МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка відеогри стратегії з використанням ігрового рушія Unity: від концепції до реалізації»

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи ЦДБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Технології цифрового дизайну»

_____ Граба Святослава Олександровича
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Редич Олександр Володимирович, к.е.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Ратушняк Тетяна Володимирівна, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 64 26.05.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки До захисту 26.05.2025 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 10.06.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії: _____ М.М. Філоненко
(підпис)

_____ Т.В. Ратушняк
(підпис)

_____ В.О. Ніжегородцев
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		Редич О.В. Франчук В.М. Філоненко М.М. Ратушняк Т.В. Ніжегородцев В.О. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	10		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	48		

Оцінювання кваліфікаційної роботи науковим керівником - 47 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
95	A	5(відмінно)	10.06.2025	Франчук В.М. Філоненко М.М. Ратушняк Т.В. Ніжегородцев В.О. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Технології цифрового дизайну»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«14» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Грабу Святославу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: **«Розробка відеогри стратегії з використанням ігрового рушія Unity: від концепції до реалізації».**
Затверджена наказом ректора від «13» листопада 2024 р. № 1717.
2. Строк здачі закінченої роботи «12» травня 2025 р.
3. Мета роботи: розробити відеогру у жанрі стратегія, використовуючи ігровий рушій Unity (мова C#).
Об'єкт дослідження: процес розробки та створення відеогри.
Предмет дослідження: кросплатформний рушій для розробки ігор і додатків Unity.
Орієнтовна база дослідження: усі галузі цифрової творчості та ігрової індустрії.
4. Вхідні дані до роботи: технічні вимоги до програмного забезпечення, методичні матеріали з програмування на C#.
5. Перелік графічного матеріалу: скріншоти, схема.
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом).

Розділ 1. Теоретико-методологічні основи розробки відеоігор
1.1. Поняття стратегічної відеогри та її особливості;
1.2. Аналіз популярних стратегічних ігор;
1.3. Обґрунтування вибору рушія Unity та мови програмування C# для розробки стратегії.

Розділ 2. Системно-аналітичні аспекти розробки відеогри стратегії на базі Unity
2.1. Концепція ігрового проекту;
2.2. Об'єктно-орієнтований підхід в грі;
2.3. Архітектура проекту та реалізація логіки гри.

Розділ 3. Реалізація відеогри стратегії у середовищі Unity
3.1. Створення ігрового прототипу: юніти, сцени, механіка бою;
3.2. Інтеграція графічних елементів, анімації та звукового супроводу;
3.3. Проведення тестування, оптимізація продуктивності,
аналіз результатів.

7. Календарний план виконання роботи:

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	до 15.12.2024	15.12.2024
2	**І розділ** (огляд літератури по об'єкту дослідження)	30.01.2025	30.01.2025
3	**ІІ розділ** (дослідження і детальний опис вибраного інструментарію, методів, моделей, бібліотек програмування; розробка структур, алгоритмів, функціональних схем тощо)	28.02.2025	28.02.2025
4	**ІІІ розділ** (розробка і програмування відеогри, практичні результати роботи, пропозиції щодо використання і покращення результатів)	30.04.2025	30.04.2025
5	**Висновки**	20.05.2025	20.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	25.05.2025	25.05.2025
7	Отримання відгуку керівника	03.06.2025	03.06.2025
8	Отримання зовнішньої рецензії	05.06.2025	05.06.2025

8. Дата видачі завдання «14» листопада 2024 р.

9. Керівник кваліфікаційної роботи _____ Редич О. В.
(підпис)

10. Керівник освітньо-професійної програми _____ Ратушняк Т.В.
(підпис)

11. Завдання до виконання отримав здобувач вищої освіти _____ Граб С. В.
(підпис)

АНОТАЦІЯ

Розробка відеогри стратегії з використанням ігрового рушія Unity: від концепції до реалізації
(назва кваліфікаційної роботи)

Мета роботи полягає в тому, щоб розробити відеогру у жанрі стратегія, використовуючи ігровий рушій Unity.

Завданнями роботи є — вигадати та розробити гру в стилі стратегії та реалізувати її на практиці на базі ігрового рушія Unity.

Предметом дослідження є кросплатформний рушій для розроблення ігор і додатків Unity.

Об’єктом дослідження виступає процес розробки та створення відеогри.

Базою дослідження виступають усі галузі цифрової творчості та ігрової індустрії.

За результатами дослідження сформульовані висновки: стратегічні ігри мають значний потенціал для розвитку логічного мислення, планування та аналітичних навичок; сучасні різновиди стратегій часто поєднують елементи різних жанрів і потребують гнучкого підходу до проектування; відеоігри, зокрема і жанру стратегія, в найближчому майбутньому будуть залишатися затребуваними на ринку ігрової індустрії; рушій Unity забезпечує зручні засоби роботи, чим полегшує створення відеоігор; створена таким чином гра може монетизуватися; навички створення відеоігор потребуються на ринку праці в IT-секторі.

Одержані результати можуть бути використані як основа для нових ігор чи навчальних проектів.

Кваліфікаційна робота містить **85** сторінок, **1** таблицю, **46** рисунків, список використаних джерел з **40** найменувань, **2** додатків.

Ключові слова: *Стратегічна відеогра; Unity; мова C#.*

Рік виконання кваліфікаційної роботи 2024 –2025

Рік захисту роботи 2025

ANOTATION

Development of a strategy video game using the Unity game engine: from concept to implementation
(title of the qualification work)

The goal of the work is to develop a video game in the strategy genre using the Unity game engine.

The tasks of the work are to invent and develop a strategy-style game and implement it in practice based on the Unity game engine.

The subject of the study is the cross-platform engine for developing games and applications Unity.

The object of the study is the process of developing and creating a video game.

The basis of the study is all areas of digital creativity and the game industry.

The research results have formulated the following conclusions: Strategy games have significant potential for the development of logical thinking, planning, and analytical skills; Modern types of strategies often combine elements of different genres and require a flexible approach to design; Video games, including the strategy genre, will remain in demand in the gaming industry market in the near future; The Unity engine provides convenient tools, which facilitates the creation of video games; The game created in this way can be monetized; Video game creation skills are needed in the labor market in the IT sector.

The results obtained can be used as a basis for new games or educational projects.

The qualification work contains **85** pages, **1** tables, **46** figures, a list of sources used from **40** titles, **2** appendices.

Keywords: Strategic video game; Unity; C# programming language.

Year of completion of the qualification work 2024 – 2025

Year of defense of the work 2025

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ ВІДЕОІГОР 7
1.1. Поняття стратегічної відеогри та її особливості 7
1.2. Аналіз популярних стратегічних ігор 11
1.3. Обґрунтування вибору рушія Unity та мови програмування C# для розробки стратегії 16
Висновки до розділу 1 22
РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНІ АСПЕКТИ РОЗРОБКИ ВІДЕОГРИ СТРАТЕГІЇ НА БАЗІ UNITY 24
2.1. Концепція ігрового проекту 24
2.2. Об’єктно-орієнтований підхід в грі 28
2.3. Архітектура проекту та реалізація логіки гри 32
Висновки до Розділу 2 35
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВІДЕОГРИ СТРАТЕГІЇ У СЕРЕДОВИЩІ UNITY 37
3.1. Створення ігрового прототипу: юніти, сцени, механіка бою 37
3.2. Інтеграція графічних елементів, анімації та звукового супроводу 44
3.3. Проведення тестування, оптимізація продуктивності, аналіз результатів 46
Висновки до Розділу 3 48
ЗАГАЛЬНІ ВИСНОВКИ 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 51
ДОДАТКИ 55
Додаток А 55
Додаток Б 67

ВСТУП

Стратегічна відеогра — це основний жанр відеоігор, який зосереджений на аналізі та розробці стратегій, а не на швидкій реакції, з метою забезпечення успіху.

Серед різноманітних жанрів ігор, покрокова стратегія займає особливе місце, вимагаючи стратегічного мислення та прийняття важливих рішень, щоб пройти складні випробування та досягти перемоги.

Цей дипломний проект присвячений розробці комп'ютерної гри у жанрі покрокової стратегії, яка буде реалізована на платформі Unity з використанням мови програмування C#.

Unity є одним з найпопулярніших і потужних інструментів для розробки ігор, який забезпечує широкі можливості для створення унікальних і захоплюючих віртуальних світів.

Цей дипломний проект є не лише можливістю продемонструвати мої навички розробки ігор на платформі Unity, але й можливістю зануритися у захопливий світ геймдеву та зробити свій власний внесок у розвиток покрокових стратегій, які знаходять свою популярність серед шанувальників відеоігор.

Актуальність роботи (проекту): ігри це один з найбільш популярних способів весело провести час і відпочити, не відходячи від свого комп'ютера. Близько 83% людей у Європі кожного дня грають у відеоігри 2-3 рази на день, що можна порівняти з прийомом їжі. Індустрія відеоігор розвивається швидкими темпами, і доходи від неї щороку зростають. Як показує прогноз, зростатимуть і надалі, тому розвиток даної індустрії є доволі перспективним [39].

Підтримка VR/AR — Unity дозволяє створювати рішення для віртуальної та доповненої реальності (використовується у медицині, інженерії, архітектурі).

Гейміфікація — можна створювати завдання, балів, рівні складності, рейтинги, що стимулює навчання.

Виклики. Потреба в базових навичках програмування (C#). Потреба в потужному обладнанні для комфортної роботи з 3D. Брак готових україномовних ресурсів, адаптованих до шкільних програм.

Перспективи. Створення державної освітньої платформи з готовими проектами на Unity для різних дисциплін. Розвиток партнерств з IT-компаніями для підтримки студентських ініціатив. Навчальні конкурси, хакатони, Unity-табори для залучення учнів до розробки.

Unity — це не лише ігровий рушій, а потужний освітній інструмент. Його інтеграція в навчальний процес здатна значно розширити можливості традиційної освіти, зробивши її більш наочною, сучасною й захопливою.

Об'єкт досліджень: процес розробки та створення відеоігор за допомогою спеціалізованих ігрових середовищ.

Предмет досліджень: кросплатформний рушій для розроблення ігор і додатків Unity.

Стан і ступінь розробки проблеми в спеціальній літературі: Ігровий рушій Unity дуже добре описаний розробниками у спеціальній літературі [17, 40].

Результати огляду літератури і джерел: була розглянута література з відкритих джерел Інтернету і визначені етапи розробки гри. Визначено, що на Unity можна розробляти різні ігри, включаючи AAA-проекти.

Методи дослідження: індукція і дедукція; аналіз і синтез; логічний підхід; спостереження; експеримент; графічні методи; метод моделювання та метод оцінки та прогнозування.

Мета: розробити відеогру у жанрі стратегія, використовуючи ігровий рушій Unity (мова C#).

Важливість досліджень визначеного об'єкту: Доходи від виробництва відеоігор вже майже в двічі перевищили прибутки від кіновиробництва і скоро зрівняються з доходами від спортивної індустрії. Наприклад, у Китаї річний дохід від індустрії відеоігор склав 24,4 млрд. дол. США. При цьому річний дохід від відеоігор України лише 68,3 млн. дол.

В Україні проходять різного роду виставки, присвячені відеоіграм. Це свідчить про те, що у вітчизняної індустрії відеоігор є великий потенціал стати потужною, орієнтованою на експорт галуззю економіки.

Багато вітчизняних проектів уже сьогодні складають гідну конкуренцію закордонним. З таких проектів явно виділяються: серія ігор «S.T.A.L.K.E.R.» у тому числі і «S.T.A.L.K.E.R. Shadow of Chernobyl», «Metro 2033» та «Metro Last Light», серія ігор «Козаки» та розроблена українським філіалом компанії Wargaming гра «World of Warplanes».

Завдання, які потрібно вирішити для досягнення мети:

- Обрати гру жанру стратегії.
- Розробити дизайн гри.
- Написати потрібні коди для роботи гри у відповідності до правил гри.
- Протестувати гру та усунути помилки.
- Опублікувати та надати доступ для скачування гри.

Unity — це багатофункціональне середовище розробки, яке дозволяє створювати як 2D-, так і 3D-ігри та інтерактивні симуляції. Його можливості активно використовуються не лише у сфері розваг, а й в освітньому процесі.

Unity підтримує C# як основну мову програмування, має потужний редактор сцен, візуальні інструменти, а також можливість розгортання проектів на десятки платформ (Windows, Android, iOS, WebGL тощо).

Завдяки цим можливостям Unity активно застосовується у STEM-освіті, навчанні програмуванню, дизайну, інженерії та інших дисциплінах.

Переваги. Міжплатформеність — можливість запуску проектів на ПК, смартфонах, в браузері (WebGL). Велика спільнота — тисячі безкоштовних туторіалів, форумів, документації.

Практична значимість і новизна результатів: я розробив гру Го, з україномовним інтерфейсом, вона розвиває культуру стратегічних відеоігор. Популяризує гру Го в Україні, яка на цей час тут маловідома.

Отримані результати: гра Ґо, в яку можна зіграти в браузері, або встановити, як додаток у смартфоні або інших гаджетах з ОС Android.

Апробація результатів досліджень: участь у І-й Міжнародній науково-практичній конференції здобувачів вищої освіти та молодих вчених: «Економіка і бізнес: нові реалії, технологічні тренди, цифрові інновації. Economyandbusiness: newrealities, technologicaltrends, digitalinnovations» (15. Травня 2025 р. м. Львів).

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ ВІДЕОІГОР

1.1. Поняття стратегічної відеогри та її особливості

Стратегічні ігри – найдавніша форма гри. Щоразу, коли два розуми зустрічаються у змаганні, обмірковуються стратегії. Від шахів до «Риску» та «Епохи імперій» — ігри, де потрібно більше думати, ніж реагувати, є одними з найдавніших.

Але що таке стратегічні ігри? По суті, це ігри, які наголошують на довгостроковому плануванні, а не на м'язових реакціях. Аналіз ситуацій, вміле мислення та прогнозування впливу кількох різних рішень – все це необхідно для успіху в стратегічних іграх. Коротше кажучи, це ігри, які можуть справді навантажити ваші мозкові клітини. Вони мають таку ж привабливість, як і головоломки — залучають ваш мозок до пошуку рішення проблеми — але там, де головоломка може бути абстрактною та одновимірною, стратегічна гра може бути обгорнута навколо певної теми та представляти собою багатошарову, багатовимірну головоломку [8, 22, 24].

Сучасні стратегічні відеоігри сприяють розвитку логічного, аналітичного й стратегічного мислення гравця. Вони унікальні, захопливі та корисні.

Жанр стратегічних відеоігор має багато різновидів та направлень. У більшості випадків складно чітко віднести гру до якогось одного піджанру, оскільки сучасні ігри часто поєднують елементи кількох піджанрів одночасно.

Два основні підвиди стратегій: «покрокові» та стратегії «в реальному часі». Також можливе їх комбінування [33].

Механіка ігрових жанрів

Ігрові жанри базуються на механіці, тобто на діях, які гравець виконує в грі [2, 3]. Жанри також визначають баланс між тим, чи гра кидає виклик фізичним (дія) чи розумовим (стратегія) навичкам гравця, а також баланс між конфліктом та дослідженням.

Розглянемо ці популярні жанри та пов'язані з ними дії гравців у табл. 1.1.

Таблиця 1.1

Жанри та пов'язані з ними дії гравців

Жанр	Дія (ї)
Пригоди	Досліджуйте, збирайте інформацію, розгадуйте головоломки
Бій	Боріться за очки
Шутер від першої особи	Біжи та стріляй
Платформер	Стрибати з платформи на платформу
Гонки	Керуйте транспортним засобом швидше, ніж інші види транспорту
Стратегія	Будуйте та боріться
Рольова гра	Боріться та використовуйте інші навички для покращення здібностей

Джерело: [3]

Тон та сеттинг гри не мають значення. Шутер від першої особи — це шутер від першої особи, якщо ви бігаєте та стріляєте. Неважливо, смішний він, захопливий чи страшний, так само як і те, чи відбувається його дія в середньовічному підземеллі, чи на Місяці. Іграння — це те, що робить гравець.

Жанр відеоігри — це неформальна класифікація відеоігри на основі способу її проведення, а не візуальних чи наративних елементів [3].

Нові жанри виникають безперервно протягом усієї історії відеоігор, часто завдяки перехресному запиленню ідей, запозичених з різних ігор, у нові [4].

Стратегічні відеоігри — це жанр, у якому гравці беруть на себе командну роль, використовуючи стратегічне мислення, управління ресурсами та планування для досягнення конкретних цілей. Ці ігри сягають свого початку класичних настільних ігор, таких як шахи та шашки, але з часом диверсифікувалися в різні піджанри. Значна частина стратегічних ігор зосереджена на військових або бойових сценаріях, де гравці переміщаються по полях битв як командири. Інші ігри можуть включати міське планування, дозволяючи гравцям керувати містами або розвитком цивілізації [13].

Жанр еволюціонував від ранніх комп'ютерних симуляцій, з такими помітними назвами, як «Eastern Front» та «Civilization», які ознаменували перехід від покрокової механіки до стратегічного ігрового процесу в реальному часі, прикладом якого є новаторські ігри, такі як «Dune II» та «Warcraft». З розвитком технологій стратегічні ігри ставали дедалі популярнішими на домашніх комп'ютерах і зрештою перейшли на мобільні платформи, що призвело до появи таких великих релізів, як «Clash Royale» та «Clash of Clans». Сьогодні стратегічні відеоігри залишаються динамічним та прибутковим сегментом ігрової індустрії, що приваблює гравців, які люблять тактичні випробування та управління ресурсами [15].

Стратегічна відеогра — це основний жанр відеоігор, який зосереджений на аналізі та розробці стратегій, а не на швидкій реакції, з метою забезпечення успіху.

Хоча багато типів відеоігор можуть містити стратегічні елементи, жанр стратегії найчастіше визначається основним акцентом на високорівневій стратегії, логістиці та управлінні ресурсами [9].

Їх також зазвичай поділяють на дві основні підкатегорії: покрокові та ігри в реальному часі , але існує також багато стратегій між/піджанрів, які містять додаткові елементи, такі як тактика, дипломатія, економіка та дослідження.

Хоча існує багато екшн-ігор, що передбачають стратегічне мислення, їх рідко класифікують як стратегічні ігри. Стратегічна гра зазвичай має більший масштаб, і її основний акцент робиться на здатності гравця перехитрити свого опонента. Стратегічні ігри рідко передбачають фізичні випробування, і, як правило, дратують гравців зі стратегічним мисленням, коли це трапляється [32].

