Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 «Інженерія програмного забезпечення»

На тему: *Розробка комп'ютерної гри-головоломки «Криптограф»*

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-21-2
_____ *Опанасюк В. В.*

Керівник кваліфікаційної роботи _____ */ Стрюк А. М. /*

Завідувач кафедри _____ */ Стрюк А. М. /*

Кривий Ріг
2025

Криворізький національний університет
Факультет: *Інформаційних технологій*
Кафедра: *Моделювання та програмного забезпечення*
Освітньо-кваліфікаційний рівень: *бакалавр*
Спеціальність: *121 "Інженерія програмного забезпечення"*

ЗАТВЕРДЖУЮ
Зав. кафедри
_____*Стрюк А. М.*
«___»_____ 2025__ р.

ЗАВДАННЯ
на кваліфікаційну роботу

студенту групи ІПЗ-21-2 Опанасюку Владиславу Володимировичу

1. Тема: *Розробка комп'ютерної гри-головоломки «Криптограф»* затверджено наказом по КНУ №119 від «10» квітня 2025 р.
2. Термін подання студентом закінченого проекту «20» червня 2025 р.
3. Вихідні дані по роботі: *Пояснювальна записка: 63 сторінки, 17 рисунків, 1 додаток, 12 використаних у роботі джерел*
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): актуальність розробки гри. Проектування програмного забезпечення. Реалізація програмного забезпечення. Тестування гри «Криптограф».
5. Перелік графічного демонстраційного матеріалу: Приклади існуючих програм. Діаграма потоків даних. Діаграми використання. Блок-схеми основних алгоритмів. Структура бази даних. Приклади роботи розробленої програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	*Формулювання мети та задач роботи*	*13.02.2025-20.02.2025*
2	*Аналіз інформаційних джерел*	*21.02.2025-09.03.2025*
3	*Визначення вимог до програмного забезпечення*	*10.03.2025-18.03.2025*
4	*Розробка функціональної схеми*	*19.03.2025-23.03.2025*
5	*Розробка алгоритмів*	*24.03.2025-31.03.2025*
6	*Розробка бази даних*	*01.04.2025-14.04.2025*
7	*Розробка програмного забезпечення*	*15.04.2025-13.05.2025*
8	*Тестування програмного забезпечення*	*14.05.2025-20.05.2025*
9	*Оформлення пояснювальної записки*	*21.05.2025-12.06.2025*
10	*Розробка демонстраційних матеріалів*	*13.06.2025-17.06.2025*

Дата видачі завдання: «_____»_____ 2025 р.
Студент _____ Опанасюк В. В.
Керівник роботи _____ Стрюк А. М.

РЕФЕРАТ

КРИПТОГРАФІЯ, КРИПТОГРАМА, ГОЛОВОЛОМКА, ГРА, PYTHON, TKINTER

Кваліфікаційна робота складається з 63 сторінок, містить 17 рисунків, 1 додаток та 12 використаних у роботі джерел.

Метою кваліфікаційної роботи є створення комп'ютерної гри-головоломки «Криптограф», що буде знайомити гравця з класичними алгоритмами шифрування та практичними методами криптоаналізу.

В роботі досліджено актуальність розробки комп'ютерної гри-головоломки «Криптограф», включно з дослідженням розвитоку криптографічних алгоритмів, оглядом існуючих криптографічних ігор та головоломок, комп'ютерних ігор з використанням криптографічних головоломок.

Були визначенні вимоги до комп'ютерної гри-головоломки «Криптограф», спроєктовано діаграми використання комп'ютерної гри, діаграма класів та базовий алгоритм гри.

Реалізація гри виконана мовою Python з використанням бібліотеки Tkinker. Наведено приклади різних режимів роботи гри-головоломки.

Створене програмне забезпечення може мати як розважальне так і освітнє застосування, а також сприяти загальному інтелектуальному розвитку гравців.

ABSTRACT

CRYPTOGRAPHY, CRYPTOGRAM, PUZZLE, GAME, PYTHON, TKINTER

The qualification work consists of 63 pages, contains 17 figures, 1 appendix, and 12 sources used in the work.

The aim of the qualification work is to create a computer puzzle game called "Cryptographer" that introduces players to classic encryption algorithms and practical methods of cryptanalysis.

The work examines the relevance of developing the computer puzzle game "Cryptographer," including an exploration of the development of cryptographic algorithms, a review of existing cryptographic games and puzzles, and computer games utilizing cryptographic puzzles.

The requirements for the computer puzzle game "Cryptographer" were defined, and use case diagrams, class diagrams, and the basic algorithm of the game were designed.

The game was implemented in Python using the Tkinter library. Examples of different gameplay modes of the puzzle game are provided.

The developed software can serve both entertainment and educational purposes, as well as contribute to the general intellectual development of players.

ЗМІСТ

ВСТУП

Деякі криптографічні алгоритми, такі як, наприклад, підстановочні шифри на кшталт шифру Цезаря, з часом втратили свою роль для забезпечення безпеки даних, оскільки стали надто простими для розкриття в умовах сучасних технологій і методів аналізу. У давні часи вони забезпечували відносно надійну охорону інформації, але розвиток обчислювальної техніки, математичних методів і криптоаналізу значно знизив їхню ефективність. Сьогодні такі шифри сприймаються більше як історичний артефакт або інструмент для навчання основам криптографії.

З плином часу прості криптографічні схеми, як-от шифр Цезаря, стали популярними у сфері інтелектуальних розваг, перетворившись на цікаві головоломки та інтерактивні завдання. Вони часто використовуються в освітніх програмах для демонстрації принципів криптографії, у настільних іграх чи квестах, де гравцям пропонується розгадати зашифроване повідомлення, щоб досягти певної мети або отримати підказку. Крім того, вони знайшли своє місце в літературі, кінематографі та сучасному мистецтві, додаючи інтриги та стимулюючи глядачів чи читачів до активної участі. Такі алгоритми продовжують залишатися джерелом натхнення і цікавості, сприяючи розвитку креативного мислення і ознайомленню людей із базовими принципами шифрування.

Криптографічні ігри-головоломки є чудовою вправою на розвиток логічного та алгоритмічного мислення, а також надають можливість глибше зрозуміти принципи криптографії, криптоаналізу та захисту інформації. Такі ігри часто базуються на завданнях, де потрібно шукати приховані повідомлення, аналізувати шифрування, розпізнавати ключі або відновлювати зашифровані дані. Секрет їхньої привабливості полягає в інтерактивності та можливості випробувати свої інтелектуальні здібності в обстановці, яка стикається з реальними проблемами вивчення та злому криптографічних систем.

У процесі проходження таких головоломок, учасники знайомляться з класичними методами шифрування, такими як шифр Цезаря, шифр Вернама, або методи перестановки, а також сучасними алгоритмами, наприклад AES, RSA або використання криптографії на еліптичних кривих. За допомогою таких вправ можна зрозуміти фундаментальні концепти, такі як симетричне та асиметричне шифрування, хешування, стійкість алгоритмів до атак із відкритим текстом або методом грубої сили.

Крім суто технічних аспектів, криптографічні задачі часто сприяють розвитку креативного мислення, адже більшість завдань вимагає нестандартного підходу до вирішення, вміння розпізнавати патерни або знаходити слабкі місця в системах захисту даних. Такі ігри також можуть виконувати освітню роль, допомагаючи новачкам і навіть досвідченим професіоналам у безпеці інформації вдосконалювати свої навички та знаходити нові способи аналізу та захисту даних.

Ігри-головоломки, спеціально побудовані на основі криптографічних методів, допомагають учасникам краще зрозуміти важливість сильних паролів, грамотного управління ключами та захисту перед передовими техніками хакерів. Вони не лише розважають, але й слугують важливим інструментом для професійної підготовки у галузі кібербезпеки, сприяючи встановленню культури уважного ставлення до збереження даних в епоху цифровізації.

Тож метою нашої кваліфікаційної роботи є створення комп’ютерної гри-головоломки «Криптограф», що буде знайомити гравця з класичними алгоритмами шифрування та практичними методами криптоаналізу.

1 АКТУАЛЬНІСТЬ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ-ГОЛОВОЛОМКИ «КРИПТОГРАФ»

1.1 Розвиток криптографічних алгоритмів

Криптографія — це наука, яка займається тим, як приховати інформацію за допомогою шифрування, щоб її зміст міг зрозуміти тільки той, кому вона призначена. Вона з'явилася дуже давно — приблизно 4000 років тому, і завжди мала одну основну мету: зробити спілкування таємним. Люди використовували шифри в різні часи — військові, дипломати, торговці, а іноді навіть закохані, щоб зберегти свої секрети від сторонніх. Раніше для цього застосовували прості методи, наприклад, писали шифровані повідомлення на папері, і це допомагало зберігати особисту або важливу інформацію у безпеці.

Криптографія виникла дуже давно, тому що людям потрібно було передавати інформацію так, щоб ніхто сторонній її не зрозумів. Це було важливо не тільки у військових чи дипломатичних справах, але й у звичайному житті. Наприклад, в Камасутрі згадується, як закохані використовували шифри для спілкування. Постійна потреба приховувати зміст повідомлень змушувала людей придумувати нові й більш складні способи шифрування. Але водночас це давало поштовх іншим людям шукати способи розгадати ці шифри. Так виникла своєрідна "гонка", де одну сторону із захисту інформації постійно намагалася обійти інша.

Історія розвитку криптографії показує, як змінилося розуміння того, що робить інформацію захищеною. Спочатку використовували методи, які приховували сам факт існування повідомлення, наприклад, стеганографію. Це могли бути секретні татуювання, як у прикладі з Геродотом. Але з часом, коли все більше людей навчилися читати, потрібно було створити способи, щоб приховувати саме зміст повідомлень, роблячи їх незрозумілими без спеціального "ключа".

У XIX столітті з'явився принцип, названий на честь Керкгоффса. Він пояснює, що захист даних залежить не від того, наскільки заплутаний чи

секретний алгоритм, а лише від того, як добре зберігається ключ. Раніше багато систем шифрування спиралися на складність алгоритмів, але це не працювало, бо такі алгоритми легко розгадували. Завдяки принципу Керкгоффса криптографія змінилася: алгоритми стали відкритими для всіх, а весь захист тепер залежить тільки від секретності ключів. Це і стало основою сучасної криптографії.

Класичні докомп'ютерні криптографічні методи традиційно поділяються на два основні типи: шифри заміни (підстановки) та шифри перестановки. Шифри заміни працюють шляхом заміни кожного символу відкритого тексту іншим символом або групою символів за певним фіксованим правилом. Цей метод був найпростішим і найбільш популярним у давнину. На противагу цьому, шифри перестановки змінюють лише порядок символів у повідомленні, не змінюючи самі символи. Хоча для коротких текстів вони вважалися ненадійними, оскільки всі варіанти легко перебрати вручну, для довших повідомлень кількість комбінацій швидко зростала, роблячи ручний перебір неможливим.

1.1.1 Шифр Цезаря

Шифр Цезаря є одним із найпростіших і найвідоміших методів шифрування, що належить до типу шифру підстановки. Його механізм полягає у циклічному зсуві кожної літери тексту на певну кількість позицій (ключ) за алфавітом. Наприклад, при зсуві на 3 позиції, літера "А" замінюється на "Г". Цей процес можна виразити математично за допомогою формули модульної арифметики:

$Y = (X + k) \bmod n$, де X представляє символ відкритого тексту, Y – символ шифрованого тексту, k – ключ (величина зсуву), а n – потужність (кількість літер) алфавіту.

Історично шифр отримав свою назву на честь римського імператора Гая Юлія Цезаря, який використовував його зі зсувом у 3 позиції для секретного листування зі своїми генералами під час військових кампаній. Його племінник Август також застосовував цей шифр, але зі зсувом вправо на одну позицію.

Незважаючи на свою історичну значимість, шифр Цезаря має суттєві вразливості. Кількість можливих ключів є вкрай обмеженою (наприклад, 25 для англійського алфавіту), що робить його надзвичайно вразливим до повного перебору (brute-force атаки). Багаторазове застосування шифру Цезаря не покращує його стійкість, оскільки послідовні зсуви еквівалентні одному більшому зсуву. Його легко зламати, навіть якщо зловмисник має доступ лише до зашифрованого тексту, використовуючи частотний аналіз або техніку "завершення простого компоненту", яка полягає у виписуванні всіх можливих зсувів зашифрованого тексту та візуальному розпізнаванні відкритого тексту.

1.1.2 Шифр Віженера

Шифр Віженера є поліалфавітним шифром, що використовує слово як ключ. Його механізм передбачає повторення ключового слова, доки його довжина не відповідатиме довжині повідомлення. Шифрування здійснюється за допомогою "Квадрата Віженера" – таблиці розміром 26x26, що містить 26 зсунутих алфавітів. Цей шифр дозволяє застосовувати багато різних зсувів для кожної літери повідомлення, що значно ускладнює злам за допомогою простого частотного аналізу, який був ефективним проти моноалфавітних шифрів.

Хоча шифр отримав назву на честь французького дипломата Блеза де Віженера, насправді його винайшов італійський криптограф Джованні Баттіста Белласо у 1553 році. Віженер лише представив його опис у 1585 році, що викликало критику з боку істориків криптографії, таких як Давід Кан. Цей факт підкреслює, що криптографічні інновації часто були результатом поступового розвитку ідей, а не одноосібних проривів, а історична атрибуція може бути складною та не завжди точно відображати реальний внесок. Це також вказує на те, що знання та методи передавалися та вдосконалювалися різними вченими та практиками протягом століть.

Шифр Віженера був достатньо простим для використання в польових умовах, особливо за наявності шифрувальних дисків. Наприклад,

"конфедерати" використовували мідний шифрувальний диск для шифру Віженера під час Громадянської війни в США. Однак, попри свою криптостійкість відносно попередніх шифрів, він не користувався такою ж високою популярністю, як шифр Цезаря, через складність ручного шифрування та дешифрування повідомлень, що вимагало значних часових витрат.

Головною вразливістю шифру Віженера є повторення ключа, якщо він коротший за повідомлення. Це дозволяє криптоаналітикам виявляти повторювані шаблони в зашифрованому тексті та використовувати такі методи, як метод Касіскі, для визначення довжини ключа. Після визначення довжини ключа, шифр стає вразливим до частотного аналізу, застосованого до окремих алфавітів.

1.1.3 Інші знакові класичні методи

Окрім шифрів Цезаря та Віженера, існували й інші важливі класичні криптографічні методи. Диск Енея, що датується IV століттям до нашої ери, був пристроєм з отворами, через які простягалася нитка, що відповідала буквам повідомлення.

Сцитала – це простий шифр перестановки, який використовувався у Стародавній Греції, де текст намотувався на циліндр для шифрування.

Книжковий шифр полягав у вказівці позиції слова в певній книзі (сторінка, рядок, номер у рядку).

Шифр Атбаш є шифром заміни, де перша літера алфавіту замінюється останньою, друга – передостанньою тощо. Нарешті,

Шифр Плейфера, винайдений Чарльзом Уїтстоном у 1854 році, був поліграфічним шифром, який шифрував пари літер, а не окремі символи.

