**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

# ГЕЙМІФІКАЦІЯ НАВЧАЛЬНОГО ПРОЦЕСУ ПІДГОТОВКИ ІТ-ФАХІВЦІВ У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

**Здобувач** _____ **Роман КОШОВИЙ**

«__» _____ 20__ р.

**Керівник роботи**

канд. пед. наук,
доцентка _____ **Катерина КІРЕЙ**

«__» _____ 20__ р.

**Миколаїв – 2025**

**Завдання на виконання кваліфікаційної роботи**

Чорноморський національний університет імені Петра Могили

| | |
|---|---|
| Факультет | Комп'ютерних наук |
| Кафедра | Інженерії програмного забезпечення |
| Рівень вищої освіти | Другий (магістерський) |
| Освітній ступінь | Магістр |
| Спеціальність | 121 Інженерія програмного забезпечення |
| Освітня програма | Інженерія програмного забезпечення |

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

**ЗАВДАННЯ**

**на кваліфікаційну магістерську роботу здобувача вищої освіти**

**Кошового Романа**

1. Тема кваліфікаційної роботи «Гейміфікація навчального процесу підготовки ІТ-фахівців у закладах вищої освіти» затверджена наказом ректора ЧНУ ім. Петра Могили № _182_ від «_2_» __липня__ 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Працездатна інформаційна система для гейміфікації навчального процесу підготовки ІТ-фахівців

4. Перелік питань, що підлягають розробці: предметна галузь та аналогічні програмні системи зі схожим функціоналом; алгоритми сортування, пошукові алгоритми, графи, структури даних, хеш-функції, візуалізація алгоритмів, оцінка складності алгоритмів, розробка легковагових ігор, досвід гейміфікації навчального процесу, педагогічна складова гейміфікації навчального процесу

5. Перелік графічних матеріалів: _____презентація_____
6. Консультанти:

| **Консультант** | **Кафедра (організація)** | **Частина роботи** |
|---|---|---|
| | | |
| | | |
| | | |

Дата видачі завдання «____» _____ 20___ р.

**КАЛЕНДАРНИЙ ПЛАН**

**виконання кваліфікаційної роботи**

Тема: Гейміфікація навчального процесу підготовки ІТ-фахівців у закладах вищої освіти

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1. | Визначення тем і керівників кваліфікаційних робіт та доведення їх переліку до здобувачів | 01.04.2025 | 01.05.2025 | Виконано |
| 2. | Подання заяви на затвердження теми та керівників КМР | 02.05.2025 | 05.05.2025 | Виконано |
| 3. | Розгляд поданих заяв | 06.05.2025 | 01.06.2025 | Виконано |
| 4. | Подання тем, керівників та консультантів КМР для затвердження наказом ректора | 02.06.2025 | 15.06.2025 | Виконано |
| 5. | Видача здобувачеві завдання на виконання КМР | 16.07.2025 | 28.07.2025 | Виконано |
| 6. | Видача здобувачеві завдання на передатестаційну практику | 25.08.2025 | 30.08.2025 | Виконано |
| 7. | Проходження передатестаційної практики, збір та аналіз матеріалів до КМР | 01.09.2025 | 05.10.2025 | Виконано |
| 8. | Приймання звітів з передатестаційної практики | 06.10.2025 | 08.10.2025 | Виконано |
| 9. | Складання календарного плану роботи на весь період виконання КМР | 09.10.2025 | 15.10.2025 | Виконано |
| 10. | Виконання КМР | 16.10.2025 | 21.11.2025 | Виконано |
| 11. | Попередній захист КМР на засіданні комісії кафедри | 24.11.2025 | 24.11.2025 | Виконано |
| 12. | Рішення про допуск здобувача до захисту роботи і призначення рецензента | 25.11.2025 | 28.11.2025 | Виконано |
| 13. | Доробка та остаточне оформлення КМР | 24.11.2025 | 12.12.2025 | Виконано |
| 14. | Написання відгуку на КМР | 15.12.2025 | 16.12.2025 | Виконано |
| 15. | Подання КМР рецензенту | 16.12.2025 | 16.12.2025 | Виконано |
| 16. | Рецензування КМР | 17.12.2025 | 19.12.2025 | Виконано |
| 17. | Подання КМР, її електронної копії та інших документів (відгуку, рецензії) до захисту | 19.12.2025 | 19.12.2025 | Виконано |
| 18. | Захист КМР перед ЕК | 22.12.2025 | 23.12.2025 | Виконано |

**Здобувач** _____ **Роман КОШОВИЙ**

«__» _____ 20__ р.

**Керівник роботи**

канд. пед. наук,

доцентка _____ **Катерина КІРЕЙ**

«__» _____ 20__ р.

# АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Гейміфікація навчального процесу підготовки ІТ-фахівців у закладах вищої освіти»

Здобувач 608М гр.: Кошовий Роман

Керівник: канд. пед. наук, доцент Кірей Катерина

Кваліфікаційна магістерська робота присвячена дослідженню актуальності гейміфікації навчального процесу для здобувачів у сфері «Інформаційні технології» на прикладі легкої гейміфікованої навчальної платформи.

**Об'єктом кваліфікаційної роботи** є освітній процес спеціальностей галузі знань «Інформаційні технології».

**Предметом кваліфікаційної роботи** є методи гейміфікації освітнього процесу закладів вищої освіти для спеціальностей галузі знань «Інформаційні технології» з використанням цифрових освітніх ігрових рішень.

**Метою роботи** є підвищення ефективності освітнього процесу в закладах вищої освіти через розробку інформаційної системи з дидактичними іграми, що сприятиме зростанню залученості здобувачів, удосконаленню підходів щодо оцінювання знань та розвитку практичних навичок майбутніх фахівців галузі знань «Інформаційні технології».

**Методами розробки** є теоретичні: аналіз предметної галузі; аналіз можливостей гейміфікації та прикладів її імплементації в навчальний процес, аналіз особливостей фахових знань галузі «Інформаційні технології». Практичні: реалізація та обслуговування інформаційної системи, ігрових застосунків, управління та централізація навчальними платформами, аналіз досвіду користувача, адаптація під потреби навчального плану та педагогічної системи; Емпіричні: тестування та налагодження розробленої інформаційної системи; моніторинг роботи системи в реальних умовах, аналіз зворотного зв'язку та отриманих системних даних.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та додатків. У першому розділі описано предметну галузь, розглянуто наявні аналоги та сформульована постановка завдань. У другому розділі проведено аналіз методів та технологій дослідження, сформовано специфікацію вимог до програмного забезпечення. У третьому розділі розглянуто процес проєктування відповідного рішення та наведено моделі функцій та використання системи. У четвертому розділі описано кодування, тестування розробленої інформаційної системи; зроблено аналіз одержаних результатів тестування та апробації.

Кваліфікаційна магістерська робота викладена на 84 сторінки, вона містить 4 розділи, 27 ілюстрацій, 1 додаток, 31 джерело в переліку посилань.

Ключові слова: *освітня ігрова платформа, гейміфікація освіти, ігрові застосунки, комп'ютерна гра, інтерактивне навчання.*

# ASTRACT

to the qualifying master's thesis

«Gamification of the educational process of training IT-specialists in higher education institutions»

Student of group 608M: Koshovyi Roman

Supervisor: Ph.D. Sc., Associate Professor Kirei Kateryna

The qualification master's thesis is devoted to the study of the relevance of gamification of the educational process for applicants in the field of «Information Technologies» using the example of a light gamified educational platform.

The **object** of the qualification work is the educational process of specialties in the field of knowledge «Information Technologies».

The **subject** of the qualification work is methods of gamification of the educational process of higher education institutions in specialties in the field of knowledge «Information Technologies».

The **purpose** of the work is to increase the efficiency of the educational process in higher education institutions through the development of an information system with didactic games, which will contribute to increasing student engagement, improving approaches to assessing knowledge, and developing practical skills of future specialists in the field of knowledge «Information Technologies».

The **development methods** are theoretical: analysis of the subject area; analysis of gamification opportunities and examples of its implementation in the educational process, analysis of the features of professional knowledge in the field of «Information Technology». Practical: implementation and maintenance of the information systems and game applications development, management and centralization of educational platforms, analysis of user experience, adaptation to the needs of the curriculum and pedagogical system; Empirical: testing and debugging of the developed information system; monitoring the operation of the system in real conditions, analysis of feedback and received system data.

The qualification work consists of an introduction, four sections, conclusions and appendices. The first section describes the subject area, considers existing analogues and formulates the task statement. The second section analyzes the methods and technologies of research, forms a specification of software requirements. The third section considers the process of designing a corresponding solution and provides models of functions and use of the system. The fourth section presents the work done on coding, testing the developed information system, analyzes the results of testing and research usage.

The master's qualification work is set out on 84 pages, it contains 4 sections, 27 illustrations, 1 appendix, 31 sources in the list of references.

Keywords: *educational gaming platform, gamification of education, gaming applications, computer game, videogame, interactive learning.*

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

| | | |
|---|---|---|
| АОК | – | Апаратно-обчислювальний комплекс |
| ПК | – | Персональний комп’ютер |
| MVP | – | Minimum Viable Product (мінімально життєздатний продукт) |
| UI | – | User interface |
| UX | – | User experience |
| DRY | – | Don’t repeat yourself |

# ВСТУП

Розвиток ІТ-індустрії та реалізація її інноваційного потенціалу потребують виховання висококваліфікованих фахівців. Еволюція програмного забезпечення вимагає спеціалістів, здатних до створення нових рішень. Без експертів галузь не зможе задовольнити зростаючі потреби, тому ключовим етапом становлення фахівця є здобуття якісної вищої освіти.

Університети мають постійно еволюціонувати, впроваджуючи інноваційні методи, що підвищують залученість здобувачів і сприяють кращому засвоєнню інформації. Сучасні методики надихають на навчання та ефективно готують майбутніх ІТ-фахівців до професійних викликів. У такому разі важливо враховувати інтереси здобувачів. Майбутні спеціалісти активно взаємодіють із технологіями, зокрема відеоіграми. Окрім розваги, ігри стимулюють пізнавальний інтерес, дозволяючи вивчати світ в інтерактивній формі.

Зважаючи на це, відеоігри доцільно використовувати як інструмент для подання матеріалу та оцінювання знань. Їхня інтеграція у навчальний процес поглиблює розуміння тем і може застосовуватися для практичних цілей, наприклад, захисту курсових проєктів. Однією з можливих реалізацій цього підходу є створення централізованої вебплатформи з колекцією освітніх ігор. До розширення бібліотеки можна залучати старшокурсників, що розвиватиме їхні навички розробки, програмування та креативність. Таке гейміфіковане навчання може покращити засвоєння матеріалу та посилити мотивацію здобувачів.

**Об’єктом кваліфікаційної роботи** є освітній процес спеціальностей галузі знань «Інформаційні технології».

**Предметом кваліфікаційної роботи** є методи гейміфікації освітнього процесу закладів вищої освіти для спеціальностей галузі знань «Інформаційні технології» з використанням цифрових освітніх ігрових рішень.

**Метою роботи** є підвищення ефективності освітнього процесу в закладах вищої освіти через розробку інформаційної системи з дидактичними іграми, що сприятиме зростанню залученості здобувачів, удосконаленню підходів щодо

оцінювання знань та розвитку практичних навичок майбутніх фахівців галузі знань «Інформаційні технології».

**Завдання**, які потрібно виконати для досягнення мети:

– аналіз теоретичних основ та концепцій;

– дослідження наявних рішень у межах зазначеної сфери;

– оцінка найкращих практик використання ігрового програмного забезпечення;

– проєктування та розробка геймифікованої навчальної платформи;

– аналіз практичного застосування програмного рішення;

– тестування та валідація розробленого програмного рішення.

**Апробація результатів КРБ.** Роботу над гейміфікацією у навчальному процесі ЗВО для спеціальностей галузі знань 12 «Інформаційні технології» представлено на Міжнародному конкурсі студентських наукових робіт «Black Sea Science 2025» (січень-березень 2025 р).

**Публікації.** Роботу опубліковано в рамках «Ольвійського форуму 2025»: Кошовий Р. В., Кірей К. О. Гейміфікація навчального процесу підготовки майбутніх ІТ-фахівців. Ольвійський форум – 2025: стратегії країн Причорноморського регіону в геополітичному просторі : тези доп. XXII Міжнар. наук. конф. / Чорном. нац. ун-т ім. Петра Могили, Миколаїв, 16-22 червня 2025 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2025. С. 63-66. DOI 10.34132/mspc2025.01.14.12.

Результати роботи опубліковано в фаховому виданні «Наука і техніка сьогодні»: Кірей К. О., Кошовий Р. В. Освітня ігрова платформа як складова навчального процесу підготовки ІТ-фахівців у закладах вищої освіти, 2025, 9(50), с. 1237-1251, https://doi.org/10.52058/2786-6025-2025-9(50)-1237-1251.

**Авторське право.** Отримано свідоцтво про реєстрацію авторського права на комп’ютерну програму «Навчальна гра «Optimize the imitation (сортування)» («Optimize the imitation») № 141513 від 15 грудня 2025 р., електронний документ з ідентифікатором CR1987151225 (sis.nipo.gov.ua).

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Для того щоб ІТ-індустрія розвивалася та могла вдало реалізовувати свій потенціал зростаючими темпами, вносячи інновації та доступність, є необхідним навчання висококваліфікованих спеціалістів. Постійна еволюція програмного забезпечення та новітніх пристроїв вимагає фахівців, які зможуть не просто розбиратися в сфері, але і з легкістю адаптуватися до вимог ринку, нових умов та різноманітних викликів. Саме створення інноваційних рішень замість роботи «за підручником» рухає прогрес та вимагає креативності та аналітичного мислення з таким же пріоритетом, як і знань у своїй сфері.

Знання потрібно не просто набути та вміти використовувати у конкретних сценаріях – їх потрібно засвоїти та застосовувати як частину свого мислення. Їх потрібно не просто завчити, а інтегрувати у власну систему цінностей, досвіду та інтуїції, щоб вони стали інструментом для прийняття рішень, вирішення проблем і досягнення цілей. Без таких спеціалістів було б неможливо постійно виправдовувати очікування та потреби ринку.

Надзвичайно важливим кроком на шляху становлення висококваліфікованого ІТ-спеціаліста є набуття вищої освіти у ЗВО. Для ефективного забезпечення здобувачів необхідними знаннями та навичками, університети також мають постійно розвиватися, впроваджуючи інноваційні методи щодо заохочення здобувачів та покращення їхнього досвіду і можливостей «поглинати» та ефективно використовувати отриману інформацію. Саме завдяки опануванню новітніх технік та ресурсів для навчання, навчальні заклади можуть надихати здобувачів та пробуджувати більш глибинний інтерес до навчання, підготовлюючи майбутніх фахівців до індустріальних викликів [1-2].

Дуже важливо звертати увагу на потреби здобувачів та їхні інтереси, які не стоять на місці так само як і сфера загалом. Більшість майбутніх ІТ-спеціалістів

не просто є просунутими користувачами ПК та новітніх технологій, але також активно взаємодіють з ними поза професійною діяльністю, часто споживаючи значні обсяги різноманітного контенту. До такого контенту, окрім іншого, можна віднести відеоігри – досить поширена форма дозвілля у спеціалістів нового покоління. Ігри не просто є розважаючими – вони можуть розпалити зацікавлення та інтерес, надаючи можливості для гравців навчитися чомусь новому або подивитися на речі під іншим кутом. Взаємодія з ігровим світом може бути захоплюючою та інтерактивною проєкцією на реальні або абстрактні принципи та образи [3].

Враховуючи популярність та привабливість відеоігор серед здобувачів, вони можуть слугувати ефективним інструментом для представлення навчального матеріалу або оцінювання знань за певними темами. Інтегруючи ігри у навчальний процес, педагоги можуть зробити викладання більш захопливим та заохочувати здобувачів розвивати більш глибоке розуміння матеріалу. Ігри також можуть бути використані у більш практичних цілях, наприклад, захист практичної роботи або оцінювання теоретичних знань [4].

Отже, вважаємо за доцільне розглянути гейміфікацію, як один зі способів покращення освітнього процесу у ЗВО, зокрема, для здобувачів галузі знань «Інформаційні технології». Для демонстрації практичного застосування у локальних масштабах розглянуто створення централізованого, легкого для розгортання вебзастосунку з колекцією дидактичних ігор за темами дисципліни «Алгоритми та структури даних». Зокрема, фокус на простоті системи обґрунтований тим, що це надаватиме можливість більш досвідченим здобувачам вносити свій вклад у розвиток системи через розширення асортименту ігор у колекції. Вважаємо, що розробка та можливість додавання ігор до готової системи є прекрасним способом для майбутніх фахівців вдосконалювати свої навички програмування та знання самої теми, за якою буде стосуватися конкретна гра.

## 1.2 Розгляд аналогів

Досить важливим у досягненні поставленої мети є аналіз наявних рішень, пов’язаних із гейміфікацією освіти, інтерактивних систем або ж ігор, які розвивають певні, наприклад, алгоритмічні навички у гравців. Ключові аспекти, які необхідно розглянути, включають:

– типи доступних ігор та їхні основні цілі;

– механізми, за допомогою яких користувачам надається контроль та інтерактивність;

– наявність або відсутність елементів змагання в іграх;

– способи, якими ігри відповідають освітнім цілям та покращують навчальний досвід.

Розгляд описаних деталей надасть ключову інформацію про досвід гейміфікації, недоліки у наявних інструментах, особливості дизайну тощо. Рішення, яке проєктується, буде розроблятися із урахуванням цих особливостей. Наявні рішення можна поділити на 3 основні види: казуальні відеоігри, ігри з пріоритетом на написання коду (програмування), та інтерактивні системи.

### 1.2.1 Казуальні ігри

До казуальних ігор можна віднести *Human Resource Machin*e [5] та *7 Billion Humans* [6] – це ігри-головоломки від студії Tomorrow Corporation, які в ігровій формі навчають основам програмування. Обидві гри мають візуальний стиль, схожий на мультфільм, та легкий гумор, з досить різною між собою механікою. Human Resource Machine зосереджена на концепції **асемблера** (мови програмування низького рівня).

– **Геймплей:** гравець керує одним маленьким офісним працівником. За допомогою простих команд (наприклад, «Взяти з вхідної скриньки», «Покласти у вихідну скриньку», «Додати», «Перейти») гравець складаєте програму (або ж алгоритм), щоб виконати завдання, які надає бос.

– **Основна ідея:** гра симулює роботу одного процесора. Гравець вчиться, як дані переміщаються, зберігаються в пам’яті (клітинки на підлозі), виконуються арифметичні операції та як створювати цикли й умовні оператори.

– **Складність:** завдання стають дедалі складнішими, вимагаючи все більш продуманої логіки. Додатковий виклик – оптимізація програми, тобто створення найкоротшого та найшвидшого коду (рис. 1.1).

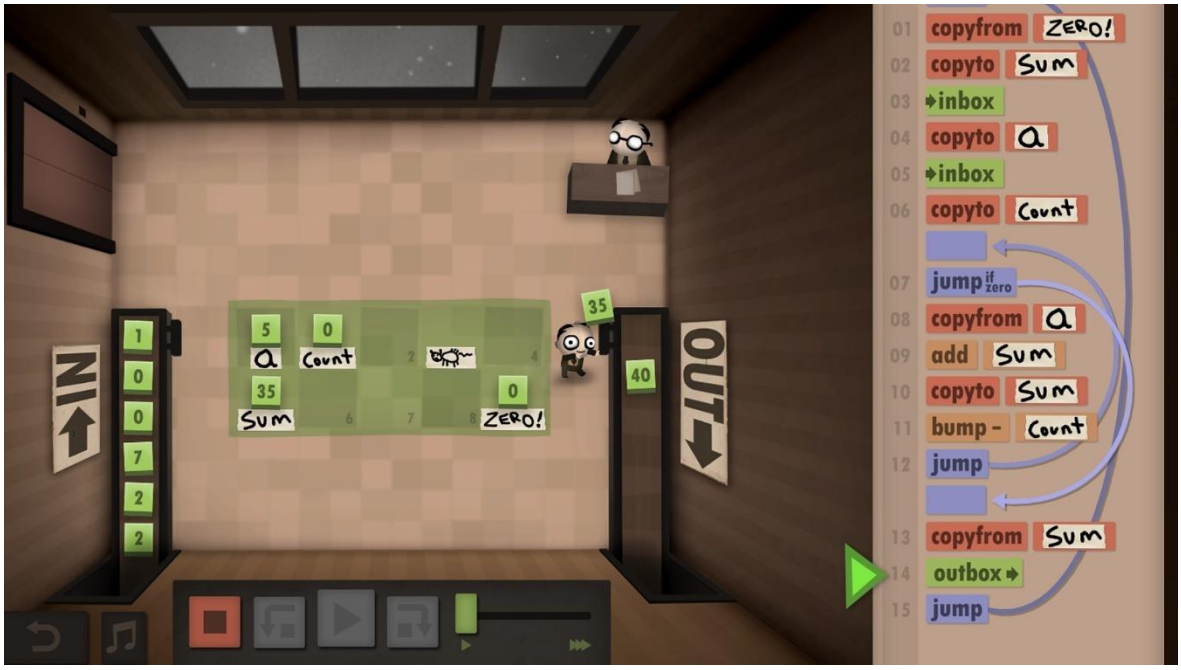



Рисунок 1.1 – Ігровий процес Human Resource Machine

*7 Billion Humans* є сиквелом *Human Resource Machine* і розширює концепцію до **паралельного програмування**.

– **Геймплей:** тепер у гравця не один, а цілий рій працівників. Задачою є написати одну й ту ж програму, яка виконуватиметься одночасно всіма працівниками. Кожен працівник має своє місце на ігровій дошці, і завданням гравця стає синхронізувати їхні дії, щоб вони не заважали один одному.

– **Основна ідея:** гра навчає принципам розподілених систем та конкурентного програмування. Гравцю доведеться враховувати, як різні «робітники» взаємодіють з однаковими даними, як розподілити завдання між ними і як уникнути конфліктів. Це значно складніше та вимагає іншого підходу до розв’язання проблем.

– **Відмінності:** якщо *Human Resource Machine* була про послідовне виконання команд, то *7 Billion Humans* – це про те, як керувати хаосом, коли багато агентів роблять різні речі, керуючись одним набором правил (рис. 1.2).


Рисунок 1.2 – Ігровий процес 7 Billion Humans

Ігри, схожі на Human Resource Machine і 7 Billion Humans – це чудовий спосіб вивчити основи програмування, алгоритмізації та керування даними на низькому рівні. Вони допомагають зрозуміти, як працює процесор, завдяки інтерактивному та доступному інтерфейсу.

Завдання в таких іграх подаються в легкій, невимушеній формі, а для їх вирішення достатньо використовувати лише мишку. Попри таку простоту, складність постійно зростає, що не тільки допомагає засвоїти нові концепції, а й розвиває навички вирішення проблем та вчить передбачати неочікувані ситуації.