Порівняно з іншими жанрами, такими як екшн-ігри або пригодницькі ігри, де один гравець протистоїть багатьом ворогам, стратегічні ігри зазвичай передбачають певний рівень симетрії між сторонами. Кожна сторона, як правило, має доступ до подібних ресурсів і дій, причому сильні та слабкі сторони кожної сторони загалом збалансовані. Стратегічні ігри схожі на рольові відеоігри

тим, що гравець повинен керувати юнітами з різноманітними числовими атрибутами [25].

Стратегічні відеоігри становлять 37% ринку відеоігор (Entertainment Software Association) та наголошують на вмілому мисленні та плануванні для досягнення успіху, що досягається здебільшого завдяки ретельному плануванню. Стратегічні відеоігри зазвичай не структуровані за рівнями чи місіями, а радше розвиваються та еволюціонують протягом тривалого періоду часу. Щоб досягти успіху, гравці повинні справлятися з швидкозмінними ігровими ситуаціями, що виникають, одночасно керуючи коштами, ресурсами та інформацією. Таким чином, більшість стратегічних ігор містять елементи тактичних та стратегічних міркувань для досягнення довгострокової мети, як-от завойовник віртуального світу або управління віртуальним містом, що зазвичай вимагає тривалої відданості протягом часу. Варто зазначити, що попередні дослідження показали, що якщо відеоігри практикуються багаторазово, то це може призвести до кількох довгострокових наслідків (Gentile et al.), включаючи зміни когнітивних конструктів (наприклад, переконань), когнітивно-емоційних конструктів (наприклад, стереотипів) та афективних рис (наприклад, емпатії) [27].

Одне нещодавнє дослідження (Adachi and Willoughby) показало, що чим більше людей повідомляли про гру в стратегічні відеоігри, тим більше покращень було помітно в навичках вирішення проблем, про які вони повідомляли самостійно. Такого ж позитивного прогностичного зв'язку не було виявлено для швидких ігор, таких як гонки та файтинги.

Через свої характеристики (тобто тривалі ігрові сесії, планування, управління ресурсами) стратегічні ігри дуже відрізняються від екшн-відеоігор. Досягнення мети у стратегічних відеоіграх передбачає тривалу відданість справі, яка іноді може тривати місяцями, витрачаючи час на збір інформації, оцінку різних варіантів, формулювання плану та розгляд зміни стратегій та/або цілей перед подальшими діями. Застосовуючи теоретичну основу Силової моделі самоконтролю, ми висунули гіпотезу про зв'язок між стратегічною грою та

індивідуальними навичками саморегуляції. Дійсно, тривале використання стратегічних відеоігор, у яких досягнення мети відбувається через постійне ігрове тренування, може функціонувати як центр саморегуляції для саморегуляції в житті [27].

Був час, коли не існувало такої категорії, як стратегії в реальному часі. Все було покроковим. Потім з'явилася маленька гра під назвою Dune, і світ змінився назавжди. З часом ігри в реальному часі захопили ринок майже так само, як і Microsoft Word, який витіснив Word Perfect. Зараз нові покрокові стратегії трапляються рідко, але чи це добре?

Теоретики ігор зазначали, що стратегічне мислення погано підходить для дій у реальному часі, а пуристи покрокових стратегій критикували стратегічні ігри в реальному часі за заміну «справжньої стратегії» ігровим процесом, який винагороджує «швидке клацання мишею». Загалом, рецензенти змогли визнати переваги, пов'язані з обома основними типами стратегічних ігор.

Стратегічні ігри можна визначити як ігри, в яких головний акцент робиться на здатності гравця перехитрити свого опонента в мисленні, будь то комп'ютер чи інша людина.

1.2. Аналіз популярних стратегічних ігор

Історія стратегічних ігор

У той момент, коли перший гомосапієнс вирішив зупинитися та подумати про те, що робити далі, народилася стратегія. Простий процес планування наперед привів нас до домінування на планеті. Цей акт став настільки захопливим для сучасних людей, що ми почали винаходити способи застосовувати його поза межами наших безпосередніх повсякденних міркувань. Настільні ігри, такі як шахи, сенет та Ґо, — це те, що люди втілюють бажання тренувати свій стратегічно зосереджений мозок на діяльності, яка не вимагає заподіяння шкоди один одному [8].

Першою справжньою стратегічною грою вважається Ґо. Згідно з легендою, Ґо виникла в стародавньому Китаї приблизно у 2356 – 2255 роках до нашої ери, коли імператор Яо захотів створити гру, щоб допомогти просвітити свого сина. Ґо трохи схоже на шахи або шашки, але є оманливо складною. Кожна гра починається з порожньої дошки, і гравці використовують необмежену кількість білих або чорних каменів, щоб утворити «території», оточуючи вільні ділянки дошки. Також можна захопити камені суперника, повністю оточивши їх. За допомогою Ґо ми можемо побачити, як принципи ведення війни застосовуються до гри [8].

Перша стратегічна гра для консолей

Було неминуче, що стародавні ігри надихнули комп'ютерних програмістів спробувати написати алгоритми, які могли б їх імітувати. Першим прикладом цього стала написання Полом Штайном і Марком Веллсом комп'ютерної програми, яка могла грати у спрощену версію шахів на MANIAC 1 (також відомому як Математичний аналізатор, Чисельний інтегратор та Автоматичний комп'ютер Модель I) у 1956 році. Цей комп'ютер був справжнім звіром і важив близько 450 кг. Це також за два роки до створення відомої гри "Теніс для двох", яку часто називають першою в історії відеогрою [8].

Коли комп'ютери ставали розумнішими, люди могли більше досліджувати їхні можливості, і першою справжньою стратегічною грою, не заснованою на шахах чи шашках, здається, стала Invasion. Випущена в 1972 році, Invasion була створена для Magnavox Odyssey — дивного пристрою, який виглядав так, ніби належав до фільму «2001: Космічна Одіссея». Invasion — це, по суті, Risk, але з битвами, схожими на Pong, які відбувалися поверх накладок, що їх потрібно було наклеїти на передню панель телевізора. Odyssey могла відображати на екрані лише кілька квадратів одночасно, але їх можна було переміщувати, обертаючи ручки, прикріплені до маленьких коробок, які служили контролерами. Окрім битв, у Invasion здебільшого грали на фізичній дошці, тому сама стратегічна гра взагалі не відбувалася на консолі [8].

Одним із помітних релізів був Herzog Zwei 1989 року, який вважається першою стратегічною грою в реальному часі. Німецькою мовою «Duke Two» гравець безпосередньо пілотує літаючий механізм, що трансформується. За допомогою механізму гравець купує наземні бойові одиниці, перевозить їх по полю бою та віддає їм накази.

З того часу події набрали обертів. Це призвело до випуску Dune II у 1992 році компанією Westwood Studios, яку часто описують як гру, що задала формулу для всіх інших стратегій у реальному часі в майбутньому. Ключові елементи, що вперше з'явилися в Dune II, пізніше з'явилися в багатьох інших іграх у стилі реального часу [8].

Такі ігри, як Age of Empires, Command & Conquer, Warcraft та багато інших ігор у жанрі RTS, не існували б без Dune II та Herzog Zwei, які заклали основу для того, що зробило розробку стратегій у реальному часі цікавою.

Піджанри стратегічних ігор

З цих скромних початків жанр стратегій розквітнув у багатьох різних напрямках. Цікаво спостерігати, як людська винахідливість і прагнення до стратегічних викликів породили ігри, які одразу не можна було б вважати грою, і наскільки далеко можна дослідити певну концепцію [16, 19]. Ось деякі з них:

Стратегія в реальному часі (RTS). Щоб додати трохи безпосередності та дії до стратегічного планування, такі ігри, як Herzog Zwei та Dune II, проклали шлях. Це призвело до неймовірно популярних серій Command and Conquer, Starcraft, Warcraft та Age of Empire [34].

Ще однією великою серією в жанрі RTS є Age of Empires. Там, де Command and Conquer пішов футуристично, Age of Empires звернулася до минулого та поєднала механіку розвитку цивілізації з безпосереднім управлінням бойовими одиницями. В Age of Empires ви розвиваєте цивілізацію в різних технологічних епохах: Темних Віках, Феодальних Віках, Віках Замків (що представляють Високе Середньовіччя) та Імперських Віках (що нагадують Відродження). Додатковий рівень стратегії, який Age of Empires пропонує завдяки

балансуванню між тим, скільки часу потрібно залишатися в кожній епосі для збору ресурсів та технологічним стрибком вперед, створює захопливий ігровий виклик. Гра настільки популярна, що в Age of Empires зростає кіберспортивна сцена [25].

4X. Хоча миттєвість мислення та реагування в режимі реального часу має свою привабливість, іноді добре уповільнити темп і мислити масштабно. Жанр 4X враховує це. 4X розшифровується як «досліджувати», «розширювати», «експлоатувати» та «винищувати», що має дати вам уявлення про широту цих ігор. Відомими іграми в цьому жанрі є серія Civilization та Stellaris.

Ігри Civilization, як випливає з назви, дозволяють вам будувати та розвивати цілу цивілізацію з нуля. Від окремого поселенця, який засновує село, до будівництва величезних мегаполісів, які торгують та воюють з іншими навколишніми цивілізаціями. Все це відбувається в покроковій структурі, тому ви можете бути вдумливими у своїх рішеннях [32].

Симулятор/ Будівництво міста. Для непосвячених, ігри City Builder можуть здатися неймовірно складними та скоріше схожими на щось, чим можна займатися для роботи, ніж для розваги. Однак, такі ігри, як Cities: Skylines, насправді є гігантськими творчими пісочницями, які дають волю вашій уяві, водночас занурюючи вас у розумну реальність. У певний момент, застрягши в заторі через дорожні роботи в центрі міста, ми всі, можливо, задавалися питанням, як насправді працює щось таке величезне та складне, як місто. Що ж, ігри City Builder пропонують симуляцію, яка дозволяє вам спробувати свої сили в цьому [33].

Стратегія тут полягає у плануванні. Як би ви спланували місто, щоб транспорт не став проблемою? Де б ви розмістили промисловий район, щоб він не забруднював парки, які ви так любовно створили? Як ви переконаєтеся, що мешканці вашого чудового міста залишаються щасливими, маючи доступ до надійного електропостачання та чистої води? Про все це можна подумати та втілити в життя в цьому піджанрі.

Захист вежі. Ігри на захист веж вважаються скоріше казуальними стратегічними іграми. Можливо, це пов'язано з їхньою популярністю на мобільних пристроях. Як випливає з назви, вони передбачають побудову відповідного комплексу оборонних споруд, щоб хвиля атакуючих ворогів не могла дістатися до вашої бази. Може бути досить захопливо спостерігати, як наступає шеренга лиходіїв, а ви розставляєте всілякі стіни, щоб спрямувати їх, турелі, щоб стріляти по них, або юніти, щоб відправляти їх угору, щоб перешкодити їм. І все це при врахуванні того, скільки ресурсів у вас залишилося, щоб оплатити всі ці оборонні споруди [25].

Багатокористувацькі версії цієї концепції також дуже популярні, найбільшою з яких є Clash Royale . Короткий приклад стратегічного мислення, де ви обмірковуєте відправку юнітів вперед, одночасно захищаючись від опонентів — і все це грається на вашому телефоні, поки ви стоїте в черзі в банку.

Картковий батлер/Складач колоди. Подібно до ігор у жанрі Tower Defense, ігри Card Battler, такі як Hearthstone, Marvel Snap та майбутня Pokémon Trading Card Game Pocket, є надзвичайно популярними на мобільних пристроях.

Вони передбачають два різних рівні стратегічного мислення. Перший — це очевидний аспект гри — коли і які карти використовувати проти опонентів. Але більш тонке та влучне стратегічне мистецтво цих ігор полягає в тому, щоб переконатися, що ваша колода має правильні карти, і що вони добре поєднуються одна з одною. Hearthstone — особливо успішна гра в цьому піджанрі [25].

Майбутнє стратегічних ігор

Куди рухатиметься жанр стратегічних ігор далі, можна тільки здогадуватися, але, оскільки ми бачили його еволюцію та інновації, очевидно, що в майбутньому ми можемо очікувати дуже цікавих речей.

Такі ігри, як Homeworld 3, Manor Lords та Ara: History Untold, розширюють межі можливостей з точки зору графічної точності та глибини систем. Manor Lords особливо цікава, оскільки це поєднання міського будівництва та стратегії в реальному часі. Також є такі ігри, як Frostpunk 2, які беруть фентезійну тему та

об'єднують її в симулятор управління містом та виживання, вимагаючи від вас змінити своє мислення, відмовившись від ідеї будівництва ідеального міста і просто намагаючись не дати йому впасти і зазнати невдачі.

Можливо, ми побачимо ще більше поєднань жанрів у жанрі стратегій. Як-от стратегія в реальному часі roguelike або гра 4X, яка запозичує елементи tower defense. Ми навіть можемо побачити ігри з надзвичайно незвичними темами, як-от гра 4X, дія якої відбувається у всесвіті мікроорганізмів.

Зрештою, зростання популярності кіберспорту може призвести до появи низки стратегічних ігор, розроблених з урахуванням змагань. Ми не згадували MOBA (Multiplayer Online Battle Arena) — багатокористувацька онлайн битва на арені, оскільки вони є піджанром стратегічних ігор, для опису яких потрібно більше, ніж кілька сотень слів, але є більше ігор, подібних до Age of Empires, які можуть надихнуть на нову кіберспортивну сцену [8].

1.3. Обґрунтування вибору рушія Unity та мови програмування C# для розробки стратегії

Ігровий рушій — це готова архітектура, яку розробники використовують для запуску гри [20].

Середньостатистичний ігровий рушій надає розробникам спосіб додавати такі речі, як: фізика, керування, рендерінг, необхідні скрипти, виявлення колізій, штучний інтелект, і т.п. без необхідності їх програмування.

Чому вони важливі? Ігрові рушії є багаторазовими компонентами, які розробники використовують для побудови основи гри. Це дає їм більше часу зосередитись на унікальних елементах, таких як моделі персонажів, текстури, взаємодія об’єктів тощо. Якби кожен робив свої ігри з нуля без допомоги цих чудових ігрових рушіїв, створення ігор зайняло би більше часу і зробити їх стало би набагато складніше.

Для цього потрібно використовувати інтерфейси додатків, таких як DirectX, OpenGL та XNA, а також комерційні та відкриті джерела бібліотек, що містять фізику, графічні сцени та бібліотеки графічного інтерфейсу. Створити

власний рушій відеоігор — це непросте завдання, але іноді необхідне, якщо гра достатньо відрізняється від усіх інших і жодні існуючі рушії не будуть працювати [5, 20].

1. Ігровий рушій Unity

Unity залишається найпопулярнішим ігровим рушієм для розробки мобільних ігрових додатків, що володіє універсальними можливостями та функціоналом.

Його ключові особливості полягають у наступному:

- Він підтримує крос-платформну розробку як для iOS, так і для Android.
- Для розробників ігрових додатків доступна велика кількість зображень та ресурсів, а також різноманітні плагіни.
- Він пропонує широку спільноту розробників, які можуть ділитися своїми знахідками та документацією для кращого досвіду клієнтів.
- Він поставляється з високоякісною графікою, яка підходить як для 2D, так і для 3D ігор.
- Він має вбудовану підтримку для розробки доповненої та віртуальної реальності.

Деякі з популярних ігор, які розробляються за допомогою Unity, включають Pokemon Go, Angry Bird 2 та Monument Valley 2 [5, 20].

2. Unreal Engine для розробки ігрових додатків

Unreal Engine відом високою якістю графіки та потужним цифровим середовищем, орієнтованим на продуктивність, для розробників ігрових додатків. Також він ідеально підходить для розробників, які шукають спосіб створювати візуально приголомшливі мобільні ігри.

Компанії з розробки мобільних ігрових додатків зазвичай використовують ігрові рушії Unreal та Unity для розробки інтерактивних ігрових рішень для кінцевих користувачів.

Його ключові особливості полягають у наступному:

- Він має розширені можливості рендерингу для реалістичної візуалізації ігрового рішення.
- Дозволяє розробляти без глибоких знань з програмування.
- Він підтримує всі платформи, на яких розробляються мобільні додатки для ігор.
- Високооптимізовано для високопродуктивних ігор, орієнтованих на високу продуктивність.

Серед цікавих прикладів рушія Unreal — Fortnite, мобільний PUBG та Infinity Blade [5, 20].

3. Движок Godot для розробки мобільних ігрових додатків

Godot — це ігровий рушій з відкритим вихідним кодом, відомий своєю гнучкістю та простотою у використанні, що надає розробникам додатків чудовий цифровий досвід.

Його ключові особливості полягають у наступному:

- Він поставляється з інтуїтивно зрозумілою і гнучкою системою сцен.
- Високо оптимізовано для мобільної продуктивності під час ігор.
- Легко експортуйте ігри на різні платформи.
- Постійні оновлення та внески спільноти, які додають цінності загальному ігровому досвіду [5, 20].

Серед прикладів інтерактивних ігор — Hyper Light Drifter та Kingdom of the Dump [4].

Як бачимо, вибір трендових ігрових рушіїв доволі велик. Кожен з них має багата плюсів і може бути використаний для створення гри. Мій вибір пав саме на рушій Unity, тому що саме його ми вивчали під час навчання в інституті.

Також використання скриптів, відносно простою в освоєнні мовою C#, дозволяє запрограмувати логіку будь-якої складності для гри будь-якого жанру та масштабу. Для порівняння, поріг входження в Unreal Engine значно вищий, адже той движок використовує мову C++, яка складніше в освоєнні, ніж C#. Втім,

і на Unity, і на Unreal Engine є інструменти для візуального програмування, які полегшують створення ігрової логіки [8, 39].

У Unity є також багато вбудованих бібліотек якими легко користуватись і при цьому є багато інструкцій які допомагають, якщо виникають питання.

Що ще важливо, Unity має надійне сховище ресурсів для плагінів, сторонніх інструментів та API, що спрощує процеси розробки ігор, зменшуючи час і зусилля на розробку, а це бюджетно [6, 7, 10].

Також у ігрового двигуна за роки існування сформувалося велике співтовариство. Тому якщо при вивченні Unity у вас виникне проблема чи нерозуміння з якогось питання, то на форумах ви обов'язково знайдете тих, хто вже стикався з цією проблемою і знайшов рішення, або тих, хто допоможе вам у всьому розібратися [29].