Еволюція від моноалфавітних до поліалфавітних шифрів є прямою відповіддю на розвиток криптоаналізу. Шифр Цезаря, будучи моноалфавітним, був легко зламаний повним перебором та частотним аналізом через обмежену кількість ключів та збереження частотного розподілу літер. У відповідь на це, для підвищення стійкості, були розроблені більш

складні одноалфавітні шифри заміни з великою кількістю ключів, але вони все ще були вразливі до частотного аналізу. Шифр Віженера, як поліалфавітний, став наступним кроком, дозволяючи застосовувати багато різних зсувів, що ускладнювало частотний аналіз. Ця послідовність розвитку демонструє постійну "гонку озброєнь" між творцями шифрів та криптоаналітиками. Кожен новий шифр був спробою подолати вразливості попереднього, а кожен успішний метод криптоаналізу стимулював створення більш складних шифрів. Це підкреслює ітеративний характер розвитку криптографії.

Занепад практичної цінності класичних шифрів для забезпечення серйозної безпеки був зумовлений двома ключовими факторами: стрімким розвитком криптоаналізу та появою комп'ютерної ери. Криптоаналіз, наука про розкриття зашифрованої інформації та криптографованих комунікацій , займається оцінкою методів шифрування та розробкою способів злому зашифрованих повідомлень.

Одним із найдавніших та найефективніших методів криптоаналізу є частотний аналіз, винайдений арабськими вченими (зокрема, Аль-Кінді у IX столітті). Цей метод базується на статистичних властивостях мови, де літери в природних текстах зустрічаються з різною, передбачуваною частотою. Завдяки цьому, криптоаналітики могли співставити частоти символів у зашифрованих текстах з відомими частотами мови оригіналу, що дозволяло розкривати шифри простої заміни. Навіть одноалфавітні шифри заміни, попри значно більшу кількість ключів (26! або понад 10^26 варіантів), залишалися вразливими до частотного аналізу, оскільки вони зберігали частотний розподіл літер вихідного тексту.

Метод Касіскі був створений для розшифровки поліалфавітних шифрів, таких як шифр Віженера. Він допомагає знайти довжину ключа, яким зашифровано текст. Ідея методу в тому, що однакові частини тексту, які повторюються через відстань, кратну довжині ключа, будуть зашифровані однаково. Якщо у зашифрованому тексті знайти такі повтори, можна здогадатися, якою довгою є ключ. Далі, знаючи приблизну довжину ключа,

можна використовувати аналіз частоти символів, щоб розшифрувати текст, розглядаючи кожний з алфавітів шифру окремо.

Класичні шифри, такі як шифри Цезаря і Віженера, перестали бути надійними не тільки через свою простоту. Головна причина їхнього занепаду — це розвиток методів криптоаналізу, які допомагали знаходити слабкі місця у їхній роботі. Наприклад, частотний аналіз та метод Касіскі дали змогу ефективно розгадувати ці шифри. Тобто ключовим фактором була не слабкість самих шифрів, а поява нових способів їх зламу. Це підкреслює динамічний характер криптології, де безпека системи є відносною і залежить від можливостей супротивника. Занепад класичних шифрів був не просто їхньою "застарілістю", а прямим наслідком успіхів криптоаналізу, що виявив їхні фундаментальні вразливості.

Принцип Керкгоффса, сформульований Огюстом Керкгофсом у XIX столітті, став фундаментальним для розуміння стійкості криптосистем. Він стверджує, що стійкість криптосистеми повинна визначатися лише таємністю ключа, а не таємністю алгоритму. Це означало, що підхід "безпека через невідомість" алгоритму більше не працював, і класичні шифри, які часто покладалися на прихованість свого механізму, виявилися недостатньо надійними.

Класичні методи мали низку обмежень, які сприяли їхньому занепаду. Шифр Цезаря, наприклад, мав лише 25 можливих ключів для англійського алфавіту, що робило його вразливим до повного перебору. Поліалфавітні шифри, як Віженера, були складнішими для ручного використання, особливо для довгих повідомлень, що обмежувало їхню практичну ефективність у швидкому та масовому обміні інформацією. Загалом, більшість класичних шифрів пропонували слабкий захист від досвідчених супротивників.

Поява комп'ютерної криптографії після Другої світової війни стала переломним моментом, що остаточно витіснив традиційні шифри з практичного використання. З появою цифрових комп'ютерів та електроніки криптографія стала значно більш просунутою. Комп'ютери уможливили

створення складніших шифрів, які могли шифрувати будь-які дані, представлені у двійковому вигляді, що зробило лінгвістичні підходи в криптоаналізі непридатними. Сучасне 128-бітове математичне шифрування є стандартом і значно надійніше за будь-який древній чи середньовічний шифр. Комп'ютери також знайшли застосування у криптоаналізі, компенсуючи підвищення складності шифрів. Це призвело до того, що класичні шифри стали небезпечними для практичного використання, оскільки їх можна було зламати за лічені секунди за допомогою обчислювальних потужностей.

Комп'ютери сильно змінили криптографію. Раніше вона була більше про мову й текст, а тепер стала частиною математики та техніки. Основна мета криптографії – захистити дані від розумного і хитрого ворога. Замість того, щоб ховати сам алгоритм, сучасна криптографія спирається на складність обчислень, які важко швидко виконати навіть найпотужнішим комп'ютерам. Комп'ютери не просто спростили шифрування, вони перевернули сам підхід до цієї науки. Старі способи, які були підходящими для ручної роботи, тепер не ефективні, бо сучасна техніка може швидко тестувати мільйони варіантів або аналізувати великі дані. Ця зміна стала справжньою революцією в захисті інформації.

1.2 Криптографічні ігри та головоломки

Хоча класичні шифри більше не підходять для забезпечення серйозної безпеки, вони стали популярними в якості головоломок і частини ігор. Їх простота та те, що їх можна розгадати вручну, зробили їх ідеальними для таких цілей.

Криптограми — це головоломки, які створені на основі старих шифрів і використовуються для розваги. Вони виглядають як короткі зашифровані тексти. Найчастіше для їх шифрування застосовують простий спосіб — заміну, де кожну букву замінюють іншою буквою або цифрою. Раніше криптограми мали серйозне призначення, але зараз їх друкують у газетах, журналах чи додають у мобільні додатки, щоб люди могли цікаво проводити час.

Щоб розгадати криптограму, зазвичай використовують деякі підказки, як-от аналіз частоти букв або схожих поєднань у словах. Наприклад, в англійській мові окремою літерою може бути лише "i", "a" або зрідка "o". Також можна помічати подвійні літери, апострофи чи дотримуватися правила, що буква не замінює саму себе. Часто автори головоломок дають кілька початкових підказок, щоб розгадувати було легше.

Криптограми згадуються й у художній літературі. Наприклад, у творі Едгара По "Золотий жук" герой розшифровує текст за допомогою логіки й обчислень, що показує, наскільки такі задачі можуть бути цікавими для розумової роботи.

У настільних іграх часто використовуються головоломки та завдання, які вимагають логічного мислення й роботи в команді. Хоча такі відомі ігри, як шахи, шашки чи доміно, не застосовують шифрування як основну ідею, існують настільні ігри, що прямо пов’язані з криптографією. Наприклад, гра "Зламай Код" побудована на принципах розшифровування. Деякі настільні ігри можуть мати назви, які нагадують про шифри чи історичні постаті, як-от "Цезар! Підкоріть Рим за 20 хвилин". Однак це не завжди означає, що шифр Цезаря використовується як частина гри.

Квест-кімнати та ескейп-руми є однією з найпопулярніших сфер застосування класичних шифрів для розваги. Ці інтерактивні ігри активно використовують різноманітні класичні методи для створення захопливих головоломок.

Приклади застосування включають:

Шифр заміни: Використовується для заміни букв числами або іншими символами, щоб гравці могли розшифрувати приховані повідомлення.

Шифр Цезаря: Часто застосовується зі зсувом на певну кількість позицій для приховування підказок, що ведуть до наступних етапів квесту.

Книжковий шифр: Вимагає від гравців знайти певну книгу та розшифрувати цифри (що вказують на сторінку, рядок та номер букви) для отримання повідомлення.

Шифр Атбаш: Використовується для шифрування фраз шляхом заміни літер на протилежні в алфавіті.

Квадрат Полібія: Цифри використовуються для кодування літер, які потім розшифровуються за допомогою відповідного квадрата.

Шифр Морзе: Повідомлення можуть бути закодовані за допомогою азбуки Морзе, іноді навіть у вигляді аудіосигналів або візуальних елементів.

Дзеркальний шифр: Текст пишеться у зворотному порядку або таким чином, щоб його можна було прочитати лише за допомогою дзеркала, додаючи елемент несподіванки.

Шифри перестановки: Можуть бути використані для складніших головоломок, де потрібно переставляти літери або блоки тексту за певним правилом.

Музичні ноти: Ноти можуть відповідати літерам або числам, що утворюють приховане повідомлення, вимагаючи від гравців знання музичної грамоти або використання спеціального ключа.

Крім того, використовуються й інші типи головоломок, що мають криптографічний характер, такі як ребуси, лабіринти, "пляшучі чоловічки" (тарабарська грамота) та решітка Кардано.

Прості шифри, як-от шифр Цезаря або прості шифри заміни, легко використовувати і розгадувати вручну. На відміну від складних комп’ютерних алгоритмів, вони настільки зрозумілі, що стали популярними в головоломках і іграх. Їх легко розв’язувати без спеціального обладнання чи складних знань, тому будь-хто може спробувати "зламати" код і отримати задоволення від цього процесу. Хоча раніше їх вважали слабкими, тепер ця простота стала їхньою перевагою для розваг.

Людям подобається розгадувати загадки, особливо коли це пов’язано з відчуттям таємниці та розслідування. У квест-кімнатах або іграх часто використовують шифри й головоломки, які створюють атмосферу загадковості. Наприклад, у розповіді Едгара По "Золотий жук" головний герой розгадує криптограму, і це стає важливою частиною сюжету. Це показує, як

людям подобається процес розгадування та відчуття успіху, коли секрет стає зрозумілим. Шифри, які раніше використовували, щоб ховати важливу інформацію, тепер стали способом створити цікаві завдання в іграх, стимулюючи нашу природну цікавість і бажання випробувати свої здібності.

У відеоіграх прямі приклади використання класичних шифрів не так поширені, як у квест-кімнатах, але деякі ігри включають логічні головоломки, що вимагають дешифрування або розпізнавання шаблонів. Цікавим паралельним прикладом є "чит-коди" у відеоіграх, які функціонують як своєрідні приховані повідомлення, які потрібно "розшифрувати" або знайти для активації певних функцій. Хоча вони не є класичними шифрами в строгому сенсі, їхня функція прихованої інформації перегукується з криптографією. Крім того, існують освітні ігри, які використовують криптографічні принципи. Наприклад, Google Doodle, присвячений Алану Тюрінгу, включав інтерактивні головоломки програмування, що імітували роботу машини Тюрінга, демонструючи обчислювальні принципи, які лежать в основі сучасної криптографії.

1.3 Огляд комп’ютерних ігор з використанням криптографічних головоломок.

Не зважаючи, що в криптографічні головоломки мало розповсюджені в комп’ютерних іграх, нам вдалось знайти декілька яскравих прикладів таких ігор.

Серед мобільних ігор можна виділити таку як Cryptograms – Puzzle Quotes (рис. 1.1).

Ця гра пропонує гравцю розгадати криптограму – закодовану цитату. Для шифрування використовується підстановочний шифр. Кожна літера замінюється якоюсь іншою буквою.

Гра повністю англомовна і для розгадування криптограм необхідне гарне знання саме англійської мови.

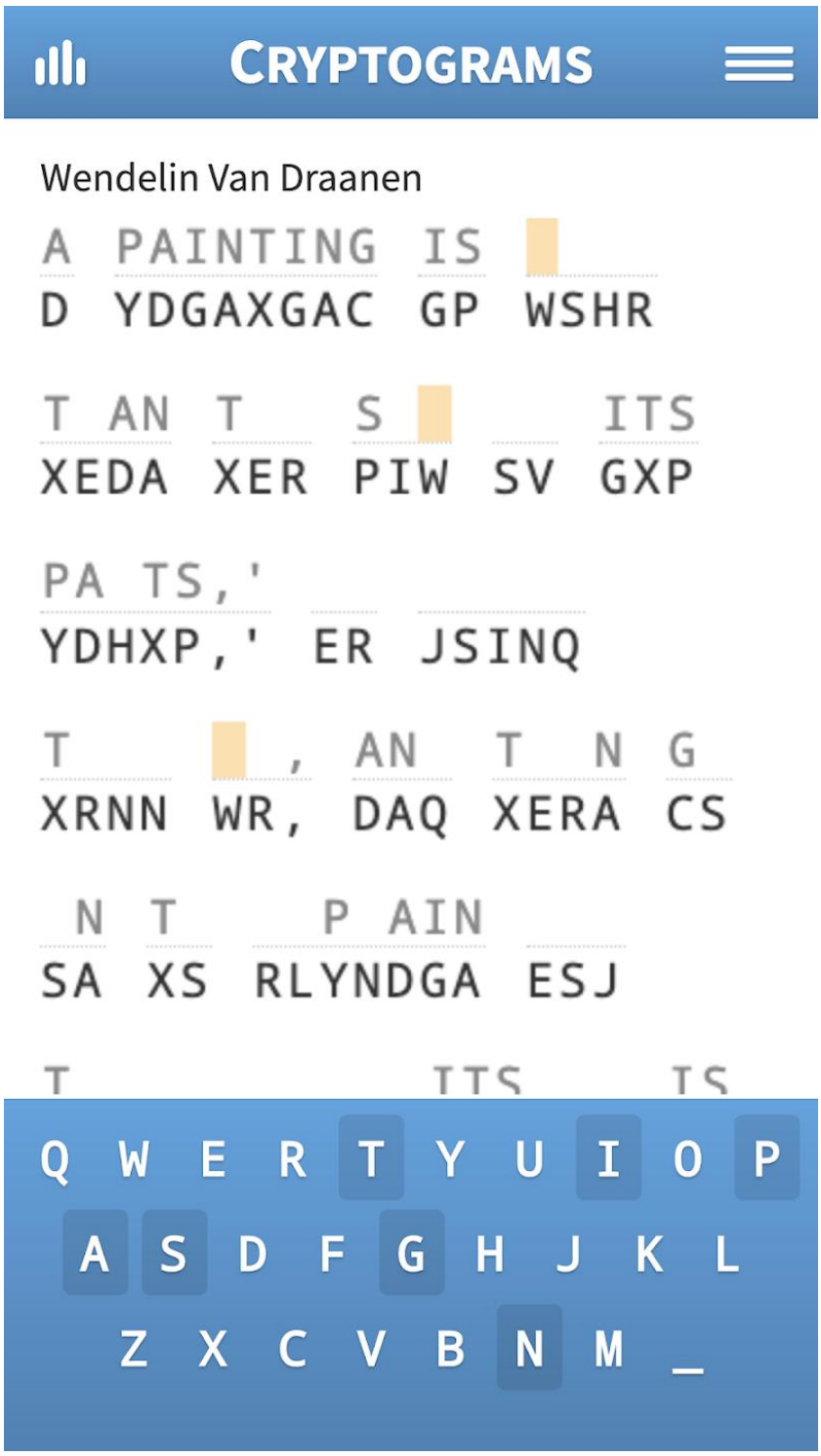

Рисунок 1.1 – Cryptograms – Puzzle Quotes

Іншим прикладом дуже схожої гри є Brain Cryptogram (рис 1.2).