Однак, такі ігри часто мають свої обмеження. Вони зазвичай орієнтовані на одиночне проходження, а їхній сюжет – лінійний. Завершення гри стає кінцевим результатом і показником того, що гравець засвоїв матеріал. Першочергово (і тільки) ігри такого виду призначені для самої гри, а не її інтеграції, але сам ігровий процес та досвід заохочення гравця у навчання комплексним речам є досить важливим та цікавим для аналізу.

### 1.2.2 Ігри з програмуванням

Дидактичні ігри з програмування, можуть мати суттєві відмінності у підході до інтеракції користувача з грою. На противагу *Human Resource Machine*, яка використовує метафоричну мову низького рівня, існують платформи, що інтегрують ігровий процес безпосередньо з реальними мовами програмування. До таких належать *CodinGame* та *CodeCombat*, кожна з яких пропонує унікальний досвід.

***CodinGame*** [7] – це не стільки гра в традиційному сенсі, скільки змагальна платформа, що геймифікує процес вирішення алгоритмічних задач. Тут гравець має можливість писати реальний код на одній із багатьох доступних мов програмування, таких як Python, Java, C++ чи JavaScript. Завдання представлені як візуальні головоломки, де гравець має запрограмувати поведінку персонажа або штучного інтелекту, щоб він успішно виконав завдання. Платформа пропонує різні режими: від однокористувацьких квестів до повноцінних багатокористувацьких боїв, де гравці змагаються, програмуючи своїх «ботів» для боротьби один з одним на спільному полі. Успіх залежить не лише від коректності коду, а й від його ефективності, швидкості виконання та алгоритмічної складності. Таким чином, *CodinGame* орієнтована на розробників, які прагнуть відточити свої навички, підготуватися до технічних співбесід або просто позмагатися у віртуальному просторі (рис. 1.3).

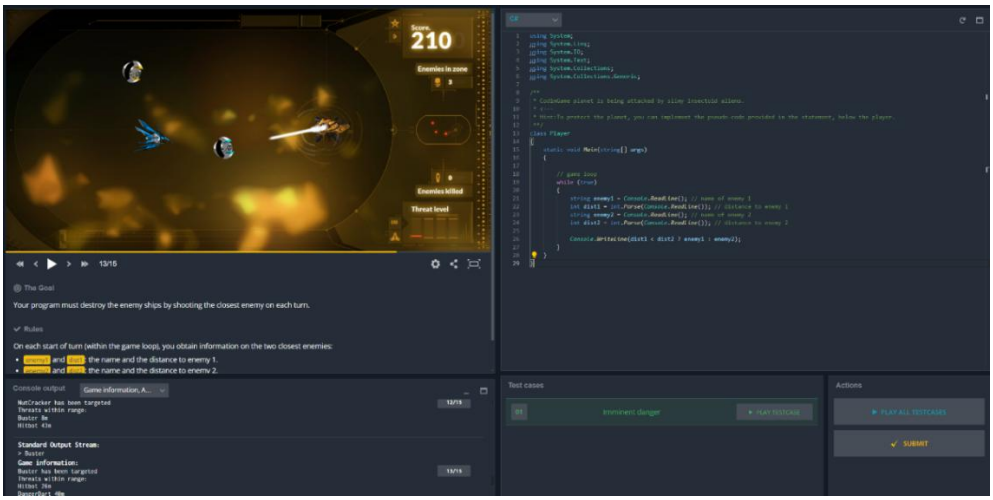



Рисунок 1.3 – Ігровий процес CodinGame

Натомість, ***CodeCombat*** [8] позиціонує себе як повноцінна рольова гра, що слугує навчальною платформою для початківців. Вона занурює гравця у вигаданий світ, де головний герой, схожий на мага чи лицаря, подорожує підземеллями, збирає скарби та б'ється з ворогами. Увесь ігровий процес керується не кліками мишки, а написанням коду у вбудованому текстовому редакторі. Наприклад, щоб герой зробив крок, гравець пише команду hero.moveRight(), а для атаки – hero.attack(enemy). Кожен рівень поступово вводить нові концепції: від простих змінних і циклів до функцій та умовних операторів. Візуальний зворотний зв'язок є миттєвим: як тільки код написаний, персонаж виконує запрограмовану дію, що робить навчання інтуїтивно зрозумілим. *CodeCombat* ідеально підходить для учнів та людей, які ніколи не програмували, оскільки вона перетворює абстрактні концепції на візуально зрозумілий та захопливий досвід (рис. 1.4).


Рисунок 1.4 – Ігровий процес CodeCombat

Об'єднуючи ці та подібні ігри, можна сказати, що вони є потужними інструментами гейміфікації навчання. Вони перетворюють складний і часто монотонний процес вивчення програмування на динамічну взаємодію. Замість сухої теорії, гравець одразу бачить практичну імплементацію своїх знань і отримує задоволення від досягнутого результату. Це не просто розвага, а інтерактивний тренажер, який вибудовує мислення розробника, розвиває логіку,

вчить структуровано підходити до розв’язання задач і допомагає уникнути типових помилок. Завдяки візуальному представленню, ці ігри роблять абстрактні поняття програмування більш доступними та зрозумілими, і можуть навіть виступати непоганими аналогами звичних практичних робіт з програмування.

### 1.2.3 Навчальні платформи

Існують також навчальні платформи, які надають гейміфіковані або інтерактивні засоби для викладання матеріалу. **Educaplay** [9] – це міжнародна платформа, створена для того, щоб перетворити будь-яку навчальну тему на інтерактивну гру. Її основна мета – надати викладачам та учням інструменти для створення та використання освітніх ресурсів у ігровій формі, без необхідності знати програмування. Замість того щоб навчати коду, ця платформа використовує гейміфікацію як засіб для засвоєння будь-якого предмету, від мов і математики до фізики та інформатики. На платформі можна знайти та створити кросворди, вікторини, ігри на відповідність, головоломки та інші інтерактивні вправи. Простий у використанні конструктор дозволяє додавати до завдань текст, зображення, аудіо та відео, роблячи процес навчання динамічним і захопливим.

**VisuAlgo** [10] – інтерактивний симулятор, який робить абстрактні поняття алгоритмів та структур даних візуально зрозумілими. Його головна цінність полягає в тому, що він дозволяє гравцю покроково спостерігати, як працює певний алгоритм, надаючи наочне уявлення про його логіку. Наприклад, коли гравець вибирає алгоритм сортування, система анімує переміщення елементів, показуючи кожне порівняння та обмін, а для обходу графу – покрокову історію дій. Ця візуалізація допомагає не просто запам’ятати назву алгоритму, а дійсно зрозуміти його механізм, що є критично важливим для таких складних концепцій, як алгоритми сортування, пошук у графах та робота з деревами. Додатково, візуалізація також містить псевдокод реалізації того чи іншого алгоритму (рис. 1.5).

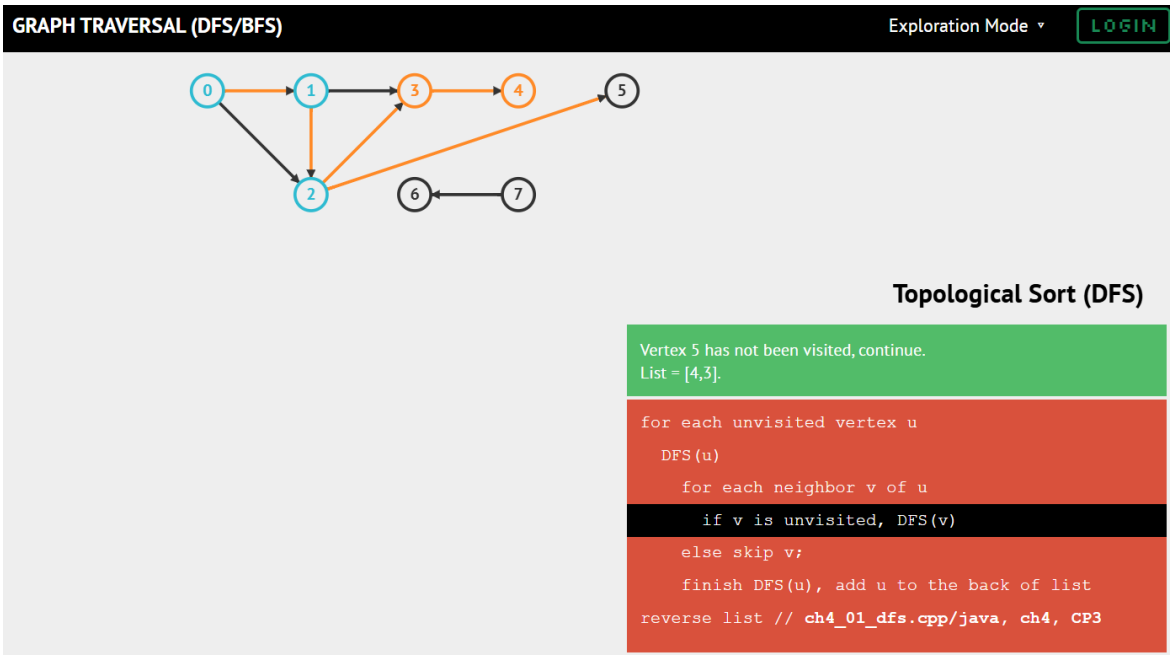



Рисунок 1.5 – Візуалізація обходу графу на VisuAlgo

Описані платформи, разом із раніше згаданими Human Resource Machine, 7 Billion Humans, CodinGame та CodeCombat, яскраво демонструють різноманітність підходів до гейміфікації освіти. Вони показують, як ігрові механіки можуть бути ефективно інтегровані в освітній процес, а які особливості будуть заважати.

Educaplay і CodinGame представляють гейміфікацію як інструмент навчання. Перша робить процес засвоєння інформації інтерактивним, додаючи елементи змагання, а друга перетворює складні професійні завдання на змагання, що мотивує гравців удосконалювати свої навички. З іншого боку, VisuAlgo, 7 Billion Humans та Human Resource Machine фокусуються на візуалізації та симуляції. Вони перетворюють абстрактні концепції (роботу процесора, логіку алгоритмів) на візуальні моделі, що дозволяє краще зрозуміти їхню внутрішню механіку. CodeCombat є прикладом навчання, вбудованого в гру, де освітній контент є частиною ігрового сюжету, а гравець вчиться програмуванню, щоб прогресувати.

Загальний стан речей показує, що гейміфікація – це набагато більше, ніж просто додавання балів та досягнень. Це фундаментальна зміна підходу до

навчання, що передбачає активну участь гравця замість пасивного споживання інформації. Елементи змагання та прогресу створюють сильну мотивацію, а миттєвий зворотний зв’язок відразу показує результат дій гравця. Такі платформи не тільки допомагають засвоїти факти, але й розвивають критичне мислення та навички вирішення проблем, перетворюючи навчання з рутинного обов’язку на захопливу пригоду.

### 1.3 Сходинки до інформатики

Найближчим аналогом до пропонованої освітньої системи є «Сходинки до інформатики» [11] – навчальний проєкт з 2011 року, комплекс навчально-розвивальних ігрових програм, які першочергово навчають учнів початкових та середніх класів правильному користуванню ПК, техніки безпеки, географії, інформатиці, базовій алгоритмізації, рідній мові та багатьом іншим темам.

Серед прикладів алгоритмізації можна взяти до уваги знаменитого «Садівника» (рис 1.6) – одна з найбільш впізнаваних ігор серед усіх в серії. Основним завданням є створення алгоритму з певних елементів «псевдокоду», де кінцевий результат виконання має бути автоматизація виконання садівником дій, необхідних для садження дерев. Дії можна як записувати в «блокнотик», який представляв собою псевдокод програми для виконання, так і виконувати дії безпосередньо при натисканні на кнопку для того, щоб зрозуміти логіку за процесом садження дерев.

Цей проєкт містить ще багато прикладів ігор, пов’язаних саме на алгоритмізації, і які були прекрасним рішенням для розвитку такого типу мислення у молодого покоління, наближаючи їх до розуміння того, як працюють АОК, та спонукаючи думати наперед. Проведення практичних занять на комп’ютерах з такими іграми супроводжувалося попередніми або слідуючими лекціями про алгоритми, візуалізацію пророблені роботи в блок-схемах тощо, що чудово вписувалося в загальний навчальний потік та розвиток школярів.


Рисунок 1.6 – Ігровий процес «Садівника»

Як програмне забезпечення, звичайно, цей комплекс поширювався незалежними копіями, які потрібно встановлювати окремо на кожен ПК та активувати ліцензію (так як продукт не забезпечувався МОН України). Це досить поширене та логічне рішення на момент середини 2010-х років. Зараз дистрибуція схожого ПЗ відбувається кардинально іншими способами, які є значно простішими для обох сторін, чим, очевидно, потрібно користуватися. Також МОН України набагато більше уваги приділяє цифровізації освіти, тому дуже актуально співпрацювати задля реалізації найкращих практик та покращення якості освіти для всіх рівнів – від початкової школи до ЗВО.

## 1.4 Постановка завдань

Із врахуванням особливостей описаних вище програмних продуктів та існуючого досвіду гейміфікації навчального процесу, необхідно дослідити можливості впровадження нових методів та централізованих підходів, орієнтуючись для початку на національний масштаб, ЗВО та сферу знань

«Інформаційні технології» з можливістю розширення у будь-якому вимірі за потреби.

Звичайно, розробка та впровадження єдиної національної геймифікованої платформи є дуже складним та косплексним завданням, навіть враховуючи лише MVP версію, це зайняло б десятки місяців. Окрім того, для підтвердження потреби та фіксації вимог до ПЗ, необхідно першочергово виконати дослідження ринку та/або цільової аудиторії. Market Research (дослідження ринку) – це систематичний процес збору, аналізу та інтерпретації інформації про цільові ринки, клієнтів, конкурентів та галузеві тенденції. Це ключовий компонент бізнес-стратегії, який допомагає компаніям приймати обґрунтовані рішення та залишатися конкурентоспроможними [12].

MVP – це версія нового продукту, яка має лише базові функції, достатні для того, щоб задовольнити перших користувачів і зібрати їхні відгуки. Основна ідея полягає в тому, щоб випустити продукт якомога швидше, не витрачаючи багато часу та ресурсів на розробку повного набору функцій, які можуть виявитися непотрібними або незатребуваними.

Головна мета MVP – перевірити гіпотезу про те, що продукт потрібен ринку, мінімізуючи ризики та витрати. Це дозволяє команді розробників та засновникам отримати цінний зворотний зв’язок від реальних користувачів і на його основі вирішити, чи варто продовжувати розвивати продукт, змінювати його напрямок (півотинг) чи взагалі відмовитися від ідеї [13].

Оскільки проведення повноцінного дослідження ринку на основі MVP очікуваної централізованої геймифікованої системи навчання не є можливим у ході дослідження зважаючи на масштаби, слід звернути увагу на більш локальний та доступний метод визначення актуальності системи, що розглядається. Для цього актуальним буде створення спрощеної інформаційної системи з локальним запуском та відсутністю серверного регулювання активності користувача та довгострокового збереження даних.

Основним концептом, який потребує перевірки та апробації, є використання ігор в якості сучасного допоміжного методичного засобу викладання та оцінювання навчальної успішності здобувачів. Отже, найбільший акцент зроблено саме на складовій гри.

Серед можливих тематичних напрямів дидактичних ігор для галузі знань «Інформаційні технології» обрано теми дисципліни «Алгоритми та структури даних» [14]. Це пояснюється низкою факторів:

– **абстрактність та складність:** дисципліна «Алгоритми та структури даних» є фундаментальною, але часто абстрактною та складною для розуміння. Здобувачам буває важко візуалізувати, як працюють алгоритми сортування, або як дані організовані в деревах чи графах. Інтерактивна гра дозволяє перетворити ці абстрактні поняття на візуальні та динамічні об’єкти, що полегшує їхнє засвоєння;

– **практичне застосування:** на відміну від теоретичних лекцій, гра пропонує здобувачам негайне практичне застосування знань. Замість того, щоб просто заучувати алгоритми, вони можуть «грати» з ними, вирішуючи головоломки та завдання, що вимагають розуміння принципів роботи структур даних. Це сприяє розвитку не лише пам’яті, а й критичного мислення та навичок виконання завдань;

– **мотивація та залученість:** навчання через гру перетворюється на захоплюючий процес. Елементи змагання, досягнення, отримання нагород та можливість взаємодії з іншими гравцями створюють додаткову мотивацію. Це допомагає підтримувати інтерес до теми, яка інакше могла б здаватися нудною чи занадто технічною. Замість пасивного споживання інформації, здобувач стає активним учасником освітнього процесу;

– **негайний зворотний зв’язок:** гра забезпечує миттєвий зворотний зв’язок. Якщо здобувач застосовує неправильний підхід, гра одразу покаже наслідки, дозволяючи йому швидко зрозуміти помилку та знайти правильне

рішення. Це набагато ефективніше, ніж очікування перевірки завдань викладачем.

Підсумовуючи, дисципліна «Алгоритми та структури даних» включно з суміжними напрямами та тематиками є однією з тих, які найбільше потребуть удосконалення в методах викладання матеріалу та оцінювання знань за сучасними вимогами та підвищення рівня знань серед здобувачів.

## Висновки до розділу 1

У першому розділі кваліфікаційної магістерської роботи описано предметну область обраної теми дослідження, виділено ключові застави теми, обґрунтовано актуальність та зроблено постановку завдань за напрямом дослідження. Розкрито об’єкт та предмет кваліфікаційної роботи, описано особливості впровадження та реалізації основної ідеї дослідження та бажаний кінцевий результат.

Проаналізовано наявні рішення з гейміфікації навчального процесу та розробки едукуючих ігор, розглянуто переваги та недоліки різних методів подачі та способів впровадження відповідного ПЗ, визначено ключові моменти для прийняття до уваги під час дослідження та проєктування способів інтеграції інформаційної системи геймифікованого навчання. Зокрема, визначено, що найбільш наближене за ідеєю рішення – це комплекс ігрових програм «Сходинки до інформатики». Зроблено його ретельний аналіз.

Поставлені та аргументовані основні завдання дослідження та способи їх виконання для досягнення поставленої мети кваліфікаційної роботи, обрано найактуальніший та реалістичний підхід щодо дослідження ринку та впровадження описаної інформаційної системи. Обрано першочергові навчальні теми для гейміфікації. Надано пріоритети, які ставляться під час проєктування описаного MVP для виконання дослідження ринку в локальним масштабах.

# 2 ОПИС МЕТОДІВ, НАУКОВОГО ТА МАТЕМАТИЧНОГО АПАРАТУ. ФОРМУВАННЯ СПЕЦИФІКАЦІЇ ВИМОГ ДО ПЗ

Врахочуючи специфіку теми дослідження, важливо в ході планування та проєктування застосунку і конкретних ігор ознайомитися з основоположними науковими підставами та математичним апаратом, який лежить в основі логіки розроблюваних ігор на навчального матеріалу, якому має навчатися здобувач під час використання системи. Для обраної дисципліни, серед головного – основні структури даних та поняття хешів, алгоритми сортування та пошуку [15].

## 2.1 Структури даних та хеш-функції

Структури даних є фундаментальними концепціями в програмуванні та інформатиці, які визначають спосіб організації, зберігання та управління даними для ефективного доступу та модифікації. Вибір правильної структури даних є критично важливим для продуктивності будь-якої програмної системи, безпосередньо впливаючи на часову та просторову складність алгоритмів, що працюють з цими даними [16].

Одними з найпростіших структур є масиви (Arrays) та зв’язані списки (Linked Lists). Масиви надають постійний час доступу (O(1)) до елементів за їхнім індексом, оскільки елементи зберігаються у безперервному блоці пам’яті. Однак, вставка або видалення елементів у середині масиву вимагає зсуву всіх наступних елементів, що призводить до часової складності O(N). Зв’язані списки, навпаки, складаються з вузлів, кожен з яких містить дані та посилання на наступний (або попередній) вузол. Це дозволяє вставляти та видаляти елементи за постійний час (O(1)) після знаходження позиції, але доступ до елемента за індексом вимагає послідовного проходу, що призводить до O(N) часової складності.

Дерева, зокрема двійкові дерева пошуку (BST – Binary Search Trees), надають збалансовану продуктивність для операцій пошуку, вставки та видалення. У BST кожен вузол має не більше двох дочірніх вузлів, і всі ключі в

лівому піддереві менші за ключ вузла, а в правому – більші. Для збалансованого BST (наприклад, AVL-дерева або червоно-чорних дерев), часова складність пошуку, вставки та видалення становить O(log(N)), де N – кількість вузлів. Це відбувається тому, що глибина дерева є логарифмічною від кількості елементів. У найгіршому випадку (незбалансоване дерево, що вироджується в зв’язаний список) продуктивність може деградувати до O(N). Просторова складність становить O(N) для зберігання вузлів.

Хеш-таблиці (Hash Tables) є однією з найефективніших структур даних для реалізації асоціативних масивів (словників), які дозволяють зберігати пари «ключ-значення» та забезпечують практично постійний час для операцій вставки, видалення та пошуку. Ядро хеш-таблиці – це хеш-функція.

Хеш-функція є математичною функцією, яка приймає вхідні дані (ключ) довільного розміру і повертає фіксоване ціле число (хеш-код або хеш-значення), яке потім використовується як індекс в масиві (бакетах хеш-таблиці). Мета наукового проєктування хеш-функції полягає в тому, щоб вона була:

– детермінованою: завжди повертає одне й те саме хеш-значення для одного й того ж ключа;

– швидкою в обчисленні: щоб не уповільнювати операції доступу;

– рівномірно розподіляючою: щоб мінімізувати колізії (коли різні ключі генерують однаковий хеш-код). Математично це означає, що ймовірність відображення будь-якого ключа на будь-який слот має бути рівномірною.

Для вирішення колізій застосовуються різні методи:

– метод ланцюгів (Chaining): кожен слот хеш-таблиці містить зв’язаний список (або іншу структуру даних) усіх елементів, які хешуються до цього слота;

– відкрита адресація (Open Addressing): у разі колізії, алгоритм шукає наступний доступний порожній слот у самій хеш-таблиці (наприклад, лінійне зондування, квадратичне зондування, подвійне хешування).

Якщо хеш-функція добре спроєктована, а завантаження хеш-таблиці (кількість елементів / кількість слотів) підтримується на низькому рівні, середній

час для операцій вставки, пошуку та видалення становить O(1). Проте, у найгіршому випадку, коли відбувається багато колізій, і всі елементи хешуються в один слот, продуктивність може деградувати до O(N), що підкреслює критичну роль якості хеш-функції та стратегії вирішення колізій у забезпеченні ефективності хеш-таблиць. Просторова складність хеш-таблиць зазвичай становить O(N).

Таким чином, від простоти масивів до складності хеш-таблиць, вибір та розуміння цих структур даних є наріжним каменем ефективної розробки програмного забезпечення [17].

## 2.2 Алгоритми сортування

У світі комп'ютерних наук сортування є однією з найбільш фундаментальних та широко досліджуваних задач. Вона полягає в упорядкуванні елементів колекції у певному порядку, що є передумовою для багатьох інших алгоритмів та структур даних. Різноманіття алгоритмів сортування вражає, і кожен з них має свої унікальні характеристики щодо часової складності, просторової складності та стабільності [18].

