувати ходи як чорних та і білих, що допоможе при навчанні інтелектуального агента.

Рисунок 2.13 – гістограма різниці рейтингу між білим та чорним гравцем

Порівняння розподілів кількості ходів гравців в залежності від того, як закінчився матч (перемога по часу або класична перемога). Як видно на рисунку 2.14, затяжні матчі мають тенденцію закінчуватись по часу, а отже не слід навчати агента на кінцівках подібних матчів через те, що через брак часу, гравці часто починають робити помилки.

Рисунок 2.14 – Залежність довжини партії від причини кінця партії

Наступний рисунок показує, що незалежно від режиму гри, довжина гри в кількості ходів має єдиний розподіл. Отже наш агент зможе однаково гарно грати в будь-який з режимів незважаючи на те, на чому він був натренований.

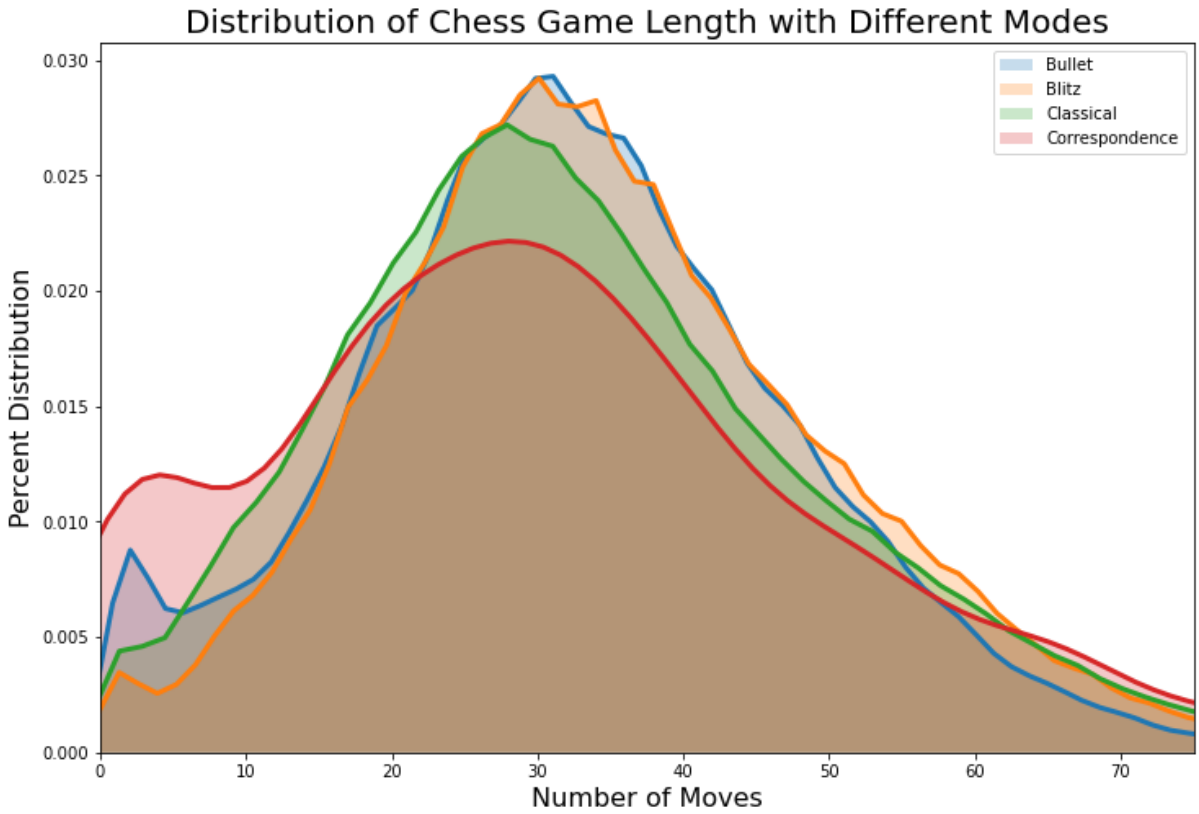


Рисунок 2.15 – Розподіл довжини партії в залежності від режиму гри

На наступному графіку показується залежність рейтингу гравця від кількості зроблених ходів. Цей графік може бути інтерпретований по різному, одна з інтерпретацій:

– професійні гравці (2250+ elo) здраво оцінюють позицію на дошці і у випадках коли вони розуміють, що партія буде закінчиться їх поразкою то вони визнають свою поразку і не сподіваються на помилку суперника;

– гравці високого рівня (1750 - 2250 elo) дуже цінують свій рейтинг та грають до кінця, навіть якщо позиція або кількість фігур не на їх користь;

– гравці початкового та середнього рівня грають до тих пір поки перевага суперника не стає занадто великою.

Рисунок 2.16 – Розподіл довжини партії в залежності від рейтингу гравців

Наступні 2 графіки показують розподіл ходів загалом та в залежності від стадії партії. В цілому для людини яка грає в шахи стає зрозумілим чому гра має саме такий розподіл, але дана інформація також може стати в нагоді при порівнянні з розподілом ходів інтелектуальних агентів для визначення їх сильних та слабких сторін.

Рисунок 2.17 – Середня кількість ходів фігуру в залежності від типу фігури

Рисунок 2.18 – Ймовірність ходу фігури в залежності від етапу партії

Останній блок теплових-карт показує частоту знаходження фігур в різних полях шахової дошки. Дана інформація стає дуже корисною при проблемі “ледачого початку” в метричних алгоритмах. Про цю проблему буде розказано більш детально пізніше.

Рисунок 2.19 – Теплова карта позицій білого та чорного короля

Рисунок 2.20 – Теплова карта позицій білої та чорної королеви

Рисунок 2.21 – Теплова карта позицій білого та чорного коня

Рисунок 2.22 – Теплова карта захвату фігури опонента

Рисунок 2.23 – Теплова карта об'явлення шаху опоненту

Висновок до розділу

В цьому розділі розповідалось про вхідні та вихідні дані кожного компонента застосунку і комплексу в цілому. Більш того, було детально розказано про формати даних і аргументацію по їх застосуванню. Крім того, були зазначені стратегії подальшого розвитку продукту з точки зору інформаційного забезпечення, а також потенційну інтеграцію з іншими програмами та сервісами пов’язаними з грою в шахи.

В кінці розділу було детально розказано і проаналізовано базу даних і як ці знання можуть бути використані як для навчання інтелектуальних агентів так і для аналізу їх дій на етапі тестування.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

В ході даної роботи розглядається три задачі, кожна з яких має свої цілі та метрики ефективності.

Задача створення доповненої реальності

Необхідно обробити відеопотік у процесі реального часу, виконти детекцію потрібних об’єктів (поверхня для відображення шахової дошки), порахувати правильні проекції на поверхню та відобразити доповнене зображення користувачу.

Задача інтелектуального агента

Отримати інформацію про поточний стан гри (положення фігур на дошці, формат гри та модифікацію правил), в залежності від обраного агента (додаток має предоставляти декілька модифікацій агентів) зробити умовно оптимальний хід який при цьому не суперечить правилам гри. Час на кожний хід визначається нефункціональними вимогами застосунку, але повинен забезпечувати гру у форматі реального часу.

Задача детектора позиції на дошці

Детектор на вхід має отримувати фотографію з шахматною дошкою і на вихід віддавати розмітку з позиціями фігур на дошці. Метрики точності та відкликання детекції також мають бути заданими нефункціональними вимогами застосунку.

3.2 Математична постановка задачі

3.2.1 Задача створення доповненої реальності

Дано: Відеопотік кожний кадр якого **x** розглядається як окреме растрове зображення висоти **h** та ширини **w** яке має 3 або 4 канали, RGB, Hex, HSV або RGBA (кольорове зображення з альфа каналом, альфа канал створює ефект часткової прозорості зображення) відповідно. Зображення може мати об’єкт

детекції (образ проектування) з множинами координат x_H та x_w, координати об'єкта x_H та x_w не є відомими для систему і мають бути визначені нею самостійно. Пари координат множин x_H та x_w формують полігон довільної заздалегідь визначеної форми, але в більшості випадків цей полігон є квадратом (тобто n = 2).

Знайти: аналогічне зображення висоти h та ширини w з відповідною кількістю каналів зображення і наявними додатковим зображенням (за умовою присутності образу проектування на кадрів) на полігоні з координатами x_H та x_w.

$$x_H = \{h_1, h_2, \dots h_n\}, n \in \mathbb{N},$$
$$x_W = \{w_1, w_2, \dots w_n\}, n \in \mathbb{N}$$

Цільова функція для задачі детекції образу проектування:

$$F(x) = \sum_{i=1}^{n} ((x_{h_i} - \widehat{x_{h_i}})^2 + (x_{w_i} - \widehat{x_{w_i}})^2) \rightarrow min$$

де:

x - поточний кадр;

x_{h_i} – ордината і-ої вершини полігона;

x_{w_i} - абсциса і-ої вершини полігона;

$\widehat{x_{h_i}}$ - передбачена детектором ордината і-ої вершини полігона;

x_{w_i}- передбачена детектором абсциса і-ої вершини полігона;

Тобто значення між реальними та передбаченими координатами об'єкта мають мінімізуватись.

3.2.2 Задача створення інтелектуального агента

Дано: Поточний стан шахової дошки. Маємо дошку розміру 8 х 8 (теоретично можливо використовувати дошки іншого розміру і агент все одно зможе знаходити умовно оптимальні ходи, але в даному застосунку такі

випадки не розглядалися). Кожна клітина дошки може приймати один з тринадцяти станів:

$k \in$ {Пуста клітина, Білий пішак, Білий кінь, Білий слон, Біла тура, Білий король, Біла королева, Чорний пішак, Чорний кінь, Чорний слон, Чорна тура, Чорний король, Чорний королева}

Дошка є комбінацією клітин при наступних обемежнях:

– кількість пішаків кожного кольору $\leqslant 8$;
– пішаки не можуть знаходитись на першому або останньому рядку;
– сумарна кількість фігур $\leqslant 16$ кожного кольору;
– на дошці присутні по одному королю кожного кольору;
– в стані шаху одночасно може знаходитись лише один король.

В даних обмеженнях немає обмежень на кількість слонів, конів, тур та королев через правило: при умові, що пішак дошовши до кінця дошки, він може перетворитися на будь-яку з перелічених згори фігур.

Розрахувати кількість можливих позицій в шахах до кінця комп'ютер не в змозі, оскільки число можливих позицій становить 10^{46} позицій.

Знайти: Умовно оптимальний хід який не суперечить правилам гри.

3.3 Опис методів розв’язання

3.3.1 Задача створення інтелектуального агента

Класичні методи

В цій роботі класичними називаються метричні методи для створення штучного інтелекту в іграх, цей підхід оцінює можливі ходи і їх наслідки і потім обирає найкращий, або для того щоб комп’ютер ходив по різному (коли це не критично) в однакових ситуаціях то інтелектуальний агент робить кроки з певною ймовірністю яка пропорційна значенню метрики ходу, прикладом такої ймовірнісної функції може стати функція Softmax [23].

Розглянемо деякі підходи, на яких базується обрахування метрики певної позиції та вплив ходу агенту на цю метрику.

Першим розглянемо мінімакс алгоритм. Це алгоритм, який використовується для визначення метрик (рахунку) в іграх з нульовою сумою [24], для найкращої гри відповідно до функції оцінки. Іншими словами алгоритм полягає в наступному: для обраного гравця розглядається множина доступних йому ходів і обраховується метрики цих ходів (будемо вважати, що маємо задачу максимізації), після цього дня кожної отриманої вершини ми знову ж розглядаємо множину ходів і обраховуємо метрику але в цей раз з протилежним знаком (через те, що інший гравець також бажає виграти та робить максимально оптимальні ходи). Таким чином ми отримуємо дерево з черугуючимися мінімумами та максимами серед яких ми обираємо найкращий хід, висота дерева є гіперпараметром і, зазвичай, для шахів приймає невелике значення від 2 до 4 оскільки кількість вершин зростає експоненційно і ми швидко упираємось в обмеження обчислювальних ресурсів. Мінімакс алгоритм візуалізовано на рисунку 3.1.

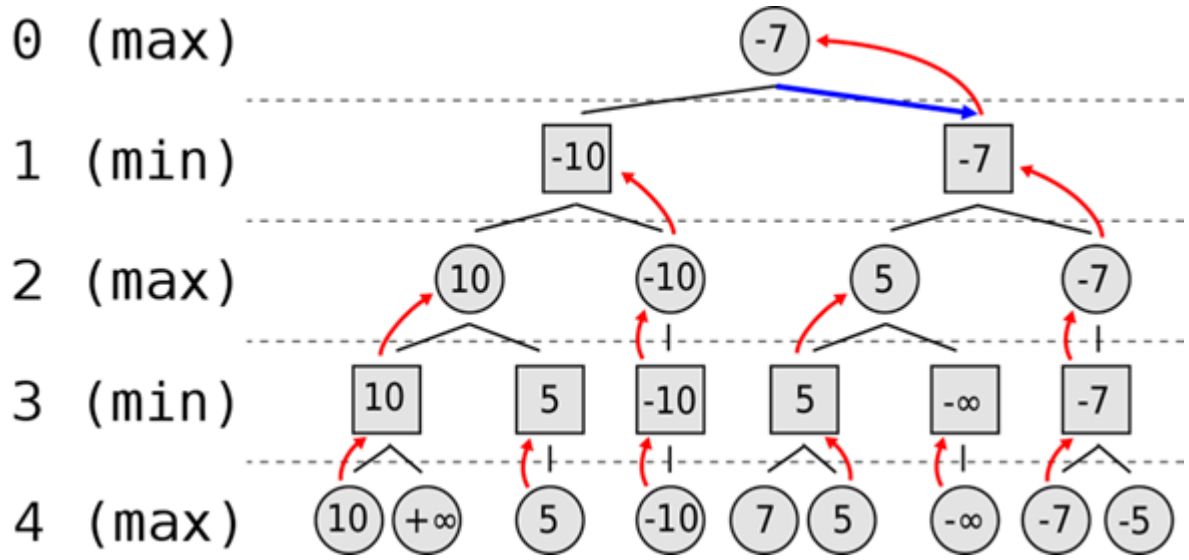


Рисунок 3.1 – візуальне представлення мінімакс алгоритму

Часто замість мінімакс алгоритму використовується схожий на нього негамакс алгоритм [25 – 26], цей алгоритм має цю ж саму ідею метричного пошуку в ширину, але базується на формулі:

$$max(a, b) = -min(-a, -b)$$

Мінімакс алгоритм знаходить метрично оптимальні ходи, але є дуже повільним методом. Для того щоб його пришвидшити використовується альфа-бета відсікання. Він позбавляє потреби прораховувати великі частини ігрового дерева, відсікаючи неперспективні ходи і не обраховуючи ці гілки до кінця (аналогічно до метод гілок та меж [27]). Алгоритм зберігає значення двух параметрів - альфа та бета. Вони є мінімальним та максимальним значенями метрики які забезпечені гравцеві який має стратегію максимізації і мінімізації, відповідно [28].

На рисунку 3.2 продемонстрована робота альфа-бета обрізання. Сірі піддерева не потрібно досліджувати (в даному випадку розрахунок був зліва направо), оскільки відомо, що група піддерев в цілому дає значення еквівалентного піддерева або гірше, і як така є неоптимальною.

Рисунок 3.2 – візуальне представлення альфа-бета відсікання

Економія обчислювальних ресурсів при використанні альфа-бета відсікання може бути значною. Якщо в стандартному мінімакс-дереві пошуку є X вузлів, альфа-бета-дерево у добре написаній програмі може мати кількість вузлів, близьку до $\sqrt{X}$. Однак скільки вузлів зможете відкинути алгоритм залежить від впорядкованості дерева пошуку. Якщо вдасться знайти найкращий можливий рух спочатку, то алгоритм відразу вилучить більшість неоптимальних вузлів. Звичайно, ми не завжди знаємо, який хід є найкращим, інакше нам б не довелось шукати його. Цей ефект працює і навпаки, якщо ми завжди шукали гірші ходи перед кращими, то алгоритму б не вдалось би

відкинути жодної безперспективної гілки дерева. Через це, якщо намагатись ідеалізувати цей підхід, то потрібно розробити алгоритм для отримання оптимального порядку обходження вершин. В цьому напрямку було проведено декілька досліджень які були спрямовані на написання оптимальної системи обходу дерева пошуку і виведення деяких правил [29] [30].

Згідно одного з цих правил була виведена таблиця (таблиця 3.1), з b=40 (середня кількість можливих ходів) економії, яка демонструє різницю між кількістю перевірених станів мінімакс алгоритму та апроксимовану кількість перевірок для альфа-бета відсікання.

Таблиця 3.1 – порівняння кількості метрик які необхідно прорахувати в залежності від використовуємого алгоритму

Глибина n	**Мінімакс b^n**	**Альфа-бета $2b^{\frac{n}{2}} - 1$**
0	1	1
1	40	40
2	1600	79
3	64,000	1639
4	2,560,000	3199
5	102,400,000	65,569
6	4,096,000,000	127,999
7	163,840,000,000	2,623,999
8	6,553,600,000,000	5,119,999

Після того як ми зрозуміли як можна оптимізувати метрики, розглянемо як задавати саму оцінку поточного стану гри. Перше що приходить на думку це підрахунок кількості фігур на кожному боці, оцінити кожну фігуру в певну кількість балів та просумувати бали кожного з гравців [31], але тоді виникає питання, в яку кількість балів оцінити кожну фігуру, чи залежить кількість балів за фігуру від стадії гри та від конкретних гравців [32]? Відповідь – безумовно так. Але в той самий час, ми не хочемо дуже багато часу

підраховувати метрику кожної позиції, тому що кожний хід ми повинні оцінити щонайменше 2000 різних позицій. Існують «класичні» оцінки вартості кожної фігури (таблиця 3.2).

Таблиця 3.2 – вартість шахових фігур

Фігура	**Вартість**
Пішак	100
Кінь	300
Слон	300
Тура	500
Королева	900
Король	∞

Іноді виникають питання чому конкретна фігура отримала певну вартість. Для цього був проведений незалежний регресійний аналіз для виявлення вартості кожної з фігур [33]. Результати експерименту продемонстровані на рисунку 3.3.