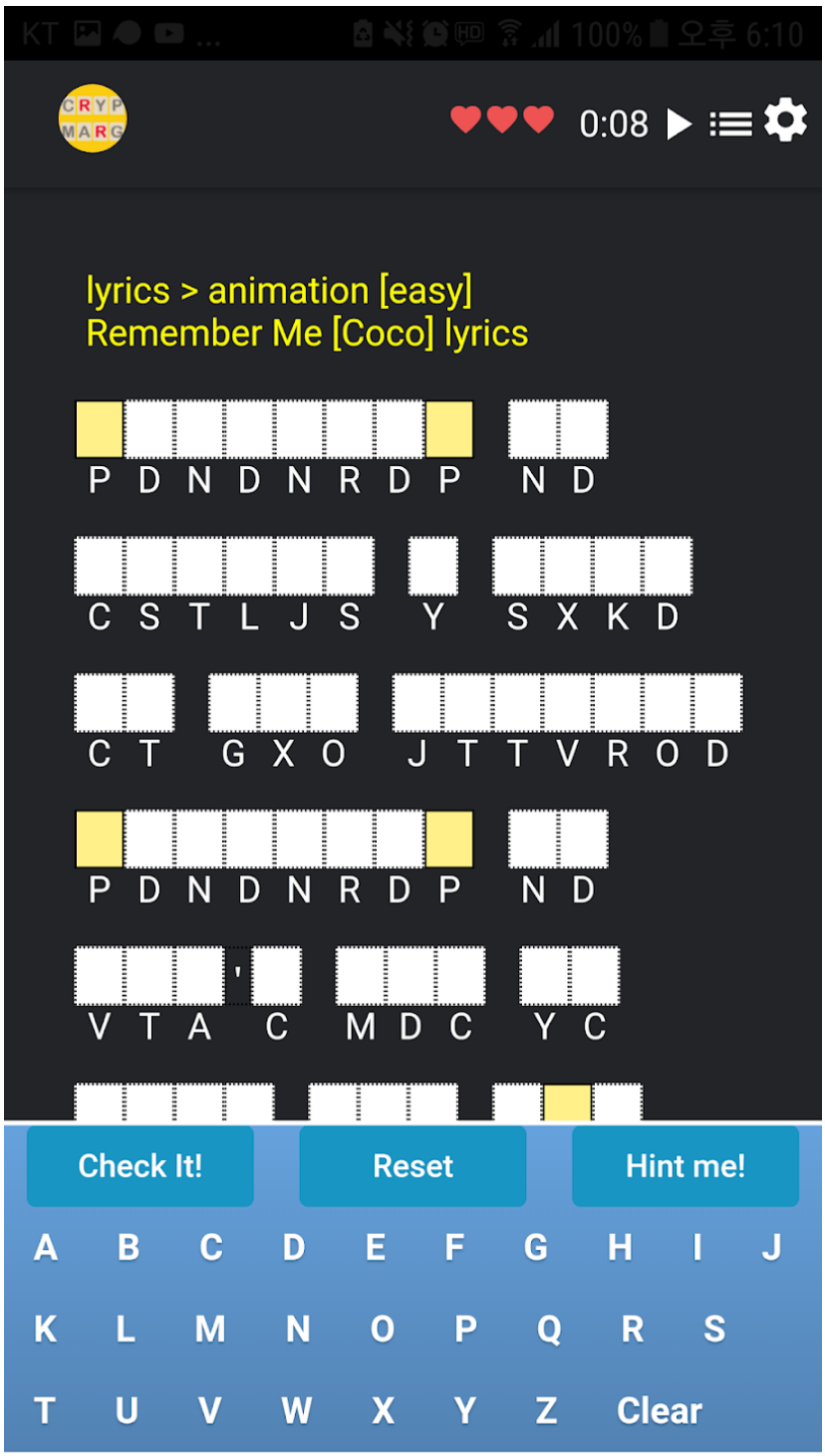

Рисунок 1.2 – Brain Cryptogram

Трохи відмінною від попередніх є гра Cryptogram - Word Puzzle Games (рис. 1.3). Вона також використовує підстановочні шифри, але підставляє числа, замість інших букв.

Проте, спосіб розв’язання такої головоломки залишається незмінним.

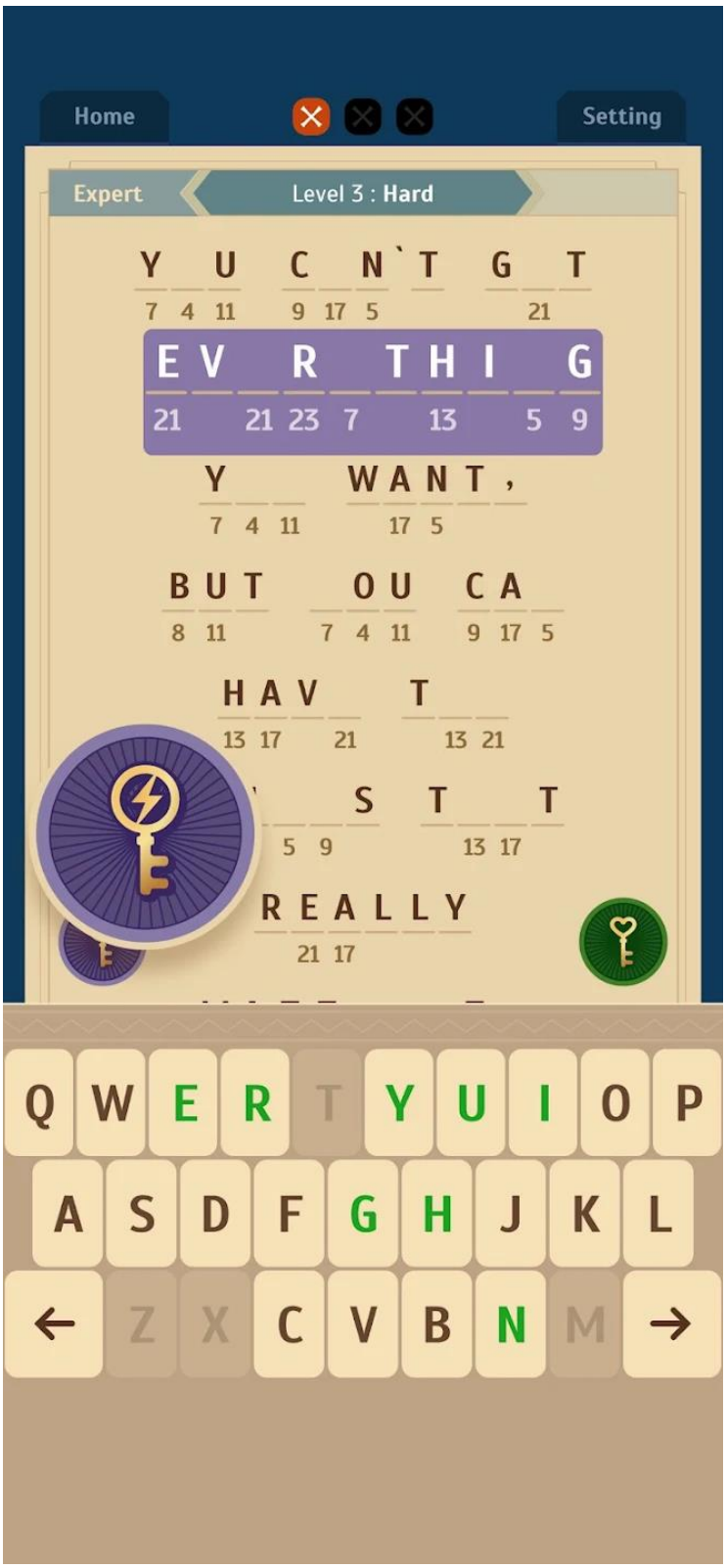

Рисунок 1.3 – Cryptogram - Word Puzzle Games

Kryptic (itch.io): Це гра-головоломка, яка, як зазначено, "базується на кодах та шифрах" (рис. 1.4).

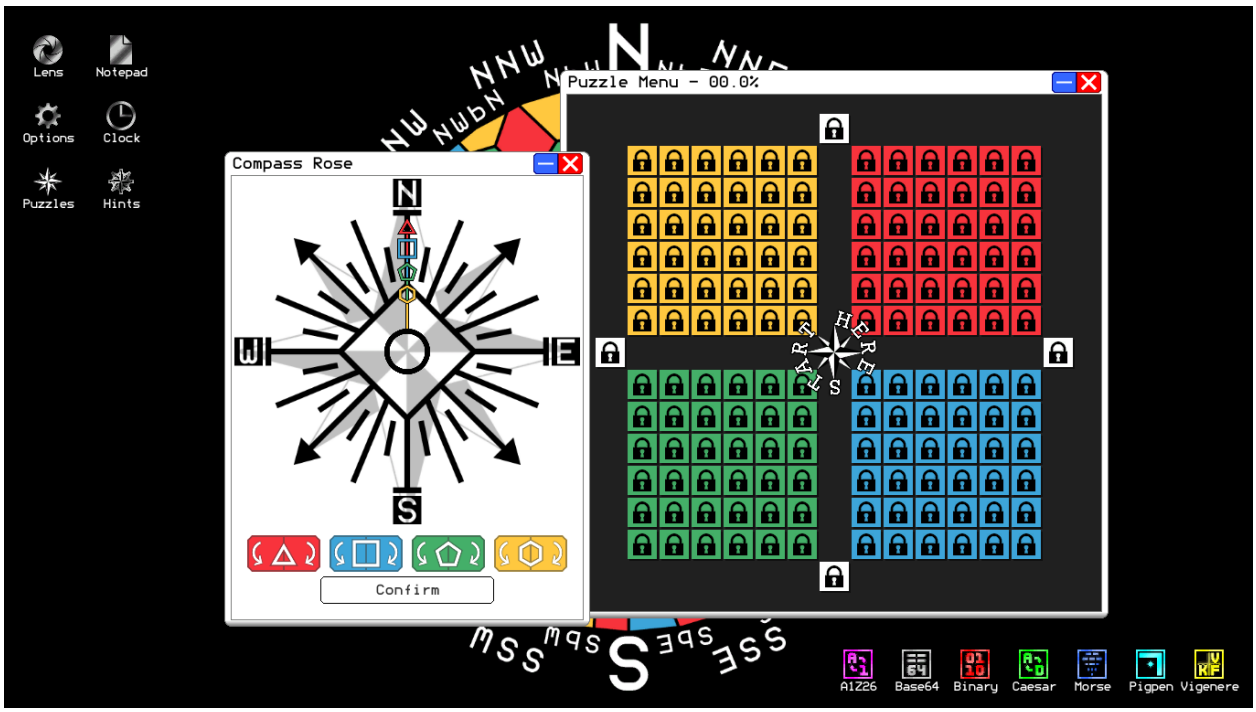

Рисунок 1.4 – Kryptic

Decipherism (itch.io): Ще одна гра-головоломка, де ви розгадуєте шифри кодувальних машин (рис. 1.5).

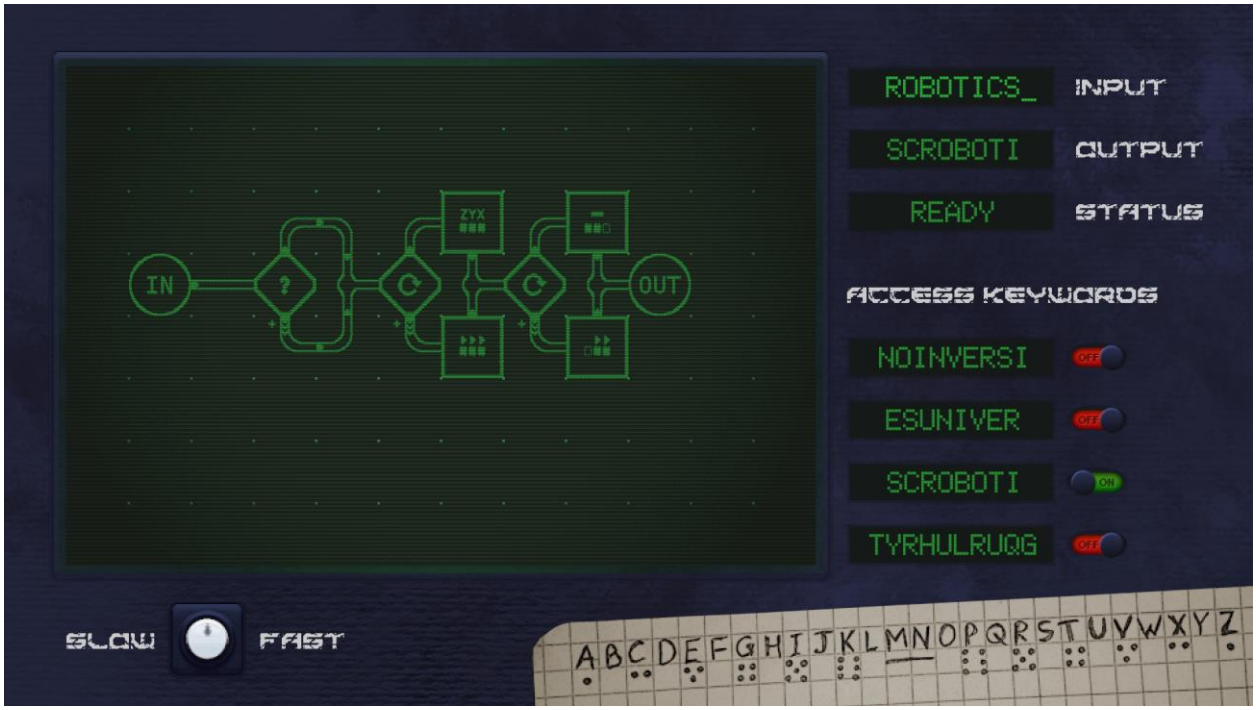

Рисунок 1.5 – Decipherism

В усіх цих іграх є одна важлива особливість – орієнтація саме на англійську мову. Це обмежує можливість їх повноцінного використання в країнах, де англійська не є основною, наприклад в Україні.

Також ці ігри орієнтовані на англомовну культуру. Більшість цитат си крилатих виразів, які використовуються в криптограмах,будуть маловідомими для інших країн. Це також ускладнює ігровий процес.

Тож, зважаючи на високий розвиваючий потенціал криптографічних головоломок в іграх та відсутність українських ігор з такими головоломками, ми вбачаємо актуальність у створені програмного забезпечення комп’ютерної гри-головоломки «Криптограф».