Знижує поріг входження і те, що всю гру ви робитимете у зрозумілому візуальному редакторі. Будувати рівні, працювати з інтерфейсом, налаштовувати шейдери та генератори частинок, створювати супротивників, анімації і навіть у деяких випадках тривимірні моделі — все це ви робитимете в одному і тому ж додатку. Хіба що для написання коду на C# потрібно буде відкривати окремо редактор Visual Studio.

Ігри, як правило, створюються шляхом постійних доробок. Unity у цьому сенсі дуже зручний, адже дозволяє запустити гру в редакторі та випробувати її без створення окремого білда, на який знадобиться час.

Більш того, поки гра запущена, ви можете щось змінити, щоб відразу подивитися, як зміна позначиться на геймплей. Наприклад, можна збільшити швидкість персонажа, змінити шкоду від зброї, додати кілька супротивників, пересунути дерево, зробити маленького дракона величезним і збільшити його HP до 1000, щоб він став босом.

Всі об'єкти в Unity збираються з компонентів, які легко видаляти і додавати. Це також полегшує створення прототипів, адже завжди можна наділити будь-який об'єкт вже опрацьованою логікою поведінки, додавши

готовий скрипт. Будь-який автомобіль можна змусити літати, якщо додати код, написаний для літака [7, 30].

Якщо ви зробили гру на Unity, припустимо, для комп'ютера, то ви дуже швидко і легко зможете зробити її версію для смартфона, планшета, ігрової приставки або, наприклад, просто для сайту, щоб пограти в неї можна було прямо у браузерному вікні. При цьому використовувати Unity можна не тільки на Windows, але і на macOS.

Як зазначив автор книги «Unity у дії» Джозеф Хокінг, це стало наслідком того, що спочатку Unity створювалася лише під MAC, а потім розширили її до роботи з Windows. Подальша робота над двигуном додала як платформи для розгортання смартфони, і такі ігрові консолі, як Xbox і PlayStation [35].

Гру в Unity можна повністю зібрати з чужих моделей, спрайтів, коду, розширень для движка і всього іншого. Все це можна знайти в Unity Asset Store, зайти в який можна прямо в движку. Є багато безкоштовних моделей. Платних варіантів, зрозуміло, набагато більше.

Мінусів у ігрового движка Unity теж вистачає, хоча плюсів на чаші терезів все ж таки більше. Ось основні проблеми, про які треба знати.

За кодом складно встежити [29].

Так, гру ви збираєте у простому та зрозумілому візуальному редакторі, але код для об'єктів доводиться писати в окремому редакторі. Якщо у вас простенька гра, то ви напишете з десяток скриптів, і все буде нормально. Але якщо гра велика і для неї потрібно написати під сотню різних скриптів, що регулюють роботу десятків різних систем і механік, то в якийсь момент ви можете усвідомити, що забули, як об'єкти та різні скрипти взаємодіють один з одним і чому, коли ви відчиняєте двері, у головного героя раптово поповнюється здоров'я. У результаті вам доведеться переглядати різні об'єкти, щоб спробувати зрозуміти, як у вас пов'язані різні елементи. Подібні проблеми вирішуються грамотною організацією коду та ієрархії об'єктів у сцені, а також використанням патернів проектування. Але все це приходить лише з більшим досвідом [29].

Проблеми з продуктивністю. Мультиплатформність двигуна — це, звичайно, благо. Але в той же час на деяких платформах продуктивність гри може бути нижчою, ніж якби ту ж гру зробили за допомогою інструментів, спеціально заточених під конкретну платформу.

Мобільний геймінг у пріоритеті. Не так мінус, як реальність, до якої варто бути готовим. На Unity можна робити різні ігри, включаючи AAA-проекти (triple-A, читається "трипл-ей") — неформальний термін, що позначає клас високобюджетних комп'ютерних ігор. Але історично так склалося, що найчастіше за його допомогою створюють мобільні тайтли. Тому якщо ви освоїте цей ігровий движок і влаштуєтеся в компанію, яка розробляє ігри, то, швидше за все, працюватимете саме над іграми для смартфонів [10, 30].

Спочатку версія Unity 1.0, що вийшла 2005 року, підтримувала три мови програмування: C#; UnityScript, створений під Unity за зразком JavaScript; та Boo, схожий на Python.

Але так склалося, що на C# писали 80,4% розробників ігор. UnityScript користувалося 18,9% користувачів. А на частку Boo припало лише 0,44% розробників. Тому у версії Unity 5.0, що вийшла 2015 року, офіційна підтримка мови Boo була припинена [36].

Ну а 2017 року в компанії оголосили, що починають відмовлятися і від UnityScript. Це рішення пояснили тим, що UnityScript заважав додавати в двигун нові функції, пов'язані з створенням коду. Мова C# розвивалася стрімко, і UnityScript за нею не встигала. А враховуючи, що на той момент лише 3,6% проектів були створені на ньому, в Unity вирішили зосередити свої сили на взаємодії двигуна з C #.

Вже у версії 2017.2 beta з меню Create Assets зник пункт, який відповідав за створення файлу мовою UnityScript (в інтерфейсі він називався JavaScript). Тому зараз C# — це основна мова для роботи в Unity [40].

C# — це об'єктно-орієнтована мова програмування. Вона була розроблена 2001 року компанією Microsoft. Це мова із C-подібним синтаксисом. Тобто вона дуже схожа на Java та C++.

У використання C# в Unity є свої особливості. Під час розробки гри ви створюєте об'єкти GameObjects. Їх поведінку визначають компоненти та скрипти, написані на C#. Кожен скрипт містить клас, який описує, як працюватиме об'єкт. У Unity за умовчанням усі класи успадковуються від базового класу MonoBehaviour. Кожен кадр Unity перевіряє всі дані в сцені і стежить, чи щось змінилося. Це схоже на велику петлю, яку проходить двигун знову і знову. Розробник за допомогою коду пояснює Unity, що має змінитись у наступному кадрі. А клас MonoBehaviour якраз і підтримує цей життєвий цикл. Для цього в ньому є багато методів. Основні з них — це Start() та Update(). Вони будуть створені всередині класу за замовчуванням [17].

Все, що ви напишете в Update(), буде виконуватися кожен кадр роботи гри. У свою чергу, Start() викликається лише один раз до оновлення першого кадру, якщо сам компонент скрипту в об'єкті включений.

Системні вимоги

Залежно від складності вашого проекту та його графіки, вимоги до комп'ютера можуть різнитися.

У різних версіях Unity для Windows мінімальні системні вимоги можуть відрізнятися одна від одної. Також вони відрізняються для версій движка, що працюють на macOS і Linux, і для використання Unity з різними пристроями віртуальної реальності та мобільних платформ. Ознайомитись з ними докладніше можна в офіційній документації [17].

Висновки до розділу 1

(Теоретико-методологічні основи)

У даному розділі було розглянуто жанр ігор стратегія, його різновиди та механіку ігрових жанрів. Проведено аналіз популярних стратегічних ігор деяких піджанрів. Висвітлено історичне походження стратегічних ігор.

Піджанри стратегії пропонують різноманітні геймплейні можливості та особливості. Покрокова стратегія дозволяє гравцеві аналізувати ситуацію, планувати кожен крок та ретельно вивчати свої дії. Вибір піджанру стратегії залежить від особистих вподобань гравця. Для тих, хто насолоджується глибоким аналітичним процесом та плануванням, покрокові стратегії можуть бути ідеальним варіантом.

У кінцевому підсумку, незалежно від обраного піджанру, жанр стратегії пропонує гравцям велику кількість варіантів та можливостей для розвитку стратегічних навичок. Важливо обирати гру, яка найкраще відповідає особистим уподобанням та геймплейним перевагам кожного гравця.

Показано, як використання стратегічних ігор може допомогати у самоорганізації у житті.

Оцінено сучасні інструменти для розробки відеоігор, серед яких обрано рушій Unity як найбільш придатний для створення ігор стратегічного жанру завдяки його функціональним можливостям, гнучкості та поширенню серед розробників. Також аргументовано вибір мови програмування C# для реалізації проекту.

РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНІ АСПЕКТИ РОЗРОБКИ ВІДЕОГРИ СТРАТЕГІЇ НА БАЗІ UNITY

2.1. Концепція ігрового проекту

Жанри ігор — це потужні інструменти для розробників ігор, які дозволяють передавати важливі теми та створювати захопливий і незабутній досвід для гравців. Розумно вибираючи та уникаючи ігрових жанрів, ви можете зробити свою гру особливою та ефективно донести своє бачення [18].

Покрокова стратегія — це жанр стратегічних відеоігор, де гравці здійснюють свої дії по черзі, впродовж кроків. Поки перший гравець не виконає свої дії, інші не можуть почати крок.

Назва жанру походить від того, що процес гри поділений на кроки (ходи). Покрокові стратегії ведуть свою історію з настільних ігор, зокрема, варгеймів.

Жанр воргейму у відеоіграх є похідним від настільних воргеймів. В більшості такі ігри мають вигляд «штабної карти» зі схематичними зображеннями територій, військ. Гравець бере на себе роль генерала чи інших чинів командного складу [25, 33].

У ширшому трактуванні до покрокових стратегій належать взагалі всі покрокові ігри, оскільки всі без винятку ігри містять стратегічну складову. У цьому розумінні в число покрокових стратегій потрапляють, наприклад, такі ігри, як Worms, шахи та преферанс [32].

Для створення свого проекту я обрав найстарішу відому стратегічну гру у світі — Ґо.

Ґо — єдина гра свого сімейства. Батьки її не знайдені, вона ніби з неба впала, а всі нащадки виникли вже у XX столітті. Ґо століттями культивувалася виключно в Азії, вона не має аналогів у Європі, Африці та Америці.

У Стародавньому Китаї гра мала назву «і» (мистецтво), а пізніше — «вейці», що означає «гра оточення». Те саме значення має корейське слово "бадук" і японське "і-ґо", яке в трохи скороченому вигляді розійшлося по світу.

Винахід Ґо приписують придворним китайського імператора Яо. За легендою, його син Даньчжу мав безглузду вдачу, і радник Шунь придумав гру, що навчає гармонії та раціонального мислення. Інша версія стверджує що древньокитайські генерали використовували папір та камені щоб позначати позиції військ [1, 26, 31, 38].

Перші гральні дошки були виявлені в гробницях династії Хань (202 р. до н. е.. — 220 р. н. е..). У період династії Тан (618 – 907) Ґо поширилося у Китаї повсюдно. Існувала навіть така посада — міністр Ґо. Вміння грати увійшло до «чотирьох достоїнств» вченого чоловіка: Ґо, гра на цині (струнний інструмент), каліграфія та живопис [28].

У VII столітті Ґо завезли до Японії. У наступних двох століттях гра була розвагою виключно імператорському будинку, і тільки в 701 році була дозволена для всіх. Саме на островах вона обросла ритуалами, отримала державну підтримку, а за часів сьоґунату перетворилася на військову дисципліну.

Стратегічну гру вважали хорошим тренуванням для управління армією на полі битви. Протягом століть японці домінували у Ґо. Найбільші корпорації виступали спонсорами змагань, було засновано національну федерацію. Проте наприкінці XX століття пальму першості перехопила Південна Корея, але в п'яти їй став наступати Китай [31].

Ґо по праву вважається найскладнішою грою у світі, хоча правила її запам'ятовуються за п'ять хвилин. Більшість ігор не ставлять завдань вище за тактичні. Вони чітко окреслені до кінцевої мети: мат, отримання дамки, виведення фішок. У Ґо перемоги можна досягти мільйоном різних способів, тому на перший план виходить стратегія: заплутати противника, приховати від нього свої наміри, розставити сіті і заманити в пастку [14].

Дивно, але за багато століть правила Ґо майже не змінилися. Все досить просто. Для гри використовується дошка, на яку нанесено 19 ліній по горизонталі та вертикалі, що утворюють 361 квадрат (ігровий пункт). Два гравці по черзі заповнюють ці пункти камінням. Для зручності камені пофарбовані в контрастні

кольори, наприклад, чорний і білий. Суть гри в тому, щоб заробити якнайбільше очок. Для цього можна захоплювати територію (оточувати порожні пункти своїм камінням) або знищувати каміння супротивника. Щоб захопити ворожий камінь, треба оточити його з усіх боків. Отут і починаються складнощі. Хоча правила досить прості, Ґо набагато складніше шахів і тим більше шашок. Початківці тренуються на дошках13x13, 9x9 і навіть 5x5. Противники по черзі виставляють камені на перехрестя ліній або пасують. Мета гри — відгородити собі якнайбільше вільних перехресть. Кожен гравець має достатньо каменів, щоб заповнити всю дошку [26].

361-клітинне ігрове поле створює безліч варіантів. Для порівняння: якщо в шахах після четвертого ходу виникає близько 100 000 можливих позицій, то в Ґо їх кількість перевищує 16 мільярдів.

Гра Ґо на дошці 19×19 дозволяє близько $2{,}081681994\times10^{\mathbf{170}}$ комбінацій.

В шахах машини давно навчилися обігравати кращих гросмейстерів, а в Ґо перемога для них наступила відносно недавно. На це є кілька причин. По-перше, як вже було сказано вище, кількість варіантів у грі настільки величезна, що навіть простий механічний перебір вкрай скрутний. По-друге, виникають серйозні проблеми з оцінкою позиції. Якщо в шахах давно вироблені критерії оцінки позиції та визначено цінність фігур, то в Ґо все набагато складніше. Навіть найпрофесійніші гравці сперечаються про можливості тієї чи іншої комбінації. Одна з найкращих програм з гри в Ґо - це MoGoTitan . Вона працює на суперкомп'ютері потужністю 15 терафлопсів, тобто обробляє 15 трильйонів операцій на секунду і грає лише на рівні сильного любителя [31].

В жовтні 2015 року рівень розвитку комп'ютерної техніки та алгоритмів досяг такого рівня, що створений інженерами лабораторії DeepMind алгоритм AlphaGo зміг вщент розбити чинного чемпіона Європи з Ґо Фен Г'юї — комп'ютер виграв 5 ігор із п'яти. У березні 2016 року відбувся матч із Лі Седолом, якого називають найвидатнішим гравцем у Ґо десятиліття (9 дан). AlphaGo було здобуто 4 перемоги з 5 ігор. У жовтні 2017 року DeepMind повідомила про значно

сильнішу версію програми — AlphaGo Zero, яка перемогла попередню версію в усіх іграх серії зі 100 ігор. Важливою відмінністю AlphaGo Zero є навчання моделі лише за рахунок ігор проти себе, без жодної попередньої інформації про стратегії Ґо [1, 14, 31].

Прості тактичні прийоми вимагають постійного переосмислення, зазвичай виграшних дебютів немає. Оцінити потенціал усіх побудов, реалізувати його і не дати це зробити супротивникові — найскладніші завдання, які вирішують одночасно.

На Сході кажуть: «Ґо вчить жити». Агресивна манера гри може призвести до втрат, а глухий захист до нестачі очок. Найчастіше зростанню заважає не відсутність навички, а вади характеру: розсіяність, забудькуватість, нерішучість, жадібність. Щоб удосконалюватися в ньому, гравцеві залишається тільки одне — вдосконалювати себе [28].

Будь-яка настільна гра є таким же тренуванням для мозку, як спорт — для м'язів. Однак складність і глибина давньої східної гри виділяє її серед інших настільних ігор.

Довга історія дозволила виявити одну важливу закономірність — серед професійних гравців вкрай рідко зустрічалася стареча деменція. Пояснення відшукали нещодавно. До хвороби Альцгеймера багато в чому призводить небажання людини напружувати свій мозок, вчитися новому, виконувати складні розумові завдання. А регулярні заняття Ґо якраз вимагає частого розумового напруження.

Білл Гейтс, Сергій Брін, Джек Ма, Генрі Кісінджер. Всі ці люди грали в Ґо і високо цінували її застосування у бізнесі та політиці.

Джек Ма, наприклад, зауважив, що "принципи, засвоєні в Ґо, заклали основу для мого бізнес-успіху". Директор з маркетингу авіакомпанії Japan Airlines і виконавчий директор готельної мережі Nikko Ясуюкі Міура переконаний, що гра в Ґо допомагає грамотно розділити ринки і потім налагодити гармонійне співіснування. Він навіть присвятив цій темі книгу.

Генрі Кісінджер звертав увагу на те, що в Ґо, на відміну від шахів, мета полягає в розширенні власних можливостей та звуженні можливостей противника. Але ніяк не його повне знищення [35].

Шанувальники Ґо із сфери бізнесу розглядали гобан як аналог ринків та територій, а камені — аналог ресурсів компанії. При такому підході кожна партія в Ґо стає моделлю завоювання нових ринків та вибудовування відносин з конкурентами. Стратегії Ґо цілком застосовні до ситуацій реального життя. Зокрема, вміння грати в Ґо дозволяє вирішувати цілий перелік завдань, які постійно постають перед кожним бізнесом.

Ґо не панацея. І не має ставати єдиним інструментом прийняття рішень. Але вона дозволяє по-новому висвітлити рутинні речі [28].

2.2. Об’єктно-орієнтований підхід в грі

Одна з найважливіших методологій розробки, про яку неодмінно слід знати майбутнім спеціалістам це об’єктно-орієнтоване програмування (ООП). Знаючи поняття та принципи ООП та розуміючи, як вони взаємодіють, розробники можуть створювати більш ефективні, структуровані, масштабовані та потужні програми [13].

Які є об'єктно-орієнтовані мови програмування

Об’єктно-орієнтовані мови програмування є потужним інструментом в руках будь-якого розробника. Використовуючи об’єкти, класи та структури даних, інженери створюють надійні, ефективні та практичні програми. Найкращими об’єктно-орієнтованими мовами програмування ввжаються Java, C++ і Python. З ними розробники можуть створювати більш чутливі та інтерактивні додатки, що, своєю чергою, дозволяє користувачам мати приємніший і продуктивніший досвід з програмою. Також популярними мовами ООП є Ruby, C# і PHP.

Переваги об'єктно-орієнтованого програмування

Об’єктно-орієнтоване програмування революціонізувало спосіб написання коду. Від великомасштабних корпоративних програм до невеликих додатків — ООП забезпечує потужний і гнучкий підхід до розробки. З ним розробники мислять в термінах об’єктів, класів і методів, роблячи код більш інтуїтивно зрозумілим і простішим в обслуговуванні. Окрім того, ООП дає змогу створювати ефективний і «багаторазовий» код, скорочуючи час і вартість розробки в цілому, а також додавати нові функції до існуючого коду.