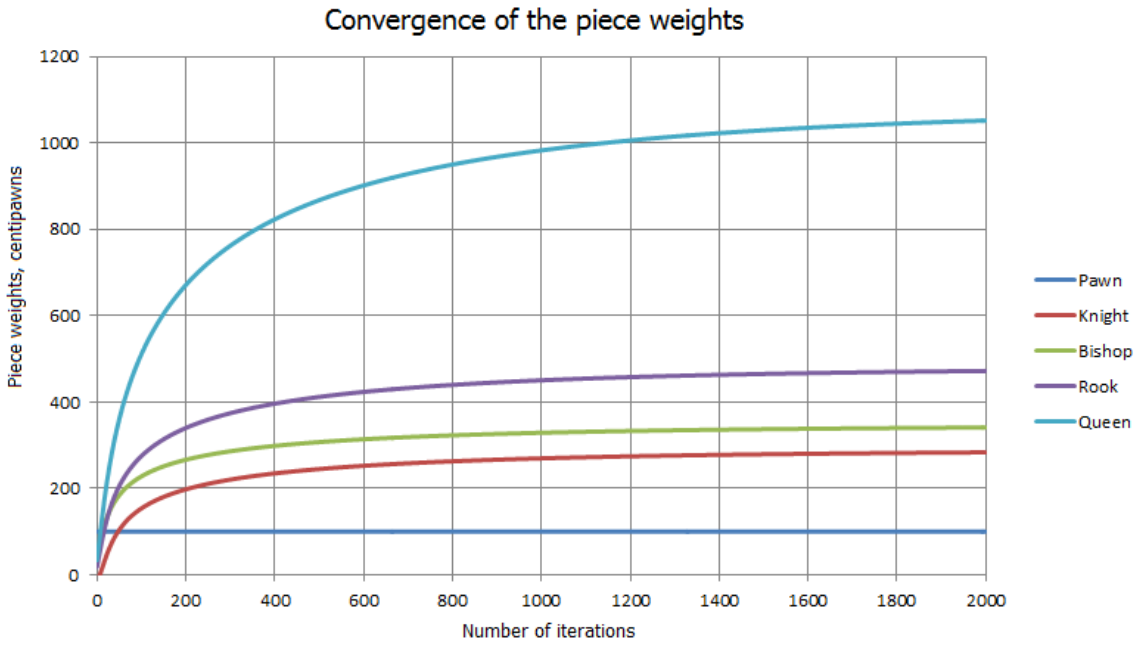


Рисунок 3.3 – графік регресійної оцінки шаховаих фігур

Після нормалізації та округлення отримали нові значення оцінок фігур які продемонстровані в таблиці 3.3.

Таблиця 3.3 – вартість шахових фігур

Фігура	**Оцінка вартості**
Пішак	100
Кінь	288
Слон	345
Тура	480
Королева	1077
Король	∞

Провалідуємо отримані значення за правилами Капабланка [34] таблиця 3.4. В цій таблиці використовуються назви фігур по англомовному скороченню. B – слон, N- кінь, P – пішак, R – тура, Q – королева.

Таблиця 3.4 – вартість шахових фігур

Умова	**Числове значення**	**Предикат**
B > N	345 > 288	Виконується
B > 3P	345 > 300	Виконується
N > 3P	288 < 300	Ні
B + N = R + 1.5P	345 + 288 = 480 + 150	Виконується (похибка < 0.5%)
Q + P = 2R	1077 + 100 > 960	Ні
B > N	345 > 288	Виконується

Результат досить вражаючий. Не знаючи нічого про фактичні події, що відбуваються на дошці, враховуючи лише результат гри, алгоритм зумів наблизити значення балів до загальноприйнятих значень.

Але кількість фігур не завжди є ключем до перемоги, в шахах дуже велике значення має позиція фігур. Дана метрика є дуже складною через те, що вона є дуже комплексною і залежить від стилю гри і рейтингу опонентів

[35]. Тем не менш, є приблизні оцінки положення кожної фігури, наприклад слон є більш корисним коли він стоїть на одній з центральних діагоналей, але майже зайвий коли він в кутку. Приклад подібних таблиць оцінки положення фігур показаний на рисунку 3.4.

Рисунок 3.4 – теплова мапа положення для короля

Сучасні підходи

В цьому блоці ми будемо досліджувати сучасні “state of the art” (англ. витвір мистецтва, тобто найкращі практики) підходи для навчання інтелектуальних агентів. Таким підходом на сьогоднішній день є навчання з підкріплення (reinforcement learning) [36].

Основні поняття навчання з підкріпленням:

– агент (Agent) - сутність, яка приймає рішення;
– середовище (Environment) - світ, який спостерігає агент;
– спостереження (Observation або State) - поточний стан агента;
– лінія поведінки (Policy) - деяке відображення зі спостережень в дії;
– дія (Action) - вчинок (дія), що здійснюється агентом згідно з лінією поведінки;
– відгук системи (Feedback, або reward) - нагорода за дії (може бути негативною).

Зауважимо, що в більшості випадків, коли використовується навчання з підкріпленням, то йде мова про машинне навчання. В свою чергу машинне навчання поділяється на три типи: з вчителем, без вчителя, з підкріпленням. Нижче представлено порівняння різних видів навчання.

Таблиця 3.5 – Порівняння методів машинного навчання

З вчителем (supervised)	**Без вчителя (unsupervised)**	**З підкріпленням (reinforced)**
вчимося передбачати релевантні відповіді	намагаємося виявити базову структуру даних	шукаємо оптимальну стратегію методом проб і помилок
потрібні правильні відповіді	не потрібні ані відповіді, ані зворотній зв'язок	агенту потрібен зворотній зв'язок на його власні дії
модель не впливає на вхідні дані	модель не впливає на вхідні дані	агент впливає на середовище, а значить і на спостереження

Тепер перейдемо до формального опису задачі:

– на кожному кроці маємо стан s ∈ S, дія a ∈ A і нагороду r ∈ R;

– хочемо максимізувати сумарну нагороду $R = \sum_t r_t \to max$;

– введемо політику π (a | s) = P (обрання дії a у стані s);

– тоді задача зводиться до знаходження $\pi * = arg\ max_\pi E_\pi[R]$.

Виникає питання як максимізувати даний функціонал? Навчання подібних агентів є ітеративним:

– граємо кілька ігор (сесій) по існуючій лінії поведінки (політику);

– оновлюємо лінію поведінки згідно з отриманим зворотнім зв'язком;

– повторюємо процедуру.

Таким чином, завдання навчання з підкріпленням полягає в тому, щоб знайти оптимальну стратегію агента, максимізуючи нагороду. Але такий підхід не завжди є оптимальним, іноді агент прагнучи отримати максимальну можливу винагороду заганяє себе до локального максимума.

Метод перехресної ентропії (Cross entropy method)

Розглянемо для початку випадок так званих табличних даних, коли є кінцеве число станів і кінцеве число дій:

- ініціалізуємо лінію поведінки (в нашому випадку нею буде матриця розміру n станів x n дій);
- починаємо N довільних сесій, на яких буде грати агент;
- з них вибираємо M елітних сесій (тобто таких, на яких було отримано найкращі результати);
- оновлюємо лінію поведінки на основі цих елітних сесій;
- повторюємо процедуру.

Якщо π (a | s) = P (здійснюємо дію a в стані s), тоді:

$$\pi_{\text{нове}}(a \mid s) = \frac{\text{скільки разів обрали дію } a \text{ в стані } s}{\text{кількість разів в стані } s}$$

На жаль табличний підхід не працює при великій кількості станів або дій, тому що кількість елементів матриці стане занадто великим і оптимізувати політику буде не ефективно. В наближеному методі крос-ентропії алгоритм знаходження оптимальної політики приблизно такий ж, як і вищеописаний, тільки замість матриці ймовірностей використовується будь-яка відповідна модель класифікації, наприклад логістична регресія або random forest classifier. Тобто $\pi_{\text{нове}}(a \mid s) = f_{\Theta}(a, s)$. Якщо простір дій нескінчений, а континуально, тоді можна або вирішувати завдання регресії, або дискретизувати простір відповідей.

В нашому випадку в шахах дуже багато станів, отже метод перехресної ентропії не буде ефективним для наших агентів.

Функція стану-значення, Q-функція

Недоліком методу перехресної ентропії є те, що при виборі дій ми використовуємо тільки розподіл ймовірностей, не враховуючи інформацію про зовнішнє середовище. Після цього дивимося на отриманий результат, обираючи кращі сесії (умовно кажучи, агент діє навмання сподіваючись знайти оптимальну стратегію). Але як пояснити агенту, чого від нього хочуть? У цьому нам допоможе гіпотеза винагороди Саттона: *наші цілі і завдання можна розуміти як максимізацію математичного очікування кумулятивної суми одержуваних скалярних величин, які ми називаємо нагородою.*

Нехай:

$$G_t = R_t + R_{t+1} + \ldots + R_T$$

- та сама кумулятивна сума нагород аж до кінця сесії. У разі, якщо сесія триває до нескінченності, сумарна нагорода також буде необмежена. Тому будемо вважати її з деяким регуризаторним коефіцієнтом γ. Тоді:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \ldots) = R_t + \gamma G_{t+1}$$

Насправді регуляризована нагорода дозволяє враховувати також вплив подій в довгостроковій перспективі (тобто певний ефект доміно). Тоді формула інтерпретується наступним чином:

$$G_0 = R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots + \gamma^T R_T =$$

$$= (1-\gamma)R_0 + (1-\gamma)\gamma(R_0 + R_1) + (1-\gamma)\gamma^2(R_0 + R_1 + R_2) + \ldots + \gamma^T \sum_{t=0}^{T} R_t$$

де коефіцієнт γ^t означає ймовірність, що дії які вже відбулись вплинуть на подію кроку t, а коефіцієнт (1 - γ) - ймовірність, що ефект скінчиться на поточному кроці.

Нагадаємо, що тепер оптимальна політика максимізує наступну величину:

$$E_{\pi\Theta}[G_0] = E_{\pi\Theta}[R_0 + \gamma R_1 + \ldots + \gamma^T R_T]$$

Задамо функцію корисності стану:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[R_t + \gamma G_{t+1} | S_t = s] =$$

$$= \sum_a \pi(a|s) \sum_{r,s`} p(r, s`|s, a)[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s`]] =$$

$$= \sum_a \pi(a|s) \sum_{r,s`} p(r, s`|s, a)[r + \gamma v_\pi(s`)]$$

Математичне очікування береться за тими величинам, які вносять весь вклад в випадковість: за лінією поведінки π (визначає здійснювану дію) і по середовищі p (визначає нагороду за крок і наступний стан).

Завдяки функції v (s) ми знаємо, які стану хороші, а які ні. Але ця інформація не допоможе нам визначити стратегію. Тому введемо функцію корисності дії:

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a] = E_\pi[R_t + \gamma G_{t+1} | S_t = s, A_t = a] =$$

$$= \sum_{r,s`} p(r, s`|s, a)[r + \gamma E_\pi[G_{t+1} | S_{t+1} - s`]] = \sum_{r,s`} p(r, s`|s, a)[r + \gamma v_\pi(s`)]$$

Неформально, q-функція показує, скільки ми будемо в середньому отримувати кумулятивної нагороди, перебуваючи в стані s і здійснюючи дію a.

У підсумку ми висловили q-функцію через state-value функцію, яка, в свою чергу, виражається через q-функцію:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

Отримавши цю формулу можемо чисельно порівнювати лінії поведінки систем:

$$\pi \geq \pi` \Leftrightarrow v_\pi(s) \geq v_{\pi`}(s)\ \forall s$$

Оптимальна лінія поведінки π^* - це така лінія поведінки, яка краще або дорівнює будь-який інший. Якщо ми використовуємо оптимальну політику зі стану s, то $v_*(s) = max_\pi\ v_\pi$ і $q_*(s,a) = max_\pi\ q_\pi(s,a)$.

Тепер для фунції $v_*(s)$можно отримати рекурсивні співвідношення через $v_*(s`)$:

$$v_*(s) = max_a \sum_{r,s`} p(r,s`|s,a)[r + \gamma v_*(s`)]$$

Пояснення: оскільки нам відомо, в які стани s’ можна перейти зі стану s, то в наведеній вище функції v(s) максимум береться по другий сумі (умовно кажучи, ймовірність переходу в найкраще стан s’ буде дорівнює одиниці).

Використовуючи динамічне програмування, можна знайти v-функцію для всіх станів (за теоремою про нерухому точку процес оновлення значень функції повинен збігатись). І потім оптимальна лінія поведінка виражається за допомогою q-функції наступним чином:

$$\pi_*(s) = arg\ max_a\ q_*(a,s)$$

Треба звернути уваги, що тут лінія поведінки, на відміну від методу перехресної ентропії, однозначно визначає дію для конкретного стану s. Для певної стохастичности можна ввести поріг епсілон, при досяжності якого дія яка буде виконуватись обирається випадково, або використовувати не максимум, а температурний softmax для отримання розподілу ймовірностей.

Q-навчання, приблизне Q-навчання, DQN

Попередній підхід містить декілька проблем. Для початку що робити, якщо нам не відомі параметри середовища p(r, s'| s, a)? Як варіант можна згенерувати багато сесій і використовувати апроксимацію вибіркою. Але в такому випадку багато часу піде на те, щоб зіграти цілу сесію, причому агент буде робити це з одним і тим же досвідом, який буде оновлюватись лише в кінці гри. Тоді зробимо так: будемо оновлювати q-функцію на кожному кроці агента.

$$Q(s_t, a_t) = E_{r_t, s_{t+1}}[r_t + \gamma * max_{a`} Q(s_{t+1}, a`)]$$

$$\approx \frac{1}{N} \sum_i [r_i + \gamma * max_{a`} Q(s_i^{\text{наступне}}, a`)]$$

Щоб не загубити інформацію про минулий досвід - будемо використовувати ковзане середнє.

$$Q(s, a) = \alpha * \hat{Q}(s, a) + (1 - \alpha) * Q(s, a),$$
$$\hat{Q}(s, a) = r(s, a) * \gamma \cdot max_{a`} Q(s`, a`)$$

Спосіб навчання, де в оновленні q-функції використовується взяття максимуму, називається Q-learning.

В іншому алгоритмі навчання - SARSA - береться мат.очікування.:

$$\hat{Q}(s, a) = r(s, a) * \gamma \cdot E_{a`} Q(s`, a`)$$

Experience replay. Ще одне поліпшення полягає в використанні буфера з накопиченим досвідом. Ідея полягає в тому, щоб оновлювати q-функцію агента не тільки по поточному стану, але і за попередньою історією. Для цього збережемо взаємодію агента з середовищем (тобто четвірку s, a, r, s`). Після цього візьмемо частину історії з буфера і оновимо на ній q-функцію. Такий

підхід дозволяє нам враховувати досвід, не рахуючи при цьому новий результат взаємодії, що допомагає збігатися швидше.

Approximate Q-learning. Згрупуємо складові у формулі поновлення q-функції:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * (r_t + \gamma \, max_{a`} \hat{Q}(s_{t+1}, a`) - Q(s_t, a_t))$$

Отримана формула нагадує градієнтний спуск (вірніше підйом, так як стоїть завдання максімізації). Тоді можна розрахувати середньоквадратичну помилку між старим значенням $Q(s_t, a_t)$ і новим:

$$L = (r_t + \gamma \, max_{a`} Q(s_{t+1}, a`) - Q(s_t, a_t))^2,$$

$$\frac{\partial L}{\partial Q} = 2 * (r_t + \gamma \, max_{a`} Q(s_{t+1}, a`) - Q(s_t, a_t))$$

Зауважимо, що в даному випадку ми не беремо похідну по $max_{a`} Q(s_{t+1}, a`)$і у цього є пояснення: по суті $max_{a`} Q(s_{t+1}, a`)$відповідає за наступні стани і якщо брати по ній похідну, то ми будемо підганяти q-функцію під значення самої себе в майбутньому.

Якщо прийняти, що наша q-функція - це деяка параметрична модель з матрицею ваг W, то на кожному кроці будемо оновлювати ваги за правилом:

$$w_{t+1} = w_t + \alpha \frac{\partial L}{\partial w}$$

Deep Q-network. Нехай нам потрібно вирішити задачу навчання з підкріпленням в деякому середовищі, яке містить просторову інформацію, наприклад переміщення об'єкта. Тоді корисно зберігати послідовність станів, за якою ми зможемо витягти зміни. З цим впорається будь-яка згорткова нейронна мережа: на її вхід подається послідовність, а на виході будуть значення q-функції.

Autocorrelation problem. Насправді описаний вище підхід працює погано, і найголовніша проблема - кореляція між сусідніми станами. Через те що сусідні стани дуже схожі, їх q-функції також будуть відрізнятися не сильно. Тому замість коректного оновлення функції, ми будемо замінювати її на максимум самої себе, що призведе до необмеженого росту значень.

Щоб виправити це, пропонується завести другу мережу, яка буде передбачати v(s`) по стану s`. А потім за поточним значенням Q(s, a) і по передбаченню рахується лосс (значення функції втрат), щоб потім оновити параметри основної мережі. Ваги додаткової мережі оновлюються раз в n кроків, тому що стану s_t і s_{t+n} вже не так корелюють між собою.

Підсумовуючи, щоб досягти гарного результату в deep q-learning треба:

- використовувати минулу історію з experience replay (працює не тільки в DQN);
- брати послідовність станів для вилучення просторової інформації;
- оновлювати параметри мережі раз в кілька кроків, щоб уникнути проблему кореляції.