Одна з ігор, розроблених в ході виконання роботи та створення демонстраційного проєкту фокусується на розумінні здобувачем/гравцем того, як працює сортування. Це не тільки загальний принцип, плюси та мінуси, складність алгоритму, але і те, як сортування відбувається з точки зору роботи апаратного забезпечення з даними на низькому рівні.

### 2.2.1 Порівняльні сортування

Однією з найпростіших категорій є порівняльні сортування, які впорядковують елементи шляхом порівняння пар елементів. До таких належать такі класичні методи, як сортування бульбашкою (Bubble Sort), сортування вибором (Selection Sort) та сортування вставками (Insertion Sort). Сортування бульбашкою багаторазово проходить по списку, порівнюючи сусідні елементи та обмінюючи їх місцями, якщо вони знаходяться в неправильному порядку. Цей

процес повторюється до тих пір, поки весь список не буде відсортований. Незважаючи на свою простоту реалізації, його ефективність є досить низькою, особливо для великих наборів даних, адже в найгіршому та середньому випадку він має часову складність $O(N^2)$, що робить його непридатним для практичного використання у більшості випадків. Сортування вибором, у свою чергу, працює шляхом повторного пошуку мінімального елемента з невідсортованої частини списку та розміщення його на початку відсортованої частини. Воно виконує менше обмінів, ніж сортування бульбашкою, але його часова складність також залишається $O(N^2)$, тому він теж не є оптимальним для великих масивів. Сортування вставками будує фінальний відсортований масив (або список) по одному елементу за раз. Кожен новий елемент вставляється у відповідне місце серед уже відсортованих елементів. Цей метод може бути ефективним для невеликих масивів або для масивів, які вже майже відсортовані, демонструючи часову складність $O(N)$ у найкращому випадку, але все ще $O(N^2)$ у найгіршому. Усі ці три алгоритми є стабільними (зберігають відносний порядок рівних елементів) і потребують мінімальної додаткової пам’яті ($O(1)$), що робить їх цікавими для навчальних цілей, але рідко використовуваними на практиці для значних обсягів даних.

На противагу цим $O(N^2)$ алгоритмам, існують значно ефективніші порівняльні сортування, що досягають оптимальної асимптотичної складності $O(N \times \log(N))$. До них належать сортування злиттям (Merge Sort), швидке сортування (Quick Sort) та сортування купою (Heap Sort). Сортування злиттям є прикладом підходу «розділяй і володарюй». Воно рекурсивно розбиває масив навпіл, сортує кожну половину, а потім зливає відсортовані половини назад в один відсортований масив. Його ключовою перевагою є гарантована часова складність $O(N \times \log(N))$ у всіх випадках (найкращому, середньому та найгіршому), а також стабільність. Проте, він вимагає додаткової пам’яті розміром $O(N)$ для операції злиття, що може бути суттєвим недоліком для дуже великих наборів даних або обмежених ресурсів.

Швидке сортування, також базуючись на принципі «розділяй і володарюй», є одним з найпопулярніших алгоритмів завдяки своїй високій швидкості на практиці. Воно обирає «опорний елемент» (pivot) з масиву і переставляє інші елементи таким чином, щоб усі елементи, менші за опорний, знаходились до нього, а більші – після нього. Потім алгоритм рекурсивно застосовується до підмасивів ліворуч і праворуч від опорного елемента. Середня часова складність швидкого сортування становить O(N×log(N)), що робить його дуже ефективним. Однак у найгіршому випадку (наприклад, коли опорний елемент обирається невдало, і масив уже відсортований у тому чи іншому порядку) його продуктивність може погіршитися до $O(N^2)$. Хоча це рідкість на практиці, стратегії вибору опорного елемента (наприклад, медіана трьох, випадковий вибір) застосовуються для мінімізації цього ризику. Швидке сортування зазвичай не є стабільним і може вимагати O(log(N)) або O(N) додаткової пам’яті залежно від реалізації та глибини рекурсії.

Сортування купою використовує структуру даних бінарна купа (binary heap) – спеціальне дерево, яке задовольняє властивості купи (кожен вузол більший або дорівнює своїм нащадкам для максимальної купи, або менший/дорівнює для мінімальної купи). Алгоритм спочатку будує купу з елементів масиву, а потім багаторазово витягує максимальний (або мінімальний) елемент з кореня купи та поміщає його в кінець відсортованої частини масиву. Цей процес повторюється, доки купа не стане порожньою. Сортування купою має гарантовану часову складність O(N×log(N)) як у найгіршому, так і в середньому випадку, і, що важливо, воно сортує «на місці», тобто вимагає лише O(1) додаткової пам’яті, що робить його привабливим для великих наборів даних з обмеженою пам’яттю. Однак воно не є стабільним, і на практиці може бути дещо повільнішим за швидке сортування через меншу локальність даних та більш складну схему доступу до пам’яті.

### 2.2.2 Непорівняльні алгоритми сортування

Крім порівняльних методів, існують непорівняльні сортування, які не базуються на порівняннях елементів, а використовують інші властивості даних (наприклад, діапазон значень або цифри). Ці алгоритми можуть досягати лінійної часової складності O(N+K) або O(N×K) (де K залежить від діапазону значень або кількості розрядів) у певних умовах. До них відносяться сортування підрахунком (Counting Sort), сортування за розрядами (Radix Sort) та сортування комірками (Bucket Sort).

Сортування підрахунком є ефективним, коли елементи є цілими числами в невеликому, обмеженому діапазоні. Воно працює шляхом підрахунку кількості входжень кожного унікального елемента, а потім використовує ці підрахунки для обчислення позиції кожного елемента у вихідному відсортованому масиві. Його часова складність становить O(N+K), де K – це діапазон значень. Сортування підрахунком є стабільним і може бути дуже швидким для відповідних даних, але вимагає додаткової пам’яті O(K) і не підходить для великих діапазонів значень або нецілочисельних даних.

Сортування за розрядами є узагальненням сортування підрахунком і може сортувати числа з довільним діапазоном значень, обробляючи їх порозрядно. Воно сортує елементи за кожним розрядом (починаючи з найменш значущого або найбільш значущого), використовуючи стабільне сортування (наприклад, сортування підрахунком) на кожному етапі. Для чисел з D розрядами та базою R (наприклад, 10 для десяткових чисел), часова складність складає O(D(N+R)). Цей метод є ефективним для великих наборів цілих чисел, особливо коли R є достатньо великим, а D – невеликим. Він також може бути стабільним, якщо використовується стабільне проміжне сортування.

Сортування комірками (Bucket Sort), або сортування відрами, розподіляє елементи по «комірках» (відрах), кожна з яких відповідає певному діапазону значень. Потім кожна комірка сортується окремо (часто за допомогою іншого алгоритму, наприклад, сортування вставками), і, нарешті, вміст комірок

об’єднується в один відсортований список. Якщо елементи розподілені рівномірно, цей метод може досягти середньої часової складності O(N+K), де K – кількість комірок. Однак у найгіршому випадку, коли всі елементи потрапляють в одну комірку, його продуктивність може погіршитися до складності алгоритму, що використовується для сортування комірок.

### 2.2.3 Оцінка ефективності сортування

Для гри про сортування, ключовим моментом для визначення ефективності сортування проведеного користувачем є розрахунок кінцевої оцінки. Це дуже важливий момент не тільки через свою загальну роль, але також і через особливості, які потрібно врахувати при реалізації:

– універсальність (метод має бути актуальним та рівнозначно об’єктивним для різних розмірів масиву);

– наочність (результат має бути достатньо зрозумілим для використання його як своєрідної метрики успішності виконаного гравцем завдання, а не тільки у порівнянні);

– ефектність (кінцевий бал, хоч і має базуватися на наукових принципах та особливостях алгоритмів, має бути поданим у ігровій формі).

Розглянемо такий метод:

```
    private calculateRating(): number {
        const n = this.array.array.length;

        const optimalComparisons = n * Math.log2(N);
        const optimalCopies = n - 1;
        const optimalCost = 2 * optimalComparisons + 3 * optimalCopies;

        // Actual cost includes penalties for errors, temp arrays, and finish
attempts
        const actualCost =
            2 * this.stats.comparisons +
            3 * this.stats.copies +
            10 * this.stats.errors +
            this.stats.tempArrayPenalty +
            this.stats.finishAttemptPenalty;

        // Ensure the return value is an integer
        return Math.floor((optimalCost / Math.max(1, actualCost)) * 1000);
    }
```

Метод calculateRating являє собою комплексну метрику для кількісної оцінки ефективності алгоритму сортування. Він виходить за рамки простого підрахунку операцій, інтегруючи теоретичні мінімуми, емпіричні витрати та штрафи за неефективне використання ресурсів чи логічні помилки.

В основі методу лежить порівняння ідеальної теоретичної вартості (optimalCost) з фактичною вартістю (actualCost), понесеною алгоритмом.

**Теоретична (оптимальна) вартість**

Оптимальна вартість розраховується як зважена сума теоретично мінімальних операцій, необхідних для сортування масиву з n елементів.

1) **Оптимальні порівняння (optimalComparisons):** Ця величина встановлюється як N×log(N). Це фундаментальна нижня межа для будь-якого алгоритму сортування, що базується на порівняннях. Вона випливає з теорії інформації: щоб розрізнити N! можливих початкових перестановок масиву, потрібно щонайменше log(N!) порівнянь, що асимптотично еквівалентно N×log(N). Цей показник є «золотим стандартом», якого досягають найефективніші алгоритми (наприклад, сортування злиттям – merge sort).

2) **Оптимальні копіювання (optimalCopies):** Ця величина встановлена як N-1. Вона представляє абсолютний теоретичний мінімум переміщень даних. Такий сценарій можливий, наприклад, у циклічній перестановці, де для сортування всього масиву потрібно, щоб кожен з N-1 елементів був переміщений один раз, а останній став на своє місце автоматично. Цей показник є більш суворим ідеалом, ніж N×log(N), і слугує для оцінки ефективності переміщення даних.

3) **Зважування операцій:** Вартість операцій зважується для відображення їх реального впливу на продуктивність. Порівнянням надається вага 2, а копіюванням – 3. Ця евристика базується на архітектурі комп’ютера, де операції запису в пам’ять (частина копіювання) є, як правило, більш «важкими» та ресурсомісткими, ніж операції читання та порівняння.

Таким чином, optimalCost – це еталон, що представляє мінімально можливу «роботу» для гіпотетично ідеального алгоритму: 2 × optimalComparisons + 3 × optimalCopies.

**Фактична вартість**

Фактична вартість (actualCost) є зваженою сумою реально виконаних операцій та додаткових штрафів, що відображають загальну ефективність стратегії сортування.

1. **Базова вартість:** 2 × this.stats.comparisons + 3 × this.stats.copies. Це прямий емпіричний підрахунок виконаних порівнянь та копіювань, зважених за тією ж логікою, що й оптимальна вартість.

2. **Штрафи (Penalties):** Це ключове нововведення, що робить оцінку більш глибокою:

– 10 × this.stats.errors: Штраф за помилки. Цей компонент має високу вагу (10), що підкреслює пріоритет коректності алгоритму. Він «карає» за будь-які неправильні дії, зроблені гравцем під час сортування;

– this.stats.tempArrayPenalty: Штраф за використання тимчасових масивів. Він враховує просторову складність алгоритму. Алгоритми, що вимагають додаткової пам’яті (як класичне сортування злиттям), отримують штраф, що робить «in-place» алгоритми (ті, що сортують масив на місці) більш привабливими з точки зору цієї метрики;

– this.stats.finishAttemptPenalty: Штраф за спробу завершення. Він карає за передчасні або невірні спроби фіналізувати процес сортування, стимулюючи гравця до впевненого визначення моменту, коли масив дійсно відсортований.

**Кінцевий рейтинг**

Підсумковий рейтинг (rating) обчислюється як відношення оптимальної вартості до фактичної, масштабоване для зручності та наочності: Math.floor((optimalCost / Math.max(1, actualCost)) × 1000).

– Співвідношення optimalCost / actualCost показує, наскільки близько фактична продуктивність алгоритму підійшла до теоретичного ідеалу.

– Використання Math.max(1, actualCost) є захисним механізмом, що запобігає діленню на нуль.

– Множник 1000 перетворює результат у зручний для читання числовий рейтинг, а Math.floor забезпечує цілочисельне значення.

Отже, метод calculateRating надає багатогранну, науково обґрунтовану оцінку. Він аналізує не лише асимптотичну ефективність (через порівняння та копіювання), але й практичні аспекти реалізації: використання пам’яті та логічну надійність, що робить його потужним інструментом для всебічного аналізу алгоритмів сортування та є чудовим прикладом оцінювання результатів гри, надаючи користувачу розуміння якості наданого рішення задачі.

## 2.3 Пошукові алгоритми

Пошукові алгоритми є фундаментальним компонентом обчислювальної науки, що забезпечує ефективний доступ до даних у колекціях. Їхня ефективність є критично важливою для продуктивності багатьох систем, від баз даних до пошукових систем. Аналіз цих алгоритмів зосереджується на їхній часовій складності (кількості операцій, необхідних для знаходження елемента) та просторовій складності (використання пам’яті) [19].

Лінійний пошук (Sequential Search) є найпростішим методом, який перевіряє кожен елемент колекції послідовно, починаючи з першого, доки не знайде потрібний елемент або не досягне кінця колекції. Цей алгоритм не вимагає, щоб дані були відсортовані. Його часова складність у найгіршому випадку та середньому випадку є O(N), де N – кількість елементів, оскільки в найгіршому випадку доведеться переглянути всі елементи. У найкращому випадку (якщо шуканий елемент є першим) складність становить O(1). Лінійний пошук вимагає O(1) додаткової пам’яті, що робить його придатним для дуже малих наборів даних або несортованих колекцій, де ефективність не є критичною.

На противагу цьому, бінарний пошук (Binary Search) значно ефективніший, але вимагає, щоб колекція була відсортованою. Він працює за принципом «розділяй і володарюй», багаторазово ділячи інтервал пошуку навпіл. Алгоритм порівнює шуканий елемент з елементом у середині інтервалу. Якщо елементи збігаються, пошук завершено. Якщо шуканий елемент менший, пошук продовжується у лівій половині; якщо більший – у правій.

Цей процес повторюється до звуження інтервалу до одного елемента або до його порожнечі. Завдяки такому підходу, часова складність бінарного пошуку становить O(log(N)) у всіх випадках (найкращому, середньому та найгіршому), що є значним покращенням порівняно з лінійним пошуком для великих N. Простірна складність також O(1) для ітеративної реалізації або O(log(N)) для рекурсивної (через стек викликів). Бінарний пошук є основою для швидкого пошуку у відсортованих масивах та структурах даних, таких як двійкові дерева пошуку.

Третій потужний підхід – це пошук за допомогою хешування (Hashing). Цей метод використовує хеш-функцію для безпосереднього відображення ключа елемента на його індекс (або адресу) у таблиці (хеш-таблиці). В ідеальному випадку, якщо хеш-функція рівномірно розподіляє ключі, а колізії (коли різні ключі відображаються на один і той же індекс) відсутні або ефективно обробляються, часова складність пошуку, вставки та видалення може становити O(1) у середньому.

Це робить хешування надзвичайно швидким методом для доступу до даних. Однак у найгіршому випадку (наприклад, при всіх колізіях, коли всі ключі відображаються на один і той же індекс), продуктивність може деградувати до O(N), подібного до лінійного пошуку.

Просторова складність хеш-таблиць зазвичай становить O(N) для зберігання елементів та додаткових структур для обробки колізій. Вибір ефективної хеш-функції та стратегії вирішення колізій є критично важливим для практичної продуктивності хешування.

## 2.4 Алгоритми пошуку шляху в графах

Алгоритми пошуку шляху в графах є фундаментальним розділом дискретної математики та інформатики, застосовуючись у навігаційних системах, маршрутизації мереж, штучному інтелекті та багатьох інших областях. Граф – це сукупність вершин (вузлів) та ребер (зв’язків), що їх з’єднують. Задача пошуку шляху полягає у знаходженні послідовності ребер, що з’єднують дві вершини (вихідну та цільову) у графі.

Пошук у ширину (BFS – Breadth-First Search) є одним з найпростіших алгоритмів для обходу графа або дерева. Він досліджує всі вершини на заданій «глибині» перед тим, як перейти до вершин наступної глибини. BFS використовує чергу (queue) для відстеження вершин, які потрібно відвідати. З наукової точки зору, BFS гарантує знаходження найкоротшого шляху (за кількістю ребер) у незважених графах. Його часова складність становить O(V+E), де V – кількість вершин, а E – кількість ребер, оскільки кожна вершина та кожне ребро відвідуються максимум один раз. Простірна складність також O(V) у найгіршому випадку, оскільки черга може містити всі вершини на одному рівні.

Пошук у глибину (DFS – Depth-First Search), на відміну від BFS, досліджує якомога глибше по кожній гілці, перш ніж повертатися та досліджувати інші гілки. DFS використовує стек (implicit stack для рекурсії або explicit stack) для відстеження вершин. З наукової точки зору, DFS не гарантує знаходження найкоротшого шляху. Він часто використовується для визначення зв’язності графа, пошуку циклів, топологічного сортування або генерації лабіринтів. Часова та просторова складність DFS також становить O(V+E) та O(V) відповідно, подібні до BFS, але зі стеком замість черги.

Для зважених графів (де ребра мають «вагу» або «вартість»), одним з найбільш відомих алгоритмів є алгоритм Дейкстри (Dijkstra’s Algorithm). Він знаходить найкоротші шляхи від однієї початкової вершини до всіх інших вершин у графі з невід’ємними вагами ребер. Алгоритм працює ітеративно,

підтримуючи множину відвіданих вершин та оцінки найкоротших відстаней до невідвіданих вершин. Він завжди обирає вершину з найменшою поточною оцінкою відстані для подальшого розширення. Його ефективність сильно залежить від реалізації пріоритетної черги: за допомогою двійкової купи часова складність становить O(ElogV), а за допомогою фібоначчієвої купи – O(E+V×log(V)). Алгоритм Дейкстри є основою для багатьох мережевих протоколів маршрутизації [20].

Нарешті, алгоритм A* (A-star Search) [21] є розширенням алгоритму Дейкстри, який використовує евристичну функцію для прискорення пошуку. Він також знаходить найкоротший шлях у зважених графах з невід'ємними вагами, але є «інформованим» алгоритмом, тобто він використовує додаткову інформацію (евристику) для оцінки відстані від поточної вершини до цільової. A* обирає наступну вершину для дослідження, мінімізуючи суму «вартості пройденого шляху» (G(N)) та «очікуваної вартості до цілі» (h(N), евристика). Якщо евристична функція є допустимою (тобто ніколи не переоцінює фактичну вартість до цілі), A* гарантовано знаходить оптимальний шлях. Його часова складність залежить від якості евристики і в найкращому випадку може бути близькою до лінійної для певних графів, але в найгіршому випадку все ще O(E) або O(V×log(V)) у залежності від реалізації купи та якості евристики. Просторова складність може бути значною, оскільки йому потрібно зберігати відкриті та закриті списки вершин.

### 2.5 Дерева та обхід дерев. Бінарні дерева

Деревоподібні структури даних є фундаментальним поняттям у дискретній математиці та інформатиці, представляючи собою ієрархічну організацію елементів, на відміну від лінійної природи масивів чи зв'язних списків. Математично дерево можна визначити як зв'язний ациклічний неорієнтований граф, в якому будь-які дві вершини з'єднані рівно одним простим шляхом. У контексті структур даних дерево зазвичай є вкоріненим, тобто має виділену

вершину – корінь (root), від якої розгалужуються зв’язки до дочірніх вузлів, утворюючи рекурсивну структуру піддерев. Важливою властивістю дерева з n вершинами є наявність рівно n-1 ребер. Висота дерева визначається як довжина найдовшого шляху від кореня до листка, що безпосередньо впливає на ефективність алгоритмів, які працюють з цією структурою. Дерева широко застосовуються для моделювання ієрархій, таких як файлові системи, синтаксичні дерева в компіляторах та DOM-моделі веб-сторінок [22].

Обхід дерева (Tree Traversal) – це систематичний процес відвідування кожної вершини дерева рівно один раз для обробки даних, що в них містяться. На відміну від лінійних структур, де обхід є тривіальним, дерева можна обходити різними шляхами, які класифікують на пошук у глибину (DFS) та пошук у ширину (BFS). У контексті бінарних дерев виділяють три основні порядки глибинного обходу: прямий (pre-order), центрований (in-order) та зворотний (post-order). Прямий обхід спочатку відвідує корінь, потім ліве і праве піддерева, що корисно для копіювання дерева. Центрований обхід відвідує ліве піддерево, корінь, а потім праве, що у випадку бінарних дерев пошуку дозволяє отримати відсортовану послідовність значень. Зворотний обхід спочатку опрацьовує нащадків, а потім корінь, що є критично важливим для видалення вузлів або обчислення математичних виразів. Часова складність усіх цих алгоритмів становить O(N), оскільки кожен вузол відвідується один раз, а просторова складність залежить від висоти дерева h і становить O(h) через використання стека викликів рекурсії [23].

Бінарні дерева пошуку (Binary Search Trees, BST) є спеціалізованим видом бінарних дерев, які накладають суворий порядок на розміщення ключів для оптимізації операцій пошуку, вставки та видалення.

Основна властивість BST полягає в тому, що для будь-якого вузла x, всі ключі в лівому піддереві менші за ключ x, а всі ключі в правому піддереві – більші або рівні ключу x. Завдяки цій властивості операції пошуку можуть виконуватися за час O(h), де h – висота дерева. У найкращому випадку, коли

дерево ідеально збалансоване, висота дорівнює $\log_2(N)$, що забезпечує логарифмічну складність O(log(N)), аналогічну бінарному пошуку в масиві. Проте у найгіршому випадку, коли дерево вироджується в лінійний ланцюжок (наприклад, при вставці вже відсортованих даних), висота досягає n, а часова складність деградує до O(N), що робить просте BST неефективним для певних сценаріїв використання. Якщо на створення таких дерев можна повпливати – потрібно уникати цих випадків шляхом оптимізації алгоритму заповнення дерева вузлами [24].

Для розв'язання проблеми деградації продуктивності незбалансованих бінарних дерев були розроблені самобалансовані бінарні дерева пошуку, такі як АВЛ-дерева (AVL trees) або червоно-чорні дерева (Red-Black trees). Ці структури даних автоматично підтримують свою висоту на рівні O(log(N)) шляхом виконання спеціальних операцій, званих обертаннями (rotations), під час вставки або видалення вузлів. Наприклад, АВЛ-дерево суворо стежить за тим, щоб різниця висот лівого і правого піддерев будь-якого вузла не перевищувала по модулю одиницю. Це гарантує, що всі основні операції – пошук, вставка та видалення – завжди виконуватимуться за логарифмічний час O(log(N)), незалежно від порядку надходження вхідних даних. Такі структури є основою для реалізації асоціативних масивів та множин у багатьох стандартних бібліотеках мов програмування, забезпечуючи стабільну та передбачувану продуктивність.