Основні принципи ООП

У цьому світі будь-що є об’єктом, чи не так? Ну, майже все. Однак, саме ствердження про те, що все є об’єктом, лежить в основі ООП. Вся парадигма об'єктно-орієнтованого програмування — це концепція об'єктів. Ці об’єкти містять дані, які ми також називаємо атрибутами або властивостями, і методи. Об'єкти можуть взаємодіяти один з одним за чотирма принципами: інкапсуляція, наслідування, поліморфізм та абстракція.

Інкапсуляція (*ENCAPSULATION)*: це стосується обгортання вмісту сутності в одну одиницю. У термінології ООП це стосується зв'язування та обгортання атрибутів класу або об'єкта (стану) з їхніми методами (поведінкою). Завдяки інкапсуляції об'єкти можуть мати свій власний приватний стан, до якого не можуть отримати доступ інші об'єкти, якщо їхні методи або атрибути не оголошено публічними. Цей аспект ООП дозволяє створювати безпечніші реалізації програмного забезпечення.

Наслідування (*INHERITANCE*): Наслідування є однією з важливих концепцій в ООП, яка дозволяє (дочірнім) класам встановлювати ієрархію шляхом успадкування атрибутів та методів іншого (батьківського) класу. Це зменшує надмірність, оскільки класи можуть мати спільну логіку, структуру та атрибути, одночасно забезпечуючи чітку ієрархію.

Поліморфізм (POLYMORPHISM): У простому сенсі, поліморфізм стосується здатності кількох об'єктів мати однакове ім'я, але різні структури або

виконувати різні функціональні можливості в різних контекстах. Поліморфізм легко спостерігати при перевантаженні функцій та перевизначенні функцій.

Абстракція (ABSTRACTION): Абстракція в термінології ООП стосується здатності класів розкривати певні атрибути даних, зберігаючи інші приватними. Зазвичай це робиться для того, щоб приховати деталі реалізації від зовнішнього світу, або зробити речі менш складними, або більш безпечними. Це робиться за допомогою різних специфікаторів доступу, які визначають видимість кожного атрибута класу [11].

Секрети успіху роботи з ООП

Секрети успіху ООП полягають в правильному застосуванні його принципів. Для того, щоб успішно розробляти програми з використанням ООП, необхідно дотримуватися наступних рекомендацій:

1. Не допускайте складності в коді. Об'єктноорієнтований код може стати дуже складним, тому зробіть простоту та ефективність пріоритетами розробки.
2. Дотримуйтесь принципів SOLID, аби уникнути проблем. Вони не є панацеєю, але допомагають зберегти код гнучким, легким у підтримці та зміні.
3. Розумійте, які класи вам потрібні та як вони пов'язані між собою. Важливо усвідомлювати, як класи взаємодіють між собою, щоб забезпечити ефективну роботу програми.
4. Не зловживайте наслідуванням. Наслідування може стати дуже складним та заплутаним, якщо використовувати його забагато. Важливо зрозуміти та відстежувати, коли цей принцип необхідно застосувати, а коли натомість звернутися до об’єктної композиції.
5. Використовуйте абстракцію для спрощення розуміння.
6. Дотримуйтеся стандартів та конвенцій програмування. Це допоможе зберегти код зрозумілим та легким у підтримці [13].

C# є об’єктно-орієнтованою мовою програмування, що означає, що вона орієнтована на створення об’єктів і класів для організації коду. Це сприяє модульності, повторному використанню коду та спрощенню розробки.

Об’єктно-орієнтований підхід дає змогу створювати чистіший і структурованіший код, що полегшує його розуміння та підтримку.

Порівнявши C# з іншими популярними мовами програмування, такими як JavaScript, Java і C++, можна виділити такі переваги:

- простий синтаксис: C# має легкий для розуміння синтаксис, що робить його доступним для новачків;
- сильна типізація: C# забезпечує високий рівень безпеки та запобігає безлічі помилок на етапі компіляції;
- Велика стандартна бібліотека: .NET Framework надає велику стандартну бібліотеку, що містить безліч готових компонентів та інструментів;
- Інтеграція з Windows: C# чудово інтегрується з операційною системою Windows, даючи змогу створювати настільні додатки та системне ПЗ [40].

Середовища та інструменти які надають можливості для розвитку високопродуктивних додатків: Visual Studio, Visual Studio Code та Ride.

Visual Studio це інтегроване середовище розробки з багатою функціональністю, яке полегшує створення високопродуктивних додатків, його особливості: інтелектуальні підказки та автодоповнення, налагодження та профілювання, інтеграція з .NET і Azure, підтримка розробки мобільних і веб-додатків.

Visual Studio Code це легковаге і безкоштовне середовище розроблення, що також надається Microsoft. Воно вирізняється високою продуктивністю і розширюваністю. Особливості VS Code включають: безліч плагінів і розширень, інтеграція з Git, крос-платформеність.

Ride — середовище розробки для C# від компанії JetBrains. Воно вирізняється високою продуктивністю і безліччю просунутих інструментів. Особливості Rider включають автоматичний рефакторинг, інтеграцію з Unity та аналіз коду.

Ці середовища для програмування на C# забезпечують усю необхідну функціональність для створення високопродуктивних додатків цією мовою.

Вибір конкретного середовища залежить від уподобань і вимог проекту.

Переваги використання C# у розробці високопродуктивних додатків.

C# має інтуїтивно зрозумілий синтаксис, що робить його відносно простим в освоєнні для початківців. Він також має безліч вбудованих функцій і бібліотек, які можуть допомогти прискорити процес розробки, наприклад:

- C# є типо-безпечною мовою, тобто компілятор перевіряє типи даних під час компіляції програми. Це допомагає уникнути помилок, які можуть призвести до зниження продуктивності.
- Об'єктно-орієнтоване програмування дає змогу створювати складні додатки, які легко масштабувати та підтримувати. Це може підвищити продуктивність, особливо для додатків, які обробляють великий обсяг даних.
- C# підтримує багатопоточність, тобто можна виконувати кілька завдань одночасно. Це підвищує продуктивність, особливо для додатків, які обробляють великий обсяг даних [40].

Інтеграція з різними платформами та технологіями

C# підтримує широкий спектр платформ, включно з Windows, macOS, Linux, iOS і Android.

Він також може бути інтегрований з різними технологіями, такими як бази даних, веб-сервіси та пристрої. Завдяки інтегруванню з різними хмарними технологіями, як-от Azure і AWS, розробники можуть підвищити продуктивність своїх додатків, переклавши частину навантаження на хмарну інфраструктуру. C# може бути інтегрований з різними базами даних, такими як SQL Server і MySQL та з різними веб-сервісами [40].

2.3. Архітектура проекту та реалізація логіки гри

Архітектура програмного продукту — це спосіб виокремлення компонентів програмної системи на найвищому рівні абстракції та спосіб їх взаємодії. На вибір архітектури програмного продукту впливають функціональні вимоги, технічні вимоги та кваліфікація розробника. Функціональні вимоги

випливають з предметної області та призначення програмного продукту. Технічні вимоги випливають з можливостей цільових пристроїв та наявних технологій розробки програмного продукту. Кваліфікація розробника визначає, чи зможе він передбачити життєвий цикл програмного продукту.

Будувати архітектуру програмного продукту можна з нуля, а можна — використовувати архітектурні шаблони. Найвідомішими та найбільш вживаними архітектурними шаблонами є Layered Architecture Pattern, EventDriven Architecture Pattern, Microkernel Architecture Pattern, Microservices Architecture Pattern, Model-View-Controller Design Pattern (MVC), Model-ViewPresenter Design Pattern та Model-View-ViewModel Design Pattern.

У розробці моєї гри я використовую шаблон MVC. Розглянемо детальніше Model-View-Controller Design Pattern (рис. 2.1).

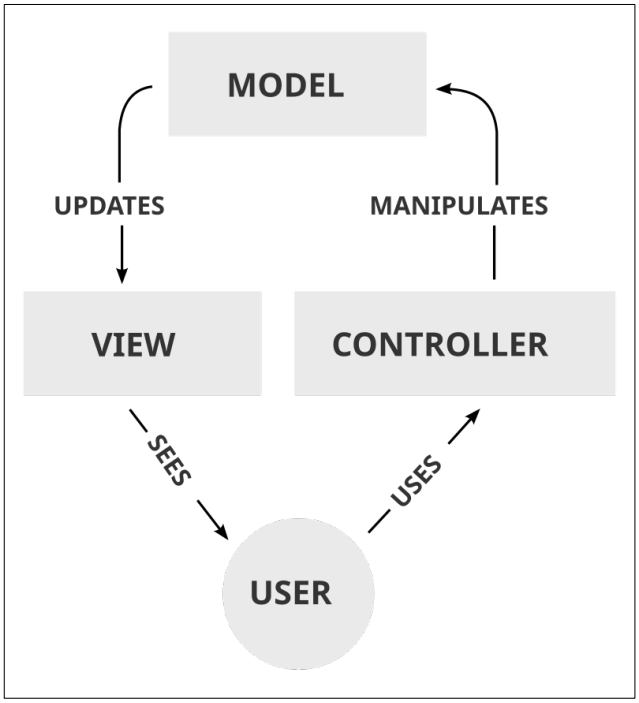


Рис. 2.1 Візуальне представлення MVC [12]

Цей шаблон містить такі архітектурні компоненти: Model — відповідає за доступ до бази даних та містить бізнес-логіку програми; View — відповідає за створення представлення та його відображення; Controller — відповідає за формування команд із отриманих користувацьких запитів [23].

Виконання користувацьких запитів відбувається наступним чином. Користувач формує запит за допомогою елементів View. View передає користувацький запит відповідному Controller. Controller при отриманні користувацького запиту викликає методи Model для маніпуляції з користувацькими даними або для зміни стану моделі. При отриманні відповіді Controller віддає її View. Якщо стан Model змінюється, то модель сповіщає про це View та передає оновлені дані. View відображає відповідь користувачу, а при оновленні стану Model перебудовує представлення та перемальовує відображення [29].

Розширення (Plug-in або Add-on) — це програмний компонент, що змінює або розширює функціональні можливості ядра програмного продукту, але програмний продукт може існувати і без них.

Гра в Unity розглядається як комплекс ігрових сцен, а кожна ігрова сцена — як організована сукупність ігрових об’єктів. Призначення та функціональні можливості ігрового об’єкта визначаються тим, які ігрові компоненти до нього прив’язані. До ігрових компонентів належать зображення, 2D або 3D модель, колізія, анімація, освітлення, камера, керування гравцем, аудіо тощо. Створення гри — це процес створення сцен, розташування ігрових об’єктів, додавання компонентів, налаштування властивостей компонентів, додавання скриптів. Його можна виконувати як в графічному інтерфейсі користувача так і в редакторі коду, наприклад, Visual Studio або Rider.

Створення користувацького інтерфейсу суттєво не відрізняється від створення гри, оскільки графічний інтерфейс користувача — це така ж ігрова сцена, на якій розміщені ігрові об’єкти — кнопки. З технічної точки зору, створення гри відбувається з використанням мови програмування C# та фреймворку Unity. Фреймворк містить набір шаблонів програмування, машину станів ігрових об’єктів та готові рішення для стандартних задача, як, наприклад, організація ігрової кімнати, синхронізація стану гри тощо.

Дані гри в Unity можуть зберігатися локально в реєстрі Windows, або у файлах файлової системи чи в базі даних. Віддалене збереження даних гри, зазвичай, виконують в базі даних або в хмарі.

Розглядаючи потенційні фінансові витрати на розробку можна сказати, що основні функції Unity доступні безкоштовно, а от на деякі покращені модулі необхідно оформлювати щомісячну або щорічну підписку.

Огляд можливостей технології .NET

.NET — це платформа з відкритим вихідним кодом для розробки програмного забезпечення різних типів. .NET — це мови програмування C#, F#, Visual Basic та набір бібліотек. Ця платформа має декілька варіантів реалізації, які відрізняються підтримуваними апаратними платформами та функціональними можливостями: .NET Core — працює під операційними системами Windows, Linux, macOS; .NET Framework — розроблена виключно для операційної системи Windows; Xamarin/Mono — оптимізована для використання на мобільних платформах. Використовуючи .NET можна розробляти: Web-застосунки; Desktopзастосунки для Windows, Linux, macOS; мобільні застосунки для iOS, Android; мікросервісні застосунки; ігрові застосунки; застосунки для реалізації машинного навчання; застосунки для реалізації інтернету речей; застосунки для реалізації хмарних технологій [21].

.NET не вимагає коштів для отримання ліцензії та інструментів розробки в тому числі для комерційного використання.

Висновки до Розділу 2

(Системно-аналітичні аспекти)

У другому розділі було обрано гру для реалізації у проекті. Це відома найстаріша гра в жанрі стратегіі — Ґо.

Була розкрита історія появи цієї гри, її вплив на суспільство та її роль в житті людини.

Проведено аналіз використаних інструментів Unity та C#.

Зроблено вибір архітектури програмного продукту, який має визначальний вплив на якість, масштабованість і підтримку гри. Архітектурний шаблон MVC (Model-View-Controller), обраний для реалізації гри, забезпечив мені чітке розділення логіки, інтерфейсу та обробки запитів, що сприяло гнучкості та зручності в розробці.

Unity як рушій ігрової розробки надає широкі можливості для створення 2D/3D гри, включаючи управління сценами, об'єктами, фізикою, анімацією та взаємодією з користувачем. Його підтримка збереження даних, як локально, так і віддалено, дозволяє ефективно реалізовувати збереження прогресу гравця.

Платформа .NET це інструмент для розробки багатоплатформених застосунків, включаючи ігрові проекти, і не потребує додаткових витрат на ліцензії. Я використовував .NET Framework 4.7.1.

Мова програмування C# має безліч переваг — це універсальна мова програмування. В неї простий синтаксис та широка стандартна бібліотека. Велика спільнота розробників роблять C# чудовим вибором для розроблення високопродуктивних додатків.

Також було розглянуто найпоширенішу парадигму програмування — ООП. По суті це фундамент програмування та рамки для розробників та організацій для того, щоб вони могли обміркувати, як вони хочуть структурувати свій код, та яка логічна організація та потік найкраще підходять для їхнього застосунку.

Узагальнюючи поєднання теорії ігор, архітектурних рішень парадигми програмування та сучасних технологій, можна зробити висновок, що Unity дозволяє створювати функціональні, масштабовані та ефективні програмні продукти у сфері ігрової індустрії.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВІДЕОГРИ СТРАТЕГІЇ У СЕРЕДОВИЩІ UNITY

3.1. Створення ігрового прототипу: юніти, сцени, механіка бою

Правила гри Ґо (див. у Додатку А).

В моїй відеогрі Ґо — юніти це каміння чорного та білого кольорів (рис .3.1), які ототожнюють собою дві армії.

Боротьба відбувається за правилами класичної гри Ґо. Сцена битви створена варіативно (рис. 3.2) — натисканням на кнопку можна обрати потрібний варіант (рис. 3.3).

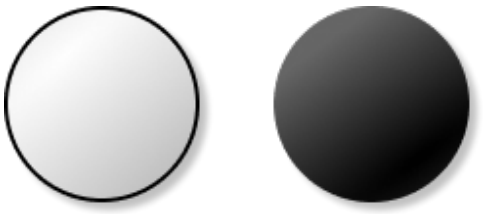


Рис. 3.1 Камені білого та чорного кольорів

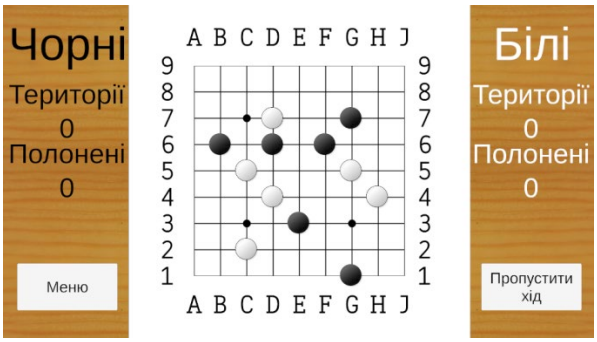


Рис. 3.2 Сцена битви 9x9 з каменями



Рис. 3.3 Меню вибору дошки

Було створено 3 дошки різних розмірів: 9x9, 13x13 та 19x19 (рис. 3.4).

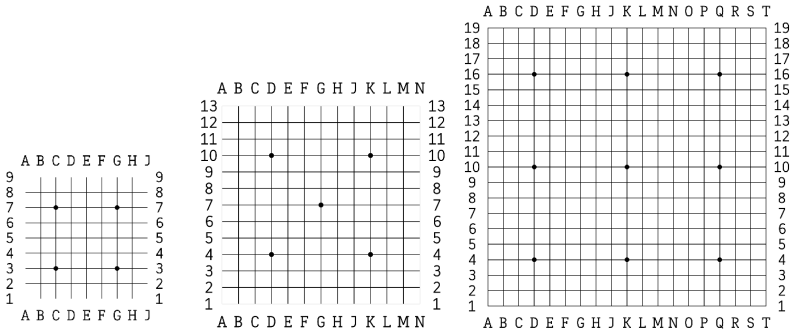


Рис. 3.4 Розміри дошок Ґо з нумерацією

Сама гра починається у головному меню де гравцям дається на вибір три кнопки: «Почати Гру», «Налаштування» та «Вийти» (рис. 3.5).



Рис. 3.5 Сцена головного меню

У меню налаштувань гравці можуть змінити гучність: окремо музики — повзунок «Музика», та окремо звукових ефектів — повзунок «Звуки» або разом одним повзунком «Гучність» (рис. 3.6).



Рис. 3.6 Меню налаштувань

Під час ігрового процесу гравці можуть натиснути на кнопку «Меню» (рис. 3.2), тим самим відкрити меню паузи (рис. 3.7). Гравці можуть одразу повернутись до битви натиснувши кнопку «Продовжити гру» або натиснути кнопку «Нова Гра», якщо є таке бажання — сцена перезапуститься, а дошка та рахунок обнуляться для нової гри. Також є можливість відкласти партію на потім

натиснувши кнопку «Зберегти та вийти». Стан партії збережеться до того моменту коли гравці захочуть її продовжити.