Double DQN. Ще одна проблема, але не така значна, - зміщення мат. очікування помилки. Ми хочемо, щоб $E_{s,a}\ L(s,a) = 0$. Але якщо кожний раз обирати максимум $Q(s`, a`)$, тоді середнє значення буде зміщуватись вправо через накопичення шуму:

Нехай $Q_*(s`, a`) = x$- справжнє значення q-функції для всіх a`; через те, що ми маємо лише оцінку, то $Q\ \ (s`, a`) = x + \epsilon_i$, де $\epsilon_i \sim N(0, \sigma)$ отримуємо $E_{s`} max_{a`} Q(s`, a`) \geq max_{a`} E_{s`} Q(s`, a`) \geq E_{s`,a`} Q(s`, a`)$

Рішення - використовувати дві оцінки q-значень Q^A і Q^B, які повинні компенсувати помилки один одного, оскільки ці помилки незалежні:

$$y = r + \gamma\ max_{a`} Q(s`, a`) = r + \gamma Q\big(s`, argmax_{a`} Q(s`, a`)\big),$$

$$y = r + \gamma\ Q^A(s`, argmax_{a`} Q^B(s`, a`))$$

За однією q-функцією будемо обирати дію, а по іншій оновлюватися, причому вибір функції для оновлення кожен раз повинен відбуватися випадково.

Policy gradient and REINFORCE algorithm

Нагадаю навіщо нам довелося ввести value based підхід. Необхідно навчатися в середовищі, а для цього потрібна деяка функція, що диференціюється (звичайна нагорода не підходить, тому що це просто число, яке не несе ніякої інформації). Тому ми ввели q-функцію і максимізувати її різними методами: простіше - дп або q-learning, складніше - approximate q-learning або DQN. Тепер же пропонується новий підхід. Запараметрізуем політику π таким чином, щоб при навчанні знайти оптимальну оцінку її параметра (так, параметризація політики вже була в методі крос-ентропії, але тоді ми оновлювалися лише раз в сесію, а не на кожному кроці, і не використали інформацію про середовище).

value based: навчаємо функції Q_θ (s, a) або V_θ(s), потім обчислюємо лінію поведінки π (s) = arg $\max_a$ Q_θ(s, a).

policy based: навчатися буде сама політика, а не якась допоміжна функція і a = πθ (s).

Знову ж таки, в однокрокової моделі будемо максимізувати середню нагороду R (s, a) за крок:

$$J = \mathop{\mathbb{E}}_{\substack{s\sim p(s)\\ a\sim\pi_\theta(s|a)}} R(s,a) = \int_s p(s)\int_a \pi_\theta(a|s)R(s,a)\ da\ ds \approx \frac{1}{N}\sum_{i=0}^{N}\sum_{s,a} R(s,a)$$

Виникає наступне питання: як обчислювати $\frac{dJ}{d\theta}$, якщо наближене значення J не залежить від θ. Тобто ми нагенерували сесії, використовуючи політику π_θ, отримали нагороду, а тепер хочемо диференціювати по якомусь усередненому набору чисел. Варіант рішення - вважати чисельну похідну:

$$\nabla J = \frac{J_{\theta+\epsilon} - J_\theta}{\epsilon}$$

Але це погана ідея, яка вимагає багато генерацій і обчислень. Другий спосіб - диференціювати по θ вираз з інтегралом. Але тоді результат перестане бути мат. очікуванням, і ми не зможемо замінити його середнім. Щоб виправити це, зробимо кілька простих перетворень:

$$\nabla \log \pi(z) = \frac{1}{\pi(z)} \nabla \pi(z),$$

$$\pi(z) \cdot \nabla \log \pi(z) = \nabla \pi(z),$$

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s,a)\ da\ ds,$$

$$\nabla J = \int_s p(s) \int_a \pi_\theta(a|s) \nabla \log \pi_\theta(a|s) R(s,a)\ da\ ds,$$

$$\nabla J = \mathop{\mathbb{E}}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} \left[\nabla \log \pi_\theta(a|s) \cdot R(s,a) \right]$$

Оскільки політика π - це деяка модель, яка передбачає дію (наприклад, нейронна мережа), то ми можемо диференціювати її логарифм за параметрами, а для обчислення ∇J використовувати усереднення по семплам:

$$\nabla J = \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a} \nabla log \pi_\Theta(a|s) * R(s,a)$$

Модель буде оновлюватись наступним чином:

$$\theta_{i+1} = \theta_i + \alpha \cdot \nabla J$$

3.3.2 Задача створення доповненої реальності

Наступний фрагмент частково взятий із наукової конференції ІСТУ-2021.

Доповнена або розширена реальність (Augmented Reality) перетворює навколишній світ в цифровий інтерфейс, розміщуючи віртуальні та реальні об'єкти в режимі реального часу. На відміну від віртуальної реальності, яка створює абсолютно штучне середовище, доповнена реальність використовує наявне середовище і накладає на нього нову інформацію. Доповнену реальність можна побачити через різноманітні відображення. Останні розробки зробили цю технологію доступною за допомогою смартфона, що призвело до розробки широкого спектру додатків доповненої реальності [18].

Програми доповненої реальності – це програмні додатки, які поєднують цифровий візуальний, аудіо та інші типи інформації із реальним середовищем користувача [18].

Даний блок немає класичних аналогів адже ця галузь є дуже молодою, але стрімко розвивається як в розважальних цілях (фільми, навігація, помічник для збірки меблів, комп’ютерні ігри) так і в професійних (навчання лікарів, асистент водія автомобіля або пілота, інтерактивні підказки солдатам). Через новітність даної сфери більшість термінів та підходів ще не мають загальноприйнятих перекладів на українську мову, тому частина з них буде дана англійською, це, при бажанні, допоможе знайти більш детальну інформацію в інтернеті [18].

Для того щоб отримати модуль який здатний аналізувати та доповнювати відеопотік у реальному часі, модуль було декомпозовано на два пункта, а саме:

- пошук поверхні для відображення;
- відображення фігур.

Розберемо кожний з них окремо [18].

Пошук поверхні для відображення

В деяких проєктах цей крок можно пропустити, але в такому випадку картинка яку буде бачити користувач буде статичною і не буде мати гарного інтерфейсу для взаємодії з користувачем [18].

Перед тим, як відображати певну додаткову інформацію на відеопотік. Треба зрозуміти в які області/точки цікавлять користувача, тобто задектувати потрібний об’єкт після цього з 2D зображення на комп’ютері треба побудувати 3D зображення для того щоб зрозуміти нахил відображення об’єктів, їх проекцію і пропорції, задетектувати зміни освітлення і застосувати цю інформацію при побудові відображень. Існує декілька способів детектувати поверхні, в цій статті ми розбиратимемо два з них [18]:

- homography підхід;
- CornerSubPix підхід.

Homography підхід

В першому підході розраховуються так звані “гомографії”. Цей процес складається з трьох компонентів, а саме: виявлення та вилучення ознак шуканого зображення, опис отриманого кадру та перевірка збігу ознак між шуканим зображенням та кадром відеопотоку [18].

У сфері комп'ютерного зору будь-які два зображення однакової площинної поверхні в просторі пов'язані за допомогою гомографії. Цей підхід має багато практичних застосувань, таких як випрямлення зображення, реєстрація зображення або обчислення руху камери - обертання та переведення - між двома зображеннями. Після того, як з матриці гомографії

прораховане обертання камери (рис. 3.5), ця інформація може бути використана для навігації або для вставки моделей тривимірних об’єктів у зображення або відео, щоб вони були відтворені з правильним ракурсом і були частиною оригінальної сцени [18].

Рисунок 3.5 – Візуалізація принципу гомографії

Далі існує багато комбінацій методів пошуку і порівняння дескрипторів. В процесі дослідження було спробувано багато різних комбійнацій, але виділити хотілось би 2 підходи [18]:

Перший: ORB (Oriented FAST and Rotated BRIEF) creator для створення ознак та BFMatcher (Brute Force Matcher) для співставлення об’єктів [18].

Другий: SIFT creator для створення ознак та FlannBasedMatcher для співставлення об’єктів [18].

Перший метод виявився найбистрішим а другий - найбільш точний. Приклади детекції та порівняння ознак для першого та другого методу можно побачити на (рис. 3.6) та (рис. 3.7) відповідно. Для того щоб порівняти методи не прискіпливо - брались один і тий самий образ та фотографія. Зліва на рисунках (рис. 3.6) (рис. 3.7) намальований образ, зправа зображення на якому шукається образ. Лініями показуються відповідні елементи порівняння образу та зображення. Щоб ускладнити задачу для детекторів було вирішено змінити кут нахилу образу і його масштаб [18].

Рисунок 3.6 – Пошук образу за допомогою ORB creator і BFMatcher

Рисунок 3.7 – Пошук образу за допомогою SIFT creator і FlannBasedMatcher

Спираючись на рисунки можно зробити висновок, що детектори працюють коректно і вибір комбінації створювача і співставлення ознак залежить лише від сценарія використання [18].

CornerSubPix підхід

Даний алгоритм “уточнює” розташування кутів, він ефективно працює лише з не дуже складними образами, але ідеально підходить для детекції шахової дошки оскільки дошка має просту геометричну форму [18].

Функція ітеративно виконує пошук точного розташування субпікселів в кутах або пошук радіальних точок [18].

Рисунок 3.8 – Метод cornerSubPix

Точний детектор кутів субпікселів базується на спостереженні, що кожен вектор від центру q до точки p, розташованої в околиці q, є ортогональним градієнту зображення при p, і враховує шум зображення та неточність вимірювання. Розглянемо вираз: [18].

$$\epsilon_i = DI_{p_i}{}^T \cdot (q - p_i)$$

де DI_{pi} - градієнт зображення в одній з точок p_i в околиці q. Для знаходження значення q треба виконати задачу оптимізації при якому ϵ_i було мінімальним. Отримаємо систему рівнянь прирівнявши ϵ_i до нуля [18]:

$$\sum_i (DI_{p_i} \cdot DI_{p_i}{}^T) \cdot q - \sum_i (DI_{p_i} \cdot DI_{p_i}{}^T \cdot p_i)$$

Градієнти підсумовуються в околиці q. Позначимо перший градієнт G, а другий позначимо як b, тоді отримаємо [18]:

$$q = G^{-1} * \mathrm{b}$$

Алгоритм встановлює центр вікна сусідства в новому центрі q, а потім циклічно повторюється, поки центр не залишиться в межах заданого порогу або поки не виконається максимальна кількість ітерацій (ці значення є параметрами даного алгоритму) (рис. 3.9) [18].

Рисунок 3.9 – Детекція об'єктів методом cornerSubPix

Відображення фігур

Після того як була знайдена поверхня для відображення і знайдена правильна проекція 2D простору в 3D простір треба відобразити фігури та дошку на поверхню. В ході експериментів були досліджені три бібліотеки з відкритим вихідним кодом [18]:

– open CV2;
– open GL;
– python PIL.

Бібліотека PIL [19] створена спеціально для обробки зображень в інтерпретованій мові програмування Python, але через те що вона не оптимізована на обробку відеопотоку, то кількість оброблених кадрів в секунду є недостатньої для комфортної гри [18].

Бібліотеки Open CV2 [20] и Open GL [21] обидві підходят для даної задачі і кількість кадрів яка обробляється даними бібліотеками достатньо для комфортної гри. Але обираючи між цими двома бібліотеками зручніше використовувати Open GL через те, що ця бібліотека спеціалізується на рендерінгу 3D графіки і використовується в розробці графіки для фільмів та комп’ютерних ігор. Отже вона має розширений функціонал і швидкий рендерінг. Крім того, бібліотека Open CV2 має низькорівневий інтерфейс, а

отже їм не зручно користуватись в високорівневих мовах програмування як Python [18].

В якості моделей був використаний формат OBJ [22]. Це простий текстовий формат даних, який представляє тривимірну геометрію - положення всіх вершини, положення УФ-перетворень вершин, їх координати нормалі, текстури об’єктів та грані. Отже цей формат дозволяє визначити кожен полігон як список вершин і текстур [18].

Фінальний отриманий результат після детекції і рендерінгу продемонстрований на (рис. 3.10) [18].

Рисунок 3.10 – Демонстрація продукту

Висновок до розділу

В розділі математичного забезпечення були сформовані змістовну та математичну постановку задачі. Крім цього були описані розглянуті методи, їх переваги та недоліки. Були детально розібрані методи та принципи роботи інтелектуальних агентів та алгоритмів створення доповненої реальності. Також були обрані найкращі комбінації технологій для даного проєкту.

Для методів детекції швидше всього працює гомографічний ORB генератор та BF порівнювач, якщо якість більш важлива за швидкість, то краще використовувати SIFT генератор і FlannBased порівнювач. У випадку коли потрібно детектувати геометричну простий образ проектування, то краще всього підходить алгоритм CornerSubPix, в нього є багато

гіперпараметрів які можно підігнати под конкретну задачу і балансувати між швидкістю та якістю детекції. Для рендерінгу зображення бібліотека Python PIL погано підходить, а інші 2 бібліотеки OpenCV та OpenGL однаково гарно підходять. Різницею є те, що OpenCV має більше методів для комп'ютерного зору, а OpenGL більше методів для створення 3D графіки, тому вибір бібліотеки залежить від області використання застосунку. Створення інтелектуальних агентів класичними методами не є складною задачею, але такі агенти не можуть адоптуватись чи донавчатись, більше того граючи з ними багато партій гравець починає відчувати повторення стратегій агента, крім цього будь-яка зміна поведінки агента потребує змін у коді. В той же час сучасні агенти можуть гарно грати навіть якщо розробник нічого не розуміє в шахах, ці агенти здатні навчатись і придумувати плани на велику кількість ходів (навідміну від класичних агентів які продумують лише декілька найближчих ходів). Але сучасні агенти потребують значного часу для навчання перед використанням. Підсумовуючи грати проти сучасного агенту більш цікаво адже він еволюціонує і починає грати все краще і видумувати ті стратегії, які б не прийшли людині в голову. Але через проблему «холодного старту» дана технологія або вимагає багато ресурсів для навчання агентів, або вдосконалення самих алгоритмів навчання.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

В даному дипломному проєкті була використання високорівнева мультипарадигмова мова програмування з динамічною типізацією Python. Ця мова була обрана з наступних міркуваня: для зручності використання додатку всі його компоненти мають бути написані на одній мові програмування, так можно було б зробити кожний модуль окремо і, наприклад, звертатися до інтелектуального агенту як до окрмого сервера, але даний піхід значно підвищує складність розгортання і знижує швидкість роботи системи в цілому, адже посилання та отримання пакетів через інтернет є дуже повільною операцією. Для створення класичних інтелектуальних агентів краще б підійшла якась назькорівнева мова програмування (наприклад Rust або C++) через те, що в цій частині багато CPU-інтенсивних обчислювань але для сучасних агентів більш підходить обрана мова програмування python оскільки вона мітсить багато оптимізованих бібліотек та пакетів для роботи з нейронними мережами та навчанням з підкріпленням. Для задачі доповненої реальності також зручно використовувати python оскільки вона має багато бібліотек для роботи з 3D графікою та бібліотек комп'ютерного зору. Ці бібліотеки зазвичай написані на назькорівневих мовах програмування, але мають зручний високорівневий інтерфейс. Так вони поєднують швидкість написання коду та швидкість виконання обрахунків. Крім цього, python є інтерпретованою мовою програмування, отже не потрібно чекати компіляції всього проєкту, що в свою чергу значно пришвидшує написання та тестування коду.

Реалізація даного програмного застосунку відбувалась на мові Python версії 3.8. При цьому були використані наступні бібліотеки:

- numpy – бібліотека, що дозволяє ефективно працювати з даними у векторному представленні. Бібліотека написана на низькорівневих мовах

програмування і є завчасно скомпільованою, що пришвидшує обробку інформації;

- Open GL – фреймворк, для створення 3D графіки та рендерінгу об’єктів, також написана на низькорівневих мовах програмування, шо робить її дуже потужною, в проєкті використовується для рендерінгу 3D моделей шахових фігур на задетектованій поверхні, створення загального вікна відображення та захвату відопотоку;
- Open CV – бібліотека для роботи з комп’ютерним зором, написана на C++. В проєкті використовується для детекції поверхонь відображення і перетворень зображень;
- pywavefront – бібліотека для роботи з 3D OBJ форматом;
- GLFW - бібліотека з відкритим кодом для розробки додатків OpenGL, OpenGL ES та Vulkan. Забезпечує простий незалежний від платформи API для створення вікон, контекстів і поверхонь, читання введення, обробки подій тощо;
- python-chess – бібліотека, з вбудованими правилами шахів і зручними класами та функціоналом для дошки, фігур та ходів;
- pytorch – сучасний фреймворк для глубокого навчання, містить багато функцій та класів для створення нейронних мереж. Цей фреймворк є дуже гнучким, він дозволяє працювати з GPU та будувати динамічний граф обчислень, що пришвидшує навчання мереж. В проєкті використовувався для створення сучасних агентів;
- datetime – бібліотека для роботи з датами та часом;
- psycopg2 – бібліотека для роботи з даними з базою даних PostgreSQL;
- SQLalchemy - бібліотека для роботи з реляційними СУБД із застосуванням технологій ORM. Служіть для синхронізації об'єктів Python та запитів реляційної бази даних. SQLAlchemy дозволяє описувати структуру бази даних та способи взаємодії з ними на мові Python без використання SQL.

Для розробки інтелектуальних агентів була використана інтерактивні зощити для розробки Jupyter Lab. При розробці модуля доповненої реальності для написання коду використовувалась IDE PyCharm, для запуску більш гнучкого запуску застосунку використовувалась Unix консоль, а для тестування детекторів об’єктів також використовувались зошити Jupyter. В подальшому, після тестування кожного з модулів код був перенесений єдиний проєкт. Для взаємодії з базою даних використовувалось IDE Datagrip.

Тестування та розробка застоснку відбувалась в операційній системі Pop OS 20.04 яка базується на ядрі Linux.

В якості бази даних використовувалась СКБД PostgeSQL, причини використання були описані в пункті 2.3.

4.2 Вимоги до технічного забезпечення

Для комфортної роботи даного застосунку були наведені наступні вимоги до технічного забезпечення:

До складу технічних засобів повинні входити:

а) комп’ютер з наступною конфігурацією:
   1) процесор з тактовою частотою не нижче 2.0 Ггц;
   2) 32 або 64 розрядна операційна система;
   3) об’єм оперативної пам’яті щонайменше 4Гб;

б) на комп’ютер має бути встановлено наступне ПО:
   1) мова програмування python версії 3.6+;
   2) бібліотеки які вказані в конфігураційному файлі requirements.txt;

в) комп’ютерна периферія, до складу якої входять:
   1) монітор;
   2) мишка;
   3) клавіатура;
   4) веб камера для захоплення відеопотоку в процесі реального часу.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма роботи застоснку

Оскільки даний застосунок написаний у функціональному стилі, то зробити диаграму класів неможливо.