## 2.6 Специфікації вимог до програмного забезпечення

### ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

**Призначення системи (застосунку), для якої розробляється програмне забезпечення**

Призначенням системи є надання доступу до тематичних ігор з метою гейміфікації навчального процесу.

**Погодження, що ухвалені в програмній документації**

Погоджено, що для створення ПЗ та його злагодженої роботи будуть використовуватися бібліотека React та ігровий рушій Phaser.

**Межі проєкту ПЗ**

Крайня дата завершення роботи над ПЗ – 01.12.2025р.

**ЗАГАЛЬНИЙ ОПИС**

**Сфера застосування**

Пріоритетна сфера застосування – навчальна, а саме – для старшокласників та здобувачів у ЗВО, з можливістю локального запуску на ПК користувача або дистрибутивного серверу з поширенням по локальній системі.

**Характеристики користувачів**

Основні характеристики користувачів: доступ до мережі, в якій розгорнуто ПЗ (якщо централізовано), пристрій (ПК, ноутбук, планшет), браузер.

**Загальна структура і склад системи**

Для максимальної легкості розгортання та використання системи локально, структура має бути монолітна. Використання баз даних чи інших допоміжних систем не передбачено на початкових стадіях.

**Загальні обмеження**

Обмеження для розгортання ПЗ – ОС з підтримкою Node.js/NPM, Python або інших аналогічних інструментів для забезпечення середовища запуску застосунку.

Обмеження для використання ПЗ – підключення до мережі, в якій розгорнуто ПЗ (якщо централізовано), сучасний веббраузер.

**ФУНКЦІЇ СИСТЕМИ**

**Пошук ігор**

**Опис функції**

Функція пошуку дозволяє користувачу здійснювати пошук ігор за назвою або її ієрархічним положенням у схемі навчального матеріалу.

**Вхідна та вихідна інформація**

Вхідна інформація – фільтр по назві або опису гри, або вибір користувачем потрібної теми чи дисципліни в ієрархії;

Вихідна інформація – список відфільтрованих ігор;

**Функціональні вимоги**

Клас-сервіс керування відображуваним списком ігор

**Функція гри**

**Опис функції**

Функція гри дозволяє користувачу починати ігрові сесії та грати в доступні ігри, досягати мети гри та бачити свої результати.

**Вхідна та вихідна інформація**

Вхідна інформація – інформація про обрану гру, введення користувача залежно від гри;

Вихідна інформація – ігровий інтерфейс, результати гри.

**Функціональні вимоги**

Реалізовані ігрові сцени Phaser.

**Функція перегляду списку лідерів**

**Опис функції**

Функція перегляду списку лідерів дозволяє користувачу переглядати найкращі результати проходження конкретної гри та деяку інформацію про відповідні найкращі сесії, включно з іменем користувача, якому сесія належала.

**Вхідна та вихідна інформація**

Вхідна інформація – інформація про обрану гру;

Вихідна інформація – список лідерів;

**Функціональні вимоги**

Сервіс для керування Local storage.

## ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

**Джерела і зміст вхідної інформації (даних)**

В цьому ПЗ джерелом вхідної інформації є користувач.

**Нормативно-довідкова інформація (класифікатори, довідники тощо)**

Використані в грі елементи наукової термінології, літератури, посилання та методів мають відповідати національним стандартам Міністерства освіти і науки України та міжнародним стандартам.

**Вимоги до способів організації, збереження та ведення інформації**

Спрощена версія інформаційної системи має використовувати Local Storage браузерного клієнту для збереження таких даних, як інформація про поточного користувача та список лідерів.

## ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Користувач повинен мати комп’ютер, ноутбук або планшет із браузером та доступом до мережі, в якій централізовано розгорнуто ПЗ, або з доступом до самої збірки ПЗ та інструментів для її самостійного локального хостингу (Node.js/NPM, Python або інших аналогічних інструментів для забезпечення середовища запуску застосунку).

Вимоги до серверу: мінімум 2 ГБ оперативної пам’яті, рекомендовано – 4 ГБ та більше.

## ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Архітектура програмної системи**

Архітектура ПЗ складається з клієнтського застосунку (Web client). Основними внутрішніми компонентами застосунку є:

- модуль списку ігор;
- ігровий модуль;
- модулі додаткових UI елементів;
- контрольний модуль застосунку (роутинг);

**Системне програмне забезпечення**

Для написання вебзастосунку використати бібліотеку React, мову програмування TypeScript. Для реалізації ігрового процесу використати ігровий рушій Phaser.

**Мережне програмне забезпечення**

Для створення ПЗ використовується Windows 11, середовище розробки Visual Studio Code і будь-який сучасний браузер.

**Програмне забезпечення ведення інформаційної бази**

Ведення локальної інформаційної бази користувача здійснюється за допомогою локального сховища браузерного клієнта (local storage).

**Мова і технологія розробки ПЗ**

Програмне забезпечення має бути розроблене з використанням бібліотеки React. Мова розробки – JavaScript/TypeScript [25].

## ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

**Інтерфейс користувача**

Вебклієнт має задовольняти усім вимогам UX та UI, що дозволить користувачу затратити найменше часу на розуміння роботи системи. Компоненти сторінки мають бути інтуїтивно зрозумілими. Відображення ігрових елементів та елементів інтерфейсу мають відповідати вкладеному інформаційному наповненню, використовуючи відповідні кольори, акценти та стилі.

**Апаратний інтерфейс**

Апаратний інтерфейс – один фізичний або віртуальний сервер для розгортання ПЗ, та пристрій користувача (ПК, ноутбук чи планшет), який він буде використовувати для взаємодії з сторінками вебклієнту.

**Програмний інтерфейс**

React [26] – JavaScript-бібліотека для створення користувацьких інтерфейсів. Phaser [27] – це безкоштовний фреймворк для розробки HTML5-ігор, що дозволяє швидко створювати кросплатформні ігри для вебу (розподілених гіпермедійних систем) та мобільних пристроїв.

**Комунікативний протокол**

Застосунок базується на використанні мережних протоколів HTTP/HTTPS.

## ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Доступність**

Застосунок є доступним для всіх користувачів, що мають бажання ним користуватися, за умови наявності підключення до мережі, в якій розгорнуто ПЗ.

**Супроводжуваність**

Рівень супроводжуваності підтримується за рахунок її модульної архітектури, що спрощує незалежне оновлення та виправлення окремих компонентів, а також завдяки використанню популярного фреймворку React, який забезпечує широкий доступ до документації та спільноти розробників. Додатково, супроводжуваність має підтримуватися шляхом залучення молодих спеціалістів до розробки ігор для розширення наявного в системі списку.

**Переносимість**

Вебклієнт має сумісність з усіма сучасними веббраузерами, незалежно від операційної системи, на якій вони працюють (наприклад, Windows, macOS, Linux, Android, iOS).

Розгортання та хостинг серверної частини застосунку є доступними на більшості операційних систем, включаючи Windows, macOS та Linux-подібні системи.

**Продуктивність**

Продуктивність ПЗ залежить від швидкості мережі та характеристик клієнтського обладнання, особливо – оперативної пам’яті та спроможностей графічного процесору. Час виконання запитів не має перевищувати 3 секунд.

**Надійність**

Для MVP, дані користувача не мають передаватися на сервер, а лише зберігатися в локальному сховищі браузерного клієнту.

У випадку розширення та подальших модифікацій системи із пріоритетом на перенесенні основних обчислювальних дій та збереження даних користувача та застосунку на сервері, вхідні дані, які передаються користувачем повинні бути приватними. Має бути виключена можливість їх отримання будь-якими іншими способами. Користувач отримує доступ виключно до своїх даних та лише після авторизації у системі.

**Безпека**

На майбутнє має бути передбачено, що функціонал, який дозволяє взаємодіяти з даними, має перевіряти токен авторизації та права доступу.

## Висновки до розділу 2

У другому розділі кваліфікаційної роботи викладено теоретичне підґрунтя математичних та наукових методів, що тісно пов'язані з розробкою дидактичних ігор в інформаційній системі для гейміфікації освітнього процесу. Виклад теоретичного апарату включає необхідний для розуміння алгоритмічний та структурний матеріал.

Описано принципи роботи та особливості використання структур даних. Наведено обґрунтування необхідності активного застосування хешування та хеш-функцій. Зроблено акцент на важливості вибору або створення якісної та релевантної хеш-функції для мінімізації колізій у хеш-таблиці. Надано теоретичне підґрунтя для розуміння алгоритмів пошуку та графо-пошукових алгоритмів.

Проведено детальний аналіз низки алгоритмів, ключових для розуміння принципів сортування. Окремо описано особливості алгоритмів, що базуються на порівнянні елементів, та, на противагу їм, непорівняльних алгоритмів. Для наочної демонстрації важливості теоретичної бази для написання програмного коду дидактичних ігор, наведено приклад коду методу оцінювання результату сортування масиву користувачем. Цей приклад взято з однієї з розроблених ігор, і він містить аргументацію щодо використання конкретних теоретичних елементів алгоритмізації.

На основі описаного наукового апарату, аналізу ринку, а також визначення потреб та пріоритетів щодо створення інформаційної системи, було розроблено специфікацію вимог до програмного забезпечення. До неї включено перелік основних функцій системи, які обов'язково мають бути реалізовані у MVP.

# 3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Функціональність та організація даних

Демонстраційний проєкт має бути маскимально простим для використання та сфокусованим на самому процесі гри. Враховуючи описані в п. 2.5 специфікації, мінімумом для проєктованого ПЗ є наступні функції:

- пошук ігор;
- власне гра;
- модифікація імені користувача;
- інструкція/навчання до гри;
- таблиця лідерів.

Хоча для мінімального потоку користувача деякі з цих функцій не обов’язкові – вони об’єктивно є невід’ємними з точки зору досвіду користувача (UX, user experience). Зазначені функції системи та їхні співвідношення (реляції) добре відображає діаграма способів використання (рис. 3.1) [28].

Тут слід виділити деякі дії, які не відносяться до напряму користувача, хоча і виконуються ним. «Restart the game» – функція, яка дозволяє користувачу перезапустити гру під час її активної фази – це необхідно для того, щоб користувачу не доводилося постійно виходити в меню чи перезавантажувати сторінку. Функція доступна лише під час гри та є опціональною, тому відноситься до гри як <<extend>> дія. Також важливо виділити вибір гри – як і «рестарт», це не ключова/кінцева функція для користувача у потоці роботи, але необхідна для інших функцій. Так, меню, яке дозволяє користувачу зіграти в певну гру, подивитися список лідерів або прочитати інструкцію до неї не є доступним без вибору гри – звідси необхідність в <<include>> зв’язку. Нарешті функції зміни імені користувача або перегляд таблиці лідерів є ключовими для користувача, впливаючи на потік для ігрового процесу (особливо під час

врахування результатів та процесу завершення гри), але не є корінними її залежностями.

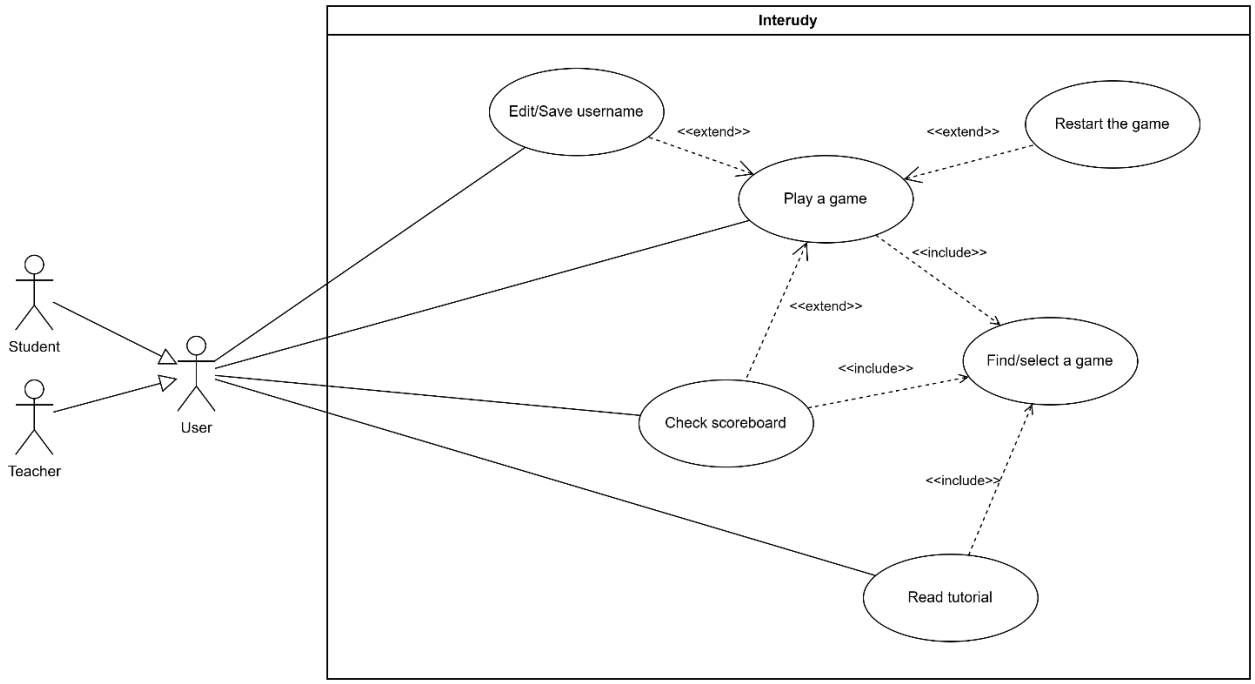



Рисунок 3.1 – Use case diagram

Ще одна деталь – це актори «Student» та «Teacher», які генералізують користувача. Це рішення зумовлено тим, що хоча ці два актори мають різні цілі, в системі демонстраційного проєкту вони не мають між собою різниці в наявному функціоналі. Розширення ж системи та централізація збережених даних мали б підстави для надання певних адміністративних особливостей певній групі користувачів, наприклад, викладач.

Оскільки застосунок матиме можливість містити ігри з різних напрямків та дисциплін, доречно продумати групування цих ігор за відповідними категоріями. Окрім покращення загальної організації ігор в системі та кодовій базі, це надасть нові можливості для пошуку ігор в списку доступних та візуалізації асортименту. Відповідно, вирішено відобразити гру в системі як листя дерева з 5 рівнів: корінь, дисципліна, тема, підтема, гра. Це найкраще буде відобразити діаграмою класів (рис. 3.2).

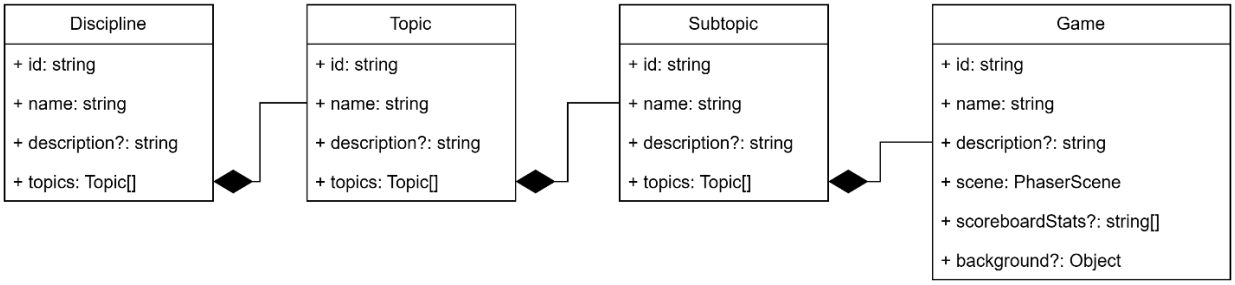



Рисунок 3.2 – Діаграма класів дерева ігор

Описана структура на діаграмі рис. 3.2 не має власної функціональності, адже першочергово потрібна для організації, тому не містить методів. Серцевиною гри є сцена, представлена компонентом PhaserScene, або таким, який її розширює – саме сцена визначатиме гру та її особливості/поведінку. Це працюватиме так:

1) користувач обирає гру в списку/дереві та натискає Play;

2) запускається Phaser Canvas, а його сценою для відображення передається збережена в полі гри сцена;

3) під час запуску ігрової сцени або ж після гри, код застосунку також може використовувати додаткові поля Game для окремих налаштувань.

Додаткові поля класу гри є конфігураційними, наприклад, background – для зміни фону конкретної гри, scoreboardStats – для зазначення полів та статистик, які потрібно зберігати в таблицю лідерів під час завершення гри користувачем. Цей список може бути розширеним музикальним треком на фоні, іншими візуальними або функціональними кастомізаціями, які специфічні для кожної окремої гри та/або потребують можливості до зміни в режимі реального часу (без зміни коду ігрової сцени або застосунку вцілому).

Такий метод представлення є важливим для майбутнього розширення системи – рішення дозволяє застосовувати максимально диверсійні підходи до візуалізації списку ігор, що може залежати від їхньої кількості та потреб користувача (список, дерево, їхня комбінація з пріоретизацією на пошуку тощо).

Тут слід звернути увагу на те, що зазначена в полі гри сцена є лише початковою – ігровий рушій дозволяє напряму взаємодіяти сценам між собою, тому після запуску, гра може продовжуватися в декількох сценах навіть одночасно.

Оскільки демонстраційний проєкт не передбачає серверної частини, відсутнє також і збереження даних на сервері. Збереження на стороні клієнта доступне в декількох варіантах.

Стан програми (в пам’яті). Дані зберігаються у змінних JavaScript або бібліотеці керування станом (наприклад, Redux або Vuex) протягом поточного сеансу. Це найшвидший метод, але дані втрачаються, коли користувач залишає сторінку або закриває вкладку/програму.

Файли cookie. Невеликі текстові файли, що зберігаються браузером. Вони надсилаються з кожним HTTP-запитом на сервер, що робить їх придатними для керування сеансами та персоналізації. Вони мають невелике обмеження розміру (зазвичай, 4 КБ) і можуть бути налаштовані на термін дії або збереження.

Вебсховище поділяється на 2 частини. Локальна пам’ять – зберігає дані без терміну дії (зберігається до явного очищення). Має більшу межу (зазвичай 5-10 МБ) і не надсилається з кожним HTTP-запитом. Сховище сеансів – подібне до локальної пам’яті, але дані очищаються, коли вкладка браузера закривається.

IndexedDB. Низькорівневий API для зберігання великих обсягів структурованих даних (включаючи файли/блоби) на клієнті. Це асинхронна система для складних потреб зберігання даних, що пропонує модель бази даних сховища об’єктів.

Веб-SQL (застаріла) – стара, нестандартна технологія, яка намагалася забезпечити SQL-подібний інтерфейс для зберігання на стороні клієнта. Вона більше не рекомендується та зазвичай замінюється IndexedDB [29].

У випадку розроблюваної системи, балансом у функціональності та складності є використання local storage – це досить простий у використанні спосіб, який не потребує чіткої схеми, але при цьому закриття вкладки або

браузеру не очищує дані, що ідеально підходить для дослідження. Для повноцінного та зручного використання цього сховища в коді доцільно створити окремий сервіс.

Цей сервіс, LocalStorageService, є обгорткою над стандартним API window.localStorage браузера. Його основне призначення – забезпечити безпечну та зручну роботу з локальним сховищем. Сервіс робить це шляхом інкапсуляції всіх базових операцій (get, set, remove) у блоки try...catch. Така реалізація гарантує, що застосунок не буде аварійно завершено, якщо localStorage недоступний, наприклад, у режимі приватного перегляду або через переповнення сховища. У разі помилок запис або видалення ігноруються, а читання повертає null. Додатково, сервіс пропонує зручні методи getJSON та setJSON, які автоматично обробляють серіалізацію (перетворення в рядок JSON) та десеріалізацію (парсинг рядка назад в об’єкт). Метод getJSON також підтримує механізм «запасного» значення (fallback), яке повертається, якщо дані відсутні за ключем або якщо збережений рядок не є валідним JSON. Таким чином, сервіс спрощує роботу з об’єктами та забезпечує стійкість до збоїв. Файл експортує одразу готовий до використання єдиний екземпляр сервісу (localStorageService), що відповідає патерну Singleton.

Поверх цієї обгортки тепер можна дописувати власне сервіси для керування конкретними даними. Такий сервіс є високорівневим інтерфейсом для роботи з локальним сховищем додатка. Його ключова роль – відокремити бізнес-логіку (що ми зберігаємо: ім’я користувача, результати ігор) від технічної реалізації (як ми зберігаємо: як рядок, як JSON, як обробляємо помилки).

Він забезпечує прості, орієнтовані на застосунок методи, знаючи, які ключі використовувати («username», «scoreboard_...») і як обробляти цільові дані. Наприклад, метод для імені користувача в цьому сервісі просто читає і пише рядок, а для таблиці результатів (Scoreboard) інший метод автоматично отримує і оновлює масив JSON, забезпечуючи, щоб нові записи додавалися до кінця існуючого списку. Уся ця робота виконується через безпечні методи базового

сервісу, що робить AppStorageService чистим, функціональним і стійким до помилок сховища.

### 3.2 Проєктування потоків операцій

Вдале проєктування ключових процесів роботи системи є важливим для написання ефективного коду. Серед комплексних процесів – запуск обраної користувачем гри та збереження результатів при її завершенні.Розбираючи ініціалізацію гри, для прикладу, можна взяти одну з ігор з обходом дерева «Closest Value Search» (рис. 3.3) [30].

Зображена діаграма має значну кількість операцій, які виконуються між моментом, коли користувач натискає на кнопку «Start game» та моментом, коли гра, власне, запустилася та готова до інтеракції. Ініціалізатором процесу є Phaser Engine, якому передається сцена відповідної гри для відображення «Closest Value Game». Головним рішенням, яке відображає діаграма, є виконання більшості операцій в базовому класі BaseTreeGame, який відповідає за збору до купи всієї інформації, необхідної для гри про обхід дерева, зі своїх залежностей, надаючи всі базові можливості та візуалізацію.

BaseTreeGame спочатку використовує локальну бібліотеку-залежність для генерації бінарного дерева з числовими значеннями в заданому діапазоні, та, що головне, детальних даних щодо візуалізації цього дерева (включаючи всі позиції елементів дерева та зв’язки). Далі, базовий клас використовує іншу залежність для візуалізації на полотні Phaser всіх зв’язків між елементами та вузлів (Node). Потік операцій далі займається ініціалізацією базових елементів інтерфейсу, до яких входить 2 панелі – шапка (містить загальний ігровий стан, деякі операції користувача та поточний рахунок) та бокова панель для логування. Логування в окрему панель на ігровому полотні наразі є головним способом надання користувачу інформації про стан та історію операцій.