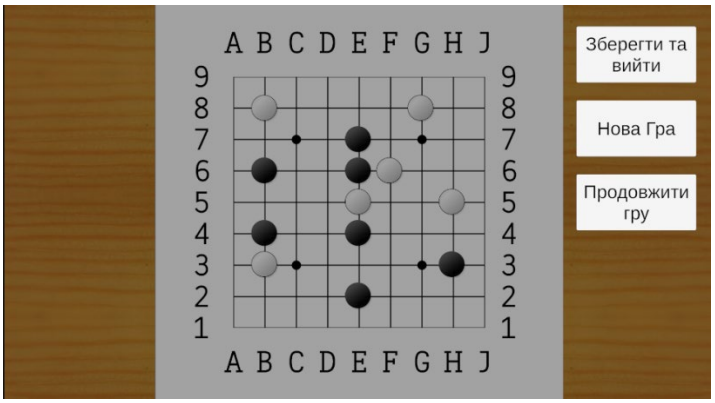


Рис. 3.7 Меню паузи

У кінці гри на екран виводиться рахунок партії, що підбиває підсумки, рахує бали та пише сторону переможця (рис. 3.8). На екрані завершення партії гравці можуть натиснути кнопку «Нова гра», щоб одразу розпочати нову партію. Також є можливість повернутися до головного меню натиснувши кнопку «Вийти».



Рис. 3.8 Вигляд екрану при завершенні партії

Пояснення до деяких кодів

Дошка

У грі Ґо основним елементом є ігрова дошка, яка складається з квадратної сітки — у мене це 9x9, 13x13 або 19x19. У моїй грі дошка реалізована через скрипт DeskManager (див. у Додатку Б ст. 68). Саме він відповідає за збереження поточного стану клітинок, ініціалізацію порожньої дошки та доступ до окремих

каменів. В Unity цей скрипт прикріплено до окремого об’єкта на сцені. Наприклад, ось так ініціалізується порожня дошка: (див. у Додатку Б ст. 69)

Тут створюється двовимірний масив, який представляє дошку, де кожна клітинка — це тип Stone (Empty, Black або White). public enum Stone { Empty, Black, White } Після ініціалізації масиву пунктів, кожен з них буде відповідати порожньому пункту — Stone.Empty.

Камені

Камені у грі Ґо — це основний елемент, яким гравці по черзі займають перетини дошки. У моїй реалізації Unity відповідає за візуальне представлення каменів, тоді як вся логіка їхнього розміщення відокремлена в інші класи. За створення та знищення каменів у сцені відповідає клас StoneManage (див. у Додатку Б ст. 67). Це метод PlaceStone, який викликається для розміщення каменя на дошці (див. у Додатку Б ст. 67). У методі обчислюється позиція на сцені відповідно до координат дошки, обирається префаб відповідного кольору (чорний або білий), а потім створюється об’єкт каменя та додається до ієрархії об'єктів у сцені.

Черга Ходу

У Ґо гравці по черзі ставлять камені, і цю черговість я реалізував у класі TurnManager. Сам скрипт дуже компактний у порівнянні з іншими: public class TurnManager (див. у Додатку Б ст. 74).

Коли гравець робить хід або пас, GameManager (див. у Додатку Б ст. 77). звертається до властивості IsBlackTurn, щоб вибрати колір каменя, а після успішного застосування ходу викликає SwitchTurn щоб змінити чергу.

Зняття каменів з дошки

У Ґо одна з головних механік — це зняття каменів, які були повністю оточені суперником. У моїй реалізації ця логіка побудована навколо класу GroupFinder (див. у Додатку Б ст. 70). Коли гравець ставить камінь, я запускаю перевірку сусідніх клітин, якщо поруч є камені супротивника, гра шукає їхню групу за допомогою методу FindGroup: (див. у Додатку Б ст. 70)

FindGroup повертає групу каменів що потім використовується для перевірки на оточення за допомоги методу IsGroupSurrounded, визначається, чи ця група повністю оточена і не має «дихань» (див. у Додатку Б ст. 71). Якщо це так — усі оточені каменізнімаються з дошки за допомоги іншого методу з StoneManager (див. у Додатку Б ст. 67), а саме RemoveStone (див. у Додатку Б ст. 68).

Перевірка недопустимих ходів

У Ґо гравець не може поставити камінь на зайняту клітинку, зробити самогубний хід, або порушити правило «Ко». У моїй реалізації, ці перевірки виконуються в момент спроби гравця поставити камінь через GameManager (див. у Додатку Б ст. 77). Класс MoveValidator містить методи, що відповідають за перевірку цих правил (див. у Додатку Б ст. 74).

У методі IsValidMove спочатку перевіряється, чи клітинка вже зайнята (див. у Додатку Б ст. 75). Це базова перевірка: якщо на цій позиції вже є камінь, гравець не може туди ходити. Потім відбувається симуляція ходу, щоб перевірити, чи не є він самогубним. Створюється тимчасова копія дошки, на нійрозміщується камінь і перевіряється, чи після цього ходу нова група має принаймні одне «дихання», IsSelfCapture (див. у Додатку Б ст. 75). Тут GroupFinder.IsGroupSurrounded — це той самий метод, що я використовую і для зняття каменів. Якщо нова група після ходу повністю оточена і не захоплює каменів суперника, то хід вважається самогубством і забороняється.

Окремо перевіряю порушення правила «Ко». Це правило забороняє робити хід, який відразу ж відновлює попередній стан дошки, це реалізовано у мене через IsKoViolation (див. у Додатку Б ст. 75). Якщо вони однакові, хід забороняється.

Обрахування захоплених територій

Після завершення партії в Ґо важливо визначити, яка частина поля вважається територією кожного гравця. У моїй грі це реалізовано за допомогою окремого класу TerritoryManager (див. у Додатку Б ст. 71). Його основна задача

— пройтися по всіх порожніх клітинках дошки, згрупувати їх і перевірити, чи ці групи оточені лише каменями одного кольору.

Підрахунок починається методом CountTerritory (див. у Додатку Б ст. 72). Цей метод перевіряє кожну порожню клітинку, яка ще не була врахована, і викликає метод FindEmptyGroup для знаходження всіх зв’язаних пустих клітинок (див. у Додатку Б ст. 72). У кінці перевіряється чи оточена група одним єдиним кольором — IsGroupSurroundedByColor (див. у Додатку Б ст. 73). Якщо так територія зараховується як захоплена.

Підрахунок балів

Підрахунок балів у моїй грі відбувається динамічно — після кожного ходу, оновлюючи UI гравців, тим самим виводячи поточний стан гри. Для цього я створив ScoreManager, а у ньому методи UpdateScores та UpdateUI (див. у Додатку Б ст. 76 – 77).

Методи звертаються до текстових обьєктів на сцені, змінюючи їх стан відповідно до рахунку партії.

ScoreManager

```
using TMPro;
using UnityEngine;

public class ScoreManager : MonoBehaviour
{
    // UI-текст для кількості полонених
    public TextMeshProUGUI blackPrisonersText;
    public TextMeshProUGUI whitePrisonersText;

    // UI-текст для кількості території
    public TextMeshProUGUI blackTeritoryText;
    public TextMeshProUGUI whiteTeritoryText;

    // Поточна кількість полонених
    public int blackPrisoners;
    public int whitePrisoners;

    // Поточна кількість території
    public int blackTerritory;
```

```
    public int whiteTerritory;

    // Оновлює значення рахунку та UI
    public void UpdateScores(int blackPrisoners, int whitePrisoners, int blackTerritory,
int whiteTerritory)
    {
        this.blackPrisoners = blackPrisoners;
        this.whitePrisoners = whitePrisoners;
        this.blackTerritory = blackTerritory;
        this.whiteTerritory = whiteTerritory;

        UpdateUI();
    }

    // Оновлює текст UI

    public void UpdateUI()
    {
        blackPrisonersText.text = $"{blackPrisoners}";
        whitePrisonersText.text = $"{whitePrisoners}";
        blackTeritoryText.text = $"{blackTerritory}";
        whiteTeritoryText.text = $"{whiteTerritory}";
    }
}
```

Завершення гри

У Го партія завершується, коли обидва гравці передають хід два рази підряд, тобто не бажають більше робити жодного ходу. У моїй реалізації це враховано через систему підрахунку пасів у класі GameManager (див. у Додатку Б ст. 77). Я зберігаю лічильник кількості поспіль зроблених пасів — PassTurn (див. у Додатку Б ст. 78), і якщо їх стає два — гра вважається завершеною.

Після другого пасу викликається метод EndTheGame (див. у Додатку Б ст. 82). Метод підраховує кількість полонених та територій для кожного з гравців. Потім на екран виводиться підсумок завершеної партії з остаточним рахунком.

3.2. Інтеграція графічних елементів, анімації та звукового супроводу

Будь яка робота по розробці гри починається з вікна редактора Unity (рис. 3.9). Робочий простір Unity можна гнучко налаштовувати під себе, змінюючи розміри інтерфейсів, змінюючи їх положення, кількість та інше.

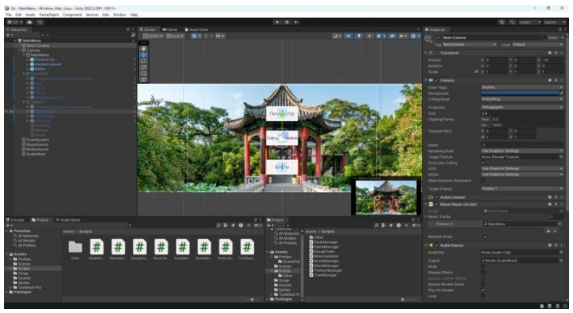


Рис. 3.9 Інтерфейс редактора Unity

Менеджер проекту це місце де зберігаються усі файли майбутньої гри. Ассети, що знадобляться при розробці, можна зберігати та сортувати по папкам. Це графічні зображення, аудіо, відео, шрифти, скрипти, префаби, налаштування для імпортованих бібліотек тощо (рис. 3.10).



Рис. 3.10 Менеджер проекту

Менеджер налаштування аудіо, або мікшер використовується для деталізованої роботи з аудіо та супроводом відеогри (рис. 3.11). У мікшері розробник може виділити та поділити на групи джерела різних аудіо таких як звуки або музика для подальшого налаштування.

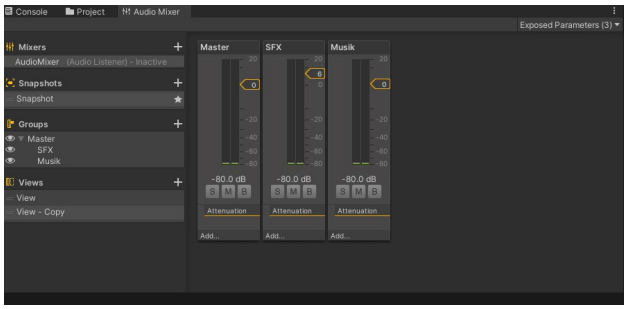


Рис. 3.11 Менеджер налаштування аудіо

Меню ієрархії об’єктів допоможе налаштувати сцену (рис. 3.12). Додаючи або видаляючи об’єкти зі сцени можна визначити їх порядок та належність один до одного.

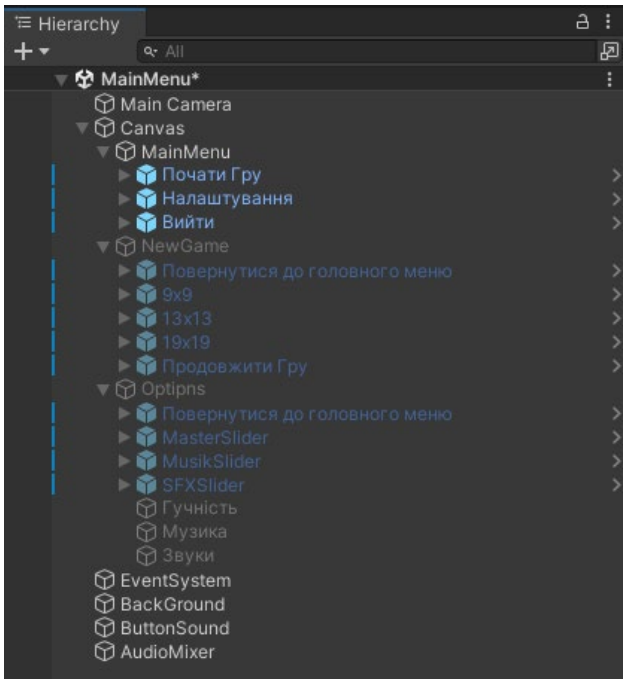


Рис. 3.12 Ієрархія об’єктів на сцені

У свою чергу вікно самої сцени візуалізує додані на неї об’єкти, дозволяючи маніпулювати їми (рис. 3.13). Виділяти, переміщувати, зменшувати, збільшувати, маштабувати, обертати та інше.

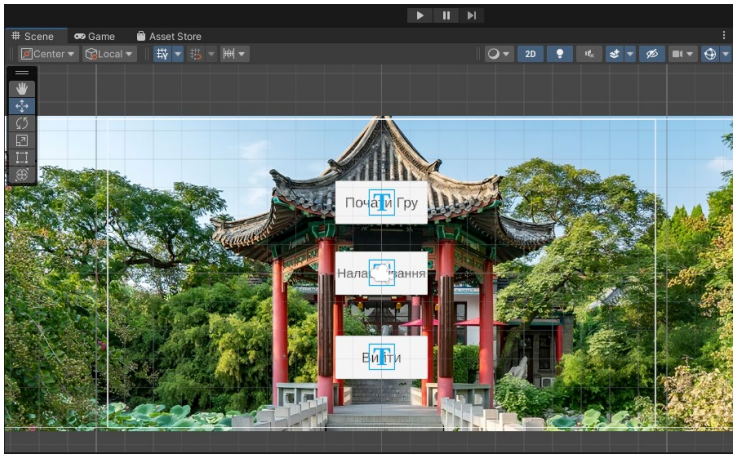


Рис. 3.13 Огляд сцени у редакторі

Але щоб працювати з об’єктами більш детально, знадобиться вікно інспектору (рис. 3.14). У вікні інспектора перераховані усі властивості обраного в Ієрархії об’єкта та налаштування обраного файлу. Параметри властивостей

об’єктів можна тонко налаштовувати під будь які потреби — додаючи нові або видаляючи попередні.

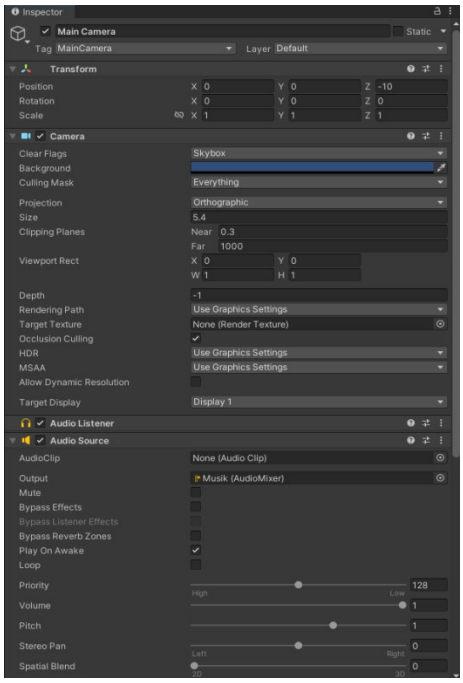


Рис. 3.14 Інспектор

3.3. Проведення тестування, оптимізація продуктивності, аналіз результатів

Фази ігрового тестування. Слід планувати проходження трьох основних етапів ігрового тестування. Перший етап починається з першої версії навчальної гри, яку ви створюєте — вашого паперового прототипу.

Друга фаза передбачає ігрове тестування з людьми поза командою розробників. Третій і останній етап буде з грою, яка виглядає майже завершеною. Кожна фаза передбачає кілька раундів ігрового тестування та аналіз результатів тестування [37].

Тестування створюваних ігор можна проводити прямо у редакторі, запускаючи майбутню гру у вікні сцени. Можна подивитися на те як об’єкти, їх властивості та скрипти взаємодіють між собою; виявити погану роботу, проаналізувати та виправити (рис. 3.15). Нові правки можна внести одразу, закінчивши процес компіляції. На основі аналізу поведінки об’єктів та скриптів можна робити висновки, змінюючи їх у потрібному напрямку, для цього використовують зовнішнне програмне, у мене це Visual Studio 2022 (рис. 3.16).

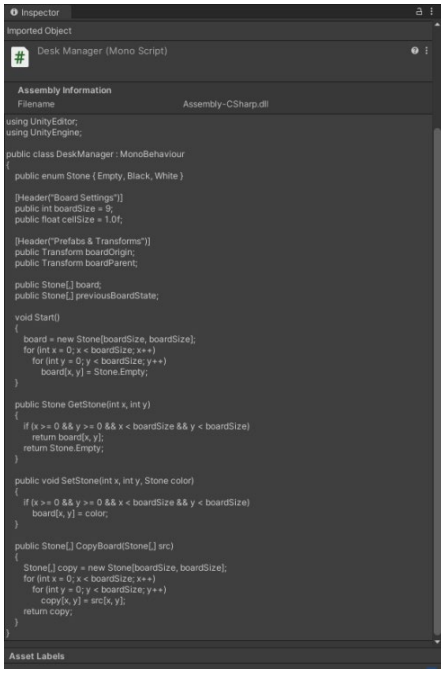


Рис. 3.15 Огляд скрипта у інспекторі

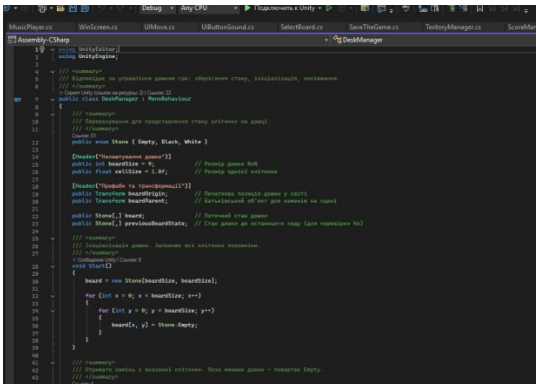


Рис. 3.16 Інтерфейс Visual Studio 2022

Для процесу дебагінгу, пошуку помилок, або їх джерел знадобиться консоль налагодження (рис. 3.17). Під час компіляції, у консолі налагодження, розробник може побачити виникаючі помилки, попередження, або заздалегідь задані повідомлення для дебагінгу.