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ ГРИ-ГОЛОВОЛОМКИ «КРИПТОГРАФ»

2.1 Визначення вимог до комп'ютерної гри-головоломки «Криптограф»

Загальні вимоги

- **Назва гри:** "Криптограф" (або варіанти, як-от "Шифри України", "Українські криптограми").
- **Жанр:** Логічна/головоломка, освітня.
- **Цільова аудиторія:** Гравці, які люблять розгадувати головоломки, а також ті, хто цікавиться українською літературою та музикою. Вікова категорія може варіюватися від підлітків до дорослих.
- **Мета гри:** Розгадувати криптограми, використовуючи надані підказки та власні знання української культури.
- **Платформи:** Можна розглянути розробку для ПК (Windows, macOS, Linux), мобільних пристроїв (Android, iOS) або навіть веб-версію.

Вимоги до ігрового процесу

1. Криптограми

- **Джерело:** Цитати українських письменників, поетів (наприклад, Тарас Шевченко, Леся Українка, Іван Франко, Василь Стус, Ліна Костенко), рядки з відомих українських пісень (як народних, так і сучасних).
- **Обсяг:** Достатня кількість криптограм для забезпечення тривалості гри та реграбельності. Можливо, понад 100-200 криптограм на старті з можливістю додавання нових.
- **Тематика:** Важливо, щоб цитати були впізнаваними або цікавими, стимулювали до розгадування.
- **Кодування:** Різні типи шифрів, які застосовуються до тексту криптограми.

2. Рівні складності

Потрібно передбачити декілька рівнів складності, що відрізнятимуться за:

- **Типом шифру:**
  - **Легкий:** Шифр заміни (кожна буква замінюється іншою фіксованою літерою або символом), можливо, з підказками щодо частоти літер.
  - **Середній:** Шифр Цезаря (зміщення літер), шифр Атбаш, дзеркальні шифри, прості поліалфавітні шифри. Можливо, з частиною тексту, що вже розшифрована, або з можливістю отримати підказку щодо кількох літер.
  - **Складний:** Більш комплексні поліалфавітні шифри, транспозиційні шифри, або навіть простіші версії вігенера або playfair (з урахуванням, що це гра, а не професійний криптоаналіз). Обмежена кількість підказок або їх повна відсутність.
- **Наявністю пробілів та розділових знаків:** На легких рівнях пробіли та розділові знаки залишаються, на складних можуть бути прибрані.
- **Довжиною криптограми:** Коротші цитати на легких рівнях, довші — на складних.
- **Кількістю підказок:**
  - **Легкий:** Багато підказок (наприклад, можливість відкрити одну літеру, показати частоту літер, дати категорію автора/пісні).
  - **Середній:** Обмежені підказки (кількість використань).
  - **Складний:** Мінімальні або відсутні підказки.

3. Механіки гри

- **Введення відповіді:** Зручний інтерфейс для введення розшифрованого тексту.
- **Система підказок:** Можливість використовувати підказки (наприклад, за ігрову валюту, або після перегляду реклами, або як нагороду за проходження рівнів). Типи підказок:
  - Відкрити одну літеру у потрібному місці.

    - Показати, яка зашифрована літера відповідає певній розшифрованій.
    - Дати додаткову інформацію про автора або твір (наприклад, епоха, жанр).
    - Показати найчастіші літери у криптограмі.
- **Прогрес гравця:** Збереження прогресу, можливість повертатися до пройдених рівнів.
- **Система нагород:** За проходження рівня (наприклад, ігрова валюта, розблокування нових цитат, досягнення).

Вимоги до інтерфейсу користувача (UI) та користувацького досвіду (UX)

- **Інтуїтивність:** Простий та зрозумілий інтерфейс, що не викликає питань у гравця.
- **Естетика:** Приємний візуальний дизайн, що відповідає тематиці гри (можливо, з використанням українських мотивів в оформленні).
- **Адаптивність:** Для мобільних платформ — адаптивний дизайн під різні розміри екранів.
- **Звук/музика:** Приємний фоновий звук, звукові ефекти при правильному/неправильному введенні, розгадуванні криптограми.

Технічні вимоги

- **База даних криптограм:** Сховище для цитат, їхніх авторів/джерел, типу шифру та рівня складності.
- **Алгоритми шифрування/дешифрування:** Реалізація різних типів шифрів.
- **Мова програмування/движок:** Залежить від обраних платформ (наприклад, Unity або Unreal Engine для кросплатформної розробки, React Native/Flutter для мобільних додатків, або чистий Python/JavaScript для веб-версії).

- **Збереження даних:** Збереження прогресу гравця, налаштувань.

Додаткові вимоги (для майбутнього розширення)

- **Режим "Виклик дня":** Щоденні унікальні криптограми.
- **Режим "На час":** Розгадування криптограм на швидкість.
- **Система рейтингів:** Таблиця лідерів серед гравців.
- **Режим "Свій шифр":** Можливість для гравців створювати свої криптограми та ділитися ними.
- **Енциклопедія:** Розділ з інформацією про авторів та твори, використані в грі.
- **Локалізація:** Можливість додавання інших мов (хоча для початку українська є ключовою).

2.2 Проєктування діаграми використання комп'ютерної гри-головоломки «Криптограф»

Можелювання програмного забезпечення гри «Криптограф» почнемо з побудови діграми використання (рис. 2.1).

Діаграма має наступні елементи:

Гравець (Actor): Головний користувач системи.

Запустити гру: Початок ігрової сесії.

Вибрати рівень складності: Гравець може обрати один з доступних рівнів (Легкий, Середній, Складний).

Розгадати криптограму: Основна функціональність гри, де гравець вводить відповідь.

Використати підказку: Додаткова функція, доступна під час розгадування криптограми. Позначено як extends, оскільки не кожне розгадування передбачає використання підказки.

Переглянути правила: Доступ до інформації про те, як грати.

Переглянути статистику: Перегляд прогресу гравця, кількості розгаданих криптограм тощо.

Вийти з гри: Завершення ігрової сесії.

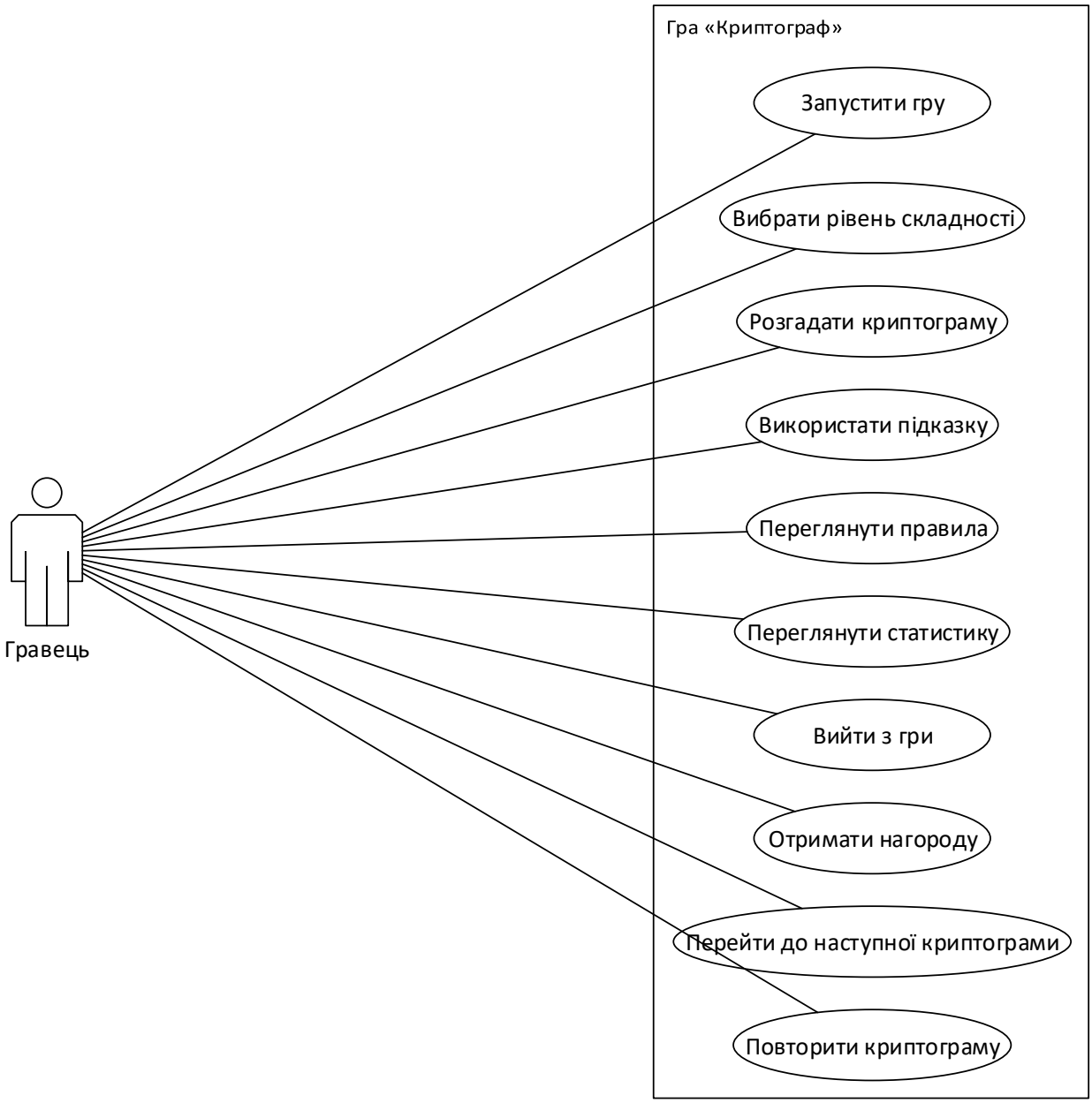

Рисунок 2.1 – Діаграма використання комп’ютерної гри «Криптограф»

Отримати нагороду: Після успішного розгадування криптограми гравець отримує певну винагороду. Позначено як includes, оскільки це є частиною процесу розгадування.

Перейти до наступної криптограми: Після успішного розгадування, гравець переходить до наступного завдання. Позначено як includes до "Розгадати криптограму".

Повторити криптограму: Гравець може вирішити повторити поточну або вже пройдену криптограму. Позначено як extends від "Перейти до наступної криптограми" (можливо, після її завершення або невдачі).

2.3 Проєктування діаграми класів комп’ютерної гри-головоломки «Криптограф»

Наступним кроком моделювання є побудова діаграми класів нашої гри (рис. 2.2).

Було виділено наступні класи:

Game: Головний клас, що управляє станом гри (меню, гра в процесі, пауза, кінець).

Player: Представляє гравця, зберігає його рахунок, кількість доступних підказок та прогрес.

Cryptogram: Базовий клас для кожної криптограми. Містить зашифрований текст, оригінал, автора, джерело, рівень складності та тип шифру.

Difficulty (Enum): Перелік рівнів складності.

CipherType (Enum): Перелік типів шифрів, що використовуються.

Hint: Клас для підказок. Містить тип підказки та її вартість (якщо є).

HintType (Enum): Перелік типів підказок.

GameManager: Управляє логікою гри: завантаження криптограм, вибір поточної, перевірка відповідей, обробка підказок та нагород.

Settings: Клас для зберігання налаштувань гри (звук, мова тощо).

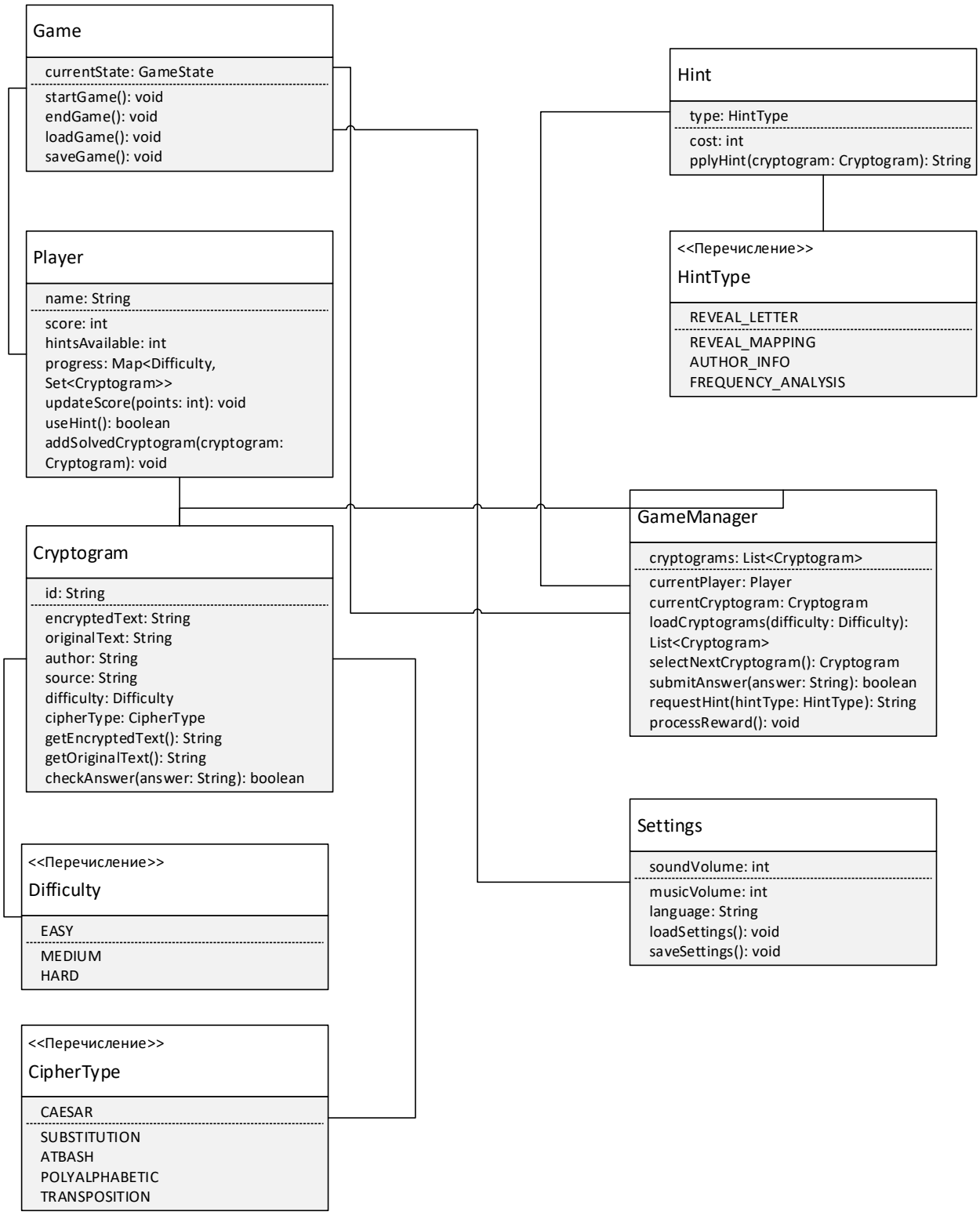

Рисунок 2.2 – Діаграма класів гри «Криптограф»

2.4 Базовий алгоритм гри «Криптограф»

Блок-схема базового алгоритму гри (рис. 2.3) ілюструє основний потік дій гравця від запуску гри до її завершення, включаючи процес вибору рівня, розгадування криптограми, використання підказок та обробку результатів.