Рисунок 4.1 – схема роботи застосунку

4.3.2 Діаграма розгортання

Рисунок 4.2 – діаграма розгортання застосунку

4.3.3 Специфікація функцій

Таблиця 4.1 – специфікація функцій

Назва функції	Опис функції	Параметри
load_models	Фунція яка загружає в пам’ять необіхдні для роботи програми ресурси: моделі фігур, конфігурація камери, тощо.	-

Продовження таблиці 4.1

main	Основна функція застосунку, ініціалізує вікно програми та зчитувач з веб камери, викликає перерисовку кадру та передає інформацію про дошку інтелектуальному агенту	Ідентифікатор камери, режим фігур (2D або 3D), режим гри, інтелектуальний агент, колір гравця
get_extrinsic_parameters	Функція знаходить на кадрі образ проектування та знаходить матрицю відображення з 2D кадру в 3D простір.	Кадр для обробки, розміри поля, матриця калібрування камери
fill_grid	Дана функція дозволяє перемалювати певне поле в довільний колір. Використовується, наприклад, для показування доступнів кроків.	Кадр, координати поля, контури кутів образу проектування, колір, прозорість
get_grid_id	Допоміжна функція для отримання номеру клітинки на дошці	X, Y
get_grid_coord	Допоміжна функція для отримання координат клітинки на дошці	Id клітинки

Продовження таблиці 4.1

make_move	Функція яка виконує крок та змінює стан дошки (крок має не суперечити правилам)	Дошка, нові координати, старі координати, тип підвищення для пішака
affine_transformation	Трансформує 2D зображення в площи з z=0 в 3D	Кадр, текстура, кути образу проектування
draw_2d_piece	Відображає 2D фігуру на зображені	Кадр, текстура, дошка, кути відображення
draw_3d_piece	Відображає 3D фігуру на зображені	Кадр, текстура, дошка, r-вектор, t-вектор, кути відображення
pieces_position	Повертає стан дошки в зручному для рендерінгу форматі	Стан дошки
corners_reorder	Змінює порядок кутів, виявлених функцією get_extrinsic_parameters (), щоб порядок був сталим серед кадрів сусідніх кадрів	Кадр, кути
compile_shader	Компіляція шейдерів	Вершини шейдерів, фрагементи шейдерів

Продовження таблиці 4.1

render	Візулізація/дідображення текстури	Скомпільваний шейдер
calibrate_camera	Функція яка автоматично калібрує параметри камери при появі в відеопотоці образу проектування під різними кутами та ракурсами. Після роботи записує параметри в конфігураційний файл	Веб камера
ai_move	Загальний інтерфейс для ходів інтелектуальних агентів	Стан дошки, тип агенту
book_metrics_move	Агент який використовує кроки професійних гравців, якщо позиція не знайдена – використовує метрику	Стан дошки
metrics_move	Агент який використовує метрику	Стан дошки, глибина пошуку
agent_move	Крок створений сучасним агентом	Стан дошки
random_move	Довільний, дозволений правилами, крок	Стан дошки
alphabeta	Реалізація альфа-бета алгоритму	Дошка, глибина пошуку
quiesce	Функція яка відсікає неперспективні галуження дерева пошуку оптимального ходу	Дошка, альфа, бета
evaluate_board	Функція оцінки стану на дошці	Стан дошки

Висновок до розділу

В цьому розділі були розглянуті засоби розробки, обгрунтован вибір мови програмування, наведений список бібліотек та фреймворків які використовуються в даному застосунку. Були розглянуті загальні вимоги для програмного забезпечення, переферії та хардверних засобів комп’ютера. Наведен список основних функцій застосунку і описаний їх сенс та параметри.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Було розроблено застосунок який може бути запущений на будь-якій сучасній операційній системі. Для того, щоб зрозуміти користуватись даною програмою, нижче буде показана робота застосунка і можливі параметри запуску. Інтерфейс програмного забезпечення та візуалізація роботи програми продемонстровано на рисунках 5.1 – 5.4. В цій демонстрації запуск буде відбуватись через Linux консоль, але застосунок можливо запустити будь-яким іншим зручним для користувача способом (наприклад з графічного інтерфейсу або використовуючи улюблене IDE).

Якщо користувач не впевнений які існують параметри програми, він завжди може дізнатися ці параметри та отримати їх короткий опис використовуючи команду ***help***, але навіть не обираючи жодних додаткових параметрів програма запуститься з параметрами по замовчуванню.

Як видно на рисунку 5.1 застосунок має 4 параметри запуску: вибір ідентифікатора камери (використовується коли на комп’ютері є дві та більше камери і користувач хоче використовувати не камеру за замовчуванням), параметр d3-piece (цей параметр потрібно вказати для використання 3D фігур замість 2D), параметр color (вибор коліру за який хоче грати гравець, за замовчуванням - білий), параметр opponent (вибір інтелектуального агента-суперника, також є режим гри self, в ньому гравець може грати проти іншої людини, за замовчуванням – класичний метричний агент).

```
ivan@doctorblinch: ~/Projects/Machine Learning/interective_chess
ivan@doctorblinch:~/Projects/Machine Learning/interective_chess$ python interact
ive_chess.py --help
usage: interactive_chess.py [-h] [--camera-id CAMERA_ID] [--d3-piece]
                            [--color COLOR] [--opponent OPPONENT]

ARchess parameters

optional arguments:
  -h, --help            show this help message and exit
  --camera-id CAMERA_ID
                        id of the camera you want to use. Start from 0. Try
                        different value if you have multiple cameras
  --d3-piece            Display 3d pieces
  --color COLOR         Choose player's color
  --opponent OPPONENT   Choose player's opponent
ivan@doctorblinch:~/Projects/Machine Learning/interective_chess$
```

Рисунок 5.1 – анатоція параметрів перед запуском застосунку

Запустимо програму з іншими параметрами, будемо використовувати 3D фігури та грати проти сучасного інтелектуального агента на основі навчання з підкріпленням. Програма виводить вікно із грою (рисунок 5.2) та веде додаткові записи у консоль (рисунок 5.3). Ми зробили декілька ходів, після чого прибрали дошку з поля зору камери і як бачимо в логах застосунку – в консоль виводиться відповідна інформація про неможливість детекції об’єкту проектування.

Рисунок 5.2 – демонстрація роботи застосунку.

```
ivan@doctorblinch: ~/Projects/Machine Learning/interective_chess
ivan@doctorblinch:~/Projects/Machine Learning/interective_chess$ python interact
ive_chess.py --d3-piece --opponent ai_rl
Starting game with white versus ai_rl
Black turn
Thinking...
Move g8f6
Time to move - 0.56
Black turn
Thinking...
Move d7d5
Time to move - 0.55
Black turn
Thinking...
Move e7e6
Time to move - 0.56
Black turn
Thinking...
Move a7a6
Time to move - 0.55
fail to find chessboard!
fail to find chessboard!
fail to find chessboard!
```

Рисунок 5.3 – демонстрація роботи застосунку з іншими параметрами та логування програми.

Коли партія завершується, на екран виводиться відповідне повідомлення про кінець партії рисунок 5.4.

Рисунок 5.4 – демонстрація завершення партії.

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій застосунку інтерактивної гри в шахи з елементами доповненої реальності вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Під час виконання мануального тестування застосунку для інтерактивної гри в шахи з елементами доповненої реальності весь функціонал був перевірений. Результати проведених випробувань було представлено таблицях нижче (5.1 – 5.3).

Таблиця 5.1 – Тестування функції розпізнавання дошки.

Мета тесту	Перевірка коректності розпізнавання образу проектування.
Початковий стан	Застосунок запущенний з довільними параметрами.
Вхідні дані	Отриманий відеопотік з веб-камери з присутнім або відсутнім образом проектування.
Схема проведення тесту	В довільному порядку, під разними нахилами та ракурсами, використовуючи різний фон та освітлення показувати та прибирати образ проектування з кадру і фіксувати вірні та хибні спрацювання детектора об’єкту проектвання.
Очікуваний результат	Детектор вірно детектує наявність та відсутність образу проектування, можливі короткострокові пропадання об’єкту при збільшенні кута до 180°.
Результуючий стан	Отримати розпізнаний образ проектування з елементами доповненої реальності або отримати повідомлення у консоль про відсутність образу на кадрі.

Таблиця 5.2 – Тестування роботи інтелектуальних агентів та дотримання правил гри

Мета тесту	Перевірити, що ані користувач ані інтелектуальні агенти не можуть зробити хід який суперечить правилам. Також перевірити, що агенти намагаються виграти та діють в межах своїх алгоритмів та здатностей.
Початковий стан	Довільна позиція шахової партії.
Вхідні дані	Хід, що суперечить правилам гри.
Схема проведення тесту	Натиснути на фігуру та намагатись зробити хід, що суперечить правилам.
Очікуваний результат	Система відмовить гравцю при здійснені хода який суперечить правилам гри. Інтелектуальні агенти роблять ходи які не суперечать правилам і діють згідно запрограмованим алгоритмам та метрикам.
Результуючий стан	Запит на повторний хід гравця (який не суперечить правилам). Агент робить консистентний, умовно оптимальний хід.

Таблиця 5.3 – Тестування роботи модуля відображення доповненої реальності.

Мета тесту	Перевірка коректності відображення елементів доповненої реальності на образі проектування.
Початковий стан	Застосунок запущенний з довільними параметрами.
Вхідні дані	Отриманий відеопотік з веб-камери з присутнім образом проектування.

Продовження таблиці 5.3

Схема проведення тесту	В довільному порядку, під разними нахилами та ракурсами, використовуючи різний фон та освітлення показувати та прибирати образ проектування з кадру і фіксувати вірні та хибні спрацювання детектора об’єкту проектування.
Очікуваний результат	Модуль вірно відображає дошку і фігури під правильними кутами і проекціями, та з правильним зовнішнім освітленням. Прямі лінії залишаються прямими, дистанції на дошці є пропорційними реальним. Зображення виглядає натурально.
Результуючий стан	Отримати правильно відображені елементи доповненої реальності на розпізнаному образі проектування.

Кінець таблиці 5.3

Висновок до розділу

В цьому розділі було описано керівництво користувача, в якому були показані можливі дії користувач застосунку, були вказані параметри запуску застосунку та роз’яснено як їми користуватись. Була поставлена мета випробувань та загальні положення, наведена мета випробувань цього застосунку. Були проведені випробування які спрямовані на тестування найважливішого функціоналу – розпізнавання образу проектування, коректне відображення елементів доповненої реальності, робота інтелектуальних агентів і консистентність ходів та правил гри. Після проведення випробувань були виявлені та виправлені наявні помилки.

ЗАГАЛЬНІ ВИСНОВКИ

Результатом даного дипломного проєкту став застосунок для інтерактивної гри в шахи в елементами доповненої реальності.

У розділі з описом предметного середовища було детально описано предметну область, процес діяльності та функціональну модель. Були висунуті функціональні вимоги та були розставлені пріоритетності для їх виконаня. Крім цього був виконан пошук аналогічних застосунків і виявлено різницю між існуючими застосунками та даним проєктом. Більшість з них мають фокус лише на одній з частин даного застосунку і також вони концентруються лише на імплементації одного алгоритму та не порівнюють роботу з іншими алгоритмам, крім того більшість аналогічних застосунків мають закритий вихідний код або пропрієтарну ліцензію. Далі в цьому розділі були розтлумачені призначення даного застосунку, встановлені цілі та задачі розробки.

У розділі інформаційного забезпечення були показані вхідні та вихідні дані. Продемонстровані особливості цих даних та їх призначення, була показана їх структура. Була детально описана структура та причини створення бази даних і проведений детальний дослідницький аналіз даних результати якого в подальшому використовувались при тренуванні інтелектуальних агентів.

У розділі математичного забезпечення сформували змістовну та математичну постановку задачі, вказані ключові функції, були детально описані алгоритми, теорія та підходи для вирішення задач детекції образу проектування, відображення доповненої реальності та створення інтелектуальних агентів.

У розділі програмного та технічного забезпечення були наведені засоби розробки, використані бібліотеки та фреймворки, були наведені вимоги до технічного забезпечення та розроблену архітектуру програмного забезпечення.

У технічному розділі показано керівництво користувача, в якому наведений перелік можливостей застосунку. Була наведена схема, мета та результати випробувань застосунку. Результатом випробувань стало виправлення виявлених помилок та неточностей.

При використанні інтелектуальних агентів класичні методи вимагають багато речей прописувати вручну або придумувати занадто складну евристику, оскільки інакше агенту складно враховувати всі аспекти гри. Крім того, коефіцієнти або регуляризатори евристики треба буде змінювати час від часу для покращення агенту. Такий підхід в свою чергу не вимагає великої обчислювальної потужності та даних для навчання, на відміну від сучасних підходів навчання інтелектуальних агентів, але вимагає значні обчислювальні потужності для обрахунку кожного ходу на відміну від сучасних інтелектуальних агентів на основі навчання з підкріпленням.

Основний недолік навчання з підкріпленням - проблема холодного старту. Агенту важко навчатися з нуля, коли простір відповідей величезний і незрозуміло, в якому напрямку потрібно рухатися, щоб отримати більшу нагороду. Тому пропонується взяти найкраще з методів навчання з вчителем і навчання з підкріпленням: спочатку вчити модель мінімізувати функцію втрат, а після цього вона стає агентом і донавчається максимізувати нагороду.

Що стосується модуля доповненої реальності, то в ході роботи було виявлено, що метод cornerSubPix показує себе краще при детекції простих об’єктів і поверхонь, в свою чергу методи які основані на принципу гомографії є більш універсальними але потребують більш детального підбору функцій та метрик. Для візуалізації краще всього підходять OBJ об’єкти через свою простоту та широку підтримку серед різних бібліотек і редакторів. Бібліотеки Open GL та Open CV2 однаково гарно підходять для ціє задачі, Open CV2 - більше підходить для перевірки гіпотез і швидкої розробки, а Open GL в свою чергу має великий поріг входження, але працює швидше і підходить для

проєктів які мають велику обчислювальну складність і багато моделей для рендерінгу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Moore's law in computer science [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/technology/Moores-law
2. Brute-Force method [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Brute-Force
3. Stockfish [Електронний ресурс] – Режим доступу до ресурсу: https://stockfishchess.org/about/
4. About AR Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html
5. ARKit vs ARCore [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@scudkot/arkit-vs-arcore-e8f4f1d7bd45
6. The AlphaGo challenge [Електронний ресурс] – Режим доступу до ресурсу: https://deepmind.com/research/case-studies/alphago-the-story-so-far
7. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [Електронний ресурс] / David Silver, Thomas Hubert, Julian Schrittwieser. – 2018 – Режим доступу до ресурсу: https://arxiv.org/pdf/1712.01815.pdf
8. How Do Convolutional Layers Work in Deep Learning Neural Networks? [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
9. What is PostgrSQL? [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/
10. What is Apache Parquet? [Електронний ресурс] – Режим доступу до ресурсу: https://parquet.apache.org/
11. How To Save Trained Machine Learning Models [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/fintechexplained/how-to-save-trained-machine-learning-models-649c3ad1c018
12. OLAP vs. OLTP: What’s the Difference [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/blog/olap-vs-oltp
13. Що таке Kubernetes? [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/
14. Public clouds [Електронний ресурс] – Режим доступу до ресурсу: https://searchcloudcomputing.techtarget.com/definition/public-cloud
15. Вимоги ACID простими словами [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/555920/

16. IBM Big data analytics [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/analytics/hadoop/big-data-analytics
17. Exploratory Data Analysis explained [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15?gi=2118ee027728
18. Огляд та порвняння сучасних підходв та методів доповненої реальності на прикладі гри в шахи, [Електронний ресурс] / Гороховський І.О. – 2021 – Режим доступу до ресурсу: http://asu.kpi.ua/istu-2021/
19. Pythob Pillow [Електронний ресурс] – Режим доступу до ресурсу: https://pillow.readthedocs.io/en/stable/
20. OpenCV Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/master/d9/df8/tutorial_root.html
21. OpenGL The Industry's Foundation for High Performance Graphics [Електронний ресурс] – Режим доступу до ресурсу: https://www.opengl.org/
22. OBJ 3D File Format [Електронний ресурс] – Режим доступу до ресурсу: https://all3dp.com/1/obj-file-format-3d-printing-cad/
23. Applications and theory of the Softmax Function [Електронний ресурс] – Режим доступу до ресурсу: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer
24. Zero-Sum Games [Електронний ресурс] – Режим доступу до ресурсу: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/zero.html
25. Negamax algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Negamax
26. Min-Max Search [Електронний ресурс] – Режим доступу до ресурсу: http://web.archive.org/web/20070820195815/www.seanet.com/~brucemo/topics/minmax.htm
27. Branch and Bound Methods [Електронний ресурс] – Режим доступу до ресурсу: https://stanford.edu/class/ee364b/lectures/bb_slides.pdf
28. Alpha-Beta Search [Електронний ресурс] – Режим доступу до ресурсу: http://web.archive.org/web/20070811182954/www.seanet.com/~brucemo/topics/alphabeta.htm
29. Michael Levin's Theorem [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Michael_Levin#Theorem
30. Optimization behind an Odd-Even Effect [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Odd-Even_Effect

31. Material Balance in the game of chess [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Material#Balance
32. Chess Board Position Evaluation [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Evaluation#:~:text=a%20heuristic%20function%20to%20determine,to%20get%20the%20best%20move.
33. Point Value by Regression Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.chessprogramming.org/Point_Value_by_Regression_Analysis
34. Capablanca's Rule in action [Електронний ресурс] – Режим доступу до ресурсу: https://www.chess.com/blog/Illingworth/capablancas-rule#:~:text=If%20you%20didn't%20get,colour%20squares%20to%20your%20bishop
35. How to Evaluate Chess Positions [Електронний ресурс] – Режим доступу до ресурсу: https://www.chess.com/article/view/how-to-evaluate-a-position
36. What is reinforcement learning? The complete guide [Електронний ресурс] – Режим доступу до ресурсу: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