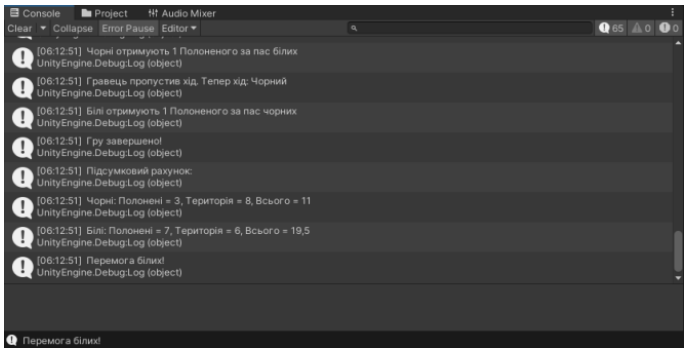


Рис. 3.17 Консоль налагодження

Посилання для перегляду розробленої гри (рис. 3.18) в браузері:



Рис. 3.18 QR-код для перегляду розробленої гри

Також можна скачати apk-файл для Android (рис. 3.19) за посиланням:



Рис. 3.19 Apk-файл для Android

Висновки до Розділу 3

(Проектно-прикладна реалізація)

У третьому розділі реалізовано розробку функціонального прототипу відеогри стратегії на платформі Unity. Створено базові сцени, моделі юнітів, механізми бою та інтерфейси користувача.

Інтегровано графічні, анімаційні та аудіоелементи, що забезпечують якісне візуальне та звукове сприйняття гри.

Проведено тестування продуктивності, виявлено та усунено основні недоліки, що дозволило досягти стабільної роботи проекту при заданих параметрах.

Отримані результати свідчать про успішне виконання практичної частини дослідження.

ЗАГАЛЬНІ ВИСНОВКИ

Як бачите, стратегічні ігри – це тип ігор, які часто мають грандіозний масштаб, зосереджуючись на виборі, який, доведеться обміркувати, а не на негайних діях, з якими може зіткнутися піхотинець, дивлячись у приціл гвинтівки. Вони вимагають іншого настрою.

У кваліфікаційній роботі «Розробка відеогри стратегії з використанням ігрового рушія Unity: від концепції до реалізації» було комплексно розглянуто всі етапи створення проекту: від теоретичного обґрунтування до практичної реалізації.

Продуктом моєї роботи є стоворенна відеогра Ґо. Описано процес побудови ігрової механіки та підрахунку балів.

Пояснено основні правила гри да додано ілюстраціонний матеріал для кращего візуального розуміння інформації.

Проведено огляд сучасних тенденцій у розробці ігор, аргументовано вибір технологічної бази (Unity, C#), розроблено архітектуру проекту та впроваджено його основні функціональні компоненти.

Сьогодні стратегічні відеоігри мають відданих шанувальників та залишаються динамічним і прибутковим сегментом ігрової індустрії, що приваблює гравців, які люблять тактичні випробування та управління ресурсами.

Практичні результати підтверджують доцільність обраних рішень і дозволяють використовувати розроблену гру як базу для подальшого вдосконалення.

У всіх нас є ігри, які нам подобаються, що дозволяють нам розслабитися та відключитися, але стратегічні ігри вимагають від нас залучення та ввімкнення. Це зовсім інший вид гострих відчуттів, ніж нокаут або постріл у голову. Стратегічно перехитрити суперника — людину чи штучний інтелект — за маневром та мисленням протягом тривалого періоду часу може змусити вас відчути себе реінкарнацією Александра Македонського [28].

Отриманий досвід і результати можуть стати підґрунтям для розробки більш складних ігрових проектів або дослідження інноваційних механік у жанрі стратегій.

На завершення ще одна думка Ясуюкі Міура. У своїй книзі "Go: An Asian Paradigm for Business Strategy" він пише, що концепція Ґо може стати новою парадигмою для вільної, справедливої та осмисленої конкуренції. Більше того, Міура вважає, що слід застосувати концепції Ґо у світовій дипломатії та підтримці миру на планеті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «AlphaGo»: перегляд документального фільму. В рамках Фестивалю з гри Го “Одеса Золота”. URL: https://surl.li/rbqcjj (Last accessed: 11.05.2025).

2. Akshat Sultania. Categorize your game among these 5 mechanics URL: https://surl.gd/yofswl (Last accessed: 11.05.2025).

3. David Mullich, The Mechanics of Game Genres. URL: https://surl.gd/fqbbxg (Last accessed: 11.05.2025).

4. Edge Staff. 50 GREATEST GAME DESIGN INNOVATIONS. November 1, 2007 URL: https://surl.lu/orftbz (Last accessed: 11.05.2025).

5. Felix Rose-Collins. Топ-5 ігрових рушіїв, які ідеально підійдуть для розробки мобільних ігрових додатків у 2024 році. Jul 22, 2024. URL: https://surli.cc/qfufir (Last accessed: 11.05.2025).

6. Jessica Clement. Biggest all-time video game industry acquisitions 2023. Feb 14, 2025. URL: https://surl.li/hqaugh (Last accessed: 11.05.2025).

7. Kevurugames. Unity — what makes it the best game engine? 18 jan 2023. URL: https://kevurugames.com/blog/unity-what-makes-it-the-best-game-engine/ (Last accessed: 11.05.2025).

8. Luke Wakeham Everything you need to know about the history of strategy games 05/29/2024. URL: https://www.redbull.com/us-en/history-of-strategy-games

9. Mark H. Walker. Strategy Gaming: Part III — Strategy Gaming. URL: https://surl.lu/mkjrmv (Last accessed: 11.05.2025).

10. Mobile Game Development Company in USA. URL: https://trangotech.com/mobile/game-development/ (Last accessed: 11.05.2025).

11. Prince Sinha. Functional vs. Procedural vs. Object-Oriented Programming. March 4, 2021. URL: https://www.scoutapm.com/blog/functional-vs-procedural-vs-oop (Last accessed: 11.05.2025).

12. RegisFrey - Own work, Public Domain. URL: https://commons.wikimedia.org/w/index.php?curid=10298177 (Last accessed: 11.05.2025).

13. Revutska Nataliya об’єктно-орієнтоване програмування (ООП). Пояснюємо чотири основні принципи. Квітень 29, 2023. URL: https://career.softserveinc.com/uk-ua/stories/what-is-object-oriented-programming-oop-explaining-four-major-principles (Last accessed: 11.05.2025).

14. Riordan Zentler. For The Spokesman-Review Game On: What happened to real-time strategy games? Thu., Oct. 12, 2023. URL: https://surl.li/qodgdu (Last accessed: 11.05.2025).

15. Sheposh Richard. Strategy video game. 2022. URL: https://surl.lu/onleff (Last accessed: 11.05.2025).

16. StrategyPlanet | Статті | Point — CounterPoint: покрокова стратегія проти стратегії в реальному часі від Dragon | 27 червня 2001 р. URL: https://surl.li/mpehem (Last accessed: 11.05.2025).

17. System requirements for Unity 2022.3. URL: https://surl.li/xegthk (Last accessed: 11.05.2025).

18. The Mechanics of Game Genres. URL: https://surli.cc/cmvliw (Last accessed: 11.05.2025).

19. Tuur Ghys. Technology Trees: Freedom and Determinism in Historical Strategy Games. URL: https://gamestudies.org/1201/articles/tuur_ghys (Last accessed: 11.05.2025).

20. Ulab.Blog. 10 найкращих ігрових рушіїв-22. URL: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv (Last accessed: 11.05.2025).

21. Why Choose .NET? Dotnet.Microsoft. URL: https://dotnet.microsoft.com/en-us/platform/why-choose-dotnet (Last accessed: 11.05.2025).

22. Wikipedia. Game classification. URL: https://surl.li/ezttcl (Last accessed: 11.05.2025).

23. Wikipedia. Model–view–controller // Wikipedia. – 2023. URL: https://en.wikipedia.org/wiki/Model–view–controller (Last accessed: 11.05.2025).

24. Wikipedia. Strategy video game. URL: https://surli.cc/hokyaa (Last accessed: 11.05.2025).

25. Wikipedia.Video game genre. URL: https://surl.li/zfvcku (Last accessed: 11.05.2025).

26. Авраамов Павло. Стратегія Ґо: Давня гра й сучасний бізнес, або Як перемогти в конкурентній боротьбі / пер. з рос. О. Кожушко. — Харків : Вид-во «Ранок» : Фабула, 2019. — 420 с. URL: https://skadi.linkbeat.net/img/315037.pdf (Last accessed: 11.05.2025).

27. Алессандро Габбіадіні, Тобіас Грайтемаєр. Університет Інсбрука, Австрія. Виявлення зв'язку між стратегічними відеоіграми та саморегуляцією: кореляційне дослідження. URL: https://surli.cc/vgxjds (Last accessed: 11.05.2025).

28. Андрій Мазур. Гра Го — як стародавня мудрість допомагає вести бізнес. URL: https://apix-drive.com/ua/blog/useful/hra-go-yak-davnia-mudrist-dopomahaie-vesty-biznes (Last accessed: 11.05.2025).

29. Архітектура твого першого проекту в Unity. URL: https://surl.li/wvuqod (Last accessed: 11.05.2025).

30. Вікіпедія. Unity (ігровий рушій). URL: https://surl.li/lmeupm (Last accessed: 11.05.2025).

31. Вікіпедія. Ґо (гра). URL: https://surl.li/bbklxe (Last accessed: 11.05.2025).

32. Вікіпедія. Жанри відеоігор. URL: https://surli.cc/mujqog (Last accessed: 11.05.2025).

33. Вікіпедія. Стратегічна відеогра. URL: https://surl.li/bdbbyb (Last accessed: 11.05.2025).

34. Вокер, Марк Х. (лютий 2002). «Стратегічні ігри: Частина III -- Стратегічні ігри» . GameSpy . Архів оригіналу від 25.05.2011 . Отримано 28.06.2009. URL: https://surl.li/ismway (Last accessed: 11.05.2025).

35. Го-клуб <Косуми> 2005-2021. URL: https://kosumi.i-go.in.ua/go.php?lng=1 (Last accessed: 11.05.2025).

36. Джозеф Хокінг. Електронна книга «Unity у дії. Мультиплатформна розробка на C#». Фрагмент для ознайомлення. URL: https://surl.lu/fbjnia (Last accessed: 11.05.2025).

37. Михайло Войтович. Грай, щоб вчитися, ч.8: Тестування нової гри 07.08.2023. URL: https://gameit.tech/play-to-learn-08/ (Last accessed: 11.05.2025).

38. Основні правила гри в Го (Українською). URL: https://www.youtube.com/watch?v=GEImaFudcS0 (Last accessed: 11.05.2025).

39. Стадник Юліанна Андріївна, Кравець Андрій Васильович Аналіз доходів від вітчизняної та світової індустрії відеоігор. URL: https://surl.li/xzexhk

40. Сергій Немчинський. Про середовище програмування C#. 01.11.2023. URL: https://surl.li/bgnras (Last accessed: 11.05.2025).

ДОДАТКИ

Додаток А

Правила гри Ґо

Ґо — це гра у побудову царства. Її учасники будують фортеці, проводять кордони та захоплюють стільки «землі», скільки можуть утримати. Якщо шахова партія закінчується знищенням армії противника, то партія в Ґо — перемир'ям та обміном полоненими [26, 38].

Шахи виховують бійця, полководця, а Ґо — правителя, організатора.

Ґо — гра на захоплення територій.

Інвентар — гра ведеться на розкресленій дошці, що називається Гобан. На дошку нанесена сітка із перпендикулярних ліній. Точки їхнього перетину називаються ігровими точками.

Класична сітка має розмір 19х19 ліній та 361 ігрову точку. Гра ведеться спеціальними предметами, які називаються каменями. Вони, як і фігури у шахах — чорного та білого кольорів. Також можно обрати інші розмери дошок. Є варіант 9х9 та 13х13. Чим менша дошка — тим легше грати.

Мета — обгородити каменями свого кольору на ігровій дошці площу більшу, ніж суперник та захватити якомога більше полоненіх каменів суперника.

Ходи — ходом називається натискання миші на перетені ліній дошки або на перетені ліній с краєм дошки на одну з ігрових точок дошки. Поставлений камінь не зсувається до кінця гри. Гру завжди розпочинають чорні.

Окремий камінь або кілька каменів вважаються такими, що потрапили в полон, якщо камені противника стоять на всіх сусідніх ігрових точках по вертикалі та горизонталі. Полонений камінь знімається (зникає) з дошки. Збоку на напису виводиться повідомлення о кількості полоненіх каменів.

Продовження додатка А

Закінчення гри — партія вважається завершеною, якщо неможливо зробити хід, який може або дати бали хóдячому, або відібрати бали у противника. Тоді оба гравця по черзі натискають кнопку пропустити хід. В цей час ігра закінчується виводом на єкран результату гри та об’явою переможця.

Умови гри — гра може відбуватися одним гравцем сам на сам або двома гравцями — хід виконується по черзі.

5 основних правил гри:

1. Гра починається з порожньої дошки (рис. А.1). Двоє гравців по черзі ставлять на неї камені свого кольору. В одного вони білі, в іншого — чорні.
2. Перемагає той, хто оточить більше території.
3. Камінь або група каменів противника, які ви оточили своїм камінням, знімається з дошки (це полонені).
4. Під час гри на дошці з'являються пункти, куди не можна ходити: заборонені зони.
5. Забороняється повторювати позицію: правило «Ко».

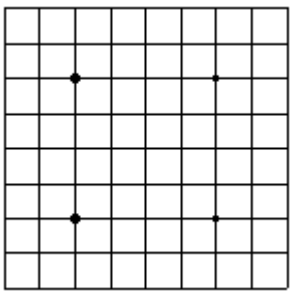


Рис. А.1 Дошка 9х9. Має дев'ять ліній по вертикалі та 9 по горизонталі

Захоплення та порятунок каменів

Для того, щоб забрати цей камінь (рис. А.2), білі повинні перекрити йому шлях за чотирма напрямками — закрити пункти, позначені хрестиками.

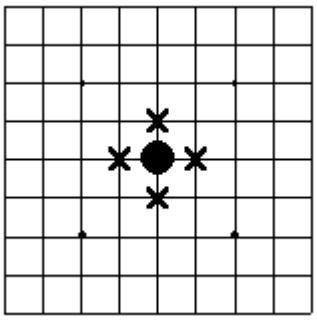


Рис. А.2 Захоплення каменю

У цього білого каменю вже закрито три пункти. Така ситуація називається "Атарі" (рис. А.3). Якщо чорні зроблять хід у пункт, позначений хрестиком, білий камінь виявиться захопленим.

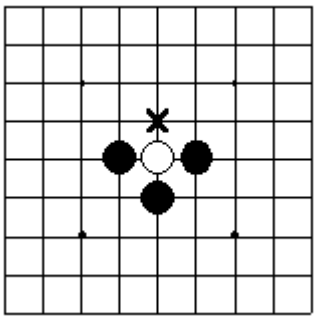


Рис. А.3 Захоплення каменю — Атарі

Чорні зробили хід 1. Білий камінь стає бранцем і знімається (зникає) з дошки (рис. А.4). Таким чином, у чорних буде на очко більше.

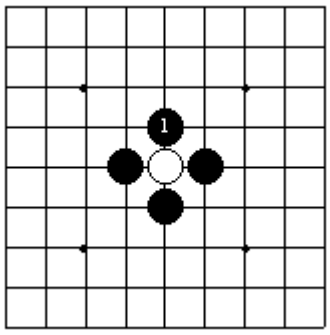


Рис. А.4 Захоплення каменю

Білі тікають від Атарі, яке вийшло на Рис. А.3, роблячи хід 1. Тепер у них вже відкрито шлях для подальшої втечі у трьох напрямах. Вони знову відзначені хрестиками (рис. А.5).

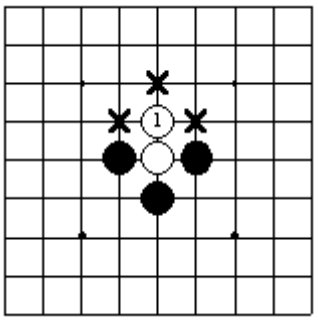


Рис. А.5 Порятунок каменю від Атарі

Захоплення каменів (рис. А.6):

a). Білий камінь у кутку можна захопити лише двома ходами.

b). Якщо він стоятиме на боці, то знадобиться вже три ходи чорних.

c). Два камені білих у центрі заникнуть, якщо чорні закриють усі сусідні пункти.

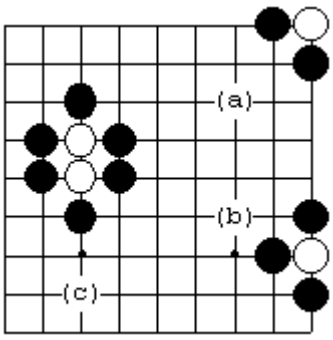


Рис. А.6 Захоплення каменів

Подивіться на позицію, яка виходить після зняття захопленого білого каміння з дошки (рис. А.7).

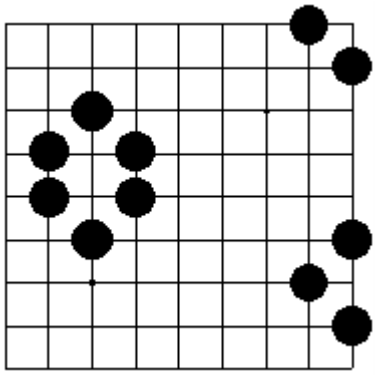


Рис. А.7 Захоплення каменів (зникнення)

Ви можете захопити і три камені білих і чотири, взагалі скільки завгодно. Головне — повністю замкнути їх, і тоді вони додадуться вам до очок, як полонені (рис. А.8).

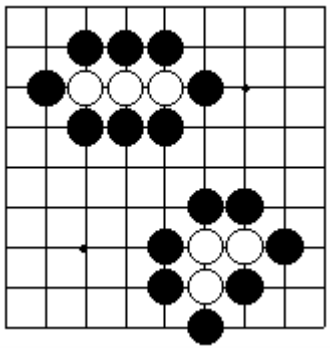


Рис. А.8 Захоплення множи́н каменів

Перед вами позиція, що виходить після зникнення каміння (рис. А.9).

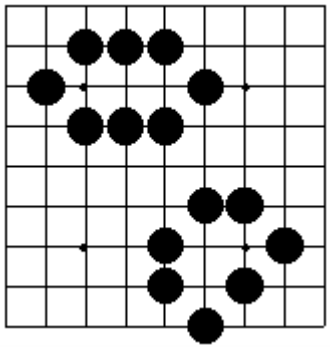


Рис. А.9 Зникнення множи́н каменів

Заборонені ходи

Ця форма вийшла після того, як чорні взяли один білий камінь. Білі не можуть піти до пункту, позначеного хрестиком. Це самогубний хід, який робити не можна (рис. А.10).

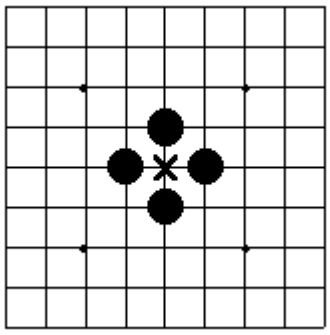


Рис. А.10 Самогубний хід (заборонен)

На цьому малюнку зазначені пункти, до яких білі не можуть ходити. Це також заборонені ходи (рис. А.11).