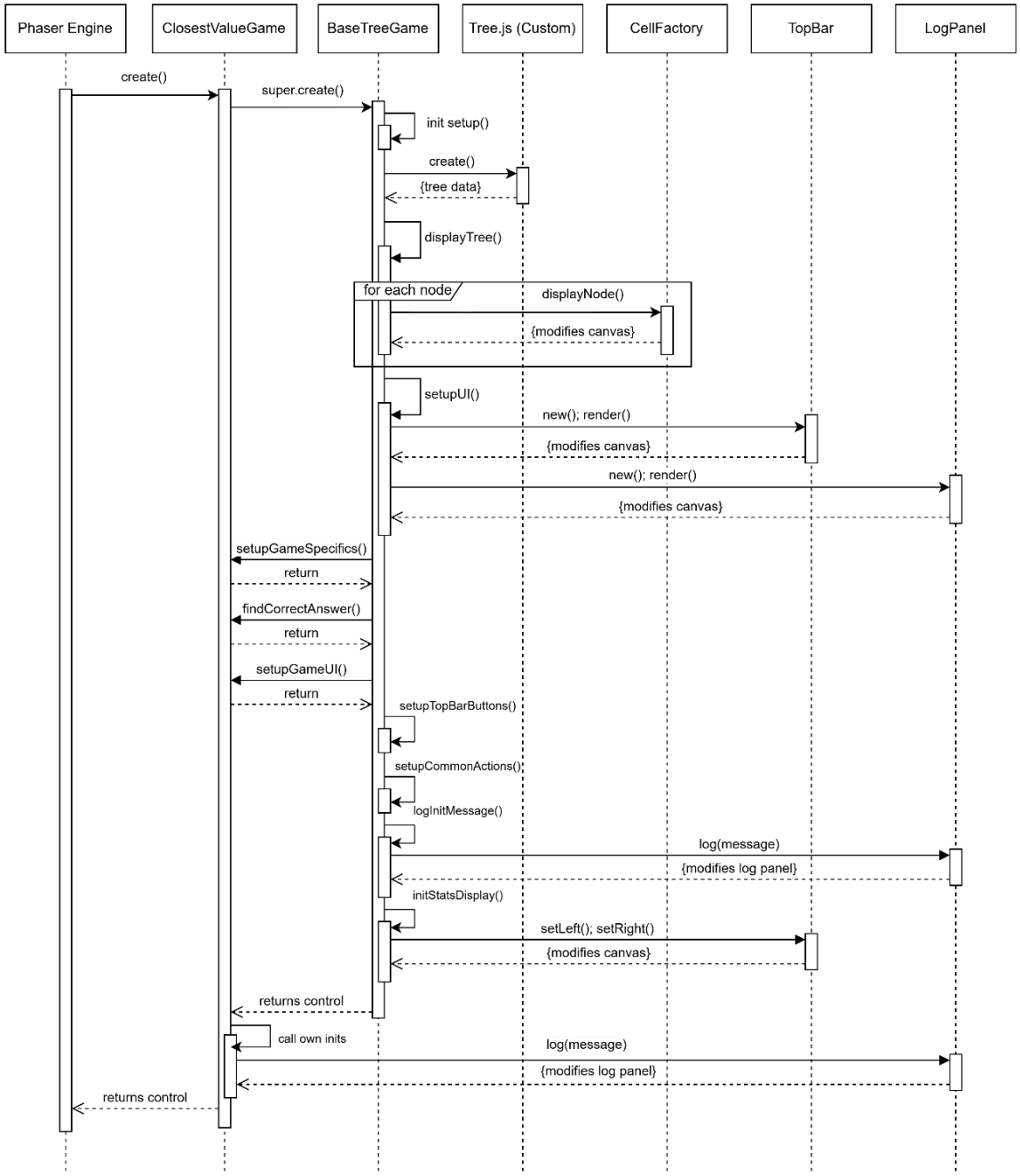



Рисунок 3.3 – Sequence diagram ініціалізації гри

Ініціалізація продовжується викликом віртуальних функцій, які має реалізувати сцена цільової гри, серед яких:

– setupGameSpecifics(): потрібна для задання конкретною грою загальних значень, специфічних для неї;

– findCorrectAnswer(): схожа на попередню, але конкретизує саме зазначення правильної відповіді, яку користувачу потрібно буде зазначити перед тим як успішно завершити гру – залежить від попередньо згенерованого дерева;

– setupGameUI(): на відміну від істановленого раніше загального інтерфейсу, цей метод відповідає за встановлення додаткових елементів, специфічних саме для наслідуваної гри, наприклад, функціонал, назви та позиціонування кнопок чи візуальних елементів інтерфейсу.

Нарешті, інтерфейс закінчує свою ініціалізацію спільними для всіх ігор з обходу бінарного дерева кнопками (наприклад, FINISH, Reset), візуальними елементами та повідомленням в логгер для позначення готовності гри для користувача – останнє виконується окремо як базовим класом (для загального повідомлення про готовність гри), так і наслідуючим (для виведення для користувача початкової інформації про ціль гри). На цьому моменті контроль передається назад Phaser Engine для завершення процесів по ініціалізації.

Серед головних принципів з використаного – поділ відповідальностей. Базовий клас бере на себе всі обов’язки з виконання операцій, спільних для будь-яких ігор з обходу бінарних дерев та спільних елементів інтерфейсу, дозволяючи реалізаціям визачати лише специфіку інтерфейсу та логіки. Це відповідає принципу DRY та сприяє пришвидшенню додавання нових ігор на цій основі.

Реалізація самого ігрового процесу відповідає схемі більшості ігор – цикл, або квітка, зав’язана на інтеракції користувача – поки користувач нічого не робить, гра зазвичай не зазнає важливих змін у стані або їх векторі, тоді як у разі введення користувачем, відбувається запрограмнована реакція. Діаграма активностей (рис. 3.4) демонструє відповідну схему роботи для гри за темою сортування [31].

Центральним елементом є один блок, який очікує дії користувача. В даному випадку він зображений як Guard (умовний блок), адже за логікою ігрового рушія, гра існує в потоці фреймів, кожен з яких має свій стан. Якщо гравець виконує якусь дію, вона реєструється на певному фреймі – тоді стан наступного фрейму буде відрізнятися від того, який очікувався інакше, вже із врахуванням інформації про дію (або набір дій) користувача. До дій відносяться прості (mouseUp, mouseDown, pressKey тощо) та складні/комбіновані інтеракції

(drag, doubleClick). В залежності від дії користувача, виконується відповідний алгоритм.

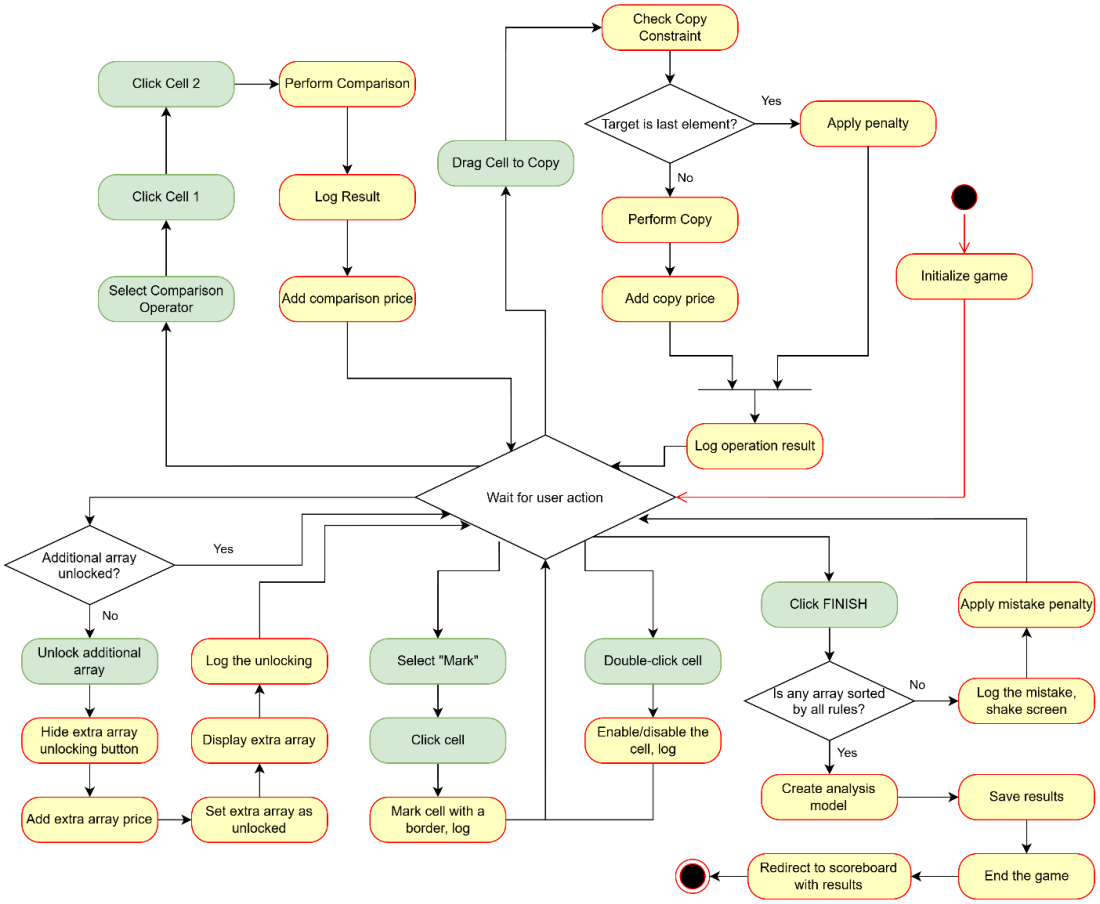



Рисунок 3.4 – Activity diagram для гри з імітацією сортування

Діаграма містить декілька головних субалгоритмів, які відповідають ключовим діям гравця. Зелені активності відображають дії користувача, тоді як жовті – реакцію системи на них.

Перетягування елементу для копіювання – означає drag користувачем однієї комірки пам’яті (зображеного в грі квадратом та індексом елементу) на інший та відпускання миші. Поверхнево, ця дія має результуватися у перенесенні даних однієї комірки в іншу, але особливість гри додає важливе обмеження – гравець не може змінювати стан елементу, якщо той є останнім екземпляром індексу з початкового масиву на полі. Це додає додатковий Guard (перевірку) для визначення відповідності дії вимогам, і виконує копіювання з невеликою ціною або ж просто назначає штрафні бали в залежності від результату перевірки. У

будь-якому разі, результат операції користувача логується в панель для інформування гравця про те, що відбулося «під капотом».

Операція порівняння включає 3 дії від користувача – вибір операції та двох комірок для порівняння кліками по них. В момент отримання обох операндів, система виконує порівняння та виводить його на панель, додаючи ціну операції до рахунку. До схожих операцій можна віднести маркування комірок або їх блокування/розблокування – останнє вже є прикладом використання подвійного натискання як тригеру операції. Всередині такої активності насправді є набагато більше нюансів стосовно того, що саме вважати успішним «дабл-кліком», але такий рівень деталізації не передбачається у зображеній діаграмі, адже є надмірним, але може бути корисним при низькорівневому проєктуванні функціональності.

Інший цікавий випадок – розблокування додаткової пам’яті користувачем. Ця операція може бути виконаною лише раз на ігрову сесію, що потрібно враховувати. В цьому випадку, дія користувача «Unlock additional array» знаходиться після Guard «Additional array unlocked?» – це зумовлено тим, що по ігровій логіці, тригер відкриття додаткового масиву пам’яті у вигляді кнопки поверх простору, що ця пам’ять займає, має бути прихованим, якщо операція вже була виконана. Іншими словами, після першого виклику операції, другий є для користувача недоступним. У іншому ж випадку, активація тригеру виконує послідовність з приховання кнопки, врахування ціни масиву, позначення всередині сцени додаткового масиву як розблокованого, додаткових операції для відображення масиву та відображення логу в панелі.

Єдина активність користувача, яка може призвести до завершення ігрового процесу (end state) – натискання на кнопку FINISH. Тут, відповідно ігровій логіці, гра не закінчується, доки всі умови для цього не були виконані – для гри про сортування, це наявність хоча б одного масиву, який є кінцевою відповіддю до задачі коректного сортування початкового масиву. Цей масив може бути як той, в якому був початковий невідсортований масив, так і масив додаткової

пам’яті. Сортування вважається коректним, якщо кінцевий масив відсортований за вказаним порядком та містить всі елементи з початкового масиву.

У випадку помилкової відповіді, користувач отримує відповідне повідомлення, яке може також бути супроводжене спеціальними візуальними ефектами (screen shake). Інакше, поточний стан ігрової сцени використовується для формування аналітичного звіту та об’єкту для збереження у таблицю лідерів, після чого гра закінчується, а застосунок переходить на сторінку результатів, що можна вважати кінцевим станом ігрової сцени.

### 3.3 Особливі проєктні рішення

Розроблювана інформаційна система має декілька компонентів або їх частин, які заслуговують особливої уваги в процесі проєктування. Одним із цікавих проєктних рішень є додатковий абстракційний шар понад сценою Phaser – в проєкті це називається ModifiedPhaserScene. Назва досить загальна, адже цей шар абстракції актуальний для кожної гри в системі, і саме його мають наслідувати базові або відповідні сцени, які представляють конкретну гру.

Створення та використання ModifiedPhaserScene є рішенням, до якого підштовхнула потреба додавання однакового функціоналу до кожної гри в системі. Найпростішим прикладом є потрушування екрану – це візуальний ефект, який доступний для камери як об’єкта Phaser, та може бути використаний як сповіщення користувача про неправильні дії або інші можливі внутрішньоігрові загрози чи інформацію. Незважаючи на те, що сам інтерфейс використання «шейку» не є складним, він все рівно потребує певної параметризації та звернення до вкладених властивостей сцени – ці особливості не будуть сильно враховуватися більшістю ігор, до того ж розробник конкретної сцени для відповідності загальній картині повинен більше часу приділити для правильного використання ефекту. Натомість один узагальнюючий метод із параметрами за замовчуванням покращить описану ситуацію, надаючи в якості

інтерфейсу лише назву методу, стандартизуючи параметри для загального використання, але залишаючи можливості для кастомізації:

```
protected shakeScreen(durationMs: number = 100, intensity: number = 0.01): void {
    this.cameras.main.shake(durationMs, intensity);
}
```

Слідуючи прикладу, можна додати і більш складні деталі базової реалізації сцени. Наступним для розгляду буде колір фону для сцени. Ця особливість раніше згадувалася як поле класу гри, і потрібна як спосіб кастомізації ігор, для додаткового візуального вирізнення їх на фоні інших. Оскільки це планується як функціональність, притаманна всім або більшості ігор, її доцільно додати сюди як частину потоку ініціалізації ігор, враховуючи те, що фон може бути кольором, градієнтом або навіть зображенням.

Нарешті, кожна гра у разі завершення має виконувати певний набір операцій:

– форматування результатів гри для відображення;

– збереження об’єкту для таблиці лідерів із врахуванням конфігурованих полів;

– виклик базового методу для завершення гри.

Це також можна об’єднати та винести на зовнішній рівень в ModifiedPhaserScene, тоді як в самих ігрових сценах достатньо буде викликати цей метод в момент, коли гру можна вважати завершеною (наприклад, натиснута кнопка FINISH із усіма виконаними обов’язковими умовами гри).

Також потрібно зазначити, що JavaScript/TypeScript не мають за замовчуванням багатьох відомих структур даних, у тому числі стеку та черги, тому їх реалізації (які також можуть бути розширеними за потреби ігор, які їх використовуватимуть) створюються додатково. Реалізації потрібних структур із використанням вбудованого масиву JavaScript та інших вбудованих можливостей винесено в окремі класи. Головна особливість цих реалізацій – це їх універсальність через узагальненість (генеративність, Generic Type) класів, що дозволяє використовувати їх для будь-якого типу даних.

Інформаційна система включає в себе реалізовану функціональність для управління «реактивними змінними». Цей механізм, що діє за патерном «Спостерігач», являє собою спеціалізований контейнер даних. Його ключовою особливістю є те, що будь-яка операція запису нового значення автоматично та миттєво ініціює виклик попередньо визначеної процедури-спостерігача (callback). У контексті ігрової логіки, у межах сцени Phaser (Phaser Scene), ця функціональність дозволяє створювати прямий та ефективний зв’язок між ігровим станом та його візуальним представленням. Наприклад, змінна, що зберігає інформацію про певну комірку пам’яті, вузол тощо, що може включати багато як відображуваних, так і прихованих від користувача даних, може бути пов’язана з текстовим об’єктом (Phaser.GameObjects.Text). При кожному оновленні рахунку процедура-спостерігач автоматично оновить вміст цього текстового елемента без необхідності додаткових перевірок у головному ігровому циклі (update loop).

Цей принцип реактивності розширено також на структури даних типу колекцій через кастомну (оригінально розроблену) функціональність «реактивного масиву». Ця система забезпечує гранулярне відстеження змін на рівні окремих елементів колекції. Внутрішньо, кожен елемент, що зберігається в такому масиві, інкапсулюється у власний «реактивний контейнер», описаний вище. Такий підхід є критично важливим для оптимізації рендерингу в Phaser. Замість того, щоб повністю перемальовувати весь список UI-елементів (в даному випадку, масиви або дерева) при зміні одного елемента, система дозволяє точково оновити лише той візуальний компонент, який пов’язаний зі зміненим елементом даних. Процедура-спостерігач для масиву отримує не лише нове значення, але й індекс елемента, що дозволяє точно ідентифікувати джерело зміни.

Для забезпечення повної цілісності реактивної системи при зміні самої структури масиву (додавання чи видалення елементів), реалізовано механізм «Проксі». Ця підсистема перехоплює стандартні операції над масивом, такі як

push (додавання) або splice (заміна/видалення). Коли до колекції додаються нові дані, механізм «Проксі» гарантує, що вони автоматично «обгортаються» у реактивні контейнери, перш ніж потрапити до сховища. Це забезпечує, що навіть динамічно додані елементи в Phaser Scene (у тому числі й у відображувані структури даних) негайно стають частиною реактивної системи та коректно зв’язуються з логікою оновлення інтерфейсу.

Архітектура створення ігор в системі базується на компонентному підході до побудови користувацького інтерфейсу, де складні візуальні елементи інкапсулюються в окремі, перевикористовувані класи. Таке рішення сприяє відокремленню відповідальності (Separation of Concerns), дозволяючи основній логіці ігрової сцени Phaser зосереджуватися на управлінні станом, а не на деталях рендерингу чи обробки взаємодії з UI. Усі компоненти отримують посилання на активну сцену Phaser (Phaser.Scene) та використовують її фабричні методи (scene.add) для безпосереднього додавання ігрових об’єктів до списку відображення.

Система надає стандартизовані композитні компоненти, такі як TopBar (верхня панель) та LogPanel (панель логування), використання яких також описано раніше в розділі 3.1. TopBar функціонує як основний елемент HUD (Heads-Up Display), пропонуючи зарезервовані області для текстової інформації (зліва та справа), а також включаючи динамічний фабричний метод (addButtons) для генерації інтерактивних кнопок. Ця підсистема кнопок автономно керує власним станом, включаючи візуальні ефекти при наведенні та логіку «вибору» (highlight), таким чином абстрагуючи цю складність від сцени. LogPanel, у свою чергу, є спеціалізованим UI-елементом для виведення діагностичних або ігрових повідомлень і часто розраховується як ключове місце інформування користувача про стан ігрового поля або виконаної операції. Архітектурною особливістю цього компоненту є реалізація власного механізму компонування тексту: він динамічно розраховує ширину символів для виконання ручного перенесення слів по рядках (word-wrapping) і керує буфером повідомлень фіксованого розміру,

гарантуючи, що відображаються лише найновіші записи, тоді як старіші стираються до того, як вони вилізуть за визначений для панелі простір (space overflow).

Для управління створенням складних ігрових сутностей, що повторюються, система використовує патерн «Фабрика» (Factory Pattern), реалізований у класі CellFactory. Єдиною відповідальністю цього компонента є програмна інстанціація та конфігурація «комірок» – часто використовуваних для відображення одиниці пам’яті, вузла в дереві тощо. Комірка визначається як композитний об’єкт (Phaser.GameObjects.Container), що об’єднує в собі набір графічних примітивів: базовий прямокутник, текстову мітку та графіку для підсвічування. Цей підхід декапсулює логіку сцени від деталей реалізації комірки; сцена просто запитує новий об’єкт, тоді як фабрика бере на себе повну відповідальність за його коректне створення, стилізацію та налаштування інтерактивності, включаючи конфігурацію для перетягування (draggable) та прийому (dropZone) – це корисно для реалізації копіювання значень в грі про сортування через перетягування.

Окрім цього, окремо винесено клас DataStructureVisualizer. Цей компонент діє як спеціалізоване «Подання» (View) у рамках архітектури MVC, призначене для візуалізації поточного стану абстрактної структури даних, наприклад, стеку чи черги. Його архітектура надає мінімалістичний публічний API (метод update), який приймає єдине значення (зазвичай результат операції «peek») і відображає його. Компонент також самостійно керує власними візуальними станами, такими як «активний» (setActive) або «вимкнений» (setDisabled), змінюючи стилі (колір рамки, прозорість), що дозволяє керуючій логіці сцени візуально позначати потік даних між різними структурами. Цей візуальний компонент використано, наприклад, в грі для обходу бінарних дерев для позначення обраної користувачем структури даних для організації алгоритму обходу. Основна логіка структури даних при цьому належить описаному раніше класу, який відповідає за конкретну структуру даних, у той час як цей візуалайзер слугує лише

візуальним поданням структури користувачу із врахуванням обмежень з точки зору апаратного забезпечення (наприклад, обчислювальний комплекс може бачити лише останній доданий елемент стеку, незалежно від їх кількості).

Розроблювана інформаційна система також включає в себе набір спеціалізованих компонентів, що є важливою частиною для реалізації ігрових застосунків, наприклад, для ігор орієнтованих на симуляцію обходу бінарних дерев. В основі цього лежить гібридна модель даних, де ключова сутність (TreeNode) інкапсулює як логічну, так і візуальну інформацію. Такий підхід є фундаментальним: кожен вузол дерева зберігає не лише свої реляційні зв’язки (посилання на батьків та нащадків) і значення, але й заздалегідь обчислені візуальні атрибути, зокрема (x, y) координати та depth (глибину в дереві). Це дозволяє ігровій сцені Phaser виступати в ролі «тонкого клієнта» (thin client) рендерингу, що просто відображає дані, замість того, щоб нести відповідальність за складні обчислення макета.

Для забезпечення варіативності ігрових рівнів система надає потужний сервіс процедурної генерації (generateRandomTree). Це керована фабрика, що надає архітектурні гарантії, критичні для ігрового процесу. Система дозволяє гнучко керувати параметрами генерації, такими як максимальна глибина, ймовірність появи нащадків (що впливає на «густоту» дерева) та діапазон значень. Важливою можливістю є примусове забезпечення унікальності значень вузлів, що може бути фундаментальною вимогою для ігрових режимів, пов’язаних із пошуком. Система також включає захисні механізми: вона перевіряє, чи можлива унікальність при заданому діапазоні значень, та використовує ітеративний підхід «генерації та перевірки», аби гарантувати, що згенероване дерево відповідає заданим обмеженням мінімального та максимального розміру, відкидаючи невідповідні варіанти. В цьому контексті, для певної гри умова максимальної та мінімальної густоти дерева надає варіативності (недолік постійно повного дерева), але забезпечує однакову

складність – відсутність обмежень може під час випадкової генерації призвести до майже порожнього дерева.