Додаток А

Тексти програмного коду

Інтерактивна гра в шахи з елементами доповненої реальності

(Найменування програми (документа))

DVD-R

(Вид носія даних)

39 арк, 40 Кб

(Обсяг програми (документа) , арк., Кб)

Київ – 2021 року

interective_chess.py

```
import argparse
import json

import chess
import cv2
import glfw
import numpy as np
from OpenGL.GL import *

import utils_3d
import utils_chess
from gl.models import BackGroundImage
from gl.piece3d import draw_3d_piece, load_models
from frame_reader import FrameReader
from ai.ai_main import ai_move

mouse_x, mouse_y = None, None
selected_x, selected_y = -1, -1
selection_flag = False  # If true, currently a grid is seleted.

board = chess.Board()
is_game_over, game_result = False, None

rvecs, tvecs = None, None
with open("./camera_parameters.json", 'r') as f:
    A = np.matrix(json.load(f))



def cursor_pos_callback(window, x, y):
    global mouse_x, mouse_y
    if rvecs is not None:
```

```
            bd_flag, mouse_x, mouse_y = utils_3d.within_boundary(x, y, 0, rvecs, tvecs, A)
            if not bd_flag: mouse_x = mouse_y = None


        def mouse_button_callback(window, button, action, mods):
            global selected_x, selected_y, selection_flag, is_game_over, game_result
            if button == glfw.MOUSE_BUTTON_LEFT and action == glfw.PRESS:
                if mouse_x is not None:
                    move = utils_chess.move_type(board, mouse_x, mouse_y, selected_x, selected_y,
selection_flag)

                    if not selection_flag:
                        if move == utils_chess.SELECT_A_PIECE:
                            selection_flag = True
                            selected_x, selected_y = mouse_x, mouse_y
                        elif move == utils_chess.SELECT_INVALID_GRID:
                            pass
                    else:  # selected something
                        if move == utils_chess.SELECT_A_PIECE:
                            selection_flag = True
                            selected_x, selected_y = mouse_x, mouse_y
                        elif move == utils_chess.SELECT_INVALID_GRID:
                            selection_flag = False
                        elif move == utils_chess.RESELECT_SAME_GRID:
                            selection_flag = False
                        elif move == utils_chess.VALID_MOVE:
                            selection_flag = False
                            if board.piece_type_at(utils_chess.get_grid_id(selected_x,
                                                                           selected_y)) == chess.PAWN and selected_y ==
utils_chess.bottom_line(
                                    board):
                                promotion_type = chess.QUEEN
                            else:
                                promotion_type = 0
```

```
                is_game_over, game_result = utils_chess.make_move(board, mouse_x, mouse_y, selected_x, selected_y,
                                                                  promotion_type)


        def main(args):
            global is_game_over, game_result, rvecs, tvecs

            opponent = args.opponent
            player_color = args.color

            opponents_list = ['self', 'ai_random', 'ai_db', 'ai_stockfish', 'ai_metrics', 'ai_book_metrics', 'ai_rl']
            if opponent == 'random':
                opponent = np.random.choice(opponents_list)
            if opponent not in opponents_list:
                opponent = 'ai_book_metrics'

            if player_color not in ['white', 'black']:
                player_color = np.random.choice([chess.WHITE, chess.BLACK])
            else:
                player_color = chess.WHITE if player_color == 'white' else chess.BLACK

            print_color = 'white' if player_color == chess.WHITE else 'black'
            print_color = print_color if opponent != 'self' else 'both'
            print(f'Starting game with {print_color} versus {opponent}')

            reader = FrameReader(camera_id=args.camera_id)

            image = BackGroundImage(1280, 720)
            last_ai_move = None

            while not glfw.window_should_close(window):
```

```
try:
    if is_game_over:
        result_img_folder = './resources/results/'
        if game_result == '1-0':
            img = cv2.imread(result_img_folder + 'white_won.jpg')
        elif game_result == '0-1':
            img = cv2.imread(result_img_folder + 'black_won.jpg')
        else:
            img = cv2.imread(result_img_folder + 'tie.jpg')

    else:

        img = reader.read()

        rvecs, tvecs, corners = utils_3d.get_extrinsic_parameters(img, 7, 7, A)

        contours = np.zeros((9 * 9, 3), np.float32)
        contours[:, :2] = np.mgrid[0:9, 0:9].T.reshape(-1, 2) - 1

        contours_corners, _ = cv2.projectPoints(contours, rvecs, tvecs, A, None)
        contours_corners = contours_corners.squeeze()

        contours_corners_2d = (contours[:, 0:2] + 1) * 60
        filled_grid_img = np.zeros((8 * 60, 8 * 60, 4), dtype=np.float32)

        if selection_flag:
            filled_grid_img = utils_3d.fill_grid(filled_grid_img, selected_x, selected_y,
                                                 contours_corners_2d,
                                                 (0, 0, 255))
            for i in range(64):
                if chess.Move(utils_chess.get_grid_id(selected_x, selected_y),
                              i) in board.legal_moves or chess.Move(
                    utils_chess.get_grid_id(selected_x, selected_y), i,
                    promotion=chess.QUEEN) in board.legal_moves:
```

```
                filled_grid_img = utils_3d.fill_grid(filled_grid_img,
utils_chess.get_grid_coord(i)[0],
                                                     utils_chess.get_grid_coord(i)[1],
                                                     contours_corners_2d, (255, 0, 0))

            if board.is_check():
                filled_grid_img = utils_3d.fill_grid(filled_grid_img,
                                                     utils_chess.get_grid_coord(
                                                         utils_chess.king_position(board))[
                                                         0],
                                                     utils_chess.get_grid_coord(
                                                         utils_chess.king_position(board))[
                                                         1], contours_corners_2d,
                                                     (255, 0, 255))

            # Moves TODO
            if last_ai_move:
                filled_grid_img = utils_3d.fill_grid(filled_grid_img,
                                                     last_ai_move[0], last_ai_move[1],
                                                     contours_corners_2d,
                                                     (0, 200, 255))
                filled_grid_img = utils_3d.fill_grid(filled_grid_img,
                                                     last_ai_move[2], last_ai_move[3],
                                                     contours_corners_2d,
                                                     (0, 200, 255))
            if opponent != 'self' and board.turn != player_color:
                last_ai_move = ai_move(board, ai_type='_'.join(opponent.split('_')[1:]))

            is_game_over, game_result = board.is_game_over(), board.result()


            #############

            if mouse_x is not None:
```

```
                filled_grid_img = utils_3d.fill_grid(filled_grid_img, mouse_x, mouse_y,
contours_corners_2d,
                                                     (0, 255, 0))

                if len(filled_grid_img != 0) != 0:
                    mask = filled_grid_img[:, :, 3] != 0

                    for i in range(3):
                        filled_grid_img[:, :, i][mask] /= filled_grid_img[:, :, 3][mask]

                    filled_grid_img[:, :, 3][mask] = 255
                    filled_grid_img = filled_grid_img.astype(np.uint8)

                piece_type = '3d' if args.d3_piece else '2d'

                img = utils_3d.draw_3d_board(img, board, rvecs, tvecs, A, contours_corners,
piece_type,
                                             filled_grid_img)


            except RuntimeError as e:
                print(e)
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
                image.set_image(img)
                image.render()
            else:
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

                image.set_image(img)
                image.render()
                if args.d3_piece and not is_game_over:
                    draw_3d_piece(board, rvecs, tvecs, A)
            finally:
                glfw.swap_buffers(window)
```

```
            glfw.poll_events()

        glfw.terminate()


    if __name__ == "__main__":
        parser = argparse.ArgumentParser(description='ARchess parameters')
        parser.add_argument('--camera-id', default=0, type=int,
                            help='id of the camera you want to use. Start from 0. Try different value if you have multiple cameras')
        parser.add_argument('--d3-piece', action='store_true', help='Display 3d pieces')  # to be finished
        parser.add_argument('--color', default='white', help='Choose player\'s color')  # to be finished
        parser.add_argument('--opponent', default='ai-random', help='Choose player\'s opponent')  # to be finished

        if not glfw.init():
            exit(-1)

        window = glfw.create_window(1280, 720, "Horokhovskyi KPI DIPLOMA", None, None)

        if not window:
            glfw.terminate()
            exit(-2)

        glfw.make_context_current(window)
        # glViewport(0, 0, 1280, 720)

        glfw.set_cursor_pos_callback(window, cursor_pos_callback)
        glfw.set_mouse_button_callback(window, mouse_button_callback)

        glClearColor(0, 0, 0, 1)
```

```
    load_models()

    main(parser.parse_args())
```

utils_3d.py

```
import numpy as np
import cv2


def affine_transform(img, texture, corners):  # transform a 2d image to the plane z=0 in 3d space
    rows, cols, _ = img.shape
    # contours = np.mgrid[0:9,0:9].T.reshape(-1,2) - 1
    w, h, _ = texture.shape

    mask = np.zeros_like(img)

    idx = np.array([72, 0, 8, 80])

    '''

    if trun:
        idx = np.array([(x) + (y)*9, (x) + (y+1)*9, (x+1) + (y+1)*9, (x+1) + y*9])
    else:
        idx = np.array([(x+1) + (y+1)*9, (x+1) + y*9, (x) + (y)*9, (x) + (y+1)*9])

    '''

    pts1 = np.float32([[0, 0], [0, h], [w, h], [w, 0]])
    pts2 = np.int32(corners).reshape(-1, 2)[idx, :]
```

```
    cv2.fillConvexPoly(mask, pts2, (255, 255, 255))

    M = cv2.getPerspectiveTransform(pts1, np.float32(pts2))
    dst = cv2.warpPerspective(texture, M, (cols, rows))

    mask = mask != 0

    img[mask] = dst[mask]

    return img


def draw_2d_piece(img, board, textures):  # draw 2d piece on a board
    current_state = board.__str__().split('\n')
    M = cv2.getRotationMatrix2D((30, 30), 180, 1)
    for i in range(8):
        pieces = current_state[i].split(' ')
        for j in range(8):
            piece = pieces[j]
            if piece != '.':
                texture = textures[piece]
                if piece.islower():
                    texture = cv2.warpAffine(texture, M, (60, 60))

                transparent_texture = img[i * 60:(i + 1) * 60, j * 60:(j + 1) * 60]
                transparent_texture[texture[:, :, 3] != 0] = texture[:, :, 0:3][texture[:, :, 3] != 0]
                img[i * 60:(i + 1) * 60, j * 60:(j + 1) * 60] = transparent_texture

    return img


def draw_3d_piece(rvec, tvec, A):
    pass
```

```
def draw_2d_board():  # draw a 2d board with 2d pieces
    img = np.zeros((8 * 60, 8 * 60, 3), dtype=np.uint8)

    for i in range(8):
        for j in range(8):
            if (i + j) % 2 == 1:
                img[i * 60:(i + 1) * 60, j * 60:(j + 1) * 60] = np.array([24, 141, 158])
            else:
                img[i * 60:(i + 1) * 60, j * 60:(j + 1) * 60] = np.array([132, 223, 236])

    return img


texture_folder = './resources/chess_piece_2d/'
textures = {'K': cv2.imread(texture_folder + 'Chess_klt60.png',
cv2.IMREAD_UNCHANGED),
            'k': cv2.imread(texture_folder + 'Chess_kdt60.png',
cv2.IMREAD_UNCHANGED),
            'R': cv2.imread(texture_folder + 'Chess_rlt60.png',
cv2.IMREAD_UNCHANGED),
            'r': cv2.imread(texture_folder + 'Chess_rdt60.png',
cv2.IMREAD_UNCHANGED),
            'N': cv2.imread(texture_folder + 'Chess_nlt60.png',
cv2.IMREAD_UNCHANGED),
            'n': cv2.imread(texture_folder + 'Chess_ndt60.png',
cv2.IMREAD_UNCHANGED),
            'B': cv2.imread(texture_folder + 'Chess_blt60.png',
cv2.IMREAD_UNCHANGED),
            'b': cv2.imread(texture_folder + 'Chess_bdt60.png',
cv2.IMREAD_UNCHANGED),
            'Q': cv2.imread(texture_folder + 'Chess_qlt60.png',
cv2.IMREAD_UNCHANGED),
```

```
            'q': cv2.imread(texture_folder + 'Chess_qdt60.png',
cv2.IMREAD_UNCHANGED),
            'P': cv2.imread(texture_folder + 'Chess_plt60.png',
cv2.IMREAD_UNCHANGED),
            'p': cv2.imread(texture_folder + 'Chess_pdt60.png',
cv2.IMREAD_UNCHANGED)}


    def draw_3d_board(img, board, rvec, tvec, A, corners, mode='2d',
                      fill_grid_img=None):  # draw a board in the 3d space
        if mode == 'letter':
            fontFace = cv2.FONT_HERSHEY_PLAIN

            objp = np.zeros((8 * 8, 3), np.float32)
            objp[:, :2] = np.mgrid[0:8, 0:8].T.reshape(-1, 2) - 0.5

            # grid_centers = np.zeros((8*8,2), np.float32)

            grid_centers, _ = cv2.projectPoints(objp, rvec, tvec, A, None)

            grid_centers = np.squeeze(grid_centers)

            current_state = board.__str__().split('\n')

            for i in range(8):
                pieces = current_state[i].split(' ')
                for j in range(len(pieces)):
                    piece = pieces[j]
                    if piece != '.':
                        text_place = tuple(grid_centers[8 * (7 - i) + j].astype(int).tolist())
                        cv2.putText(img=img, text=piece, org=text_place, fontFace=fontFace,
fontScale=2,
                                    color=(255, 0, 255))
```

```
    if mode == '2d':
        d2_board = draw_2d_board()
        d2_board = draw_2d_piece(d2_board, board, textures)

        if fill_grid_img is not None and len(fill_grid_img[:, :, 3] != 0) != 0:
            mask = fill_grid_img[:, :, 3] != 0
            d2_board[mask] = d2_board[mask] // 3 * 2 + fill_grid_img[:, :, 0:3][mask] // 3

        img = affine_transform(img, d2_board, corners)

    if mode == '3d':
        d2_board = draw_2d_board()

        if fill_grid_img is not None and len(fill_grid_img[:, :, 3] != 0) != 0:
            mask = fill_grid_img[:, :, 3] != 0
            d2_board[mask] = d2_board[mask] // 3 * 2 + fill_grid_img[:, :, 0:3][mask] // 3

        img = affine_transform(img, d2_board, corners)

    return img


def project_to_3d(u, v, z, rvec, tvec, A):  # get the 3-d coordinate of a point given the z coordinate of the point
    R = np.zeros((3, 3))
    cv2.Rodrigues(rvec, R)
    R = np.matrix(R)
    t = tvec
    uv = np.matrix([u, v, 1]).transpose()
    left_hand_side = R.getI() * A.getI() * uv
    s = float(((R.getI() * t)[2] + z) / left_hand_side[2])
    x = float((s * left_hand_side - R.getI() * t)[0])
    y = float((s * left_hand_side - R.getI() * t)[1])
```

```
    return x, y, z


last_starting = np.array([0, 0])  # store the last starting point of the chessboard.


def corners_reorder(img, corners, num_x, num_y):  # reorder corners detected by get_extrinsic_parameters() so that the order is consistent among frames
    global last_starting

    mask = np.zeros_like(img, dtype=np.uint8)
    cv2.fillConvexPoly(mask, np.int32(corners[[0, 6, 48, 42], :]), (255, ))
    avg_color = np.average(img[mask != 0])  # take the average color of chessborad as threshold

    sample1 = (corners[0] + corners[8]) / 2
    sample2 = (corners[48] + corners[40]) / 2

    if img[int(sample1[1]), int(sample1[0])] > avg_color and img[int(sample2[1]), int(sample2[0])] > avg_color:  # white
        corners = np.reshape(corners, (num_x, num_y, 2))
        for x in range(num_x):
            corners[x, :, :] = corners[x, ::-1, :]
        corners = np.reshape(corners, (num_x * num_y, 2))

    # if corners[0,0] > corners[-1,0]: #use the left black square as first square
    #     corners = corners[::-1,:]

    if np.linalg.norm(corners[0] - last_starting) > np.linalg.norm(corners[-1] - last_starting):
        corners = corners[::-1, :]

    last_starting[:] = corners[0]
```

```
        if np.cross(corners[0] - corners[-1], corners[0] - corners[num_x - 1]) < 0:  # scanning
every row then column
            corners = np.reshape(corners, (num_x, num_y, 2), order='F')
            corners = np.reshape(corners, (num_x * num_y, 2))

        return corners


    def get_extrinsic_parameters(img, num_x, num_y, A):  # caliberate camera
        # img should be in bgr format

        objp = np.zeros((num_x * num_y, 3), np.float32)
        objp[:, :2] = np.mgrid[0:num_x, 0:num_y].T.reshape(-1, 2)

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        ret, corners = cv2.findChessboardCorners(gray, (num_x, num_y), None)
        if not ret:
            raise RuntimeError('fail to find chessboard!')

        corners = np.squeeze(corners)
        corners = corners_reorder(gray, corners, num_x, num_y)
        ret, rvecs, tvecs = cv2.solvePnP(objp, corners, A, None)

        if not ret:
            raise RuntimeError('fail to get camera extrinsic parameters!')

        return rvecs, tvecs, corners


    def within_boundary(mouse_x, mouse_y, z, rvecs, tvecs,
                        A):  # check wherther a mouse position is within the boundary of the borad
and return the x,y the mouse is selecting
        x, y, _ = project_to_3d(mouse_x, mouse_y, 0, rvecs, tvecs, A)
```

```
    x, y = x + 0.5, y + 0.5

    if x > -0.5 and x < 7.5 and y > -0.5 and y < 7.5:
        ret = True
    else:
        ret = False

    return ret, round(x), round(y)


def fill_grid(img, x, y, contours_corners, color, alpha=0.5):  # fill a gird in 2d board
    assert img.shape[2] == 4  # an RGBA image

    # idx = np.array([(x) + (y)*9, (x) + (y+1)*9, (x+1) + (y+1)*9, (x+1) + (y)*9])
    x = x
    y = 8 - y

    idx = np.array([(x) + (y) * 9, (x + 1) + (y - 1) * 9])
    imgpts = np.int32(contours_corners).reshape(-1, 2)[idx, :]

    img[imgpts[-1, 1]:imgpts[0, 1], imgpts[0, 0]:imgpts[-1, 0]] += np.array([color[0],
color[1], color[2], 1])

    # img = np.add(img>>1, cv2.fillConvexPoly(img, imgpts, color)>>1)

    return img
```