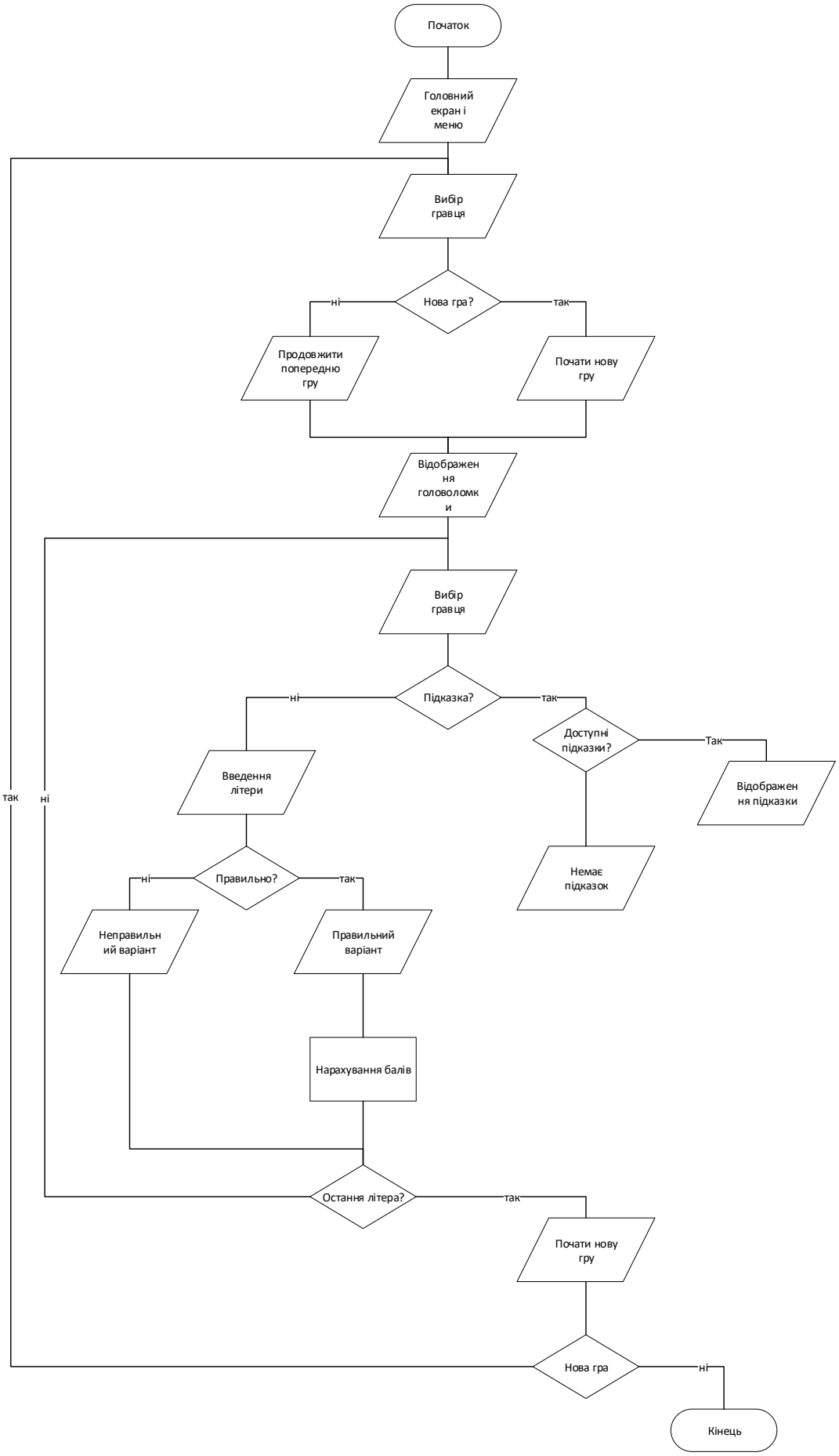

Рисунок 2.3 – Базовий алгоритм гри «Криптограф»

3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ-ГОЛОВОЛОМКИ «КРИПТОГРАФ»

3.1 Добір засобів розробки комп'ютерної гри-головоломки «Криптограф»

Для реалізації гри «Криптограф», яка включає різноманітні криптографічні головоломки, ми обрали мову програмування Python завдяки її зручності та широким можливостям для опрацювання текстових даних. Python пропонує велику кількість готових бібліотек та функцій, які дозволяють реалізувати шифрування, дешифрування, створення алгоритмів для розв'язання криптографічних загадок та генерацію ключів. Його зручний синтаксис забезпечує швидке написання коду, а велика кількість відкритих ресурсів та документації дозволять ефективно вирішувати завдання будь-якого рівня складності.

У грі будуть присутні різні завдання, такі як шифрування тексту за допомогою класичних методів Крім того, можна інтегрувати задачі на основі стеганографії, що передбачають приховування інформації всередині інших даних, наприклад зображень чи звукових файлів.

Завдяки можливостям Python у роботі з об'єктами та модульності можна розділити програму на кілька компонентів: окремі модулі для шифрування, дешифрування, генерації задач та взаємодії з користувачем. Такий підхід сприяє легкому оновленню та масштабуванню гри — наприклад, доданню нових типів головоломок або більш складних рівнів.

Таким чином, вибір Python як основної платформи для створення гри «Криптограф» є оптимальним рішенням, що забезпечує гнучкість у розробці, багатство функціоналу та можливість експериментувати з різноманітними форматами задач.

Візуальний інтерфейс гри «Криптограф» ми побудували, використовуючи бібліотеку Tkinter, яка є вбудованою в Python та забезпечує просту і зручну платформу для створення графічних інтерфейсів додатків.

Tkinter дозволяє створювати різні графічні елементи, такі як кнопки, текстові поля, етикетки, панелі меню, випадаючі списки та інші компоненти, необхідні для інтерактивного управління ігровим процесом. У нашому проекті ми використовували базові можливості цієї бібліотеки для реалізації основних функцій гри, таких як введення даних, відображення результатів криптографічних обчислень і взаємодія з користувачем.

Завдяки Tkinter ми організували логічну структуру гри: створили головне вікно, в якому розміщені всі елементи, необхідні для управління грою. Наприклад, передбачені кнопки для запуску обчислень, очищення полів введення та виходу з програми. Окрім того, графічний інтерфейс містить текстові поля для введення вихідних даних і відображення результатів.

Tkinter також дозволив нам стилізувати інтерфейс гри, налаштувавши кольори, шрифти та розміри елементів, щоб забезпечити естетичну привабливість і зручність використання. Ми врахували потреби користувачів різних вікових груп, зробивши інтерфейс максимально зрозумілим і інтуїтивним. Подібний підхід до розробки GUI за допомогою Tkinter є оптимальним для швидкого прототипування та створення невеликих програм з широкими можливостями інтерактивної взаємодії.

3.2 Приклади роботи комп’ютерної гри-головоломки «Криптограф»

Гра починається з авторизації (рис. 1.1). Так як мережева частина гри ще не реалізована, то для авторизації достатньо ввести ім’я користувача.

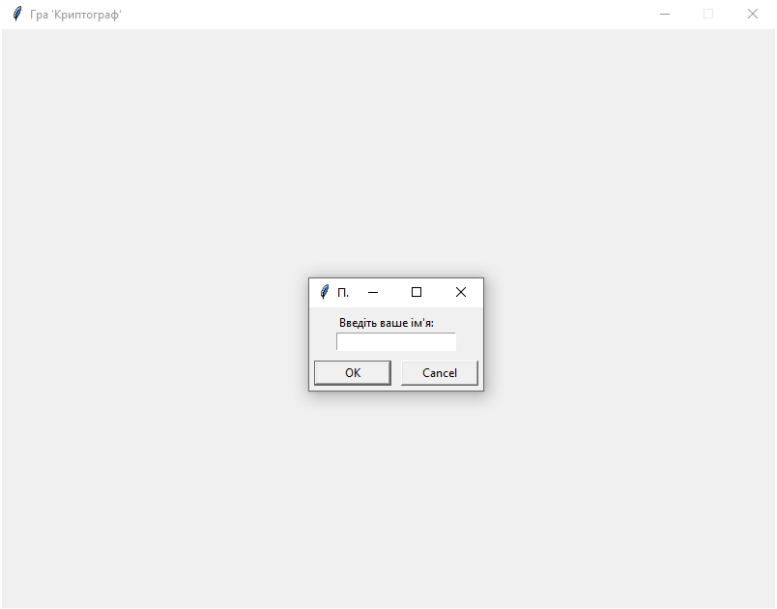

Рисунок 3.1 – Авторизація в грі

Після авторизації гравець попадає в головне меню гри (рис. 3.2).

В головному меню присутні такі пункти як:

– Почати гру

– Статистика

– Правила гри

– Вийти

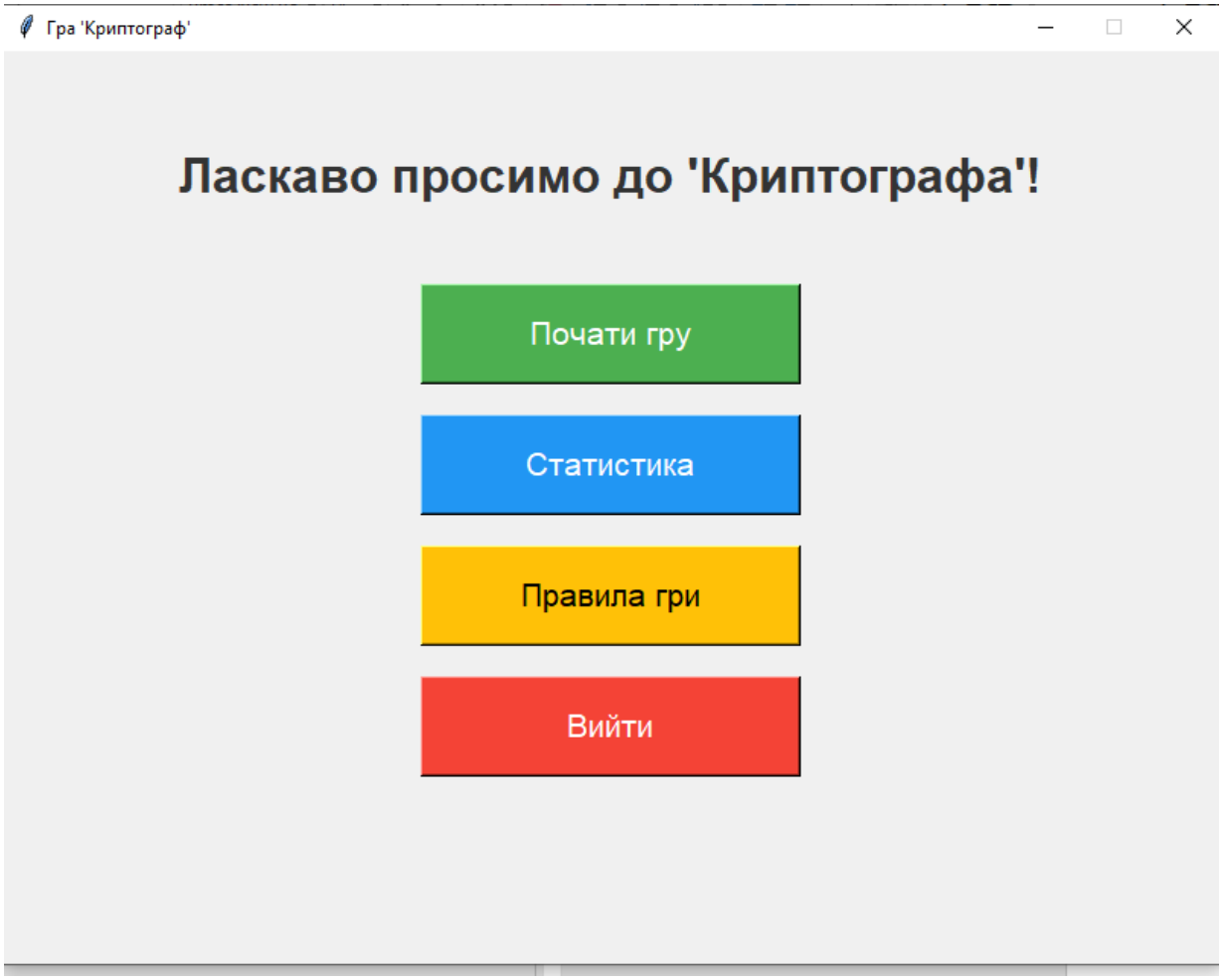

Рисунок 3.2 – Головне меню гри

Для гравця, який вперше запускає гру, буде важливо попередньо ознайомитись з її правилами.

Сторінка з правилами гри показана на рисунку 3.3.

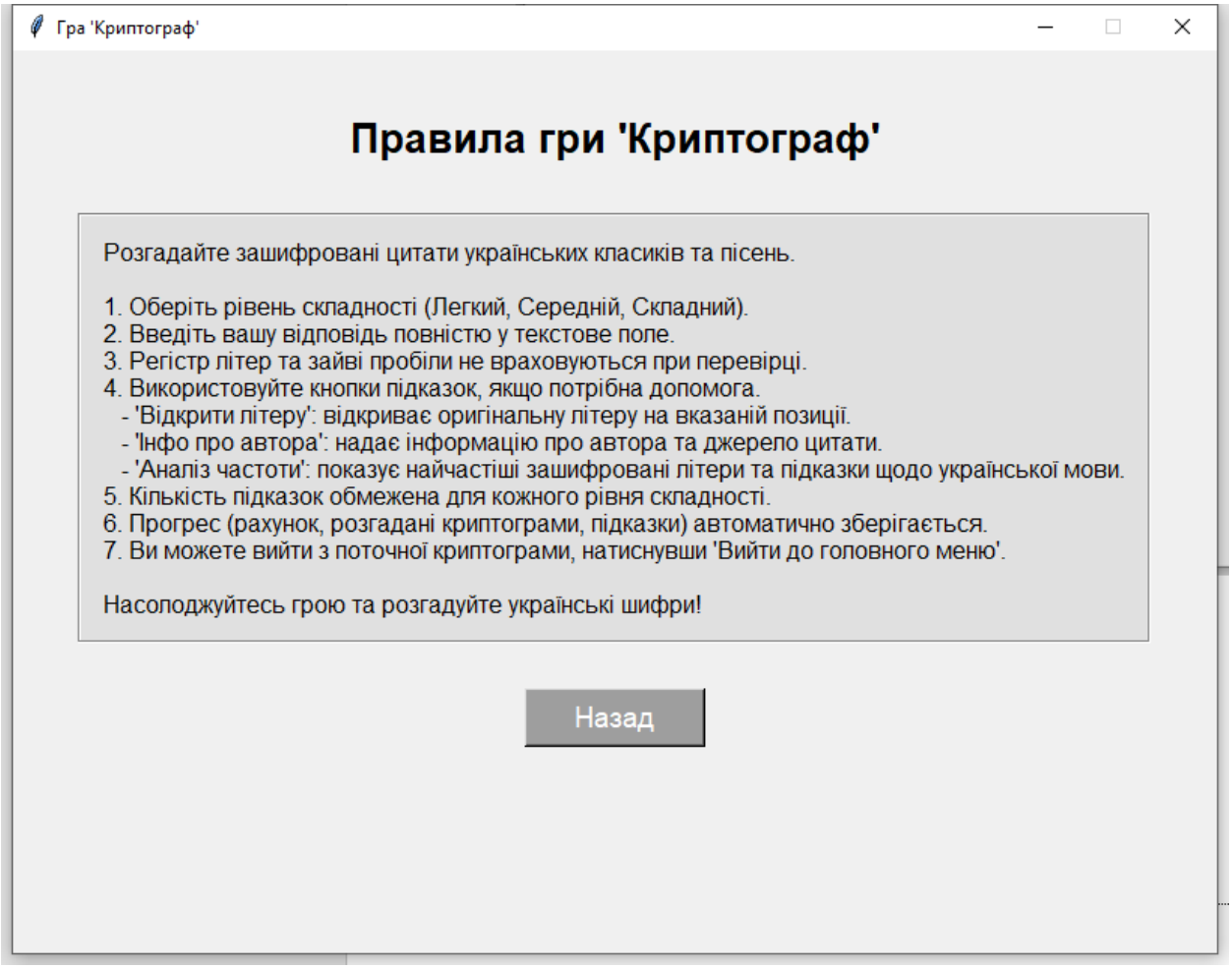

Рисунок 3.3 – Правила гри

Ознайомившись з правилами, можна перейти до самої гри.