Найскладнішим архітектурним компонентом у цьому наборі є алгоритм візуального компонування (assignPositions), який перетворює абстрактну згенеровану структуру на естетичну та вільну від перетинів 2D-схему. Система використовує для цього надійний багатопрохідний алгоритм, що базується на post-order (зворотному) обході. Замість примітивного розміщення «згори-вниз», він спочатку рекурсивно обчислює відносні позиції для найнижчих піддерев (листків), а потім піднімається вгору, центручи батьківські вузли над їхніми нащадками. Його ключова перевага – це активне уникнення колізій: під час об’єднання лівого та правого піддерев, алгоритм аналізує їхні «контури» (крайні ліві та праві координати). Якщо піддерева розташовані занадто близько або перетинаються, система інтелектуально зміщує все праве піддерево вбік, гарантуючи мінімально необхідний простір, перш ніж розмістити над ними батьківський вузол. Метод також враховує особливу деталь для ігрового процесу обходу бінарного дерева – якщо вузол має лише одного нащадка, він буде чітко лівим або правим та буде відображатися відповідно, а не посередині під батьківським вузлом – при цьому порожній простір відсутнього вузла все ще може бути використаний сусідніми піддеревами.

На завершальному етапі, після того, як вся відносна сітка координат дерева розрахована, система виконує фінальне масштабування та центрування. Вона аналізує загальну ширину та висоту отриманого макета і застосовує афінне перетворення (масштабування та зсув) до кожного вузла. Це гарантує, що незалежно від згенерованої форми (чи є дерево широким, вузьким, глибоким або незбалансованим), воно завжди буде акуратно вписане у надані розміри ігрового полотна (canvas) з урахуванням візуальних відступів. Такий підхід та увага до деталей покращує естетичний елемент гри та полегшує розуміння ігрової ситуації, що впливає на загальний UX.

## Висновки до розділу 3

У третьому розділі кваліфікаційної роботи описано деталі та особливості проєктування інформаційної системи для гейміфікації навчального процесу. Тут особливу увагу приділено проєктуванню функціональності, організації даних, потоків операцій, а також особливих архітектурних та розроблених програмних рішень.

Складено діаграму використання, шо надає уявлення про функціональність системи та зв’язки між окремими її елементами, обґрунтовано залежності. Описано особливості зберігання даних у розроблюваній системі на прикладі ігор – наведено діаграму класів для дерева ігор, як способу зберігання та подавання їх користувачу. Описано технічні деталі та архітектурні рішення, які стали підгрунтямм для виконання завдання зі збереження даних.

На прикладі ініціалізації ігрового процесу описано способи реалізації комплексних низько- та середньорівневих операцій. Тут до уваги взято принципи чистого коду та оптимальне використання архітектурних патернів. Усе це відображено на діаграмі послідовностей дла зазначеного процесу. Також, за допомогою діаграми активностей надано опис того, як функціонує більшість ігрових застосунків у системі на прикладі гри з імітації сортування.

Окрім цього, приділено особливу увагу власним архітектурним та програмним рішенням, серед яких:

- базовий рівень абстрації над ігровою сценою Phaser;
- реактивні змінні та масиви;
- шаблони для елементів UI та допоміжні методи;
- спеціалізовані компоненти на прикладі Tree.

Опис зазначених рішень супроводжено важливими деталями реалізації, описом потреби та способів використання, важливості для системи та її розширення, посиланнями на архітектурні паттерни.

# 4 КОДУВАННЯ, ТЕСТУВАННЯ, АПРОБАЦІЯ ТА КЕРІВНИЦТВО КОРИСТУВАЧА

## 4.1 Кодування ПЗ

Оскільки вихідний програмний код досить обширний, особливо включаючи елементи UI, доцільно навести лише деякі ключові високорівневі елементи коду, які слідують загальній концепції та виконують важливу задачу відносно системи чи її частини.

Для прикладу, частина коду, яка відповідає за перевірку коректності сортування:

```
    private hasAllOriginalElements(checkedArr: ReactiveVariable<Variable>[]):
boolean {
        const n = this.array.array.length;
        const checkedIndices = new Set(checkedArr.map(v => v.value.index));
        for (let i = 0; i < n; i++) {
            if (!checkedIndices.has(i)) return false;
        }
        return true;
    }

    private isArraySorted(arr: ReactiveVariable<Variable>[]): boolean {
        let prev: number | null = null;
        let count = 0;
        for (const v of arr) {
            const curr = v.value.value;
            if (curr === null || (prev !== null && prev > curr)) return false;
            prev = curr;
            count++;
        }
        return count === this.array.array.length;
    }

    private isSortedAndComplete (arr: ReactiveVariable<Variable>[]): boolean {
        return this.isArraySorted(arr) && this.hasAllOriginalElements(arr);
    }
    private isGoalArchived(): boolean {
        return this.isSortedAndComplete(this.array.array) ||
this.isSortedAndComplete(this.tempArray.array);
    }
```

Розбираючи по методах підряд, hasAllOriginalElements() відповідає за перевірку масиву на те, що він містить всі елементи початкового масиву за

індексом. Це важлива перевірка, адже вона виключає можливість «махлювання» шляхом задання як відповіді відсортованого, але некоректного масиву (наприклад, з усіма однаковими значеннями). Це одна з ключових вимог до гри про імітацію сортування. Цей метод створює нову множину з індексів вхідного масиву та перевіряє, чи всі індекси в очікуваному діапазоні наявні в цій множині.

Наступний метод, isArraySorted(), відповідає за те, щоб наданий масив був повний та відсортований. За замовчуванням, порядок сортування є неспадаючим, тому при знаходженні пари, яка порушує умову, метод відразу повертає негативний результат – це також стосується невалідних значень. Перевірки на порожні значення дуже актуальні, враховуючи те, що, наприклад, додаткова пам’ять може містити в результаті порожні комірки, але її також потрібно брати до уваги.

Метод isSortedAndComplete() об’єднує результати двох попередніх методів для одного масиву, повертаючи як результат, чи є масив коректною відповіддю. Нарешті, isGoalArchived() викликає isSortedAndComplete() для обох доступних користувачу масивів, і якщо хоча б один з них є валідною відповіддю – ціль гри вважається досягненою.

Сервіси в інформаційній системі містять певний рівень бізнес логіки конкретного потоку даних або операції – код виконано із врахуванням потреб для розширення та перевикористання, в шаровій архітектурі. Для прикладу, сервіс AppStorageService є інтерфейсом для керування основними і необхідними для застосунку даними, які зберігаються в localStorage(). Завдяки такому підходу, у разі заміни способу збереження даних з локального сховища на інше, клієнтський код зазнає мінімальних змін, адже достатньо лише реалізувати новий сервіс із ідентичним інтерфейсом:

```
class AppStorageService {
    getUsername(): string {
        return localStorageService.get(USERNAME_KEY) || '';
    }

    setUsername(username: string): void {
        localStorageService.set(USERNAME_KEY, username);
```

```
    }

    getScoreboard(gameId?: string): unknown[] {
        const key = `scoreboard_{gameId || 'unknown'}`;
        return localStorageService.getJSON<unknown[]>(key, []);
    }

    appendScoreboard(result: unknown, gameId?: string): void {
        const key = `scoreboard_{gameId || 'unknown'}`;
        const existing = this.getScoreboard(gameId);
        const next = [...existing, result];
        localStorageService.setJSON(key, next);
    }
}
```

В цьому випадку, сервіс містить достатній мінімум для потреб застосунку – всі дані, які потребують збереження це ім'я користувача та таблиці лідерів для ігор. Серед цікавих рішень – це динамічні ключі для ігор: оскільки кожна гра має своє ID в системі, для їх ефективного зберігання ключі формуються з цих ідентифікаторів. В кінцевому результаті, це виглядає так, як зображено на рис. 4.1.

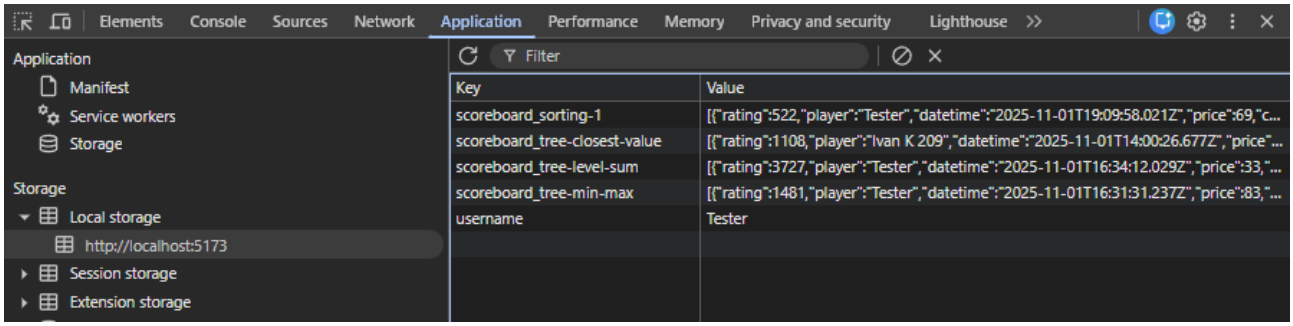



Рисунок 4.1 – Збереження даних системи в локальному сховищі

Реалізація згаданого у попередньому розділі представлення структур даних є здебільшого уніфікованим представленням UI в коді:

```
export class DataStructureVisualizer {
    private title: Phaser.GameObjects.Text;
    private itemText: Phaser.GameObjects.Text;
    private border: Phaser.GameObjects.Rectangle;

    constructor(scene: Phaser.Scene, x: number, y: number, width: number, height:
number, title: string) {
        this.border = scene.add.rectangle(x, y, width, height, 0x000000, 0.2)
            .setOrigin(0, 0)
            .setStrokeStyle(2, 0xffffff, 0.7);
```

```
        this.title = scene.add.text(x + width / 2, y + 15, title, {
            fontFamily: 'Arial',
            fontSize: '16px',
            color: '#ffffff',
        }).setOrigin(0.5);

        this.itemText = scene.add.text(x + width / 2, y + height / 3 * 2, 'EMPTY',
{
            fontFamily: 'monospace',
            fontSize: '20px',
            color: '#ffffff',
        }).setOrigin(0.5);
    }

    update(peekValue: number | string | null | undefined): void {
        if (peekValue !== null && peekValue !== undefined) {
            this.itemText.setText(peekValue.toString());
        } else {
            this.itemText.setText('EMPTY');
        }
    }

    setActive(isActive: boolean): void {
        const color = isActive ? 0xffff00 : 0xffffff;
        const alpha = isActive ? 1 : 0.7;
        this.border.setStrokeStyle(isActive ? 3 : 2, color, alpha);
        this.title.setColor(isActive ? '#ffff00' : '#ffffff');
    }

    setDisabled(isDisabled: boolean): void {
        const alpha = isDisabled ? 0.3 : 1;
        this.border.setAlpha(alpha);
        this.title.setAlpha(alpha);
        this.itemText.setAlpha(alpha);
    }
}
```

Клас інкапсулює декілька ігрових об’єктів Phaser: Rectangle для тла або рамки (border), а також два об’єкти Text для заголовка (title) та для відображення поточного елемента (itemText). Така інкапсуляція дозволяє керувати візуалізатором як єдиним цілим.

При створенні екземпляра класу в конструкторі, він негайно додає всі необхідні графічні елементи на ігрову сцену (scene.add). Використання методів setOrigin(0, 0) для прямокутника та setOrigin(0.5) для тексту є типовою практикою в Phaser для точного позиціонування та центрування. Прямокутник

створюється з напівпрозорим фоном та контуром, а текстове поле для даних за замовчуванням показує «EMPTY», вказуючи на те, що структура даних порожня.

Ключова функціональність реалізована в методі update. Він приймає один аргумент, peekValue, який, є верхнім елементом стека або першим елементом черги (отриманим через peek()). Метод перевіряє, чи не є це значення null або undefined; якщо значення існує, воно конвертується в рядок і відображається в itemText. В іншому випадку, текст повертається до «EMPTY». Це дозволяє абстрагувати візуалізатор від конкретної реалізації структури даних.

Крім того, клас надає два методи для управління візуальним станом: setActive та setDisabled. Метод setActive змінює колір рамки та заголовка (зазвичай на жовтий для виділення) і збільшує товщину рамки, що корисно для індикації активної структури, з якою взаємодіє користувач. Метод setDisabled, у свою чергу, змінює прозорість (альфа-канал) усіх елементів візуалізатора, «приглушуючи» його. Це ефективний спосіб показати, що структура даних наразі неактивна або недоступна для операцій.

Структура, для прикладу, використана в іграх для обходу бінарного дерева, де користувач сам обирає структуру даних – наведене представлення чудово підходить для обох можливих структур (рис. 4.2).

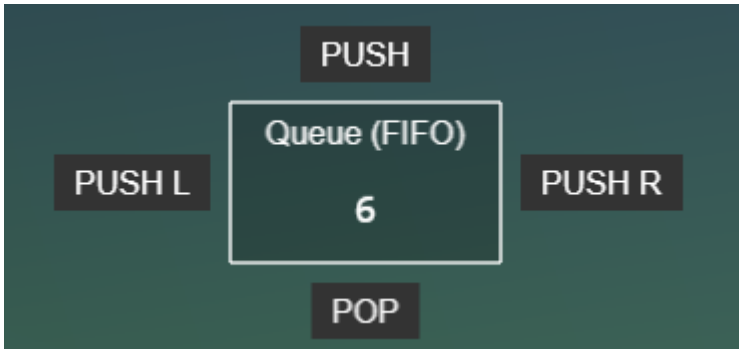



Рисунок 4.2 – Представлення структури даних

До особливостей реалізації можна віднести кастомне представлення розмітки для інструкцій до ігор. Початково, інструкція відноситься до конфігурабельних файлів, тобто, їх можна змінювати під час роботи системи, завантажувати до них різні матеріали тощо. Для коректного та гарного

відображення, схожого до .md формату, створено власний парсер, включно з переформатуванням розмітки в контент як наведено нижче:

```
    const renderMarkdown = (content: string) => {
        // Simple markdown to HTML conversion
        let html = content
            // Headers
            .replace(/^### (.*)/gim, '<h3 class="text-xl font-bold text-yellow-400
mb-0">1</h3>')
            .replace(/^## (.*)/gim, '<h2 class="text-2xl font-bold text-yellow-400
mb-1">1</h2>')
            .replace(/^# (.*)/gim, '<h1 class="text-3xl font-bold text-yellow-400
mb-1">1</h1>')

            // Bold text
            .replace(/\*\*(.*?)\*\*/g, '<strong class="text-orange-
400">1</strong>')

            // Images with {filename} syntax
            .replace(/\{([^}]+\.(png|jpg|jpeg|gif|svg))}/g, (_, filename) => {
                return `<img src="{baseUrl}{filename}" alt="{filename}" class="mx-
auto rounded-lg shadow-lg block" />`;
            })

            // Code blocks
            .replace(/```([\s\S]*?)```/g, '<pre class="bg-gray-800 p-4 rounded-lg
my-4 overflow-x-auto"><code class="text-green-400">1</code></pre>')

            // Inline code
            .replace(/`([^`]+)`/g, '<code class="bg-gray-800 text-green-400 px-2
py-1 rounded">1</code>')

            // Lists
            .replace(/(?:^|\n)\* (.*)/gim, '<li class="ml-8 mb-1">1</li>')
            .replace(/(?:^|\n)- (.*)/gim, '<li class="ml-8">1</li>')

            // Line breaks
            .replace(/\n\n/g, '</p><p class="mb-1">')
            .replace(/\n/g, '<br />');

        // Wrap in paragraphs
        html = '<p class="mb-4">' + html + '</p>';

        return { __html: html };
    };
```

Пропускаючи контент інструкції через цей метод, на виході отримується гарно відформатована розмітка із різними типами форматування тексту та зображення (рис. 4.3). Головна перевага використання такого підходу над спеціалізованими бібліотеками та рішеннями – простота та можливість повного

налаштування задля вписування вихідного результату в загальний дизайн системи.

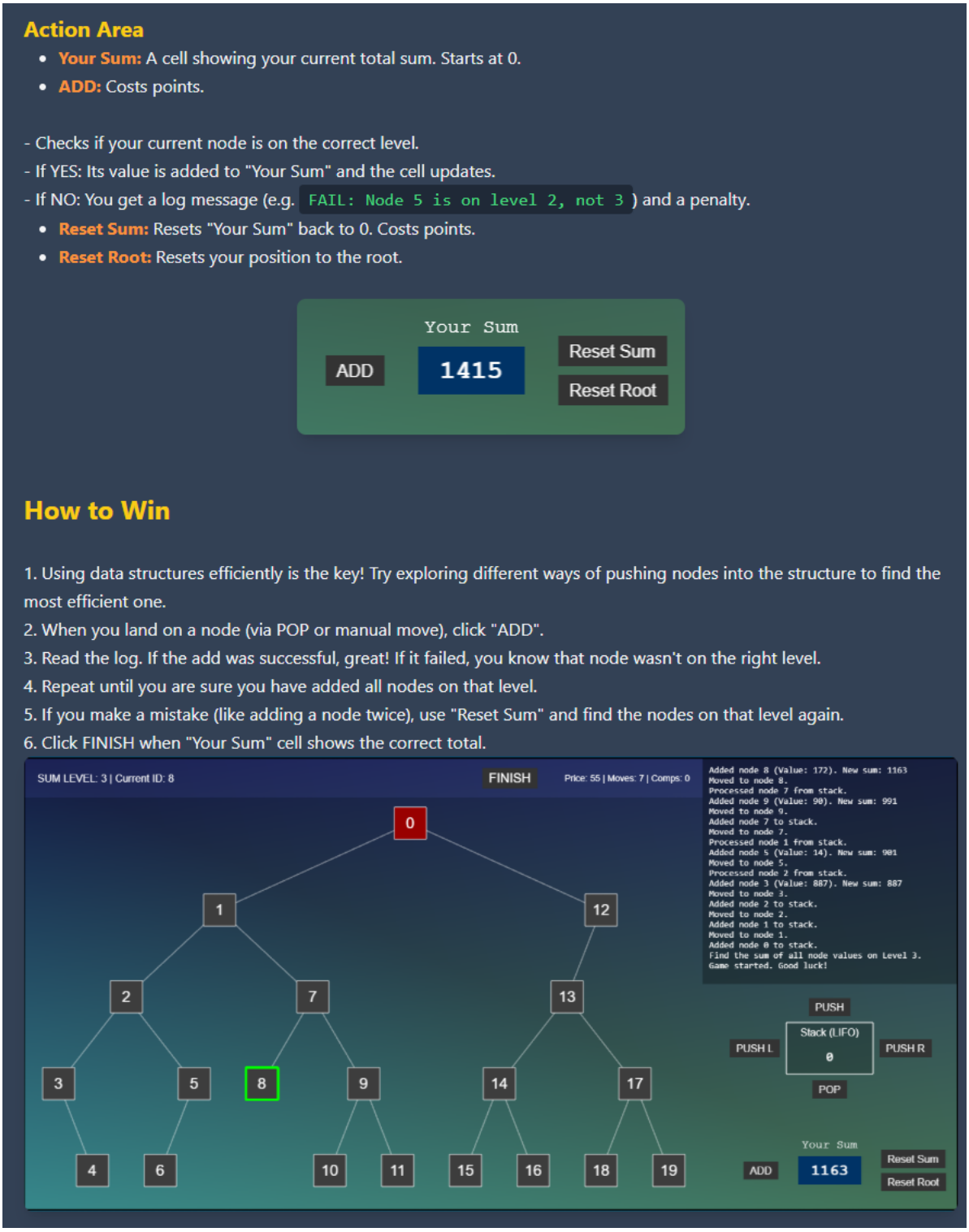



Рисунок 4.3 – Частина інструкції до гри по знаходженню суми вузлів дерева

Нарешті, код файлу Tree.js є досить обширним – найбільшу частину займає параметризація, перевірка на похибки та візуалізація дерева. Однією з основних

частин модулю є власне генерація дерева – код направлений на підвищення різноманіття ігрового процесу шляхом його структуризованої випадковості:

```
function createNode(depth: number, minVal: number, maxVal: number, uniqueValues:
boolean): TreeNode {
    let value;
    if (uniqueValues) {
        do {
            value = Phaser.Math.Between(minVal, maxVal);
        } while (usedValues.has(value));
        usedValues.add(value);
    } else {
        value = Phaser.Math.Between(minVal, maxVal);
    }
    return new TreeNode(NodeIdCounter++, value, depth);
}

function generateSubtree(parent: TreeNode, maxDepth: number, minVal: number,
maxVal: number, childProbability: number, uniqueValues: boolean,
preventBarrenInnerBranches: boolean = false): void {
    if (parent.depth >= maxDepth) {
        return;
    }
    if (Math.random() < childProbability) {
        const leftChild = createNode(parent.depth + 1, minVal, maxVal,
uniqueValues);
        parent.left = leftChild;
        leftChild.parent = parent;
        generateSubtree(leftChild, maxDepth, minVal, maxVal, childProbability,
uniqueValues);
    }
    if (Math.random() < childProbability) {
        const rightChild = createNode(parent.depth + 1, minVal, maxVal,
uniqueValues);
        parent.right = rightChild;
        rightChild.parent = parent;
        generateSubtree(rightChild, maxDepth, minVal, maxVal, childProbability,
uniqueValues);
    }
    if (preventBarrenInnerBranches && !parent.left && !parent.right &&
parent.depth < maxDepth - 1) {
        const child = createNode(parent.depth + 1, minVal, maxVal, uniqueValues);
        if (Math.random() > 0.5) {
            parent.left = child;
        } else {
            parent.right = child;
        }
        child.parent = parent;
        generateSubtree(child, maxDepth, minVal, maxVal, childProbability,
uniqueValues);
    }
}
```

Представлений код деталізує дві ключові функції, createNode та generateSubtree, які складають ядро механізму процедурної генерації дерева. Функція createNode є, по суті, «фабричним методом» для створення окремих вузлів. Вона інкапсулює логіку генерації значення вузла, використовуючи Phaser.Math.Between для вибору випадкового числа у заданому діапазоні. Важливим параметром є uniqueValues, який, у разі активації, змушує функцію перевіряти згенероване значення на унікальність у зовнішній колекції/множині usedValues. Це гарантує, що кожен вузол у дереві матиме унікальне значення, що може бути критичним для певних алгоритмів обходу або пошуку, які будуть застосовані до цього дерева пізніше в ігровому процесі (це також може залежати від цілей гри – саме тому параметризація важлива).