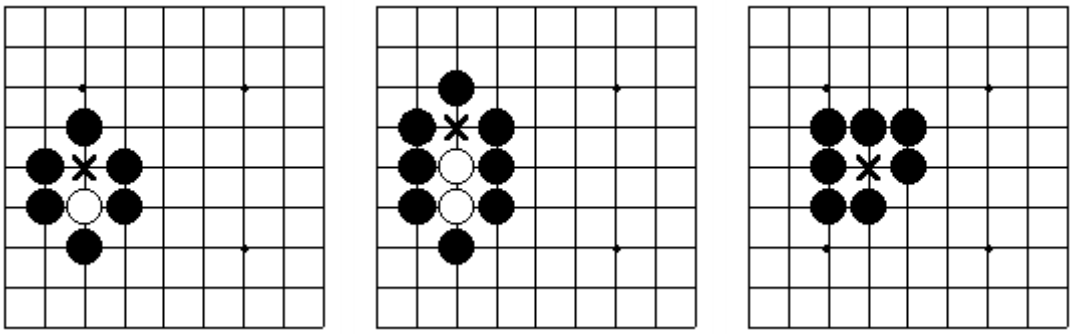
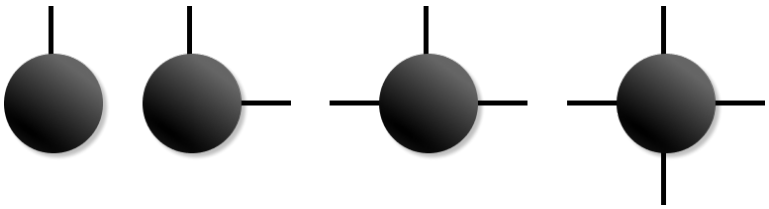


Рис. А.11 Різновиди самогубних ходів (заборонені)

Однак, якщо чорна група буде зовні повністю оточена, то білі можуть піти в пункт, позначений хрестиком, і захопити все чорне каміння. Зняти каміння до ходу білих у місто відмічене хрестиком неможна (рис. А.13). В цьому випадку говорять, що каміння ще мая дихання або життя (рис. А.12).



1 дихання 2 дихання 3 дихання 4 дихання

Рис. А.12 Дихання каміння

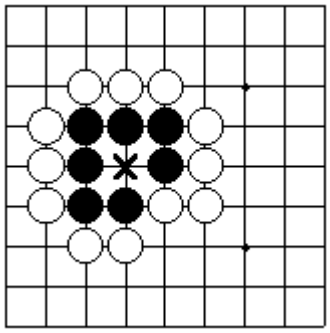


Рис. А.13 Повний захват чорного каміння

Тобто білі роблять хід 1 і знімають із дошки всю чорну групу (рис. А.14).

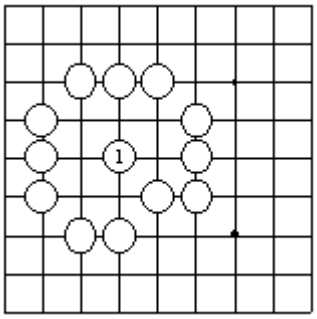


Рис. А.14 Зняття великої групи чорних каменів

Ось що вийшло після того, як білі зняли з дошки все чорне каміння. Це була велика група із 7 каменів (рис. А.15). Це каміння після закінчення гри буде додани до балів білих.

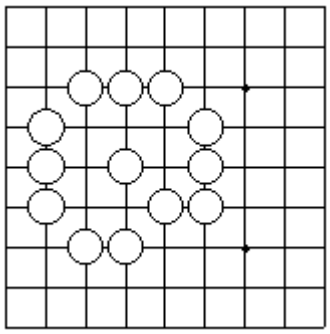


Рис. А.15 Захват території білими (чорні зникають)

На цьому малюнку (рис. А.16) ви бачите два пункти, помічені хрестиком. Білим туди ходити не можна. Навіть якщо білі щільно оточать групу чорних, як показано тут, у них все одно не буде двох ходів разом, щоб завершити оточення. Тому цю чорну групу не можна захопити. Вона називається "Фортеця". У неї два "Ока".

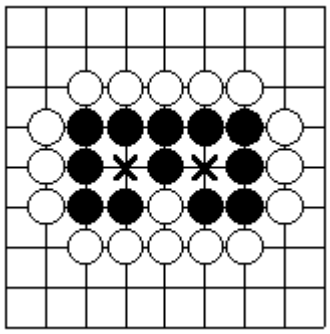


Рис. А.16 Першій варіант Фортеці

А ось ще одна фортеця з двома очима (рис. А.17).

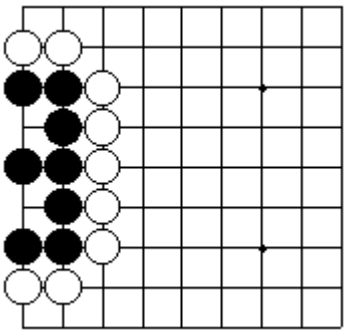


Рис. А.17 Другий варіант Фортеці

Продовження додатка А

Що таке "Ко"

Чорні можуть взяти зазначений трикутником білий камінь, якщо зроблять хід у пункт "a" (рис. А.18).

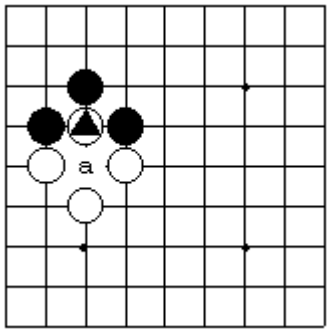


Рис. А.18 Ситуація Ко_1

Якщо тепер білі підуть у пункт "a" і заберуть зазначений камінь чорних, то виникне повторення позиції, яке може тривати до безкінечності. Ця ситуація називається "Ко". У Ґо повторення позиції заборонено. Тож білі не можуть зараз узяти камінь чорних. Ход у цей пункт зараз заборонено. Але вже через хід білі можуть ходити туди!

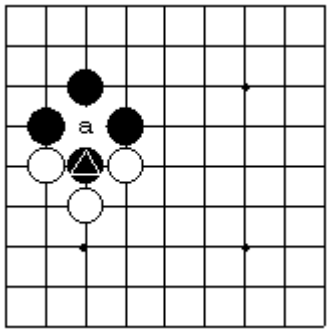


Рис. А.19 Ситуація Ко_2

Результат партії та підрахунок балів за територіями.

Існує кілька різних систем підрахунку, залежно від правил, які встановлені на турнірі, або за якими погодилися грати гравці. В моїй грі я обрав підрахунок за японськими правилами: гравець отримує по одному балу за кожен пункт вільної території і за кожен полонений камінь супротивника.

Той, хто грає чорними, ходить першим — на Сході чорний колір символізує агресію. Для компенсації переваги чорних білі одержують 6,5 балів. Через дробовість фори ("Комі") в Ґо не буває нічиїх [31].

Приклади:

Праворуч вгорі частина дошки оповита чорним камінням. Ліворуч унизу білими. Противники одержують по стільки балів, скільки вільних пунктів дошки вони обгородили. Тут у чорних 8 очок, а у білих — 9 (рис. А.20).

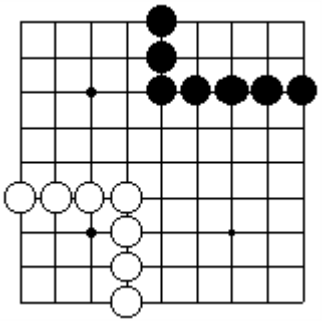


Рис. А.20 Схема підрахунку балів_1

Партнери розділили дошку. Чорні забрали праву частину, а білі — ліву. У чорних виявилося 30 балів, а у білих — 31. Білі виграли у цій партії 1 бал (рис. А.21).

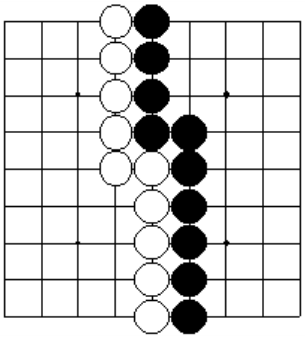


Рис. А.21 Схема підрахунку балів_2

Тут позиція складніша. Якщо ви гратимете самі, то можете отримати в кінці партії і більш хитромудру позицію. Буває, що вам вдається захопити в якомусь місці лише 1 або 2 бали. Тут біля чорних територія у двох місцях — справа вгорі та зліва внизу (рис. А.22).

Продовження додатка А

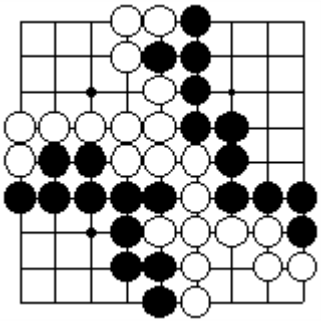


Рис. А.22 Схема підрахунку балів_3

Територія білих зверху зліва і знизу праворуч. Підрахуємо, чим закінчилася ця партія. У чорних справа вгорі 13 балів, а зліва внизу 10, отже, всього 23 бала. Білі вгорі ліворуч 10 балів, а внизу праворуч — 4 бала. Разом — 14 балів. У чорних на 9 балів більше. Чорні виграли 9 балів (рис. А.22).

Способи з'єднання каменів

Для того, щоб захопити територію, треба побудувати міцну огорожу. Таку огорожу захопити можна дуже рідко і насилу [26, 38]. Це каміння ніби приросло одне до одного. Вони утворюють єдине ціле — групу (рис. А.23).

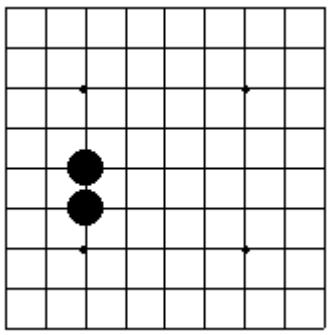


Рис. А.23 Міцне з'єднання

Ці два камені теж з'єднані досить міцно. Справа в тому, що якщо білі підуть у пункт "a", чорні побудують міцне з'єднання ходом "b" і навпаки, якщо білі ходять в "a", чорні міцно з'єднуються в "b" (рис. А.24).

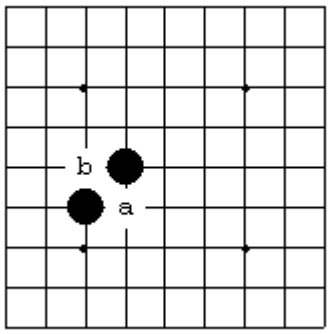


Рис. А.24 Діагональне з'єднання

Чорні ніби перестрибнули через один пункт. Це теж з'єднання, але менш міцне: чорне каміння можна розділити (рис. А.25).

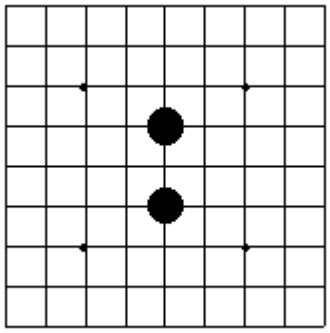


Рис. А.25 Стрибок через один пункт

Якщо білі вклиняться між камінням чорних ходом 1, а чорні підуть 2 і нападуть на білий камінь, білі зможуть вибігти ходом 3. Вийшло два пункти "a" і "b", в один з яких наступним ходом можуть піти білі та розділити чорне каміння на дві групи (рис. А.26).

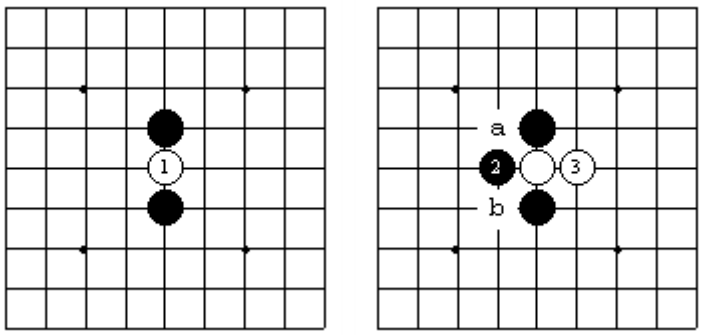


Рис. А.26 Дві групи

Додаток Б

Програмний код моєї програми

StoneManager

```
using UnityEngine;
using System.Collections.Generic;
using UnityEngine.EventSystems;
using System;

public class StoneManager : MonoBehaviour
{
   public static event Action<Vector2Int> OnStonePlaced;

   // Префаб чорного каменя
   public GameObject blackStonePrefab;

   // Префаб білого каменя
   public GameObject whiteStonePrefab;

   // Батьківський об’єкт для усіх каменів
   public Transform boardParent;

   [Header("Stone Settings")]
   // Масштаб для каменів
   public Vector3 stoneScale = Vector3.one;

   [Header("Activation Check")]
   // Об'єкт, активність якого перевіряється перед встановленням каменя
   public GameObject requiredActiveObject;

   // Зберігає посилання на інстанси каменів, розміщених на дошці
   private Dictionary<Vector2Int, GameObject> stoneObjects = new();

   // Розміщує камінь заданого кольору на вказаній клітинці
   public void PlaceStone(int x, int y, DeskManager.Stone color, Transform
boardOrigin, float cellSize)
   {
      // Якщо контрольований об'єкт не активний — не виконуємо дію
      if (requiredActiveObject != null && !requiredActiveObject.activeInHierarchy)
         return;
```

```
        // Обчислюємо світову позицію для розміщення каменя
        Vector3 spawnPos = boardOrigin.position + new Vector3(x * cellSize, y * cellSize, 0);

        // Вибираємо потрібний префаб в залежності від кольору
        GameObject prefab = (color == DeskManager.Stone.Black) ? blackStonePrefab : whiteStonePrefab;

        // Викликаємо подію та створюємо об’єкт
        OnStonePlaced?.Invoke(new Vector2Int(x, y));
        GameObject stone = Instantiate(prefab, spawnPos, Quaternion.identity, boardParent);

        // Застосовуємо масштаб

        stone.transform.localScale = stoneScale;

        // Зберігаємо посилання на об’єкт за його координатами
        stoneObjects[new Vector2Int(x, y)] = stone;
    }

    // Видаляє камінь з дошки (та зі сцени) за координатами
    public void RemoveStone(Vector2Int pos)
    {
        if (stoneObjects.ContainsKey(pos))
        {
            Destroy(stoneObjects[pos]);
            stoneObjects.Remove(pos);
        }
    }
}
```

DeskManager

```
using UnityEditor;
using UnityEngine;

public class DeskManager : MonoBehaviour
{
    public enum Stone { Empty, Black, White }
```

Продовження додатка Б

```
    [Header("Налаштування дошки")]
    public int boardSize = 9;           // Розмір дошки NxN
    public float cellSize = 1.0f;       // Розмір однієї клітинки

    [Header("Префаби та трансформації")]
    public Transform boardOrigin;       // Початкова позиція дошки у світі
    public Transform boardParent;       // Батьківський об'єкт для каменів на сцені

    public Stone[,] board;              // Поточний стан дошки
    public Stone[,] previousBoardState; // Стан дошки до останнього ходу (для перевірки Ко)

    void Start()
    {
        board = new Stone[boardSize, boardSize];

        for (int x = 0; x < boardSize; x++)
        {
            for (int y = 0; y < boardSize; y++)
            {
                board[x, y] = Stone.Empty;
            }
        }
    }

    public Stone GetStone(int x, int y)
    {
        if (x >= 0 && y >= 0 && x < boardSize && y < boardSize)
            return board[x, y];

        return Stone.Empty;
    }

    public void SetStone(int x, int y, Stone color)
    {
        if (x >= 0 && y >= 0 && x < boardSize && y < boardSize)
            board[x, y] = color;
    }

    public Stone[,] CopyBoard(Stone[,] src)
    {
        Stone[,] copy = new Stone[boardSize, boardSize];
```

```
        for (int x = 0; x < boardSize; x++)
        {
            for (int y = 0; y < boardSize; y++)
            {
                copy[x, y] = src[x, y];
            }
        }

        return copy;
    }
}
```

GroupFinder

```
using System.Collections.Generic;
using static DeskManager;
using UnityEngine;

public class GroupFinder
{
    // Пошук групи каменів одного кольору, зв'язаних між собою
    public static List<Vector2Int> FindGroup(Stone[,] board, int x, int y, DeskManager.Stone color)
    {
        var visited = new HashSet<Vector2Int>();   // Вже перевірені позиції
        var stack = new Stack<Vector2Int>();       // Стек для обходу в глибину
        var group = new List<Vector2Int>();        // Результуюча група

        stack.Push(new Vector2Int(x, y));

        while (stack.Count > 0)
        {
            var current = stack.Pop();

            // Якщо камінь ще не відвіданий і має потрібний колір
            if (!visited.Contains(current) && board[current.x, current.y] == color)
            {
                visited.Add(current);
                group.Add(current);

                // Додаємо сусідів до стеку для перевірки
```

Продовження додатка Б

```
            foreach (var neighbor in GetNeighbours(current.x, current.y, board.GetLength(0)))
                stack.Push(neighbor);
        }
    }

    return group;
}

// Перевірка, чи група каменів оточена (немає вільних клітин поруч)
public static bool IsGroupSurrounded(Stone[,] board, List<Vector2Int> group)
{
    foreach (var pos in group)
    {
        foreach (var neighbor in GetNeighbours(pos.x, pos.y, board.GetLength(0)))
        {
            // Якщо хоча б одна клітина сусіда пуста — група не оточена
            if (board[neighbor.x, neighbor.y] == DeskManager.Stone.Empty)
                return false;
        }
    }
    return true; // Якщо всі сусіди зайняті — група оточена
}

// Отримання списку сусідніх клітин (ліворуч, праворуч, вгору, вниз)
public static List<Vector2Int> GetNeighbours(int x, int y, int boardSize)
{
    var list = new List<Vector2Int>();

    // Додавання сусідів по горизонталі та вертикалі, якщо вони в межах дошки
    if (x > 0) list.Add(new Vector2Int(x - 1, y));  // Ліворуч
    if (x < boardSize - 1) list.Add(new Vector2Int(x + 1, y));  // Праворуч
    if (y > 0) list.Add(new Vector2Int(x, y - 1));  // Вгору
    if (y < boardSize - 1) list.Add(new Vector2Int(x, y + 1));  // Вниз

    return list;
}
}
```