Почавши нову гру, гравцю пропонується перш за все обрати складність головоломки (рис. 3.4).

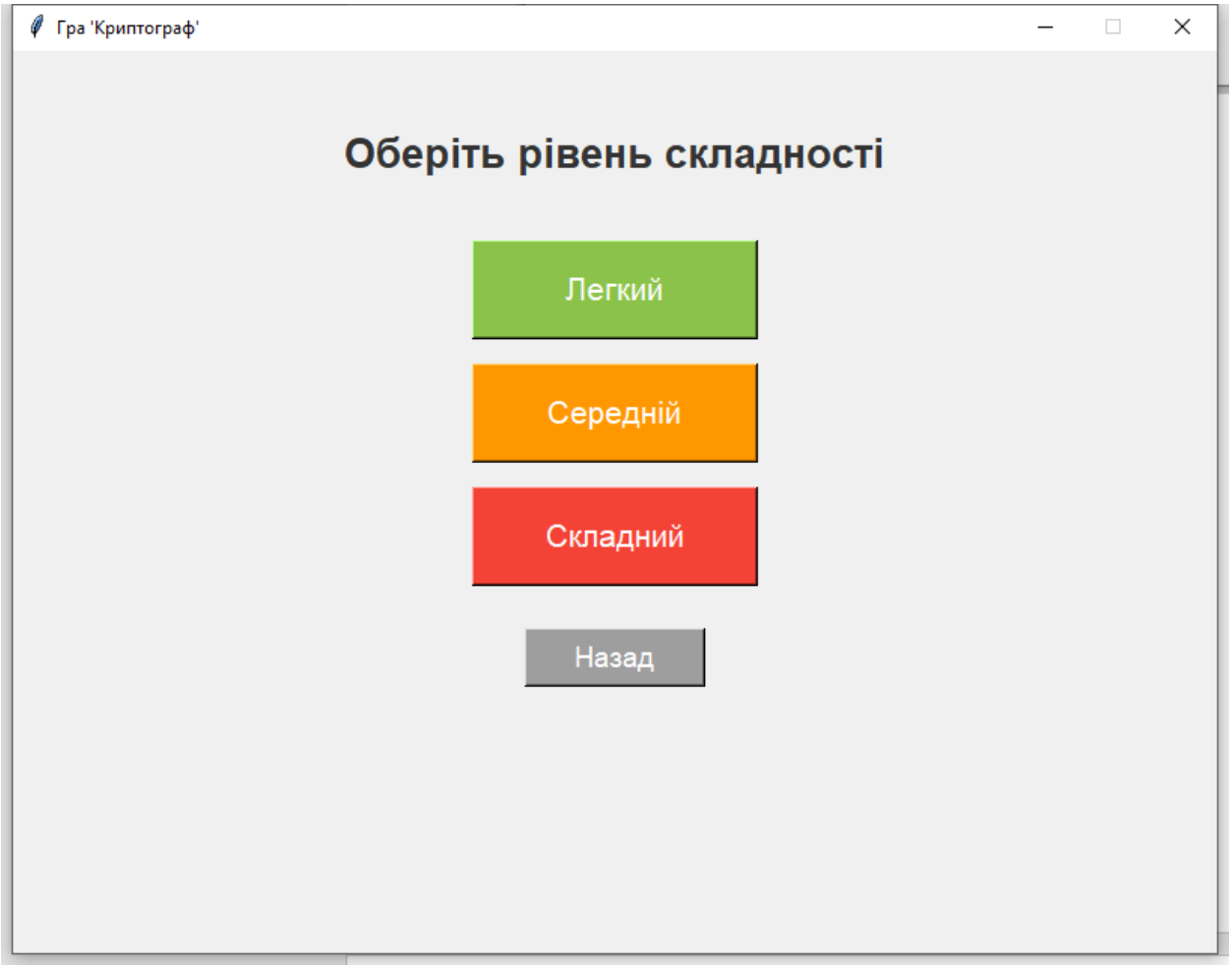

Рисунок 3.4 – Вибір складності головоломки

Після вибору рівня складності, відображається сама головоломка (рис. 3.5).

На екрані розгадування головоломки можна побачити саму закодовану фразу, та поле для введення відгаданої фрази.

Під час відгадування можна використовувати наступні підказки:

– відкрити літеру (показує, яка літера зашифрована з обраною гравцем літерою шифрограми;

– Інформація про автора (значно полегшити вгадування може інформація про автора цитати, що зашифрована)

– Аналіз частоти (основним криптоаналітичним методом для таких головоломок є частотний аналіз, тож ця підказка надає інформацію, які літери

в криптограмі має найбільші частоти. Так їх легше зіставити з реальними літерами у фразі.

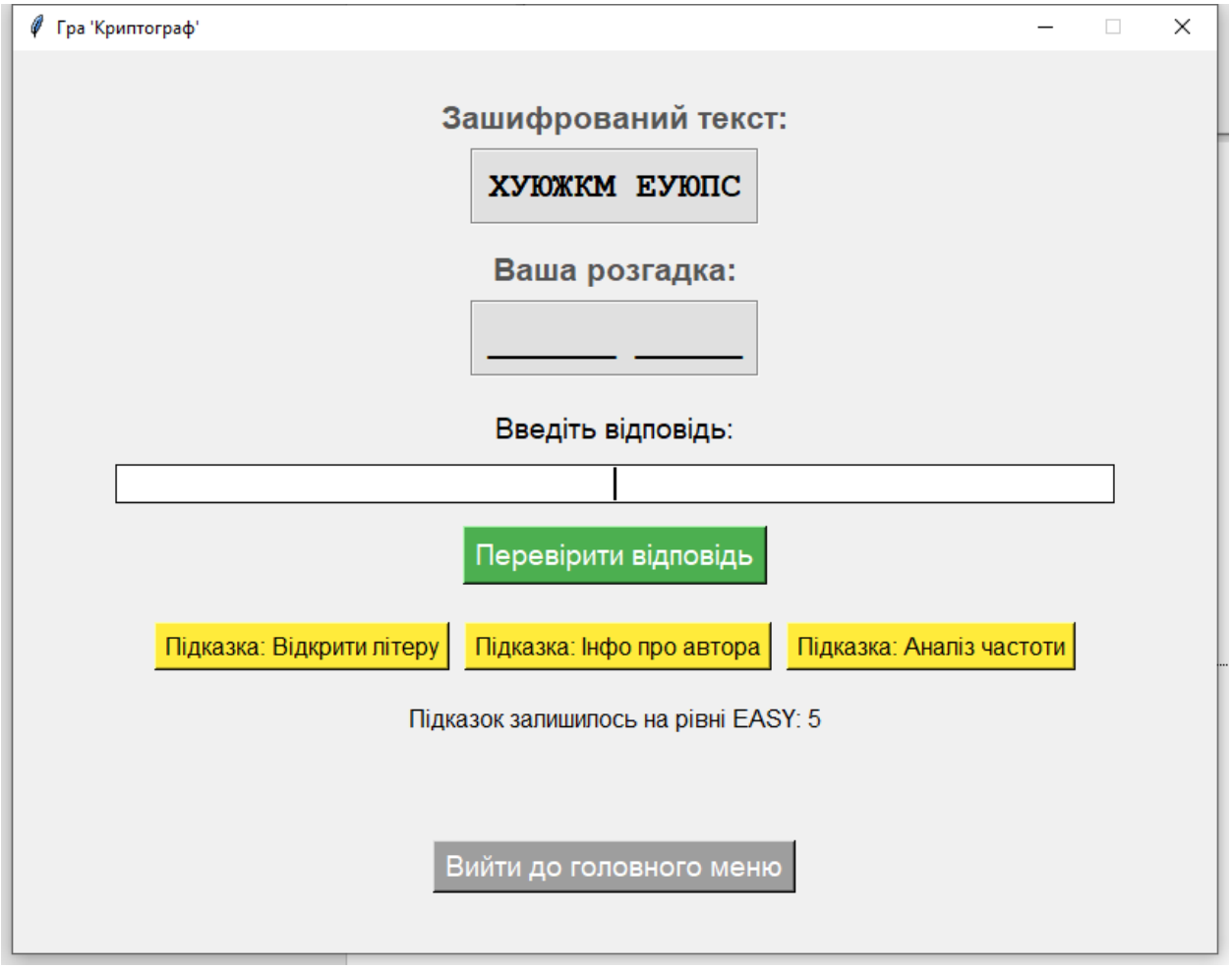

Рисунок 3.5 – Екран розгадування головоломки

На рисунку 3.6 відображенно повідомлення з підказкою про автора.

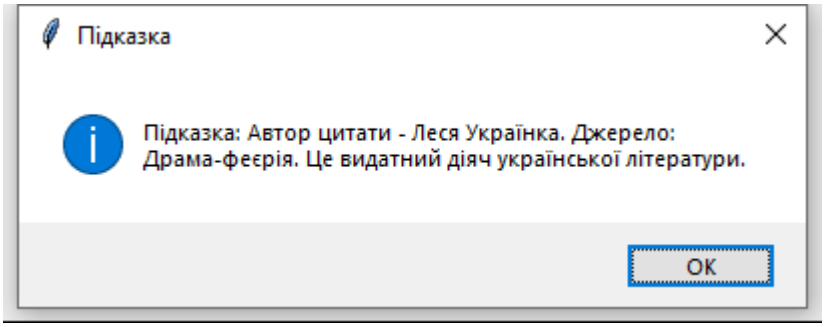

Рисунок 3.6 – Підказка щодо автора

На рисунку 3.7 показана підказка з розподілом частоти літер в криптограмі.

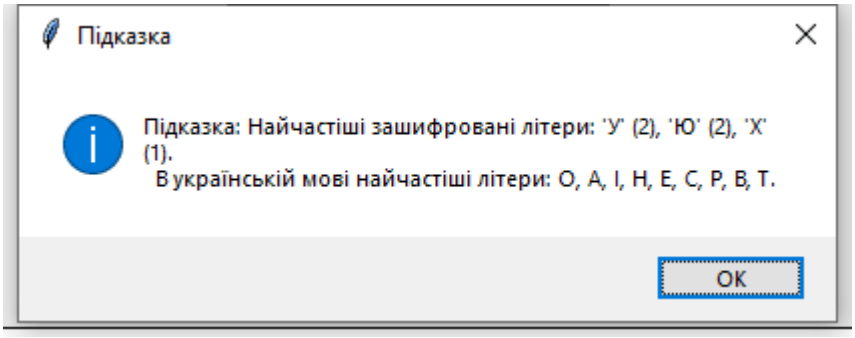

Рисунок 3.7 – Аналіз частоти

А на рисунках 3.8 та 3.9 можна побачити як реалізована підказка з відкриттям обраної літери.

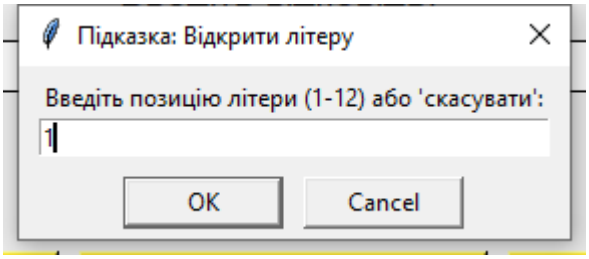

Рисунок 3.8 – Вибір літери, яку потрібно відкрити

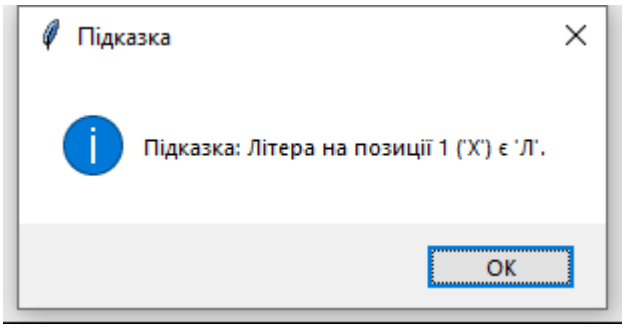

Рисунок 3.9 – Відображення літери, що замінена на обрану

Реалізована також можливість відгадувати не всю фразу одразу, а пробувати по окремим літерам, що може спростити вгадування та зменшити кількість невдалих спроб (рис. 3.10).