Центральну роль у процесі відіграє функція generateSubtree, яка є класичною реалізацією рекурсивного алгоритму для побудови бінарного дерева. Рекурсія в програмуванні – це техніка, за якої функція викликає саму себе для розв’язання менших підзадач. У цьому випадку, «велика задача» – це побудувати дерево від заданого parent, а «менші підзадачі» – це побудувати ліве та праве піддерева для цього вузла. Алгоритм має чіткий базовий випадок (умова припинення рекурсії): if (parent.depth >= maxDepth). Це запобігає нескінченному зростанню дерева та гарантує, що його глибина не перевищить заданого ліміту.

Стохастичний (випадковий) характер генерації досягається за допомогою параметра childProbability. Для кожного потенційного нащадка (лівого та правого) відбувається перевірка Math.random() < childProbability. Це дозволяє гнучко контролювати топологію дерева: низька ймовірність призведе до «рідкісних», витягнутих дерев, тоді як висока – до густих, кущистих структур. Це і є «структурована випадковість»: процес не суто хаотичний, а підпорядковується чітким імовірнісним правилам.

Особливий інтерес становить параметр preventBarrenInnerBranches. Він втручається в процес, якщо вузол (який ще не досяг максимальної глибини) випадково не отримав жодного нащадка. Ця логіка примусово створює хоча б

одного нащадка для таких «безплідних» внутрішніх гілок, запобігаючи передчасному обриву гілки. Таким чином, код поєднує фундаментальні концепції комп'ютерних наук – рекурсивне визначення та маніпуляцію деревною структурою даних – з імовірнісними методами для створення динамічного та варіативного контенту.

### 4.2 Керівництво користувача

Стартова сторінка вебзастосунку складється з поля для встановлення/зміни імені користувача в системі та списку ігор в особливому візуальному представленні дерева (рис. 4.4).

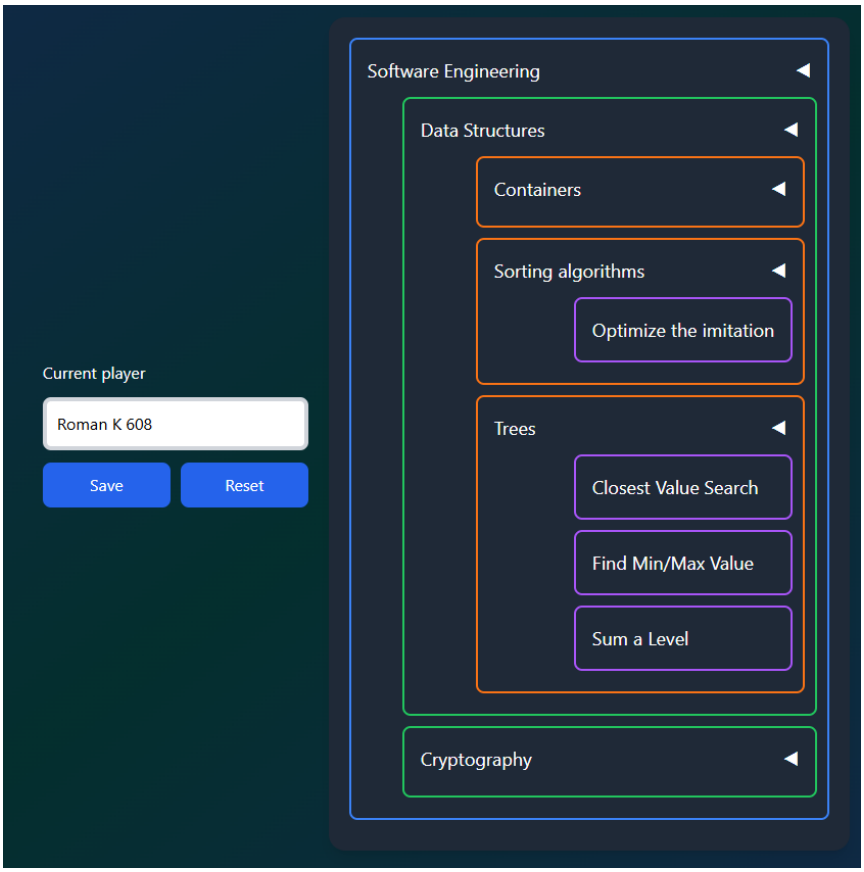



Рисунок 4.4 – Стартова сторінка застосунку

Для зміни імені користувача після введення бажаного потрібно натиснути на кнопку Save – поле буде виділене зеленим кольором. Поки змінене значення поля не буде збережено, поле буде позначене жовтим кольором. Кнопка Reset дозволяє повернути ім'я користувача до останнього збереженого.

Дерево ігор у наданому представленні надає можливість згортати та розгортати кожен вузол окремо. Натискання на певну гру (листя) перенаправляє користувача в меню, пов’язане саме з цією грою (рис. 4.5).

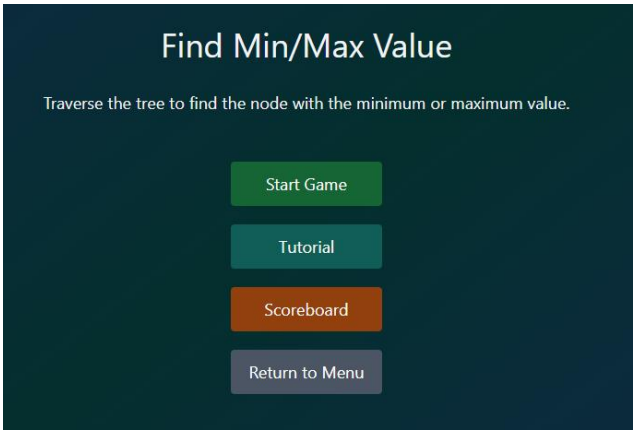



Рисунок 4.5 – Меню гри

Меню дозволяє почати гру, перейти на сторінку інструкції до гри, списку лідерів або повернутися на головне меню зі списком ігор. Інструкція є окремою сторінкою та містить опис до гри, створений в спосіб, описаний в розділі 4.1. Інструкція також дозволяє напряму запустити гру (рис. 4.6).

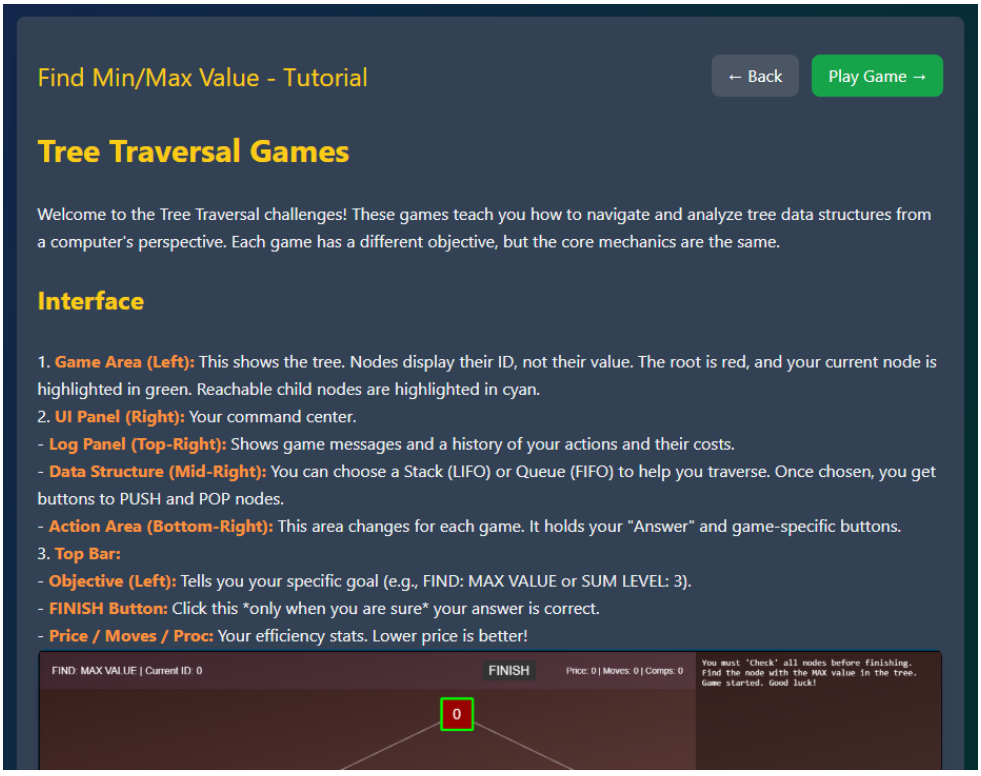



Рисунок 4.6 – Інструкція до гри (туторіал)

Список лідерів представляє собою таблицю, яка відображає найкращі збережені результати гри. Таблиця також позначає результат останньої ігрової сесії для зручності у відстежуванні. Додаткова інформація про результати сесії доступні при натисканні на результат у списку (рис. 4.7).

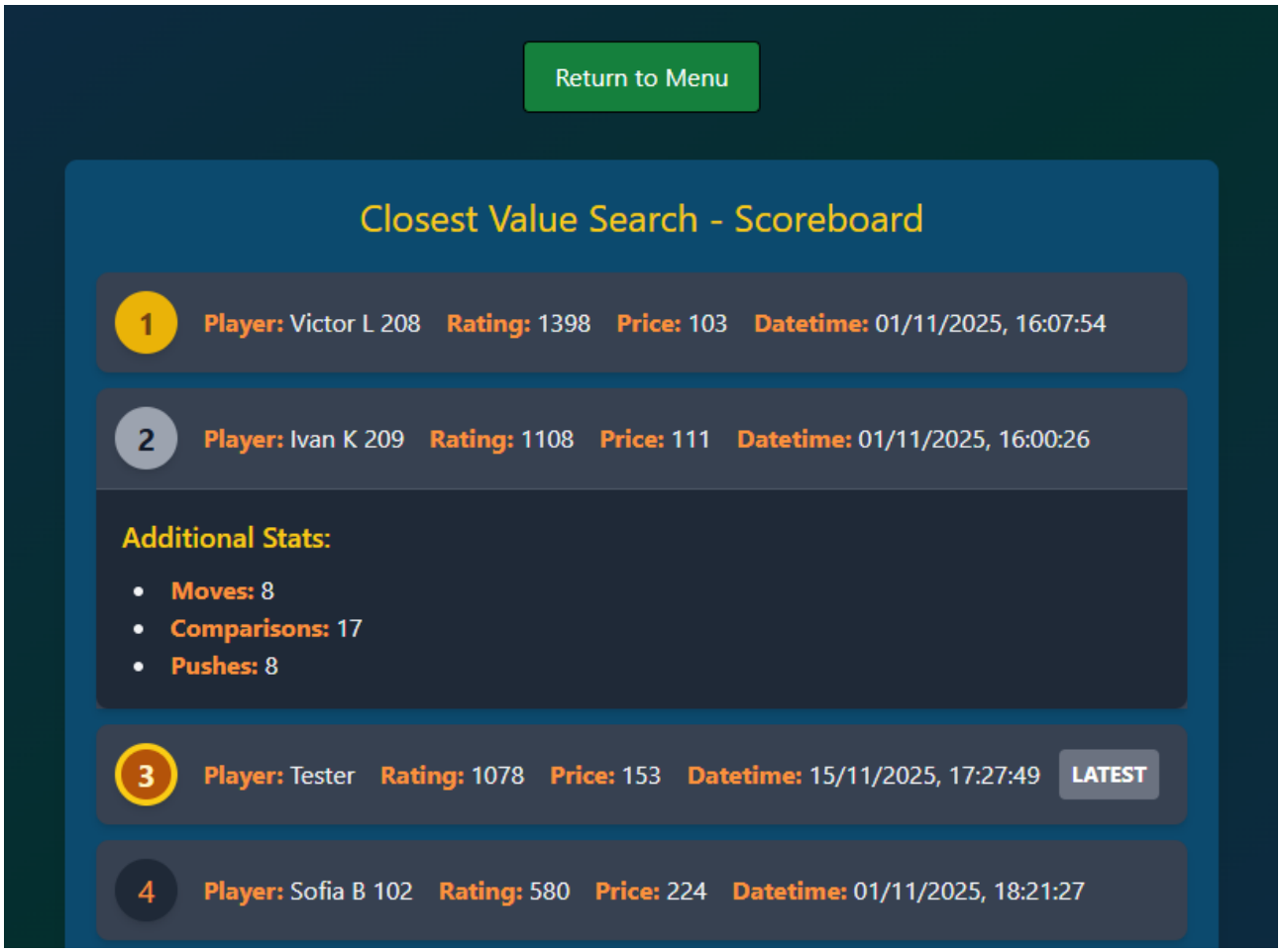



Рисунок 4.7 – Таблиця лідерів

Нарешті, сам геймплей відбувається у полотні Phaser, відповідно до правил, описаних у інструкції до конкретної гри (рис. 4.8-4.9). В наведених прикладах реалізованих ігор, управління гравцем здійснюється лише мишкою – сюди входять натискання, подвійні натискання, перетягування. Блок логів, хоч і має певні недоліки з точки зору дизайну, є досить реалістичним для майбутніх розробників – структуризація та аналіз інформації, наданої програмним забезпеченням при його налагодженні є дуже важливим.

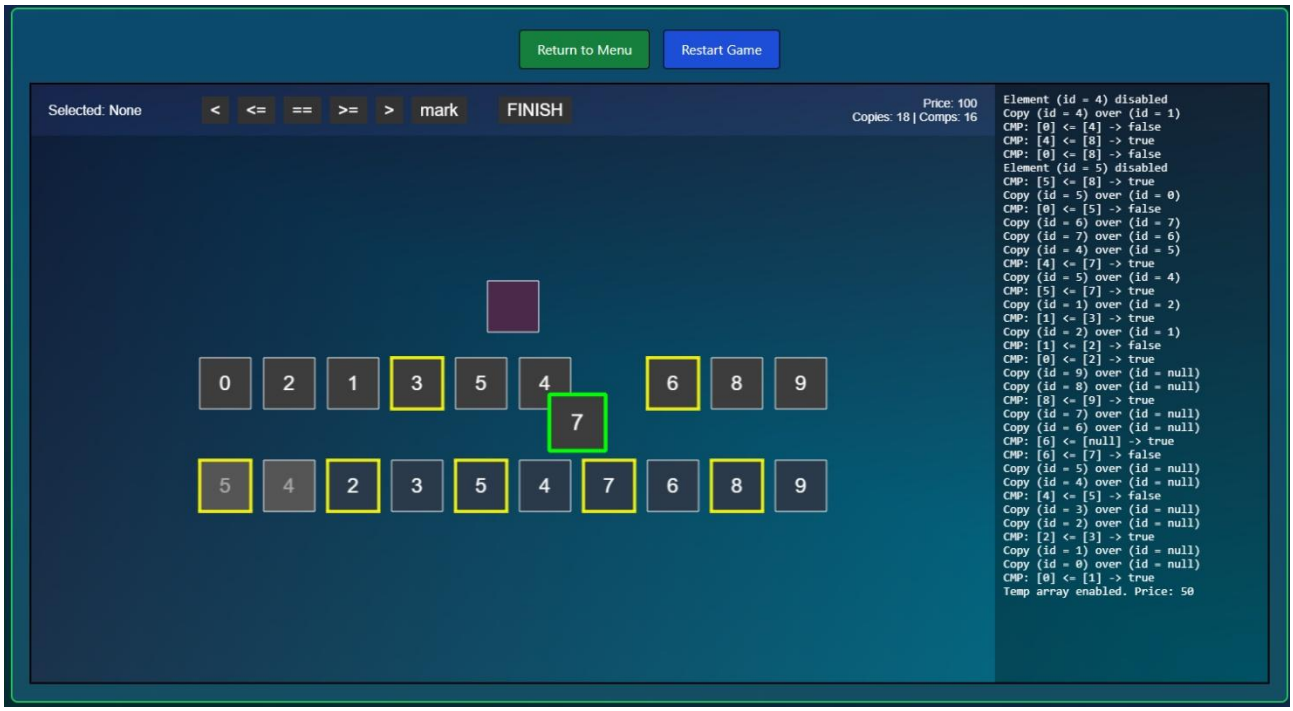



Рисунок 4.8 – Ігровий процес гри «Optimize the imitation»

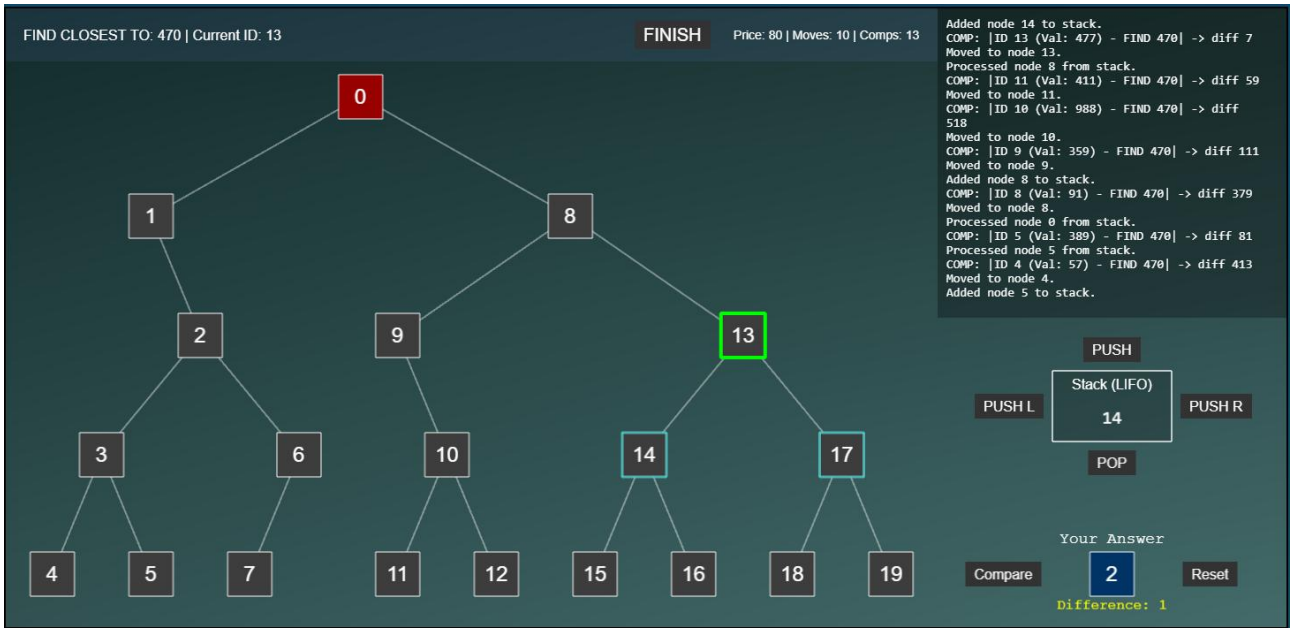



Рисунок 4.9 – Ігровий процес гри «Closest Value Search»

Після проходження гри, користувача зустрічає відповідний екран з аналізом результатів та оновлена таблиця лідерів. Аналіз додається до кожної гри окремо та базується на вхідних і вихідних даних розв’язання, наданого користувачем (рис. 4.10).

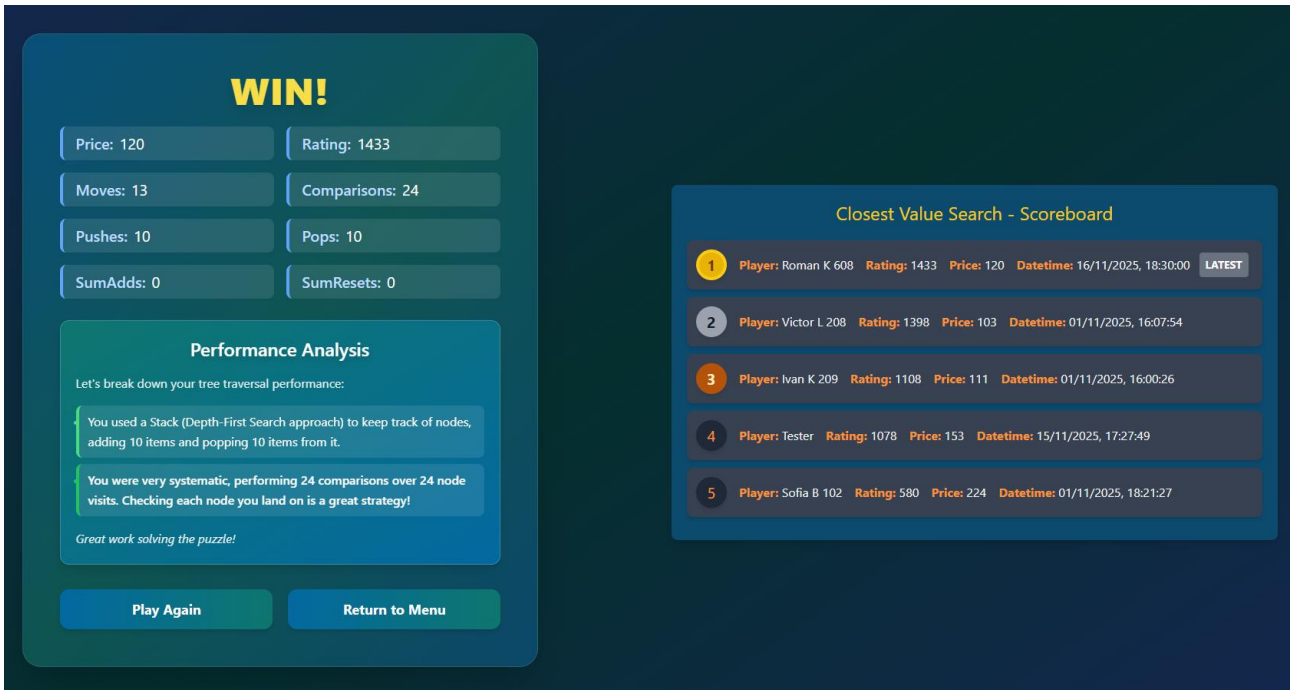



Рисунок 4.10 – Екран результатів

Перспектива системи заключається у модульності – її можна розширювати іграми постійно, просто додаючи їх до конфігурації із передаванням коду сцен. Це дозволить навіть здобувачам додавати до системи свої ігри в якості практики для засвоєння вивченого матеріалу. Звичайно, для ефективного менеджменту іграми та іншими даними потрібно розробити більш глобальний та централізований модуль і підсистему для конфігурабельності, але це не входить до запланованих можливостей для реалізації у рамках дослідження. Головне – це перевірка потенціалу навчальних ігор як способу закріплення здобувачами знань та їх оцінювання.

### 4.3 Тестування та впровадження

Тестування програмного забезпечення – це процес перевірки комп'ютерної програми з метою визначення її відповідності заданим вимогам та очікуванням користувачів. Воно включає запуск та дослідження програми в різних умовах для виявлення помилок, дефектів або неточностей у логіці роботи ще до моменту офіційного випуску. Тестування ігрового програмного забезпечення є окремим напрямом, який, окрім пошуку технічних збоїв, зосереджується на перевірці

ігрового балансу, коректності механік, зручності керування та загального враження від ігрового процесу. Загалом цей етап розробки необхідний для гарантування якості кінцевого продукту, мінімізації фінансових та репутаційних ризиків, а також для забезпечення надійності та безпеки, адже виправлення проблем на стадії створення коштує значно дешевше, ніж після релізу.