utils_chess.py

```
import chess

SELECT_A_PIECE = 0
SELECT_INVALID_GRID = 1
RESELECT_SAME_GRID = 2
```

```
VALID_MOVE = 3


def get_grid_id(x, y):  # get the grid id from x and y
    return 8 * y + x


def get_grid_coord(id):  # get x and y from grid id defined in python-chess
    return (id % 8, id // 8)


def peices_position(board):  # get the positions where pieces exist

    def isupper(string):
        return string.isupper()

    def islower(string):
        return string.islower()

    creteria = isupper if board.turn == chess.WHITE else islower
    result = []
    current_state = board.__str__().split('\n')
    for i in range(8):
        pieces = current_state[i].split(' ')
        for j in range(len(pieces)):
            piece = pieces[j]
            if creteria(piece):
                result.append(8 * (7 - i) + j)

    return result


def king_position(board):  # get the position of the king belongs to current player
```

```
        creteria = 'K' if board.turn == chess.WHITE else 'k'
        current_state = board.__str__().split('\n')
        for i in range(8):
            pieces = current_state[i].split(' ')
            for j in range(len(pieces)):
                piece = pieces[j]
                if creteria == piece:
                    return 8 * (7 - i) + j


    def move_type(board, x, y, last_x, last_y, selection_flag):  # check whether a move is valid, etc.
        if not selection_flag:  # now no grid has been selected
            if get_grid_id(x, y) in peices_position(board):
                return SELECT_A_PIECE
            return SELECT_INVALID_GRID

        else:
            if x == last_x and y == last_y:
                return RESELECT_SAME_GRID
            if get_grid_id(x, y) in peices_position(board):
                return SELECT_A_PIECE
            if chess.Move(get_grid_id(last_x, last_y), get_grid_id(x, y)) in board.legal_moves or chess.Move(
                    get_grid_id(last_x, last_y), get_grid_id(x, y), promotion=chess.QUEEN) in board.legal_moves:
                return VALID_MOVE
            return SELECT_INVALID_GRID


    def make_move(board, x, y, last_x, last_y, promotion_type):  # make a move. Move must be valid
        if promotion_type != 0:
```

```
            board.push(chess.Move(get_grid_id(last_x, last_y), get_grid_id(x, y),
promotion_type))
        else:
            board.push(chess.Move(get_grid_id(last_x, last_y), get_grid_id(x, y)))

        return board.is_game_over(), board.result()


    def bottom_line(board):  # the row id of bottom row
        return 7 if board.turn == chess.WHITE else 0
```

ai_main.py

```
    import chess
    import chess.polyglot
    import numpy as np
    from datetime import datetime

    from ai.stockfish import stockfish
    from ai.metrics import alphabeta


    def ai_move(board: chess.Board, ai_type='random', verbose=1):
        ai_type2func = {
            'random': random_move,
            'db': db_move,
            'stockfish': stockfish,
            'metrics': metrics_move,
            'book_metrics': book_metric_move,
            'rl': stockfish,
        }

        start = datetime.now()
        if verbose:
```

```
                color = 'White' if board.turn else 'Black'
                print(f'{color} turn')
                print('Thinking...')

        move = ai_type2func[ai_type](board)
        # move.xboard()
        board.push(move)

        if verbose:
                print(f'Move {move}')
                time_taken = datetime.now() - start
                print(f'Time to move - {time_taken.seconds + round(time_taken.microseconds / 10**6, 2)}')

        return move.from_square % 8, move.from_square // 8, move.to_square % 8, move.to_square // 8

    def book_metric_move(board, depth=3, book='human'):
        book2path = {
                'human': 'ai/books/human.bin',
                'computer': 'ai/books/computer.bin',
                'pecg': 'ai/books/pecg_book.bin'
        }
        try:
                move = chess.polyglot.MemoryMappedReader(book2path[book]).weighted_choice(board).move
                return move
        except:
                return metrics_move(board, depth)


    def metrics_move(board, depth=3):
        bestMove = chess.Move.null()
        bestValue = -99999
```

```
        alpha = -100000
        beta = 100000
        for move in board.legal_moves:
            board.push(move)
            boardValue = -alphabeta(board, -beta, -alpha, depth - 1)
            if boardValue > bestValue:
                bestValue = boardValue
                bestMove = move
            if boardValue > alpha:
                alpha = boardValue
            board.pop()
        return bestMove


    def random_move(board: chess.Board):
        return np.random.choice(list(board.legal_moves))


    def db_move(board: chess.Board, limit=100):
        from sqlalchemy import create_engine, text
        import psycopg2
        engine = create_engine('postgresql+psycopg2://ivan:chess-
master@localhost:7777/chess-master', echo=False)
        query = f'''select move from games_people
                    where board like '{board.fen().split(' ')[0]}%'
                    order by black_elo + white_elo desc
                    limit {limit};'''
        res = [i[0] for i in engine.execute(text(query))]
        if res:
            return chess.Move.from_uci(np.random.choice(res))

        print('Position not found!')
        return metrics_move(board)
```

metrics.py

```
import chess


def alphabeta(board: chess.Board, alpha, beta, depthleft):
        bestscore = -9999
        if (depthleft == 0):
                return quiesce(board, alpha, beta)
        for move in board.legal_moves:
                board.push(move)
                score = -alphabeta(board, -beta, -alpha, depthleft - 1)
                board.pop()
                if score >= beta:
                        return score
                if score > bestscore:
                        bestscore = score
                if score > alpha:
                        alpha = score
        return bestscore


def quiesce(board: chess.Board, alpha, beta):
        stand_pat = evaluate_board(board)
        if stand_pat >= beta:
                return beta
        if alpha < stand_pat:
                alpha = stand_pat

        for move in board.legal_moves:
                if board.is_capture(move):
                        board.push(move)
                        score = -quiesce(board, -beta, -alpha)
```

```
                board.pop()

                if score >= beta:
                        return beta
                if score > alpha:
                        alpha = score
        return alpha


pawntable = [
        0, 0, 0, 0, 0, 0, 0, 0,
        5, 10, 10, -20, -20, 10, 10, 5,
        5, -5, -10, 0, 0, -10, -5, 5,
        0, 0, 0, 20, 20, 0, 0, 0,
        5, 5, 10, 25, 25, 10, 5, 5,
        10, 10, 20, 30, 30, 20, 10, 10,
        50, 50, 50, 50, 50, 50, 50, 50,
        0, 0, 0, 0, 0, 0, 0, 0]

knightstable = [
        -50, -40, -30, -30, -30, -30, -40, -50,
        -40, -20, 0, 5, 5, 0, -20, -40,
        -30, 5, 10, 15, 15, 10, 5, -30,
        -30, 0, 15, 20, 20, 15, 0, -30,
        -30, 5, 15, 20, 20, 15, 5, -30,
        -30, 0, 10, 15, 15, 10, 0, -30,
        -40, -20, 0, 0, 0, 0, -20, -40,
        -50, -40, -30, -30, -30, -30, -40, -50]

bishopstable = [
        -20, -10, -10, -10, -10, -10, -10, -20,
        -10, 5, 0, 0, 0, 0, 5, -10,
        -10, 10, 10, 10, 10, 10, 10, -10,
        -10, 0, 10, 10, 10, 10, 0, -10,
```

```
        -10, 5, 5, 10, 10, 5, 5, -10,
        -10, 0, 5, 10, 10, 5, 0, -10,
        -10, 0, 0, 0, 0, 0, 0, -10,
        -20, -10, -10, -10, -10, -10, -10, -20]

    rookstable = [
        0, 0, 0, 5, 5, 0, 0, 0,
        -5, 0, 0, 0, 0, 0, 0, -5,
        -5, 0, 0, 0, 0, 0, 0, -5,
        -5, 0, 0, 0, 0, 0, 0, -5,
        -5, 0, 0, 0, 0, 0, 0, -5,
        -5, 0, 0, 0, 0, 0, 0, -5,
        5, 10, 10, 10, 10, 10, 10, 5,
        0, 0, 0, 0, 0, 0, 0, 0]

    queenstable = [
        -20, -10, -10, -5, -5, -10, -10, -20,
        -10, 0, 0, 0, 0, 0, 0, -10,
        -10, 5, 5, 5, 5, 5, 0, -10,
        0, 0, 5, 5, 5, 5, 0, -5,
        -5, 0, 5, 5, 5, 5, 0, -5,
        -10, 0, 5, 5, 5, 5, 0, -10,
        -10, 0, 0, 0, 0, 0, 0, -10,
        -20, -10, -10, -5, -5, -10, -10, -20]

    kingstable = [
        20, 30, 10, 0, 0, 10, 30, 20,
        20, 20, 0, 0, 0, 0, 20, 20,
        -10, -20, -20, -20, -20, -20, -20, -10,
        -20, -30, -30, -40, -40, -30, -30, -20,
        -30, -40, -40, -50, -50, -40, -40, -30,
        -30, -40, -40, -50, -50, -40, -40, -30,
        -30, -40, -40, -50, -50, -40, -40, -30,
        -30, -40, -40, -50, -50, -40, -40, -30]
```

```
def evaluate_board(board: chess.Board):
    if board.is_checkmate():
        if board.turn:
            return -9999
        else:
            return 9999
    if board.is_stalemate():
        return 0
    if board.is_insufficient_material():
        return 0

    wp = len(board.pieces(chess.PAWN, chess.WHITE))
    bp = len(board.pieces(chess.PAWN, chess.BLACK))
    wn = len(board.pieces(chess.KNIGHT, chess.WHITE))
    bn = len(board.pieces(chess.KNIGHT, chess.BLACK))
    wb = len(board.pieces(chess.BISHOP, chess.WHITE))
    bb = len(board.pieces(chess.BISHOP, chess.BLACK))
    wr = len(board.pieces(chess.ROOK, chess.WHITE))
    br = len(board.pieces(chess.ROOK, chess.BLACK))
    wq = len(board.pieces(chess.QUEEN, chess.WHITE))
    bq = len(board.pieces(chess.QUEEN, chess.BLACK))

    material = 100 * (wp - bp) + 320 * (wn - bn) + 330 * (wb - bb) + 500 * (wr - br) +
900 * (wq - bq)

    pawnsq = sum([pawntable[i] for i in board.pieces(chess.PAWN, chess.WHITE)])
    pawnsq = pawnsq + sum([-pawntable[chess.square_mirror(i)]
                for i in board.pieces(chess.PAWN, chess.BLACK)])
    knightsq = sum([knightstable[i] for i in board.pieces(chess.KNIGHT,
chess.WHITE)])
    knightsq = knightsq + sum([-knightstable[chess.square_mirror(i)]
                for i in board.pieces(chess.KNIGHT, chess.BLACK)])
```

```
        bishopsq = sum([bishopstable[i] for i in board.pieces(chess.BISHOP,
chess.WHITE)])
        bishopsq = bishopsq + sum([-bishopstable[chess.square_mirror(i)]
                        for i in board.pieces(chess.BISHOP, chess.BLACK)])
        rooksq = sum([rookstable[i] for i in board.pieces(chess.ROOK, chess.WHITE)])
        rooksq = rooksq + sum([-rookstable[chess.square_mirror(i)]
                        for i in board.pieces(chess.ROOK, chess.BLACK)])
        queensq = sum([queenstable[i] for i in board.pieces(chess.QUEEN,
chess.WHITE)])
        queensq = queensq + sum([-queenstable[chess.square_mirror(i)]
                        for i in board.pieces(chess.QUEEN, chess.BLACK)])
        kingsq = sum([kingstable[i] for i in board.pieces(chess.KING, chess.WHITE)])
        kingsq = kingsq + sum([-kingstable[chess.square_mirror(i)]
                        for i in board.pieces(chess.KING, chess.BLACK)])

        eval = material + pawnsq + knightsq + bishopsq + rooksq + queensq + kingsq
        if board.turn:
                return eval
        else:
                return -eval
```

stockfish.py

```
import chess
import chess.engine


def stockfish(board: chess.Board):
    engine = stockfish_create_engine()
    return stockfish_move(engine, board)


def stockfish_create_engine():
    engine = chess.engine.SimpleEngine.popen_uci(
        'ai/engines/stockfish_13_linux_x64_bmi2/stockfish_13_linux_x64_bmi2')
```

```
    return engine


def stockfish_move(engine, board: chess.Board):
    move = engine.play(board, chess.engine.Limit(time=0.4))
    return move.move
    # board.push(move.move)
```

piece3d.py

```
from .matrix4x4 import *
from .models import OBJModel
import numpy as np

pieces_models = {}


def load_models():
    global pieces_models
    pieces_models = {'Q': OBJModel('resources/chess_piece_3d/queen.obj'),
                     'R': OBJModel('resources/chess_piece_3d/rock.obj'),
                     'N': OBJModel('resources/chess_piece_3d/knight.obj'),
                     'B': OBJModel('resources/chess_piece_3d/bishop.obj'),
                     'K': OBJModel('resources/chess_piece_3d/king.obj'),
                     'P': OBJModel('resources/chess_piece_3d/pawn.obj')
                     }


def draw_3d_piece(board, rvec, tvec, A):
    scales = {
    'K': 1.75,
    'R': 1.0,
```

```
'N': 1.2,
'B': 1.5,
'Q': 1.6,
'P': 1.0,
}

camera_view = translate(tvec) @ rodrigues(rvec)
light_source = - np.linalg.inv(rodrigues(rvec)) @ np.append(tvec, 1)
light_source = light_source / np.linalg.norm(light_source) * 5

focal = A[0, 0], A[1, 1]
principal = A[0, 2], A[1, 2]
camera_perspective = intrinsics_to_perspective(focal,
                                               principal,
                                               0.1, 100,
                                               1280, 720)

current_state = board.__str__().split('\n')
for i in range(8):
    pieces = current_state[i].split(' ')
    for j in range(8):
        piece = pieces[j]
        if piece != '.':
            piece_scale = scales[piece.upper()]
            pieces_models[piece.upper()].render(
                translate([(1/piece_scale) * (j - 0.5), (1/piece_scale) * (-7 + i + 0.5) * (1 if
piece.isupper() else -1) , 0]),
                camera_view @ scale([piece_scale, piece_scale, piece_scale]) @ flip() if
piece.isupper() else camera_view @ scale([piece_scale, piece_scale, piece_scale]),
                camera_perspective,
                light_source,
                np.array([1, 1, 1]) if piece.isupper() else np.array([0, 0, 0]))
```

calibrate_camera.py

```
import json

import cv2
import numpy as np

from frame_reader import FrameReader

reader = FrameReader()

# Define the chess board rows and columns
rows = 7
cols = 7

# Set the termination criteria for the corner sub-pixel algorithm
criteria = cv2.TERM_CRITERIA_MAX_ITER + cv2.TERM_CRITERIA_EPS, 30, 0.001

# Prepare the object points: (0,0,0), (1,0,0), (2,0,0), ..., (6,5,0). They are the same for all images
objectPoints = np.zeros((rows * cols, 3), np.float32)
objectPoints[:, :2] = np.mgrid[0:rows, 0:cols].T.reshape(-1, 2)

# Create the arrays to store the object points and the image points
objectPointsArray = []
imgPointsArray = []

valid_frame = 0

while True:
    # Load the image and convert it to gray scale
    img = reader.read()

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
            # Find the chess board corners
            ret, corners = cv2.findChessboardCorners(gray, (rows, cols), None)

            # Make sure the chess board pattern was found in the image
            if ret:
                # Refine the corner position
                corners = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

                # Add the object points and the image points to the arrays
                objectPointsArray.append(objectPoints)
                imgPointsArray.append(corners)

                # Draw the corners on the image
                cv2.drawChessboardCorners(img, (rows, cols), corners, ret)

                piece = 'Valid Frame: ' + str(valid_frame + 1) + '/30'

                print(piece)

                valid_frame += 1

            # Display the image
            cv2.imshow('chess board', img)
            cv2.waitKey(1000)

            if valid_frame >= 30:
                break

        # Calibrate the camera and save the results
        ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objectPointsArray, imgPointsArray,
gray.shape[::-1], None, None)
        print(mtx)
```

```
# Print the camera calibration error
error = 0

for i in range(len(objectPointsArray)):
    imgPoints, _ = cv2.projectPoints(objectPointsArray[i], rvecs[i], tvecs[i], mtx, dist)
    error += cv2.norm(imgPointsArray[i], imgPoints, cv2.NORM_L2) / len(imgPoints)

print("Total error: ", error / len(objectPointsArray))

with open('./camera_parameters.json', 'w') as f:
    json.dump(mtx.tolist(), f)
```

frame_reader.py

```
import cv2


class FrameReader:
    """
    read frame from camera.
    """

    def __init__(self, width=1280, height=720, camera_id=0):
        self.width = width
        self.height = height
        self.capture = cv2.VideoCapture(camera_id)
        self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, self.width)
        self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, self.height)

    def read(self):
        ret, img = self.capture.read()
        if not ret:
            raise RuntimeError("fail to read frame!")
```

```
        return img  # img in gbr format
```