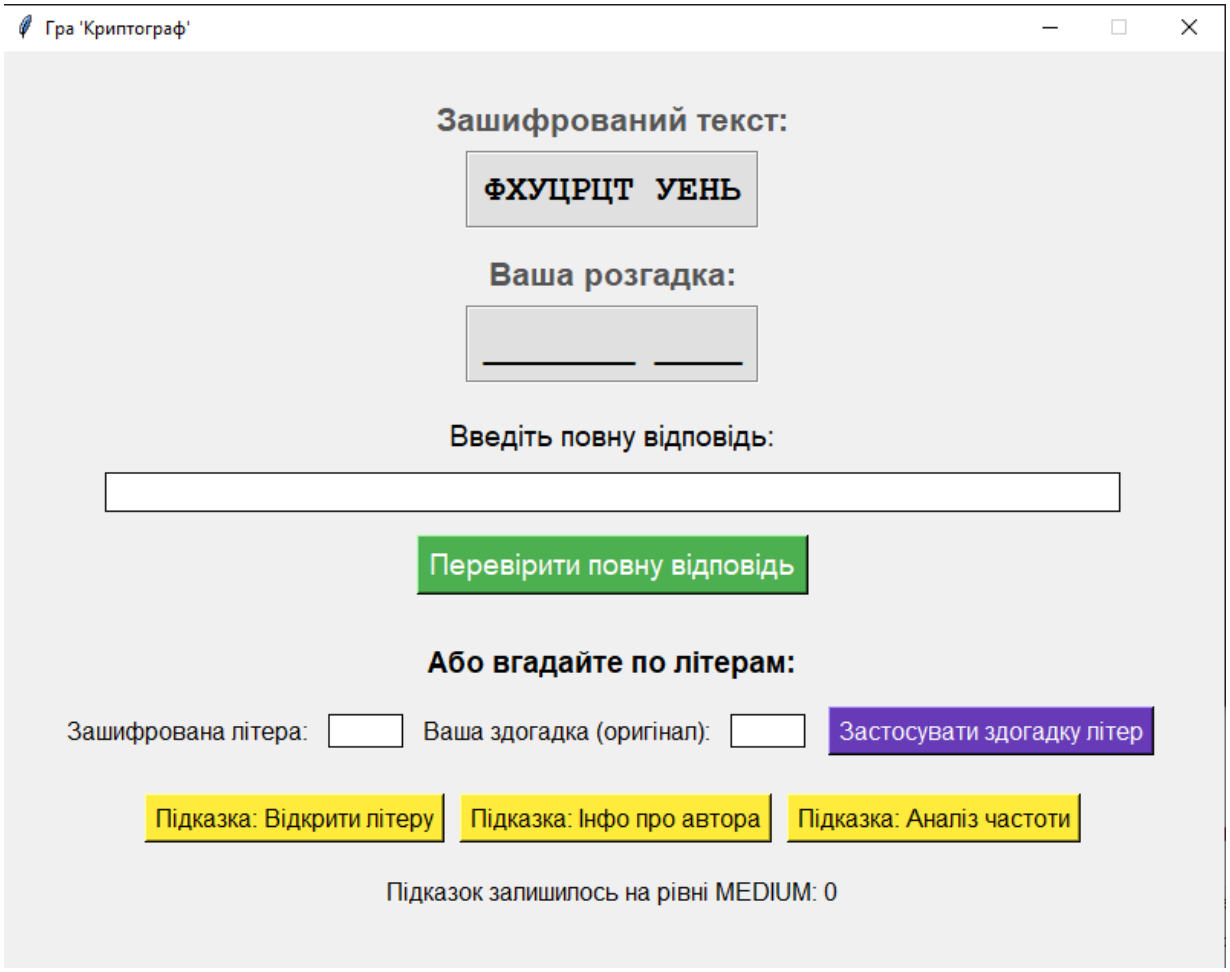

Рисунок 3.10 – Можливість відгадування фрази по окремим літерам

Результати ігор зберігаються, а бали, що отримав гравець, накопичуються.

Статистика гравця відображається в окремому вікні, що показано на рисунку 3.11.

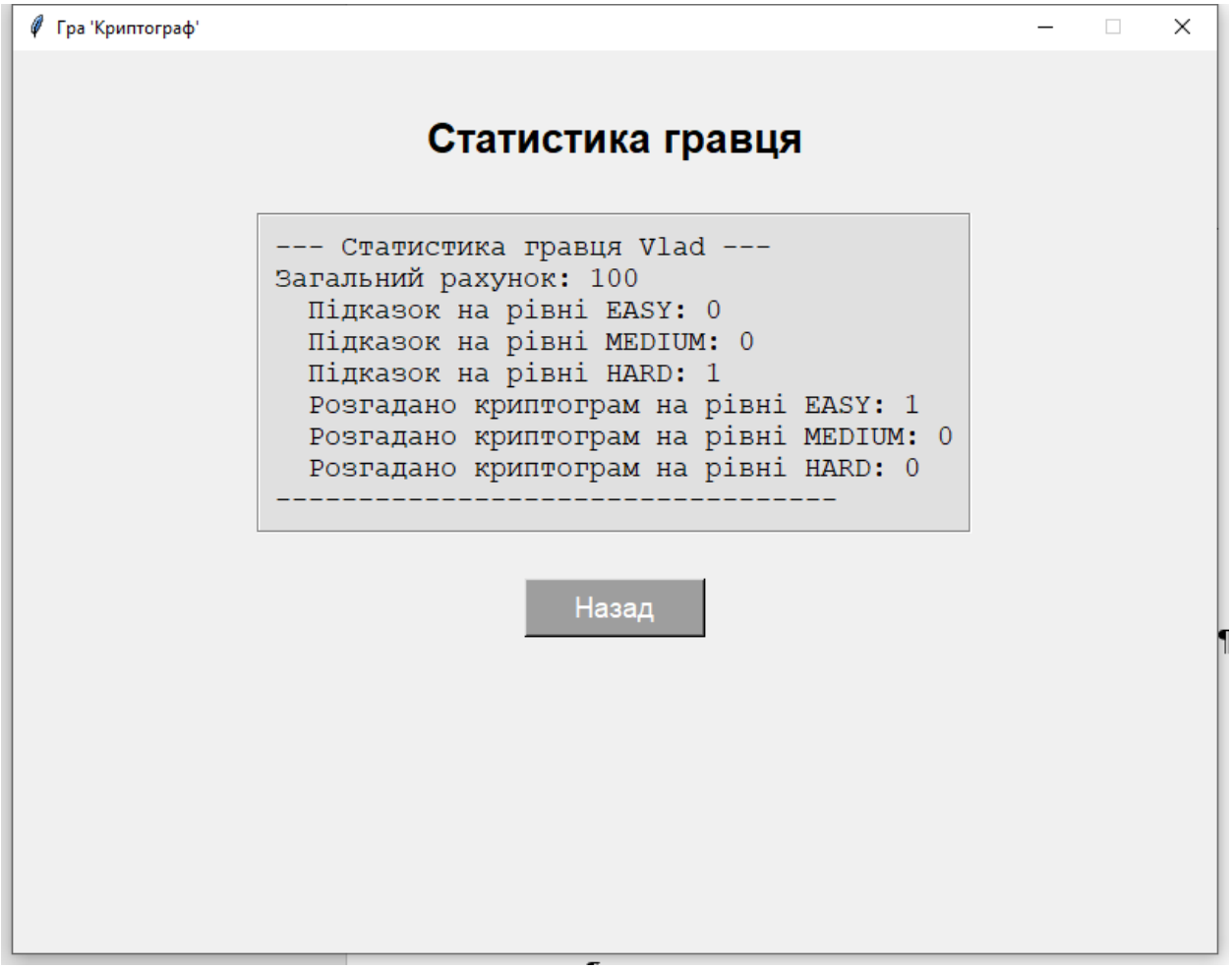

Рисунок 3.11 – Статистика гравця

Створене базове програмне забезпечення можна розвивати та доповнювати. Наприклад, реалізувати інші типи головоломок та додати мережевий режим, що надасть можливість створити веб та мобільний варіант гри.

ВИСНОВКИ

В нашій кваліфікаційній роботі було спроєктовано та реалізовано комп’ютерну гру «Криптограф», яка орієнтована на розвиток логічного мислення та опанування базових принципів криптографії. Основою механіки гри є розгадування криптографічних головоломок, заснованих на підстановочних шифрах. Гра пропонує користувачам різноманітні завдання, в яких необхідно дешифрувати текст, отримуючи ключ до шифру шляхом аналізу символів і логічних взаємозв’язків.

Проєкт передбачає систему рівнів, яка поступово ускладнюється, стимулюючи інтелектуальний ріст гравця. У процесі проходження гри користувач отримує базові знання про історію криптографії, її значення в інформаційній безпеці та практичне застосування.

Особливість гри полягає в тому, що в криптограмах використані цитати видатних українських письменників та поетів, які є символами національної культури й духовності, а також рядки з популярних сучасних українських пісень, що вже стали частиною культурного надбання. Цей підхід не тільки робить гру цікавою та захопливою, але й сприяє популяризації української літератури та музики серед гравців. Гравець отримує можливість розгадувати загадкові тексти, поглиблюючи свої знання культурної спадщини, відкриваючи для себе нові ім'я та твори, що збагачують його світогляд і надихають. Таким чином, інтерактивний формат гри перетворюється на спосіб культурного просвітництва, об'єднуючи приємне проведення часу із пізнавальною складовою.

Створене базове програмне забезпечення може бути основою для подальшого масштабування та розвитку, залежно від ідей та потреб користувачів. Наприклад, можна розширити функціонал, реалізувавши інші типи головоломок, таких як логічні задачі, головоломки на швидкість вирішення або багаторівневі квести зі складним сюжетом. Це дозволить значно урізноманітнити ігровий процес.

Окрім цього, можна додати мережевий режим, що забезпечить можливість гри в реальному часі з іншими користувачами, включаючи режим змагання, де гравці змагатимуться за найшвидше вирішення задач або співпрацюватимуть для досягнення спільної мети. Такий функціонал відкриває перспективу для інтеграції системи рейтингу, досягнень та нагород, що збільшить зацікавленість та залученість гравців.

Для кращої доступності варто створити веб-версію гри, яка працюватиме без необхідності завантажувати додаткові файли, а також мобільний варіант, оптимізований для платформ iOS та Android. Це забезпечить широке охоплення аудиторії, дозволяючи користувачам насолоджуватися грою з будь-якого пристрою та місця. Мобільна версія також може включати інтеграцію з соціальними мережами або функції сповіщень, щоб залучати гравців до активного використання.

Таким чином, розвиток базового програмного забезпечення може перетворити його на багатофункціональну платформу, яка не лише задовольнить потреби існуючих користувачів, але й сприятиме залученню нових.

У результаті реалізації проєкту вдалося створити інтерактивний інструмент, який не лише розважальний, але й освітній, сприяючи популяризації криптографії серед широкого кола користувачів. Такий підхід демонструє, як сучасні технології можуть служити ефективним засобом навчання, забезпечуючи глибоке занурення в тему через ігровий процес та практичні завдання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Історія криптографії – URL: https://uk.wikipedia.org/wiki/%D0%86%D1%81%D1%82%D0%BE%D1%80%D1%96%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%97

2. Криптографія: Від давніх шифрів до блокчейну. Повний посібник з інформаційної безпеки у цифровому світі – URL: https://blog.mexc.com/uk/what-is-cryptography-guide-ru/

3. Криптографія від Середніх століть до Нового часу – URL: https://wiki.cusu.edu.ua/index.php/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F_%D0%B2%D1%96%D0%B4_%D0%A1%D0%B5%D1%80%D0%B5%D0%B4%D0%BD%D1%96%D1%85_%D1%81%D1%82%D0%BE%D0%BB%D1%96%D1%82%D1%8C_%D0%B4%D0%BE_%D0%9D%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D1%87%D0%B0%D1%81%D1%83

4. Криптографія від історії до сучасних стандартів – URL: https://eir.zp.edu.ua/server/api/core/bitstreams/6189646a-7b55-442a-8100-da5e56c96f50/content

5. Класичні методи криптології: методичні рекомендації для здобувачів спеціальностей «Прикладна математика» та «Системний аналіз» / М.М. Повідайчик, І.Я. Шпонтак. Ужгород: Видавництво УжНУ «Говерла», 2020. 28 с.

Шифрування та дешифрування шифром Цезаря, Вернама , Віженера – URL: https://teteryakv12.wordpress.com/2014/10/07/%D1%88%D0%B8%D1%84%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B4%D0%B5%D1%88%D0%B8%D1%84%D1%80%D1%83%D0%B2%D

0%B0%D0%BD%D0%BD%D1%8F-%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%BC-%D1%86%D0%B5/

6. Шифр Віженера – URL: https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80_%D0%92%D1%96%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B0

7. Cryptograms · Puzzle Quotes – URL: https://play.google.com/store/apps/details/Cryptograms_Decrypt_Quotes?id=com.razzlepuzzles.cryptograms&hl=uk

8. Brain Cryptogram – URL: https://play.google.com/store/apps/details?id=game.pondol.cryptogram.app&hl=uk

9. Cryptogram - Word Puzzle Games – URL: https://play.google.com/store/apps/details?id=cryptogram.cypher.decrypt.code.word.puzzle.number&hl=uk

10. Decipherism – URL: https://quasilyte.itch.io/decipherism

11. Kryptic – URL: https://foxchaotica.itch.io/kryptic

12. tkinter — Python interface to Tcl/Tk – URL: https://docs.python.org/3/library/tkinter.html