Окрім ручного тестування застосунку під час та по завершенню розробки власне розробником, створену інформаційну систему для гейміфікації навчального процесу також доцільно передати викладачам та здобувачам. Отже, навчальні ігри впроваджені в курс «Алгоритми та структури даних» для здобувачів груп 2-го курсу спеціальності «Інженерія програмного забезпечення».

Очікуваним результатом тестування було отримання зворотнього зв’язку від цільової аудиторії: викладачів та здобувачів. Для збору результатів використано форму в Google Forms. Опитування включали питання про зручність та інтуїтивність інтерфейсу, складність ігор, загальні враження, побажання та думку про те, чи зручніше, цікавіше та ефективніше засвоювати матеріал в геймифікованому поданні

Тестування проводилося в 2 етапи. На першому етапі для використання надано систему з однією грою «Optimise the imitation». Ця версія була доступна лише в локальній мережі університету та не мала функціоналу інструкцій, описаних в розділі 4.2. На рис. 4.11–4.13 наведено результати опитування здобувачів.


Рисунок 4.11 – Результати опитування щодо інтерфейсу

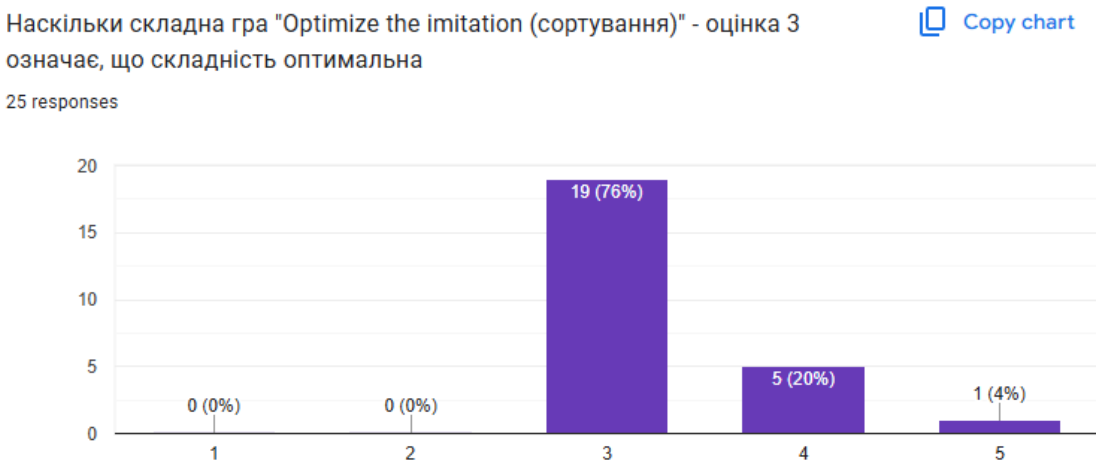



Рисунок 4.12 – Результати опитування щодо складності гри про сортування

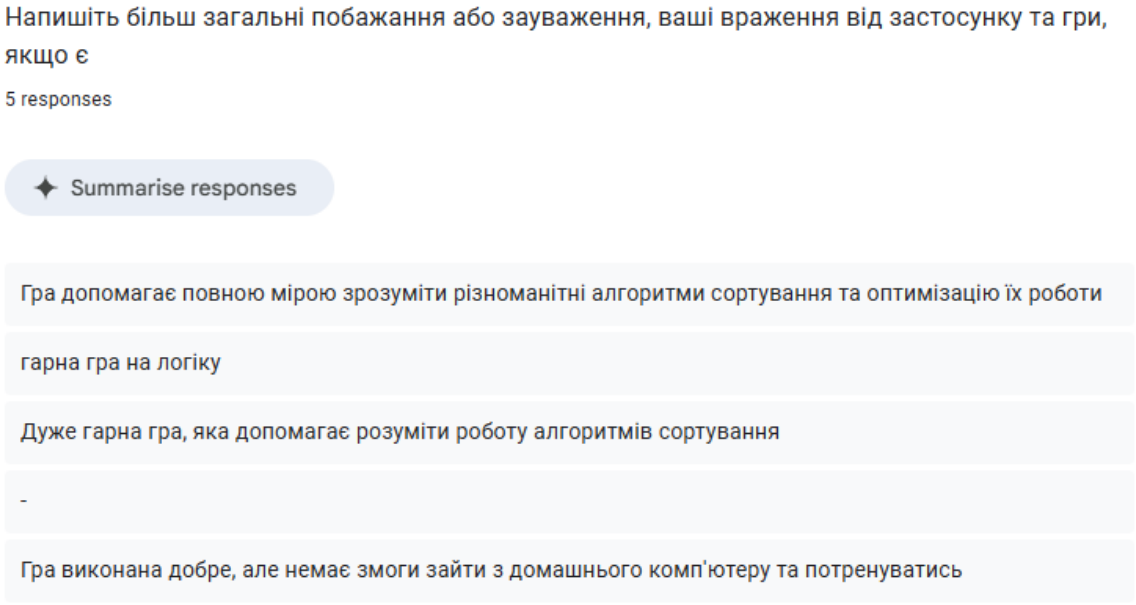



Рисунок 4.13 – Побажання та враження від першої версії системи

Серед отриманих результатів, головними деталями, на які варто було звернути увагу – це потреба у покращенні інтерфейсу, додаванні інструкцій та підсказок до ігор, а також – проблема локальності.

Для підвищення зручності дистрибутивного використання застосунку було прийнято рішення щодо його розгортання на публічному IP-адресі. Таке рішення дає змогу здобувачам мати доступ до системи через Інтернет з будь-якого пристрою через веббраузер.

Наступна версія системи включала покращення загального користувацького та ігрового інтерфейсу, а також були додані інструкції до ігор, які за потреби можна було редагувати в режимі реального часу. Оновлена система також мала розширений асортимент ігор – сумарно чотири дидактичні гри. За новою версією було отримано нові результати тестування та опитувань (рис. 4.14–4.15).


Рисунок 4.14 – Результати другого опитування по інтерфейсу

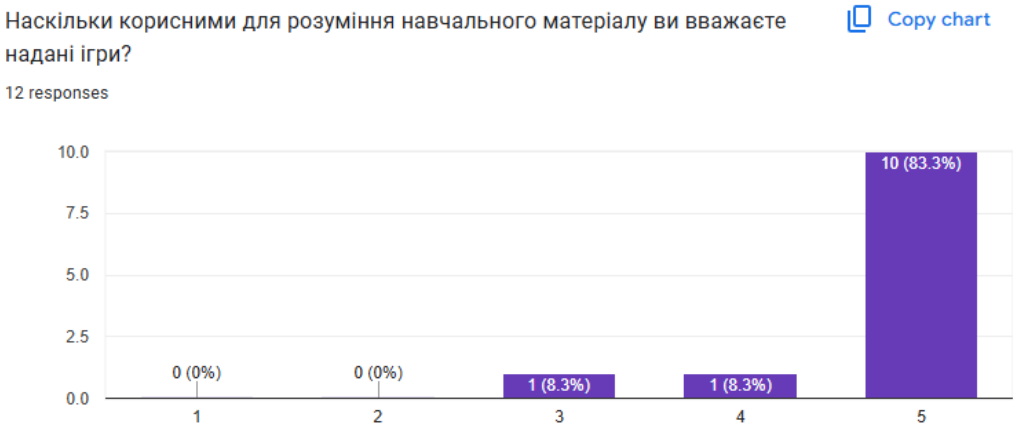



Рисунок 4.15 – Результати опитування про корисність ігор

Загальні враження покращилися, а текстові відповіді доповнилися новими пропозиціями та ідеями, вартими уваги у подальшому процесі розробки.

Також вельми важливим є таке загальне питання, як актуальність впровадження ігор у навчальний процес. Тут результати опитування серед

здобувачів за обома версіями гри були одноголосними (рис. 4.16–4.17). Це свідчить про те, що навіть часткова гейміфікація в навчанні може суттєво покращити його ефективність та залученість здобувачів.

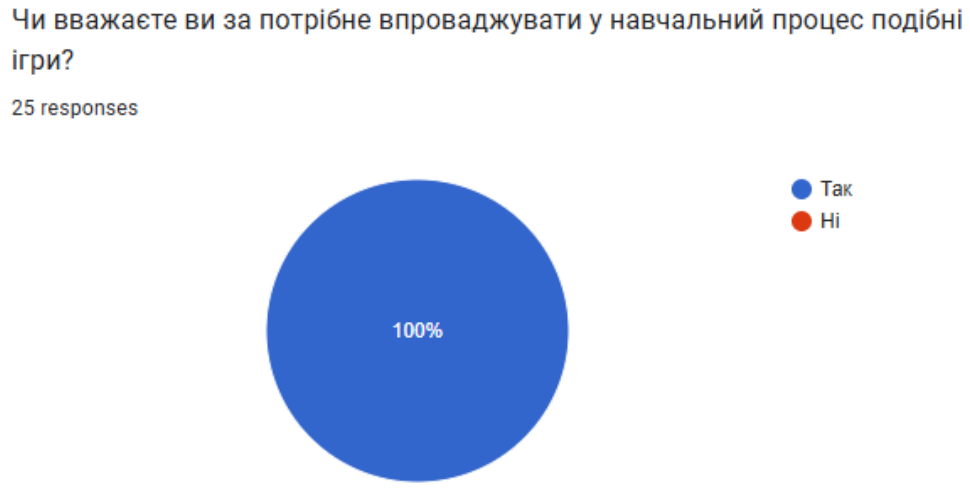



Рисунок 4.16 – Результати першого опитування про актуальність гейміфікації


Рисунок 4.17 – Результати другого опитування про актуальність гейміфікації

Наведені результати тестування у вигляді вихідних даних опитування є як чудовим матеріалом для аналізу досліджуваної теми роботи, так і «маяком» напряму розвитку гейміфікації у навчальному процесі. Беручи за основу цю інформацію, можна розширювати та покращувати систему надалі, після чого розширювати коло дослідження маркету. В подальшому це може стати підґрунтям щодо створення повноцінної та глобальної системи, інтегрованої з МОНУ та іншими інформаційними системами у сфері освіти.

## Висновки до розділу 4

В четвертому розділі кваліфікаційної роботи описано головні особливості кодування інформаційної системи для гейміфікації навчального процесу. Наведено конкретні приклади програмного коду важливих елементів та модулів застосунку, включно з перевіркою коректності сортування користувачем масиву в грі, керуванням та збереженням даних застосунку та користувача, візуалізації структур даних, кастомною реалізацією представлення розмітки інструкцій до ігор та роботи з деревами (їх генерацією) для ігор. За потреби, програмний код, окрім опису та контексту, супроводжувався рисунками вихідних результатів.

Надано керівництво користувача, яке також є демонстрацією виконаної роботи з інтерфейсом та функціональністю в інформаційній системі. Описано функціонал меню, дерево ігор, інструкції, ігрове вікно, вікно результатів та таблиці лідерів. Розділ також включає в себе скріншоти ігрових процесів ігор за темами сортування масивів та обходів дерева.

Також описано проведену роботу з тестування та впровадження інформаційної системи для дисципліни «Алгоритми та структури даних», що викладається здобувачам другого курсу спеціальності «Інженерія програмного забезпечення» на факультеті комп’ютерних наук ЧНУ ім. Петра Могили. Описано етапи впровадження та удосконалення системи грунтовані на результатах двох етапів практичного тестування. Зокрема, наведено результати опитування здобувачів у якості дослідження маркету. Зроблено загальні висновки щодо корисності та потенціалу розроблюваної системи, а також напрямів її розвитку у майбутньому.

# ВИСНОВКИ

У результаті виконання кваліфікаційної магістерської роботи розроблено централізовану вебплатформу з колекцією освітніх ігор. Проведено дослідження щодо ефективності та потреби у гейміфікації навчального процесу підготовки ІТ-фахівців у закладах вищої освіти.

Для досягнення поставленої мети виконано наступні завдання:

– проведено аналіз теоретичних основ та концепцій;

– досліджено нявні рішення у межах зазначеної сфери;

– оцінено найкращі практики використання ігрового програмного забезпечення;

– спроєктовано та розроблено геймифіковану навчальну платформу;

– проаналізовано практичне застосування програмного рішення;

– протестовано та провалідовано розроблене програмне рішення.

Проаналізовано та описано предметну область, виконано порівняння наявних аналогів інформаційної системи із зазначенням їхніх переваг та недоліків. Описано науковий та математичний апарат роботи. Визначено основні функції та характеристики системи, складено специфікацію вимоги до ПЗ. Проведено роботу з моделювання інформаційної системи, складання сценаріїв користувача. З використанням UML-діаграм відображенно ключові елементи алгоритмізації бізнес-процесів та концептуальних особливостей побудови системи. Обґрунтовано вибір стеку технологій для реалізації системи, відображено ключові архітектурні рішення. Розглянуто та описано роботу з кодування системи та дизайну інтерфейсу користувача, впровадження та тестування системи.

Навчальну платформу розроблено та впроваджено (Акт від 4 листопада 2025 р. (додаток А)) на замовлення факультету комп’ютерних наук ЧНУ ім. Петра Могили. У процесі розроблення та впровадження враховані вимоги замовника.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Арістова Н. О. Проблема поняття «мотивація учіння» в науковій літературі. Теоретичні питання культури, освіти та виховання, 22, 2002. С. 97–100.

2. Арістова Н. О. Формування мотивації вивчення іноземної мови у студентів вищих навчальних закладів: монографія. Київ: ТОВ «ГЛІФМЕДІЯ», 2015, 240 с.

3. Бугаєва В. Гейміфікація як спосіб формування активної професійної поведінки майбутніх фахівців ІТ галузі. Educational Challenges, 0(56), 2018. С. 129–135. doi: 10.5281/zenodo.577567.

4. Малихін О. В. Формування індивідуальних стратегій навчання засобами комп'ютерних технологій як педагогічна проблема. Вісник Чернігівського національного педагогічного університету імені Т. Г. Шевченка, 133, 2016. С. 124–126.

5. Tomorrow corporation : human resource machine. Tomorrow Corporation. URL: https://tomorrowcorporation.com/humanresourcemachine (дата звернення: 23.09.2025).

6. Tomorrow corporation : 7 billion humans. Tomorrow Corporation. URL: https://tomorrowcorporation.com/7billionhumans (дата звернення: 23.09.2025).

7. Coding games and programming challenges to code better. CodinGame. URL: https://www.codingame.com/start/ (дата звернення: 23.09.2025).

8. CodeCombat - Coding games to learn Python and JavaScript. CodeCombat. URL: https://codecombat.com/ (дата звернення: 23.09.2025).

9. Free educational games generator | Educaplay. Free educational games generator | Educaplay. URL: https://www.educaplay.com/ (дата звернення: 23.09.2025).

10. Visualising data structures and algorithms through animation - VisuAlgo. visualising data structures and algorithms through animation - VisuAlgo. URL: https://visualgo.net/en (дата звернення: 23.09.2025).

11. Сходинки до інформатики® – офіційний сайт проєкту. URL: http://dvsvit.com.ua/cxodunku/ (дата звернення: 23.09.2025).

12. Twin A. How to Do Market Research, Types, and Example. Investopedia. URL: https://www.investopedia.com/terms/m/market-research.asp (дата звернення: 23.09.2025).

13. What is a Minimum Viable Product (MVP)?. Agile Alliance | Promoting a more effective, humane, and sustainable way of working. URL: https://agilealliance.org/glossary/mvp/ (дата звернення: 23.09.2025).

14. Кірей К. О. Алгоритми та структури даних: методичні рекомендації для виконання лабораторних робіт здобувачами денної форми навчання спеціальності 121 «Інженерія програмного забезпечення». Миколаїв: ЧНУ ім. Петра Могили, 2019. 90 с.

15. Jon Kleinberg, Eva Tardos. Algorithm Design, 1st Edition, 2009. 864 с.

16. Barnett G., Del Tongo L. ata Structures and Algorithms: Annotated Reference with Examples. 2008. 101 с. URL: https://mta.ca/~rrosebru/oldcourse/263114/Dsa.pdf (дата звернення: 09.10.2025).

17. Hashing in Data Structure. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/hashing-data-structure/ (дата звернення: 09.12.2025).

18. Sorting Algorithms. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/sorting-algorithms/ (дата звернення: 09.12.2025).

19. Data Structures - Searching Algorithms. Free Tutorials on Technical and Non Technical Subjects. URL: https://www.tutorialspoint.com/data_structures_algorithms/searching_algorithms.htm (дата звернення: 09.12.2025).

20. Path Finding Algorithms. Medium. URL: https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40 (дата звернення: 09.12.2025).

21. GeeksforGeeks. A* Search Algorithm. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/a-search-algorithm/ (дата звернення: 09.12.2025).

22. Tree Data Structure. Programiz: Learn to Code for Free. URL: https://www.programiz.com/dsa/trees (дата звернення: 09.12.2025).

23. Tree Traversal Techniques. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/tree-traversals-inorder-preorder-and-postorder/ (дата звернення: 09.12.2025).

24. DSA Binary Trees. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/dsa/dsa_data_binarytrees.php (дата звернення: 09.12.2025).

25. JavaScript with syntax for types. TypeScript. URL: https://www.typescriptlang.org/ (дата звернення: 15.05.2025).

26. React. URL: https://react.dev/ (дата звернення: 15.05.2025).

27. Phaser - A fast, fun and free open source HTML5 game framework. URL: https://phaser.io/ (дата звернення: 30.08.2025).

28. UML Use Case Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/tutorial/uml-use-case-diagram (дата звернення: 09.12.2025).

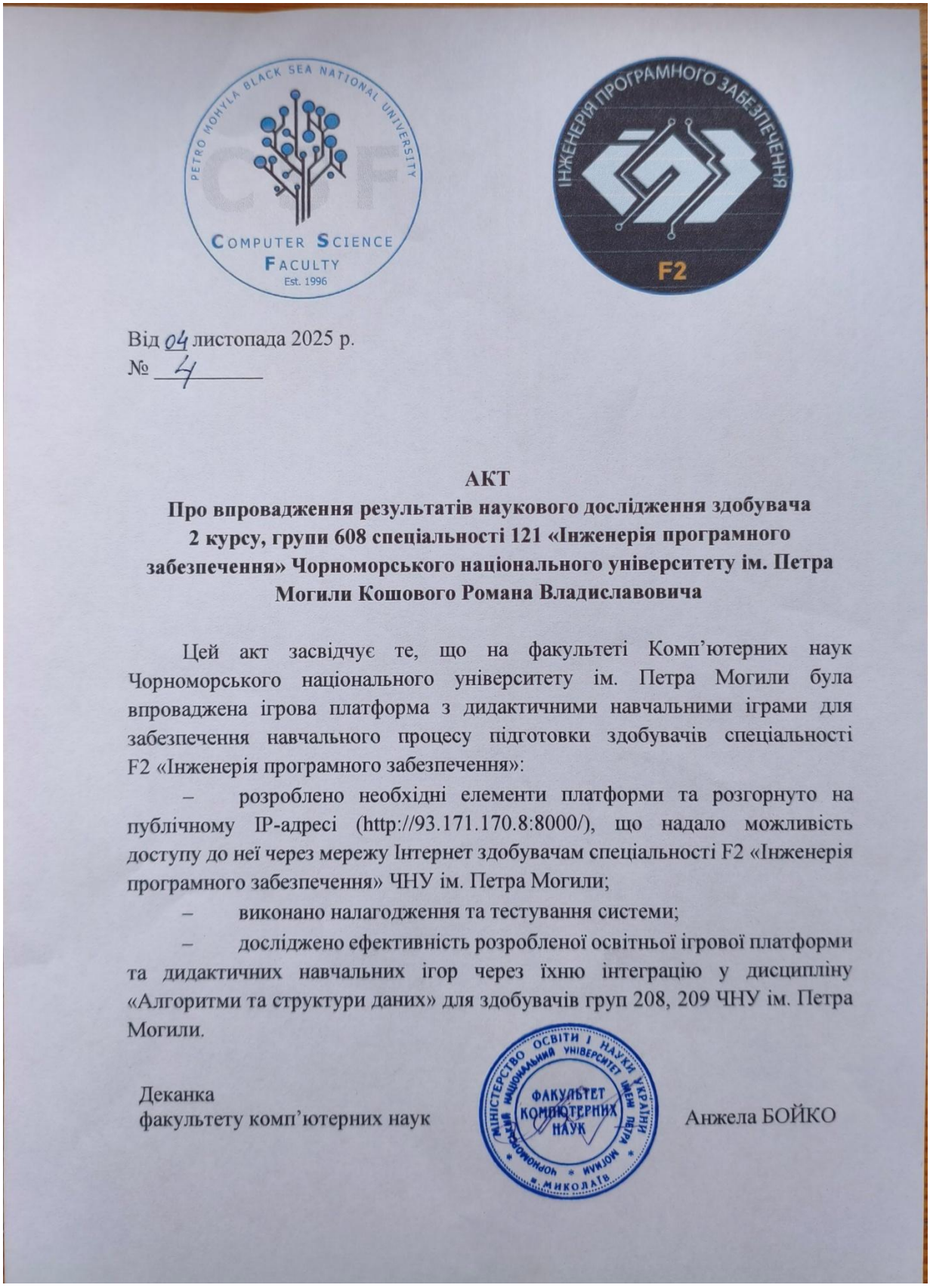
29. Client-side storage - Learn web development. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Client-side_APIs/Client-side_storage (дата звернення: 15.11.2025).

30. UML Sequence Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/uml-sequence-diagram (дата звернення: 09.12.2025).

31. UML Activity Diagram Tutorial. Lucidchart. URL: https://www.lucidchart.com/pages/tutorial/uml-activity-diagram (дата звернення: 09.12.2025).

# ДОДАТОК А

## Акт впровадження інформаційної системи



Від 04 листопада 2025 р.
№ 4

**АКТ**
**Про впровадження результатів наукового дослідження здобувача 2 курсу, групи 608 спеціальності 121 «Інженерія програмного забезпечення» Чорноморського національного університету ім. Петра Могили Кошового Романа Владиславовича**

Цей акт засвідчує те, що на факультеті Комп'ютерних наук Чорноморського національного університету ім. Петра Могили була впроваджена ігрова платформа з дидактичними навчальними іграми для забезпечення навчального процесу підготовки здобувачів спеціальності F2 «Інженерія програмного забезпечення»:

– розроблено необхідні елементи платформи та розгорнуто на публічному IP-адресі (http://93.171.170.8:8000/), що надало можливість доступу до неї через мережу Інтернет здобувачам спеціальності F2 «Інженерія програмного забезпечення» ЧНУ ім. Петра Могили;
– виконано налагодження та тестування системи;
– досліджено ефективність розробленої освітньої ігрової платформи та дидактичних навчальних ігор через їхню інтеграцію у дисципліну «Алгоритми та структури даних» для здобувачів груп 208, 209 ЧНУ ім. Петра Могили.

Деканка
факультету комп'ютерних наук Анжела БОЙКО