matrix4x4.py

```
import numpy as np


def translate(v):
    m = np.eye(4, dtype=np.float32)
    m[:3, 3] = np.squeeze(v)
    return m


def scale(v):
    m = np.eye(4, dtype=np.float32)
    for i in range(3):
        m[i, i] = v[i]

    return m

def flip():
    m = np.eye(4, dtype=np.float32)
    m[1,1] = -1

    return m


def rodrigues(r):
    r = np.array(r, dtype=np.float32)
    theta = np.linalg.norm(r)
    r /= theta

    R = np.eye(4)
```

```
    R[:3, :3] = np.cos(theta) * np.eye(3) \
            + (1 - np.cos(theta)) * np.outer(r, r) \
            + np.sin(theta) * np.array([[0, -r[2], r[1]],
                                        [r[2], 0, -r[0]],
                                        [-r[1], r[0], 0]
                                        ])
    return R.astype(np.float32)


def perspective(fov_deg, aspect, z_near, z_far):
    assert (aspect != 0.0)
    assert (z_near != z_far)

    fov = np.radians(fov_deg)
    tan_half_fov = np.tan(fov / 2.0)

    m = np.zeros((4, 4), dtype=np.float32)
    m[0, 0] = 1.0 / (aspect * tan_half_fov)
    m[1, 1] = 1.0 / tan_half_fov
    m[2, 2] = -(z_far + z_near) / (z_far - z_near)
    m[2, 3] = -(2.0 * z_far * z_near) / (z_far - z_near)
    m[3, 2] = -1.0
    return m


def orthographic(r, t, z_near, z_far):
    m = np.array([[1 / r, 0, 0, 0],
                  [0, 1 / t, 0, 0],
                  [0, 0, -2 / (z_far - z_near), -(z_far + z_near) / (z_far - z_near)],
                  [0, 0, 0, 1]], dtype=np.float32)

    return m
```

```
def intrinsics_to_perspective(focal, principal, z_near, z_far, width, height):
    fx, fy = focal
    cx, cy = principal

    m = np.array([[2 * fx / width, 0, 2 * cx / width - 1, 0],
                  [0, -2 * fy / height, 1 - 2 * cy / height, 0],
                  [0, 0, (z_far + z_near) / (z_far - z_near), -(2.0 * z_far * z_near) / (z_far -
z_near)],
                  [0, 0, 1, 0]], dtype=np.float32)
    return m
```

models.py

```
import numpy as np
import pywavefront
from OpenGL.GL import *
from OpenGL.GL import shaders
from OpenGL.arrays.vbo import VBO


def compile_shader(vertex_shader, fragment_shader):
    with open(vertex_shader) as f:
        vertex_shader = ''.join(f.readlines())
    with open(fragment_shader) as f:
        fragment_shader = ''.join(f.readlines())
    return shaders.compileProgram(shaders.compileShader(vertex_shader,
GL_VERTEX_SHADER),
                                  shaders.compileShader(fragment_shader,
GL_FRAGMENT_SHADER))


class OBJModel:
    vs_source = 'gl/mvp.vs.glsl'
    fs_source = 'gl/phong.fs.glsl'
```

```
        def __init__(self, obj_file):
            obj = pywavefront.Wavefront(obj_file, strict=True, collect_faces=True)
            assert len(obj.mesh_list) == 1 and len(obj.mesh_list[0].materials) == 1
            material = obj.mesh_list[0].materials[0]
            #
            self.num_vertices = len(material.vertices) // material.vertex_size
            self.vbo = VBO(np.array(material.vertices, dtype=np.float32),
GL_STATIC_DRAW, GL_ARRAY_BUFFER)

            self.vao = glGenVertexArrays(1)
            glBindVertexArray(self.vao)

            self.vbo.bind()
            glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * 4, self.vbo)
            glEnableVertexAttribArray(0)
            glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * 4, self.vbo + 3 * 4)
            glEnableVertexAttribArray(1)

            glBindVertexArray(0)

            self.shader = compile_shader(self.vs_source, self.fs_source)

        def render(self, model, view, perjection, light_position, color):
            glEnable(GL_DEPTH_TEST)

            glUseProgram(self.shader)

            glUniformMatrix4fv(0, 1, True, model)
            glUniformMatrix4fv(1, 1, True, view)
            glUniformMatrix4fv(2, 1, True, perjection)
            glUniform3fv(3, 1, light_position)
            glUniform3fv(4, 1, color)
```

```
        glBindVertexArray(self.vao)
        glDrawArrays(GL_TRIANGLES, 0, self.num_vertices)


    class BackGroundImage:
        vs_source = 'gl/empty.vs.glsl'
        fs_source = 'gl/image.fs.glsl'

        def __init__(self, width, height):
            vertices = [-1, -1, 0,
                        1, -1, 0,
                        1, 1, 0,
                        1, 1, 0,
                        -1, 1, 0,
                        -1, -1, 0]

            self.num_vertices = len(vertices) // 3
            self.vbo = VBO(np.array(vertices, dtype=np.float32), GL_STATIC_DRAW,
GL_ARRAY_BUFFER)

            self.vao = glGenVertexArrays(1)
            glBindVertexArray(self.vao)

            self.vbo.bind()
            glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * 4, self.vbo)
            glEnableVertexAttribArray(0)

            glBindVertexArray(0)

            self.shader = compile_shader(self.vs_source, self.fs_source)

            self.texture = glGenTextures(1)
            glBindTexture(GL_TEXTURE_2D, self.texture)
```

```
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)

            glTexParameteri(GL_TEXTURE_2D,          GL_TEXTURE_MIN_FILTER,
GL_LINEAR)
            glTexParameteri(GL_TEXTURE_2D,          GL_TEXTURE_MAG_FILTER,
GL_LINEAR)
            glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGB8, width, height)

            # img = Image.open('board.jpg').transpose(Image.FLIP_TOP_BOTTOM)
            # glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
            #         img.size[0], img.size[1], 0,
            #         GL_RGB, GL_UNSIGNED_BYTE, img.tobytes())

        def set_image(self, image):
            glBindTexture(GL_TEXTURE_2D, self.texture)
            np.flipud(image)

            glTexSubImage2D(GL_TEXTURE_2D, 0,
                    0, 0, image.shape[1], image.shape[0],
                    GL_BGR, GL_UNSIGNED_BYTE, np.flipud(image).tobytes())

        def render(self):
            glDisable(GL_DEPTH_TEST)

            glUseProgram(self.shader)

            glUniform1i(0, 0)

            glActiveTexture(GL_TEXTURE0)
            glBindTexture(GL_TEXTURE_2D, self.texture)

            glBindVertexArray(self.vao)
            glDrawArrays(GL_TRIANGLES, 0, self.num_vertices)
```

piece3d.py

```
from .matrix4x4 import *
from .models import OBJModel
import numpy as np

pieces_models = {}


def load_models():
    global pieces_models
    pieces_models = {'Q': OBJModel('resources/chess_piece_3d/queen.obj'),
                     'R': OBJModel('resources/chess_piece_3d/rock.obj'),
                     'N': OBJModel('resources/chess_piece_3d/knight.obj'),
                     'B': OBJModel('resources/chess_piece_3d/bishop.obj'),
                     'K': OBJModel('resources/chess_piece_3d/king.obj'),
                     'P': OBJModel('resources/chess_piece_3d/pawn.obj')
                     }


def draw_3d_piece(board, rvec, tvec, A):
    scales = {
    'K': 1.75,
    'R': 1.0,
    'N': 1.2,
    'B': 1.5,
    'Q': 1.6,
    'P': 1.0,
    }

    camera_view = translate(tvec) @ rodrigues(rvec)
    light_source = - np.linalg.inv(rodrigues(rvec)) @ np.append(tvec, 1)
    light_source = light_source / np.linalg.norm(light_source) * 5
```

```
        focal = A[0, 0], A[1, 1]
        principal = A[0, 2], A[1, 2]
        camera_perspective = intrinsics_to_perspective(focal,
                                                       principal,
                                                       0.1, 100,
                                                       1280, 720)

        current_state = board.__str__().split('\n')
        for i in range(8):
            pieces = current_state[i].split(' ')
            for j in range(8):
                piece = pieces[j]
                if piece != '.':
                    piece_scale = scales[piece.upper()]
                    pieces_models[piece.upper()].render(
                        translate([(1/piece_scale) * (j - 0.5), (1/piece_scale) * (-7 + i + 0.5) * (1 if
piece.isupper() else -1) , 0]),
                        camera_view @ scale([piece_scale, piece_scale, piece_scale]) @ flip() if
piece.isupper() else camera_view @ scale([piece_scale, piece_scale, piece_scale]),
                        camera_perspective,
                        light_source,
                        np.array([1, 1, 1]) if piece.isupper() else np.array([0, 0, 0]))
```

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008158038

Дата перевірки:
03.06.2021 14:45:18 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2021 16:00:56 EEST

ID користувача:
77149

Назва документа: **Gorohovskij_bachelor_is72**

Кількість сторінок: **62** Кількість слів: **11791** Кількість символів: **83997** Розмір файлу: **167.96 KB** ID файлу: **1008224488**

6.13%
Схожість

Найбільша схожість: **1.42%** з джерелом з Бібліотеки (ID файлу: **1008224490**)

2.9% Джерела з Інтернету	158	Сторінка 64
5.93% Джерела з Бібліотеки	623	Сторінка 66

0.03% Цитат

Цитати	1	Сторінка 67

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

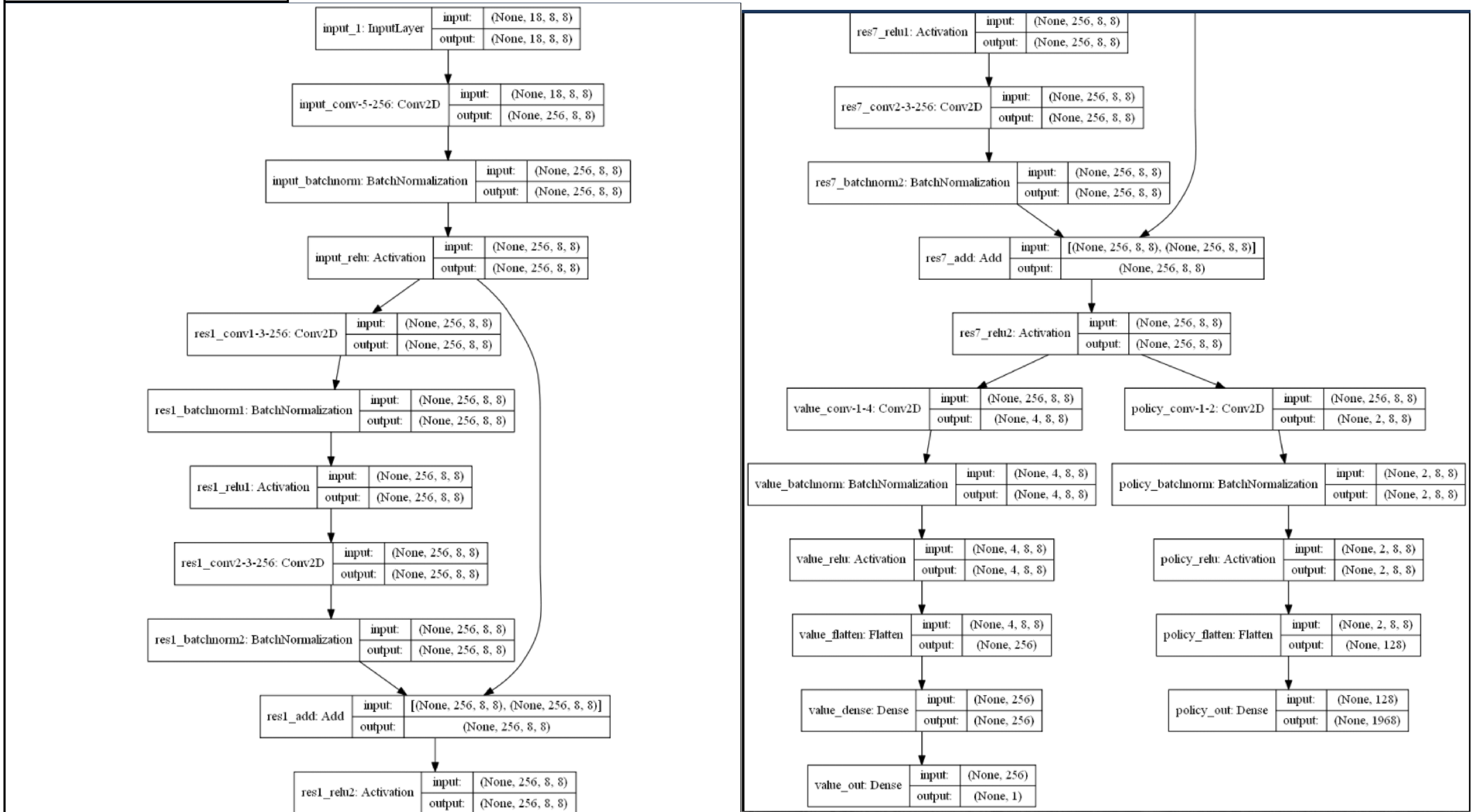
Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	55

Графічний матеріал
до дипломного проєкту

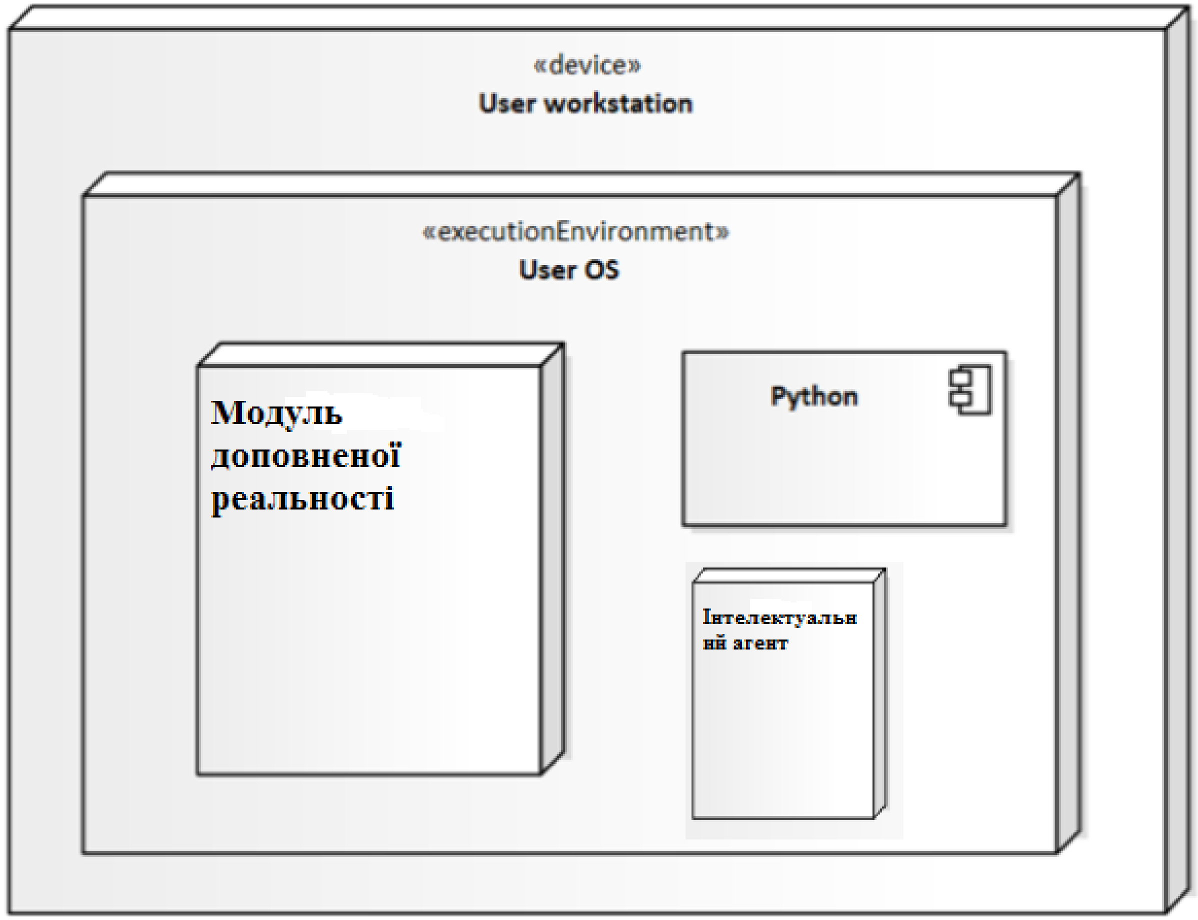
на тему: Інтерактивна гра в шахи з елементами доповненої реальності

Київ – 2021 року

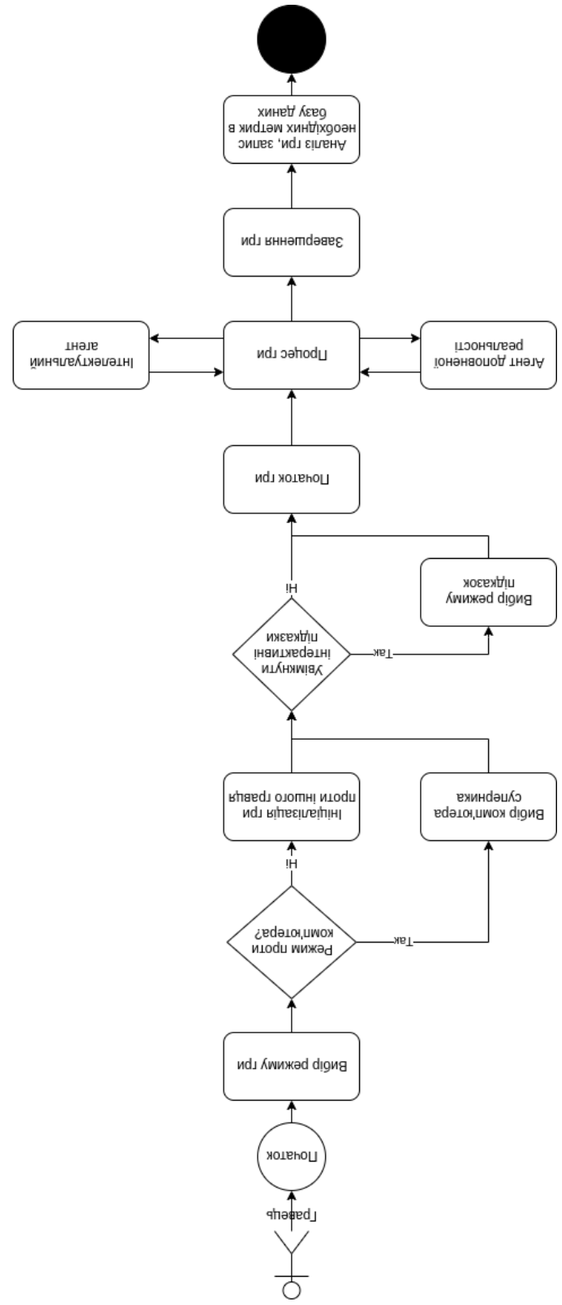


Зм.	Арк.	№ документа	Підпис	Дата	ДП 7205.02.000 СНМ
Розробив		Гороховський І.О.			Схема нейронної мережі сучасного інтелектуального агента
Перевірив		Олійник Ю.О.			Літера / Маса / Масштаб; Аркуш 1 / Аркушів 1
Консульт.					Інтерактивна гра в шахи з елементами доповненої реальності
Н. кон.		Новінський В.П.			КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72
Затвердив		Олійник Ю.О.			