TerritoryManager

```
using System.Collections.Generic;
```

Продовження додатка Б

```
using UnityEngine;

public class TerritoryManager : MonoBehaviour
{
    // Посилання на менеджер дошки для доступу до стану гри
    public DeskManager deskManager;

    // Підраховує кількість території для заданого кольору
    public int CountTerritory(DeskManager.Stone color)
    {
        int territoryCount = 0;

        // Масив для відслідковування перевірених клітин
        bool[,] visited = new bool[deskManager.boardSize, deskManager.boardSize];

        for (int x = 0; x < deskManager.boardSize; x++)
        {
            for (int y = 0; y < deskManager.boardSize; y++)
            {
                // Перевірка тільки порожніх клітинок, що не були відвідані
                if (deskManager.GetStone(x, y) == DeskManager.Stone.Empty &&
!visited[x, y])
                {
                    // Знаходження групи суміжних порожніх клітинок
                    List<Vector2Int> emptyGroup = FindEmptyGroup(x, y, visited);

                    // Перевірка, чи група оточена лише каменями заданого кольору
                    if (IsGroupSurroundedByColor(emptyGroup, color))
                    {
                        territoryCount += emptyGroup.Count;
                    }
                }
            }
        }

        return territoryCount;
    }

    // Знаходить групу суміжних порожніх клітинок, починаючи з початкової
клітинки
    List<Vector2Int> FindEmptyGroup(int startX, int startY, bool[,] visited)
    {
```

Продовження додатка Б

```
        List<Vector2Int> group = new();
        Stack<Vector2Int> stack = new();
        stack.Push(new Vector2Int(startX, startY));

        while (stack.Count > 0)
        {
            Vector2Int current = stack.Pop();
            int x = current.x;
            int y = current.y;

            // Перевірка меж та стану клітинки
            if (x >= 0 && y >= 0 && x < deskManager.boardSize && y < deskManager.boardSize &&
                !visited[x, y] && deskManager.GetStone(x, y) == DeskManager.Stone.Empty)
            {
                visited[x, y] = true;
                group.Add(current);

                // Додаємо сусідів для перевірки
                foreach (var neighbor in GroupFinder.GetNeighbours(x, y, deskManager.boardSize))
                {
                    stack.Push(neighbor);
                }
            }
        }

        return group;
    }

    // Перевірка, чи група порожніх клітинок оточена тільки каменями заданого кольору
    bool IsGroupSurroundedByColor(List<Vector2Int> group, DeskManager.Stone color)
    {
        foreach (var cell in group)
        {
            List<Vector2Int> neighbours = GroupFinder.GetNeighbours(cell.x, cell.y, deskManager.boardSize);

            foreach (var neighbour in neighbours)
```

Продовження додатка Б

```
            {
                DeskManager.Stone neighbourStone = deskManager.GetStone(neighbour.x,
neighbour.y);

                // Ігноруємо порожні клітинки
                if (neighbourStone == DeskManager.Stone.Empty)
                    continue;

                // Якщо серед сусідів є камінь іншого кольору — група не вважається
оточеною
                if (neighbourStone != color)
                    return false;
            }
        }

        return true;
    }
}
```

TurnManager

```
public class TurnManager
{
    // true — хід чорного, false — хід білого
    private bool isBlackTurn = true;

    // Показує, чий зараз хід
    public bool IsBlackTurn => isBlackTurn;

    // Змінює чергу ходу
    public void SwitchTurn()
    {
        isBlackTurn = !isBlackTurn;
    }
}
```

MoveValidator

```
using static DeskManager;

public class MoveValidator
```

Продовження додатка Б

```
{
    // Перевірка на самогубство: група одразу не має дихань
    public bool IsSelfCapture(DeskManager deskManager, int x, int y, Stone color)
    {
        Stone[,] simulatedBoard = deskManager.CopyBoard(deskManager.board);
        simulatedBoard[x, y] = color;

        var group = GroupFinder.FindGroup(simulatedBoard, x, y, color);
        return GroupFinder.IsGroupSurrounded(simulatedBoard, group);
    }

    // Перевірка на порушення правила Ко: дошка не повинна збігатися з попереднім станом
    public bool IsKoViolation(Stone[,] previousBoard, Stone[,] simulatedBoard)
    {
        if (previousBoard == null)
            return false;

        return IsBoardEqual(previousBoard, simulatedBoard);
    }

    // Перевірка, чи клітинка пуста
    public bool IsValidMove(DeskManager deskManager, int x, int y)
    {
        return deskManager.GetStone(x, y) == Stone.Empty;
    }

    // Порівняння двох дошок
    private bool IsBoardEqual(Stone[,] a, Stone[,] b)
    {
        int sizeX = a.GetLength(0);
        int sizeY = a.GetLength(1);

        for (int x = 0; x < sizeX; x++)
        {
            for (int y = 0; y < sizeY; y++)
            {
                if (a[x, y] != b[x, y])
                    return false;
            }
```

```
        }
        return true;
    }
}
```

ScoreManager

```
using TMPro;
using UnityEngine;

public class ScoreManager : MonoBehaviour
{
    // UI-текст для кількості полонених
    public TextMeshProUGUI blackPrisonersText;
    public TextMeshProUGUI whitePrisonersText;

    // UI-текст для кількості території
    public TextMeshProUGUI blackTeritoryText;
    public TextMeshProUGUI whiteTeritoryText;

    // Поточна кількість полонених
    public int blackPrisoners;
    public int whitePrisoners;

    // Поточна кількість території
    public int blackTerritory;
    public int whiteTerritory;

    // Оновлює значення рахунку та UI
    public void UpdateScores(int blackPrisoners, int whitePrisoners, int blackTerritory,
int whiteTerritory)
    {
        this.blackPrisoners = blackPrisoners;
        this.whitePrisoners = whitePrisoners;
        this.blackTerritory = blackTerritory;
        this.whiteTerritory = whiteTerritory;

        UpdateUI();
    }

    // Оновлює текст UI
```

```
    public void UpdateUI()
    {
        blackPrisonersText.text = $"{blackPrisoners}";
        whitePrisonersText.text = $"{whitePrisoners}";
        blackTeritoryText.text = $"{blackTerritory}";
        whiteTeritoryText.text = $"{whiteTerritory}";
    }
}
```

GameManager

```
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using static DeskManager;
using System.Collections;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public DeskManager deskManager; // Менеджер дошки
    public StoneManager stoneManager; // Менеджер каменів
    public MoveValidator moveValidator; // Перевірка ходів
    public TurnManager turnManager; // Менеджер чергування ходів
    public TerritoryManager territoryManager; // Менеджер територій
    public ScoreManager scoreManager; // Менеджер рахунку
    public ChangeUI changeUI; // UI для змін
    public WinScreen winScreen; // UI для результату

    public AudioClip cancelSound; // Звук скасування ходу
    public AudioClip TakeSound; // Звук взяття каменя
    public AudioSource audioSource; // Джерело звуку

    public int blackHavePrisoners = 0; // Кількість полонених чорних каменів
    public int whiteHavePrisoners = 0; // Кількість полонених білих каменів
    public int blackTerritory = 0; // Кількість захопленої чорними території
    public int whiteTerritory = 0; // Кількість захопленої білими території

    private int consecutivePasses = 0; // Лічильник послідовних пасів

    // Дані останнього поставленого каменя
    private Vector2Int lastMove;
```

Продовження додатка Б

```
    private Stone lastMoveColor;

    IEnumerator Start()
    {
        moveValidator = new MoveValidator(); // Ініціалізація перевірки ходів
        turnManager = new TurnManager(); // Ініціалізація менеджера ходів

        yield return new WaitForEndOfFrame();

        // Перевірка, чи є збережена гра
        if (PlayerPrefs.GetInt("NewGame", 0) == 1)
        {
            LoadGame(); // Завантажуємо гру
            PlayerPrefs.SetInt("NewGame", 0);
            PlayerPrefs.Save();
        }
    }

    void Update()
    {
        // Обробка натискання миші
        if (Input.GetMouseButtonDown(0))
        {
            HandleClick();
        }
    }

    // Пропуск ходу
    public void PassTurn()
    {
        if (turnManager.IsBlackTurn)
        {
            whiteHavePrisoners++; // Якщо чорний пропускає хід, білий отримує полонених
        }
        else
        {
            blackHavePrisoners++; // Якщо білий пропускає хід, чорний отримує полонених
        }

        consecutivePasses++;
```

Продовження додатка Б

```
        // Якщо було два паси поспіль — завершуємо гру
        if (consecutivePasses >= 2)
        {
            EndTheGame();
            return;
        }

        turnManager.SwitchTurn(); // Перемикаємо хід
        UpdateScore(); // Оновлюємо рахунок
    }

    // Обробка натискання на дошку
    private void HandleClick()
    {
        Vector3 mouseWorldPos = Camera.main.ScreenToWorldPoint(Input.mousePosition); // Отримуємо позицію миші у світі
        Vector2 mousePos2D = new Vector2(mouseWorldPos.x, mouseWorldPos.y);
        RaycastHit2D hit = Physics2D.Raycast(mousePos2D, Vector2.zero); // Виконуємо радіальний промінь для перевірки натискання
        if (hit.collider == null) return; // Якщо нічого не натиснуто, виходимо

        // Обчислюємо координати клітинки на дошці
        Vector2 localPoint = hit.point - (Vector2)deskManager.boardOrigin.position;
        int x = Mathf.RoundToInt(localPoint.x / deskManager.cellSize);
        int y = Mathf.RoundToInt(localPoint.y / deskManager.cellSize);

        // Якщо клітинка не пуста, повертаємося
        if (!moveValidator.IsValidMove(deskManager, x, y))
        {
            return;
        }

        ApplyMove(x, y); // Застосовуємо хід
    }

    // Застосування ходу
    private void ApplyMove(int x, int y)
    {
        Stone currentColor = turnManager.IsBlackTurn ? Stone.Black : Stone.White; // Визначаємо колір каменя
```

Продовження додатка Б

```
        Stone opponentColor = turnManager.IsBlackTurn ? Stone.White : Stone.Black; // Визначаємо колір супротивника

        Stone[,] simulatedBoard = deskManager.CopyBoard(deskManager.board); // Копіюємо дошку для симуляції
        simulatedBoard[x, y] = currentColor; // Ставимо камінь на симульовану дошку

        // Перевірка на захоплення каменів
        List<Vector2Int> captured = CaptureStones(simulatedBoard, x, y, currentColor, opponentColor);

        // Якщо камені не захоплені, перевіряємо, чи не самогубний хід
        if (captured.Count == 0)
        {
            var group = GroupFinder.FindGroup(simulatedBoard, x, y, currentColor);
            if (GroupFinder.IsGroupSurrounded(simulatedBoard, group))
            {
                audioSource.PlayOneShot(cancelSound); // Якщо самогубство — відтворюємо звук скасування
                return;
            }
        }

        // Перевірка на порушення правила Ко
        if (moveValidator.IsKoViolation(deskManager.previousBoardState, simulatedBoard))
        {
            audioSource.PlayOneShot(cancelSound); // Якщо порушення — відтворюємо звук скасування
            return;
        }

        deskManager.previousBoardState = deskManager.CopyBoard(deskManager.board); // Оновлюємо попередній стан дошки
        deskManager.SetStone(x, y, currentColor); // Ставимо камінь на дошку

        // Видаляємо захоплені камені
        foreach (var stone in captured)
        {
            deskManager.SetStone(stone.x, stone.y, Stone.Empty);
```

Продовження додатка Б

```
            stoneManager.RemoveStone(stone);
            audioSource.PlayOneShot(TakeSound); // Відтворюємо звук захоплення
        }

        // Розміщуємо камінь на дошці
        stoneManager.PlaceStone(x, y, currentColor, deskManager.boardOrigin,
deskManager.cellSize);

        // Оновлюємо рахунок полонених
        if (opponentColor == Stone.Black)
            whiteHavePrisoners += captured.Count;
        else
            blackHavePrisoners += captured.Count;

        lastMove = new Vector2Int(x, y); // Оновлюємо останній хід
        lastMoveColor = currentColor; // Оновлюємо колір останнього каменя

        consecutivePasses = 0; // Скидаємо лічильник пасів
        turnManager.SwitchTurn(); // Перемикаємо хід
        UpdateScore(); // Оновлюємо рахунок
    }

    // Захоплення каменів супротивника
    private List<Vector2Int> CaptureStones(Stone[,] board, int x, int y, Stone
currentColor, Stone opponentColor)
    {
        List<Vector2Int> captured = new List<Vector2Int>();

        // Перевіряємо сусідів клітинки
        foreach (var neighbor in GroupFinder.GetNeighbours(x, y,
deskManager.boardSize))
        {
            if (board[neighbor.x, neighbor.y] == opponentColor)
            {
                var group = GroupFinder.FindGroup(board, neighbor.x, neighbor.y,
opponentColor);
                if (GroupFinder.IsGroupSurrounded(board, group))
                {
                    foreach (var pos in group)
                        board[pos.x, pos.y] = Stone.Empty; // Видаляємо захоплені камені

                    captured.AddRange(group); // Додаємо захоплені камені до списку
```

```
            }
        }
    }

    return captured;
}

// Оновлення рахунку
private void UpdateScore()
{
    blackTerritory = territoryManager.CountTerritory(Stone.Black); // Обчислюємо територію чорного
    whiteTerritory = territoryManager.CountTerritory(Stone.White); // Обчислюємо територію білого
    scoreManager.UpdateScores(blackHavePrisoners, whiteHavePrisoners, blackTerritory, whiteTerritory); // Оновлюємо UI
}

// Завершення гри
private void EndTheGame()
{
    UpdateScore(); // Оновлюємо рахунок

    float blackTotal = blackHavePrisoners + blackTerritory; // Загальний рахунок чорного
    float whiteTotal = whiteHavePrisoners + whiteTerritory + 6.5f; // Загальний рахунок білого (з бонусом за територію)

    changeUI.OpenMenu(); // Відкриваємо меню
    winScreen.CountWiner(blackTerritory, whiteTerritory, blackHavePrisoners, whiteHavePrisoners); // Обчислюємо переможця
}

// Збереження поточного стану гри
public void SaveGame()
{
    GameState state = new GameState(); // Створюємо об'єкт для збереження стану гри
    state.boardSize = deskManager.boardSize; // Зберігаємо розмір дошки

    // Збереження даних дошки
    for (int x = 0; x < state.boardSize; x++)
```

Продовження додатка Б

```
        {
            for (int y = 0; y < state.boardSize; y++)
            {
                // Додаємо значення кожної клітинки (порожня, чорна чи біла) до списку
                state.boardData.Add((int)deskManager.board[x, y]);
            }
        }

        // Збереження поточного ходу (чи хід чорного)
        state.isBlackTurn = turnManager.IsBlackTurn;
        // Збереження кількості полонених для кожного гравця
        state.blackPrisoners = blackHavePrisoners;
        state.whitePrisoners = whiteHavePrisoners;

        // Збереження координат та кольору останнього ходу
        state.lastMoveX = lastMove.x;
        state.lastMoveY = lastMove.y;
        state.lastMoveColor = (int)lastMoveColor;

        // Перетворюємо об'єкт на JSON та зберігаємо в PlayerPrefs
        string json = JsonUtility.ToJson(state);
        PlayerPrefs.SetString("GameState", json);
        PlayerPrefs.SetInt("HasSavedGame", 1); // Позначаємо, що є збережена гра
        PlayerPrefs.SetInt("BoardSize", state.boardSize); // Зберігаємо розмір дошки
        PlayerPrefs.SetInt("NewGame", 1); // Вказуємо, що гра нова
        PlayerPrefs.Save(); // Зберігаємо налаштування
    }

    // Завантаження стану гри з PlayerPrefs
    public void LoadGame()
    {
        // Якщо збереженої гри немає, повертаємося
        if (PlayerPrefs.GetInt("HasSavedGame", 0) == 0)
            return;

        string json = PlayerPrefs.GetString("GameState", ""); // Читаємо збережений JSON
        if (string.IsNullOrEmpty(json)) // Якщо дані порожні, повертаємося
            return;

        // Перетворюємо JSON в об'єкт стану гри
```

Продовження додатка Б

```
        GameState state = JsonUtility.FromJson<GameState>(json);

        // Відновлюємо розмір дошки та її дані
        deskManager.boardSize = state.boardSize;
        deskManager.board = new DeskManager.Stone[state.boardSize, state.boardSize];

        int index = 0;
        for (int x = 0; x < state.boardSize; x++)
        {
            for (int y = 0; y < state.boardSize; y++)
            {
                // Відновлюємо колір кожної клітинки дошки
                int value = state.boardData[index++];
                DeskManager.Stone color = (DeskManager.Stone)value;
                deskManager.board[x, y] = color;

                // Якщо клітинка не пуста, додаємо камінь
                if (color != DeskManager.Stone.Empty)
                {
                    stoneManager.PlaceStone(x, y, color, deskManager.boardOrigin,
deskManager.cellSize);
                }
            }
        }

        // Оновлюємо попередній стан дошки
        deskManager.previousBoardState =
deskManager.CopyBoard(deskManager.board);
        consecutivePasses = 0; // Скидаємо лічильник пасів

        // Відновлюємо кількість полонених
        blackHavePrisoners = state.blackPrisoners;
        whiteHavePrisoners = state.whitePrisoners;

        // Відновлюємо хід (якщо треба, змінюємо хід)
        if (state.isBlackTurn != turnManager.IsBlackTurn)
            turnManager.SwitchTurn();

        // Відновлюємо останній хід
        Vector2Int pos = new Vector2Int(state.lastMoveX, state.lastMoveY);
        Stone colorLast = (Stone)state.lastMoveColor;
```

Продовження додатка Б

```
        stoneManager.RemoveStone(pos); // Видаляємо попередній камінь
        stoneManager.PlaceStone(pos.x, pos.y, colorLast, deskManager.boardOrigin, deskManager.cellSize); // Ставимо останній камінь

        lastMove = pos; // Оновлюємо останній хід
        lastMoveColor = colorLast; // Оновлюємо колір останнього каменя

        UpdateScore(); // Оновлюємо рахунок
    }

    // Видалення збереженого стану гри
    public void SaveDel()
    {
        PlayerPrefs.SetInt("HasSavedGame", 0); // Вказуємо, що збережена гра відсутня
    }

    // Перезапуск сцени (початок нової гри)
    public void RestartScene()
    {
        Scene currentScene = SceneManager.GetActiveScene(); // Отримуємо поточну сцену
        SceneManager.LoadScene(currentScene.name); // Завантажуємо її знову
    }
}
```