ДП 7205. 03.000 ССД



					ДП 7205.03.000 ССР
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна розгортання
Розробив		Гороховський І.О.			Літера / Маса / Масштаб
Перевірив		Олійник Ю.О.			Аркуш 1 / Аркушів 1
Консульт.					Інтерактивна гра в шахи з елементами доповненої реальності
Н. кон.		Новінський В.П.			КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72
Затвердив		Олійник Ю.О.			



Зм.	Арк.	№ документа	Підпис	Дата							
Розробив		Горохівський І.О.									
Перевірив		Олійник Ю.О.									
Консульт.											
Н. кон.		Новінський В.Л.									
Затвердив		Олійник Ю.О.									

ДП 7205.04.000 ЛСР

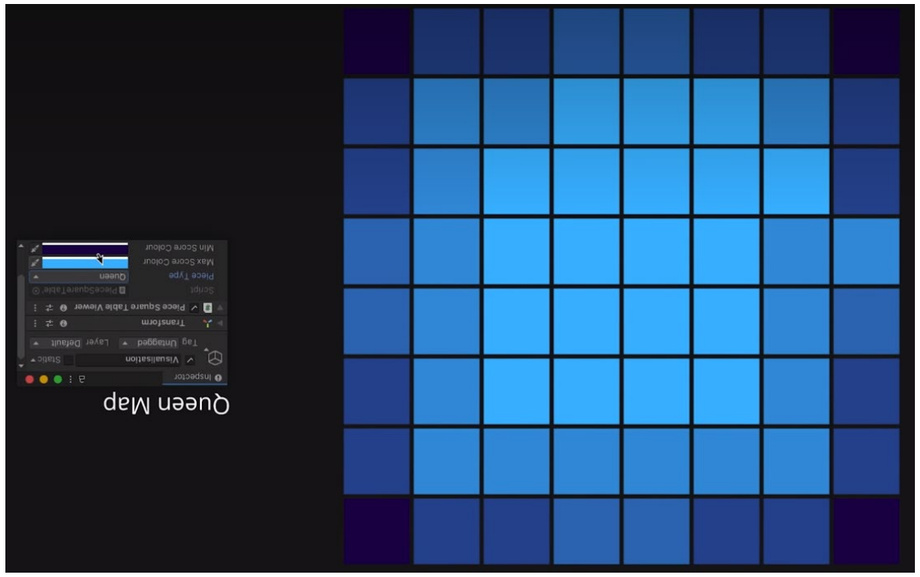
Схема структурна діяльності

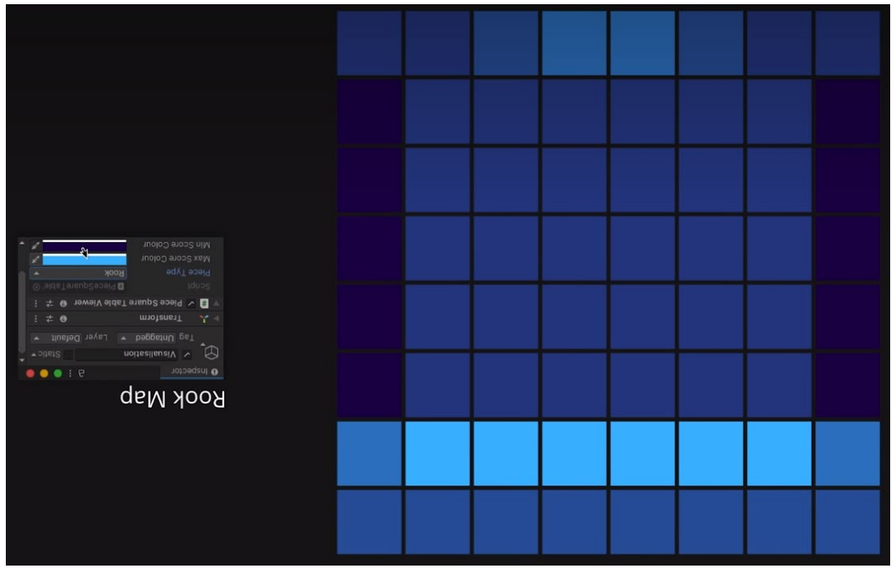
Інтерактивна гра в шахи з елементами доповненої реальності

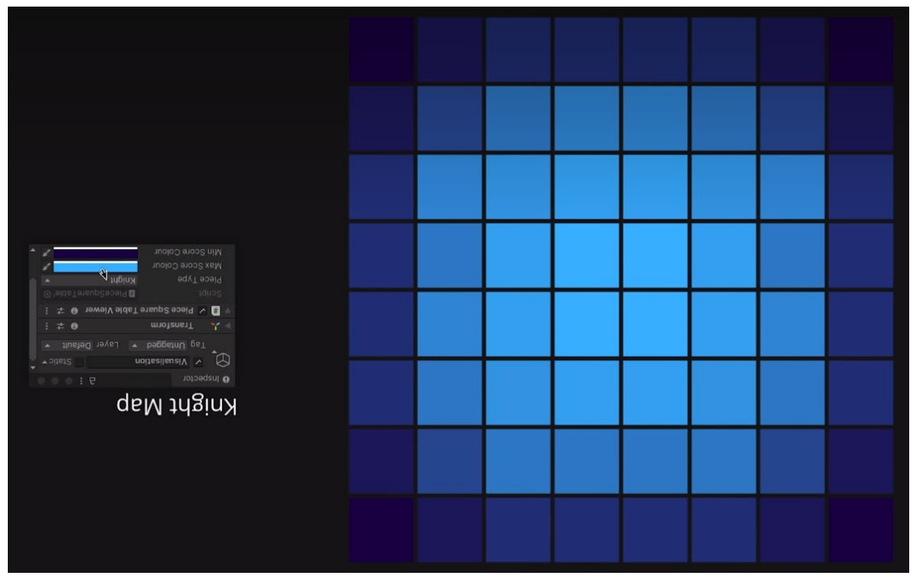
КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-72

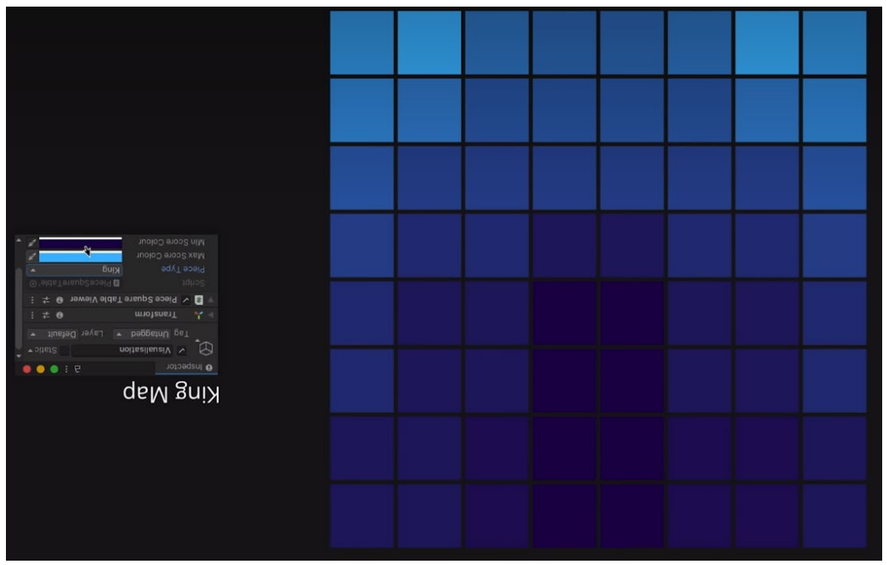
Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

ДП 7205. 04.000 ЛСР



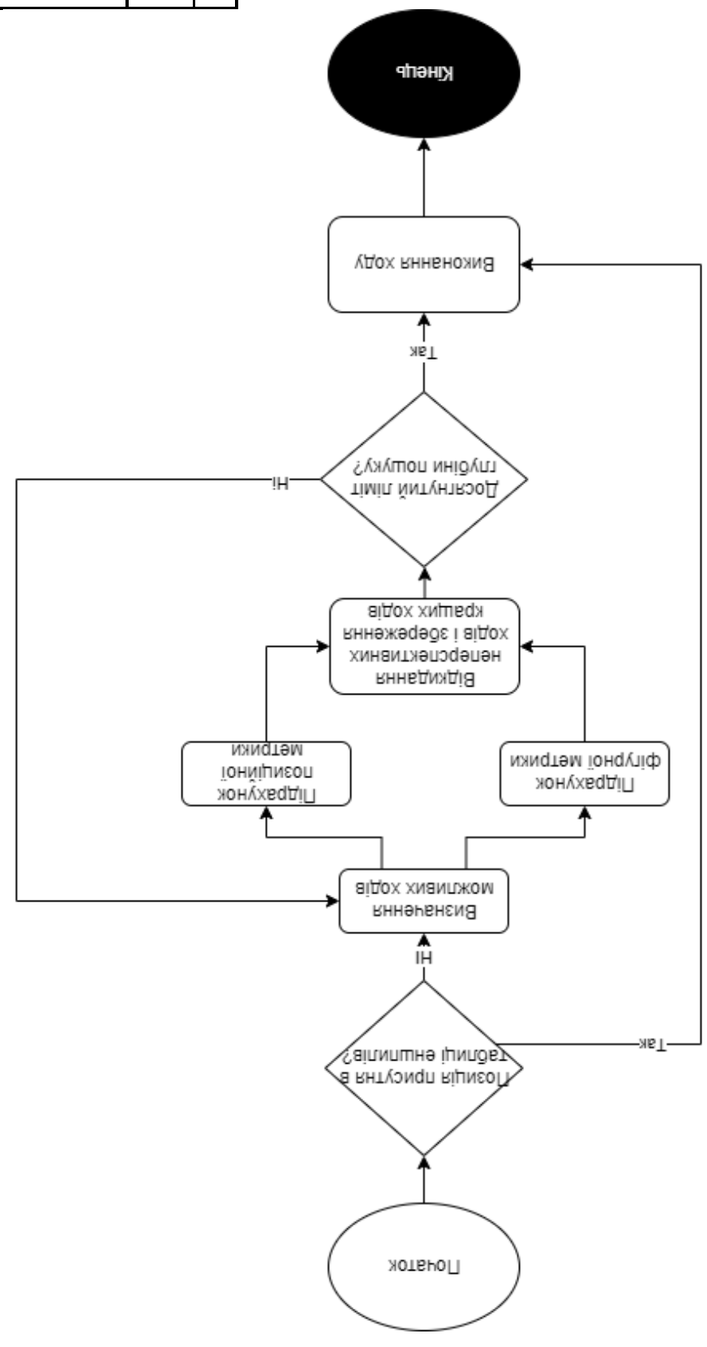






Змн.	Арк.	№ докумeнта	Підпис	Дата					
Розробив		Гороховський І.О.			*Типові карти позицій шахових фігур*			Літера	Маса	Масштаб
Перевірив		Олійник Ю.О.								
Консульт.								Аркуш 1		Аркушів 1
Н. кон.		Новіцький В.Л.								
Затвердив		Олійник Ю.О.			*Імперактивна гра в шахи з елементами доповненої реальності*			*КПІ ім. Ігоря Сікорського Кафедра АСОІУ Гр. ІС-72*		

ДП 7205.05.000 ТКП



Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Горохівський І.О.			Схема роботи класичного алгоритма			Літера	Маса	Масштаб
Перевірив		Олійник Ю.О.								
Консульт.								Аркуш 1	Аркушів 1	
Н. кон.		Новіньський В.Л.			Імперактивна дра в цахі з елементами доповненої реальності			КПІ ім. Ігоря Сікорського Кафедра АСОІУ Гр. ІС-72		
Затвердив		Олійник Ю.О.								

ДП 7205.06.000 СКА

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”

Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО

Керівник проєкту

_____ *Юрій ОЛІЙНИК*

(підпис) (вл. ім’я, прізвище)

“5” квітня 2021 р.

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*

(підпис) (вл. ім’я, прізвище)

“6” квітня 2021 р.

Інтерактивна гра в шахи з елементами доповненої реальності

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7205.01.000 ТЗ*

на 12 сторінках

Київ – 2021 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: Інтерактивна гра в шахи з елементами доповненої реальності.

Умовне позначення: Інтерактивні шахи

1.2 Найменування організації-замовника та організацій учасників робіт

Замовником проекту є кафедра Автоматизованих систем обробки інформації та управління НТУУ «КПІ ім. Ігоря Сікорського». Представником замовника є Олійник Юрій Олександрович.

Розробником системи є студент факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського»: студент групи ІС-72 Гороховський Іван Олегович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створенні проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи – 8 лютого 2021 року.

Плановий термін по закінченню роботи над створенням системи – не пізніше 29 травня 2021 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Призначеням системи є забезпечення інтерактивних можливостей гравця в шахи за рахунок додавання елементів доповненої реальності та штучного інтелекту для гри в шахи, порівняння сучасних та класичних підходів, отримання висновків та рекомендацій для використання.

2.2 Цілі створення системи

Цілями створення є:

- збільшення інтерактивних можливостей гравця;
- зробити системи розпізнавання образів;
- створення системи доповненої реальності;
- створення множини інтелектуальних агентів для гравців різного рівня.

Для досягнення поставлених цілей необхідно розв’язати наступні задачі:

- виконати порівняльний аналіз і реалізувати різні методи детекції об’єктів;
- зробити точну систему проекцій з площини в 3D простір;
- виконати порівняльний аналіз і реалізувати різні методи створення елементів доповненої реальності;
- проаналізувати методи і алгоритму створення інтелектуальних агентів.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Даний проект має три об'єкта автоматизації, а саме:

- автоматизація детекції поверхні проектування;
- автоматизація створення доповненої реальності;
- автоматизація ходів інтелектуальних агентів.

Для того, щоб автоматизувати детекцію поверхні необхідно визначити множину ключових точок образу проектування, знайти відповідну множину на відеопотоці в реальному часі та вірно виділити координати образу проектування.

Для того, щоб автоматизувати створення доповненої реальності, треба вирахувати проекції та освітленість образу проектування, виконати рендер потрібних моделей та доповнити зображення фігурами на місцях відповідно до поточного стану партії.

Для того, щоб автоматизувати ходи інтелектуальних агентів потрібно визначити правильні метрики та цілі інтелектуальних агентів, запрограмувати/навчити агентів для досягнення оптимального значення метрики/цілі.

По завершенню роботи ми отримаємо систему, яка дозволить

автоматизувати всі зазначені процеси.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Система має виконувати наступні функції:

- Користувач може підключити камеру для взаємодії з застосунком в процесі реального часу;
- Користувач має можливість грати з комп’ютерним інтелектуальним агентом;
- Користувач має можливість грати з іншим користувачем;
- Користувач має можливість ознайомитись зі списком вже зіграних матчів користувачів різного рівня.

4.2 Вимоги до надійності

Система має адекватно реагувати на помилки які можуть статись під час роботи програми і видавати користувачу відповідне повідомлення. Наприклад: ігрова дошка не знайдена, користувач задає неможливі параметри при запуску застосунку або користувач намагається зробити хід який суперечить правилам гри.

Система має вірно визначати образ проектування та коректно відображати елементи доповненої реальності. Інтелектуальні агенти повинні робити ходи які не суперечать правилам гри та діяти відповідно до закладеної програми (оптимізація метрики, використання нейронної мережі).

4.3 Умови експлуатації

Для адекватної роботи системи мати комп’ютер та периферію які відповідає вимогам зазначеним в розділі 4.4.

Усі користувачі системи мають дотримуватися правил експлуатації електронної обчислювальної техніки.

4.4 Вимоги до складу і параметрів технічних засобів

Для комфортної роботи даного застосунку були наведені наступні вимоги до технічного забезпечення:

До складу технічних засобів повинні входити:

- комп’ютер з наступною конфігурацією:
    1) процесор з тактовою частотою не нижче 2.0 Ггц;
    2) 32 або 64 розрядна операційна система;
    3) об’єм оперативної пам’яті щонайменше 4Гб;
- на комп’ютер має бути встановлено наступне ПО:
    1) мова програмування python версії 3.6+;
    2) бібліотеки які вказані в конфігураційному файлі requirements.txt;
- комп’ютерна периферія, до складу якої входять:
    1) монітор;
    2) мишка;
    3) клавіатура;
    4) веб камера для захвати відеопотоку в процесі реального часу.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 5.1 містить календарний план робіт та терміни їх виконання.

Таблиця 5.1 – Календарний план виконання робіт

№ з/п	Назва етапів створення продукту	Строк виконання
1.	Вивчення рекомендованої літератури	12.02.2021 р.
2.	Аналіз існуючих методів розв’язання задачі	19.02.2021 р.
3.	Постановка та формалізація задачі	05.03.2021 р.
4.	Розробка інформаційного забезпечення	12.03.2021 р.
5.	Алгоритмізація задачі	26.03.2021 р.
6.	Обґрунтування використовуваних технічних засобів	09.04.2021 р.
7.	Розробка програмного забезпечення	23.04.2021 р.
8.	Налагодження програми	30.04.2021 р.
9.	Виконання графічних документів	16.04.2021 р.
10.	Оформлення пояснювальної записки	07.05.2021 р.
11.	Подання ДП на попередній захист	14.05.2021 р.
12.	Подання ДП на основний захист	21.05.2021 р.
13.	Подання ДП рецензенту	31.05.2021 р.

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1 Види випробувань

Види випробувань узгоджуються із замовником до проведення випробувань. Здача - прийом робіт виконується поетапно на комп'ютерах замовника в аудиторіях кафедри АСОІУ у відповідності з робочою програмою та календарним планом.

Усі програмні продукти, що створюються в рамках даної системи передаються замовнику як у вигляді готових модулів, так і у вигляді вихідних кодів, представлених в електронній формі.