Додаток А

Текст програми

```
import random
import string
import json
import os
import tkinter as tk
from tkinter import messagebox, simpledialog

# --- Constants for Ukrainian alphabet and save file ---
UKRAINIAN_ALPHABET = "АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ"
SAVE_FILE = "player_data.json"

# --- Enums ---
from enum import Enum

class Difficulty(Enum):
    EASY = 1
    MEDIUM = 2
    HARD = 3

class CipherType(Enum):
    CAESAR = 1
    SUBSTITUTION = 2

class HintType(Enum):
    REVEAL_LETTER = 1
    AUTHOR_INFO = 2
    FREQUENCY_ANALYSIS = 3


# --- Cryptogram Class ---
class Cryptogram:
    def __init__(self, original_text, author, source,
difficulty, cipher_type, id=None):
        self.original_text = original_text.upper()
        self.author = author
        self.source = source
        self.difficulty = difficulty
        self.cipher_type = cipher_type
        self.id = id if id else str(random.randint(10000,
99999))
        self.encrypted_text = self._encrypt_text()

    def _normalize_text_for_cipher(self, text):
        """Removes everything except Ukrainian alphabet
letters from text."""
        return "".join(char for char in text if char in
UKRAINIAN_ALPHABET)
```

```
        def _encrypt_text(self):
            """
            Private method for encrypting text based on cipher
type.
            Symbols that are not Ukrainian alphabet letters
remain unchanged.
            """
            if self.cipher_type == CipherType.CAESAR:
                shift = 0
                if self.difficulty == Difficulty.EASY:
                    shift = random.randint(1, 5)
                elif self.difficulty == Difficulty.MEDIUM:
                    shift = random.randint(6, 15)
                elif self.difficulty == Difficulty.HARD:
                    shift = random.randint(16, 32)
                return   self._caesar_cipher(self.original_text,
shift)
            elif self.cipher_type == CipherType.SUBSTITUTION:
                return
self._substitution_cipher(self.original_text)
            else:
                return self.original_text

        def _caesar_cipher(self, text, shift):
            """Caesar cipher implementation for the Ukrainian
alphabet."""
            result = ""
            alphabet_len = len(UKRAINIAN_ALPHABET)
            for char in text:
                if char in UKRAINIAN_ALPHABET:
                    original_pos = UKRAINIAN_ALPHABET.find(char)
                    new_pos   =   (original_pos   +   shift)   %
alphabet_len
                    result += UKRAINIAN_ALPHABET[new_pos]
                else:
                    result += char
            return result

        def _substitution_cipher(self, text):
            """
            Simple substitution cipher implementation for the
Ukrainian alphabet.
            Generates a fixed letter mapping for a specific
cryptogram.
            """
            alphabet_list = list(UKRAINIAN_ALPHABET)
            shuffled_alphabet = list(UKRAINIAN_ALPHABET)
            random.shuffle(shuffled_alphabet)
            mapping          =          dict(zip(alphabet_list,
shuffled_alphabet))

            result = ""
            for char in text:
```

```
                if char in mapping:
                    result += mapping[char]
                else:
                    result += char
            return result

        def check_answer(self, answer):
            """Checks if the entered answer is correct (case-
insensitive and ignores extra spaces)."""
            return          answer.upper().strip()          ==
self.original_text.strip()

        def __str__(self):
            return (f"--- Cryptogram ---\n"
                    f"Encrypted text: {self.encrypted_text}\n"
                    f"Difficulty: {self.difficulty.name}\n"
                    f"Cipher type: {self.cipher_type.name}\n"
                    f"--------------------")


    # --- Player Class ---
    class Player:
        def __init__(self, name="Гравець"):
            self.name = name
            self.score = 0
            self.hints_available = {
                Difficulty.EASY.name: 5,
                Difficulty.MEDIUM.name: 3,
                Difficulty.HARD.name: 1
            }
            self.solved_cryptograms_ids = {
                Difficulty.EASY.name: [],
                Difficulty.MEDIUM.name: [],
                Difficulty.HARD.name: []
            }
            self.load_progress()

        def update_score(self, points):
            self.score += points

        def use_hint(self, difficulty_name):
            if self.hints_available[difficulty_name] > 0:
                self.hints_available[difficulty_name] -= 1
                return True
            return False

        def    add_solved_cryptogram_id(self,    cryptogram_id,
difficulty_name):
            if          cryptogram_id          not          in
self.solved_cryptograms_ids[difficulty_name]:

self.solved_cryptograms_ids[difficulty_name].append(cryptogram_i
d)
```

```
        def get_stats(self):
            stats = f"--- Статистика гравця {self.name} ---\n"
            stats += f"Загальний рахунок: {self.score}\n"
            for diff_name, count in self.hints_available.items():
                stats += f"   Підказок на рівні {diff_name}:
{count}\n"
            for diff_name, solved_ids in
self.solved_cryptograms_ids.items():
                stats += f"   Розгадано криптограм на рівні
{diff_name}: {len(solved_ids)}\n"
            stats += "-----------------------------------"
            return stats

        def save_progress(self):
            """Saves player progress to a JSON file."""
            data = {
                "name": self.name,
                "score": self.score,
                "hints_available": self.hints_available,
                "solved_cryptograms_ids":
self.solved_cryptograms_ids
            }
            try:
                with open(SAVE_FILE, 'w', encoding='utf-8') as f:
                    json.dump(data, f, ensure_ascii=False,
indent=4)
                print("Прогрес збережено.") # Console print for
debugging
            except IOError as e:
                print(f"Помилка збереження прогресу: {e}") #
Console print for debugging

        def load_progress(self):
            """Loads player progress from a JSON file."""
            if os.path.exists(SAVE_FILE):
                try:
                    with open(SAVE_FILE, 'r', encoding='utf-8')
as f:
                        data = json.load(f)
                        self.name = data.get("name", self.name)
                        self.score = data.get("score",
self.score)
                        self.hints_available =
data.get("hints_available", self.hints_available)
                        self.solved_cryptograms_ids =
data.get("solved_cryptograms_ids", self.solved_cryptograms_ids)
                    print("Прогрес завантажено.") # Console print
for debugging
                except (IOError, json.JSONDecodeError) as e:
                    print(f"Помилка завантаження прогресу: {e}.
Починаємо нову гру.") # Console print for debugging
            else:
```

```
                print("Файл збереження не знайдено. Починаємо
нову гру.") # Console print for debugging

    # --- Hint Class ---
    class Hint:
        def __init__(self, hint_type, cost=1):
            self.type = hint_type
            self.cost = cost

        def  apply_hint(self,  cryptogram,  current_guess_list,
pos=None):
            """
            Applies a hint to the cryptogram, returns a message.
            Can  update  current_guess_list  (list  of  characters
representing the player's current guess).
            'pos' is used for REVEAL_LETTER hint to specify the
position.
            """
            if self.type == HintType.REVEAL_LETTER:
                if pos is None: # If position is not provided,
reveal a random unsolved letter
                    unsolved_indices = [i for i, (e_char, o_char)
in                      enumerate(zip(cryptogram.encrypted_text,
cryptogram.original_text))
                                        if       e_char       in
UKRAINIAN_ALPHABET and current_guess_list[i] != o_char]
                    if not unsolved_indices:
                        return "Всі літери вже розгадані, або
немає букв для підказки!"
                    pos = random.choice(unsolved_indices)

                encrypted_char = cryptogram.encrypted_text[pos]
                original_char = cryptogram.original_text[pos]

                if encrypted_char not in UKRAINIAN_ALPHABET:
                    return "Це не буква. Оберіть іншу позицію."

                if current_guess_list[pos] == original_char:
                    return "Ця літера вже розгадана. Оберіть іншу
позицію."

                current_guess_list[pos] = original_char
                return  f"Підказка: Літера на позиції  {pos+1}
('{encrypted_char}') є '{original_char}'."

            elif self.type == HintType.AUTHOR_INFO:
                return     (f"Підказка:     Автор     цитати     -
{cryptogram.author}. "
                        f"Джерело:    {cryptogram.source}.    Це
видатний діяч української літератури.")

            elif self.type == HintType.FREQUENCY_ANALYSIS:
                char_counts = {}
```

```
                for char in cryptogram.encrypted_text:
                    if char in UKRAINIAN_ALPHABET:
                        char_counts[char]                          =
char_counts.get(char, 0) + 1

                if not char_counts:
                    return "Немає літер для аналізу частоти."

                sorted_chars    =    sorted(char_counts.items(),
key=lambda item: item[1], reverse=True)
                top_3_encrypted    =    ",    ".join([f"'{char}'
({count})" for char, count in sorted_chars[:3]])

                return   (f"Підказка:   Найчастіші   зашифровані
літери: {top_3_encrypted}.\n"
                        f"  В українській мові найчастіші літери:
О, А, І, Н, Е, С, Р, В, Т.")

            return "Невідомий тип підказки."


    # --- GameManager Class ---
    class GameManager:
        def __init__(self):
            self.player = Player()
            self.cryptograms_data                                  =
self._load_initial_cryptograms()
            self.current_cryptogram = None
            self.current_difficulty = None
            self.current_guess = []

        def _load_initial_cryptograms(self):
            """
            Loads initial cryptograms. This simulates loading
from a database.
            In a real project, this could be a JSON file,
database, etc.
            Adding more variety, especially for harder levels.
            """
            cryptograms = {
                Difficulty.EASY: [
                    Cryptogram("ДУМИ МОЇ ДУМИ", "Тарас Шевченко",
"Поезія", Difficulty.EASY, CipherType.CAESAR, id="easy-1"),
                    Cryptogram("ЛІСОВА ПІСНЯ", "Леся Українка",
"Драма-феєрія",     Difficulty.EASY,     CipherType.SUBSTITUTION,
id="easy-2"),
                    Cryptogram("ЩЕ НЕ ВМЕРЛА УКРАЇНА", "Павло
Чубинський",    "Гімн",    Difficulty.EASY,    CipherType.CAESAR,
id="easy-3"),
                    Cryptogram("КОБЗАР",    "Тарас    Шевченко",
"Збірка   поезій",   Difficulty.EASY,   CipherType.SUBSTITUTION,
id="easy-4"),
```

```
                    Cryptogram("ВІЧНИЙ РЕВОЛЮЦІОНЕР", "Іван
Франко", "Поезія", Difficulty.EASY, CipherType.CAESAR, id="easy-
5"),
                ],
                Difficulty.MEDIUM: [
                    Cryptogram("ВЕЛИКИЙ ЛЬОХ", "Тарас Шевченко",
"Поема", Difficulty.MEDIUM, CipherType.SUBSTITUTION, id="medium-
1"),
                    Cryptogram("ЗАХАР БЕРКУТ", "Іван Франко",
"Повість", Difficulty.MEDIUM, CipherType.CAESAR, id="medium-2"),
                    Cryptogram("ЧЕРВОНА КАЛИНА", "Народна
пісня", "Пісня", Difficulty.MEDIUM, CipherType.SUBSTITUTION,
id="medium-3"),
                    Cryptogram("МОЙСЕЙ", "Іван Франко", "Поема",
Difficulty.MEDIUM, CipherType.CAESAR, id="medium-4"),
                    Cryptogram("СТОЮ НА СКЕЛІ", "Леся Українка",
"Поезія", Difficulty.MEDIUM, CipherType.SUBSTITUTION, id="medium-
5"),
                ],
                Difficulty.HARD: [
                    Cryptogram("СОНЦЕ ВСТАЄ ЗА ГОРОЮ", "Ліна
Костенко", "Поезія", Difficulty.HARD, CipherType.CAESAR,
id="hard-1"),
                    Cryptogram("Я (РОМАНТИКА)", "Микола
Хвильовий", "Новела", Difficulty.HARD, CipherType.SUBSTITUTION,
id="hard-2"),
                    Cryptogram("ТІНІ ЗАБУТИХ ПРЕДКІВ", "Михайло
Коцюбинський", "Повість", Difficulty.HARD, CipherType.CAESAR,
id="hard-3"),
                    Cryptogram("КАТЕРИНА", "Тарас Шевченко",
"Поема", Difficulty.HARD, CipherType.SUBSTITUTION, id="hard-4"),
                    Cryptogram("КАМЕНЯРІ", "Іван Франко",
"Поезія", Difficulty.HARD, CipherType.CAESAR, id="hard-5"),
                ]
            }
            return cryptograms

        def select_difficulty(self, choice):
            """Selects difficulty level."""
            try:
                self.current_difficulty =
Difficulty(int(choice))
                return True
            except ValueError:
                return False

        def get_next_cryptogram(self):
            """Selects the next cryptogram for the current
difficulty level."""
            if not self.current_difficulty:
                return None

            difficulty_name = self.current_difficulty.name
```

```
            solved_ids_for_diff                                    =
self.player.solved_cryptograms_ids[difficulty_name]

            available_cryptograms = [
                c             for             c             in
self.cryptograms_data[self.current_difficulty]
                if c.id not in solved_ids_for_diff
            ]

            if not available_cryptograms:
                return None

            self.current_cryptogram                                =
random.choice(available_cryptograms)
            self.current_guess    =    ['_'    if    char    in
UKRAINIAN_ALPHABET      else      char      for      char      in
self.current_cryptogram.original_text]
            return self.current_cryptogram

        def submit_answer(self, answer):
            """Processes the player's entered answer."""
            if not self.current_cryptogram:
                return False

            if self.current_cryptogram.check_answer(answer):
                self.player.update_score(100                       *
self.current_difficulty.value)

self.player.add_solved_cryptogram_id(self.current_cryptogram.id,
self.current_difficulty.name)
                self.player.save_progress()
                return True
            else:
                return False

        def  process_letter_mapping(self,  encrypted_char_input,
guessed_char_input):
            """
            Processes a player's guess for a mapping from an
encrypted character to an original character.
            Updates the current_guess list and checks if the
cryptogram is solved.
            Returns a tuple: (message, is_solved_now)
            """
            if not self.current_cryptogram:
                return "Криптограма не завантажена.", False

            encrypted_char_input = encrypted_char_input.upper()
            guessed_char_input = guessed_char_input.upper()

            if encrypted_char_input not in UKRAINIAN_ALPHABET or
guessed_char_input not in UKRAINIAN_ALPHABET:
```

```
                return "Обидва символи мають бути літерами
українського алфавіту.", False

            correct_mapping_made = False
            incorrect_mapping = False

            for i, encrypted_char in
enumerate(self.current_cryptogram.encrypted_text):
                if encrypted_char == encrypted_char_input:
                    if self.current_cryptogram.original_text[i]
== guessed_char_input:
                        # Apply the correct guess
                        self.current_guess[i] =
guessed_char_input
                        correct_mapping_made = True
                    else:
                        incorrect_mapping = True

            is_solved_now = "".join(self.current_guess) ==
self.current_cryptogram.original_text

            if is_solved_now:
                self.player.update_score(50 *
self.current_difficulty.value) # Smaller reward for letter by
letter

self.player.add_solved_cryptogram_id(self.current_cryptogram.id,
self.current_difficulty.name)
                self.player.save_progress()
                return "!!! ВІТАЄМО! Ви розгадали криптограму по
літерам !!!", True
            elif correct_mapping_made:
                return "Літери успішно замінено!", False
            elif incorrect_mapping:
                return "Ця здогадка невірна для деяких літер.",
False
            else:
                return "Такої зашифрованої літери не знайдено,
або здогадка не змінила текст.", False


        def request_hint(self, hint_type, pos=None):
            """Applies a hint."""
            if not self.current_cryptogram:
                return "Криптограма не завантажена."

            difficulty_name = self.current_difficulty.name
            if not self.player.use_hint(difficulty_name):
                return "У вас немає доступних підказок для цього
рівня складності."

            hint = Hint(hint_type)
```

```
            # For REVEAL_LETTER hint, 'pos' will be passed,
otherwise it's None.
            hint_message                                    =
hint.apply_hint(self.current_cryptogram, self.current_guess, pos)
            self.player.save_progress()
            return hint_message


    # --- Main GUI Application Class ---
    class CryptographerApp:
        def __init__(self, master):
            self.master = master
            master.title("Гра 'Криптограф'")
            master.geometry("800x600") # Set initial window size
            master.resizable(False, False) # Prevent resizing for
simpler layout

            self.game_manager = GameManager()

            # Check if player name needs to be set
            if self.game_manager.player.name == "Гравець" and
self.game_manager.player.score      ==      0      and      not
self.game_manager.player.solved_cryptograms_ids[Difficulty.EASY.
name]:
                self._ask_player_name()

            self.frames = {}
            self._create_main_menu_frame()
            self._create_difficulty_frame()
            self._create_game_frame()
            self._create_stats_frame()
            self._create_rules_frame()

            self._show_frame("main_menu")

        def _ask_player_name(self):
            name = simpledialog.askstring("Привіт!", "Введіть
ваше ім'я:", parent=self.master)
            if name:
                self.game_manager.player.name = name
                self.game_manager.player.save_progress()
            else:
                messagebox.showinfo("Ім'я гравця", "Ім'я гравця
не введено. Буде використано ім'я за замовчуванням 'Гравець'.")

        def _create_frame(self, name):
            frame = tk.Frame(self.master, padx=20, pady=20,
bg="#F0F0F0")
            self.frames[name] = frame
            frame.grid(row=0, column=0, sticky="nsew") # All
frames occupy the same grid cell
            self.master.grid_rowconfigure(0, weight=1)
            self.master.grid_columnconfigure(0, weight=1)
```

```
            return frame

        def _show_frame(self, name):
            for frame in self.frames.values():
                frame.grid_remove() # Hide all frames
            self.frames[name].grid() # Show the desired frame

        def _create_main_menu_frame(self):
            frame = self._create_frame("main_menu")

            tk.Label(frame,     text="Ласкаво     просимо     до
'Криптографа'!",   font=("Arial",   24,   "bold"),   bg="#F0F0F0",
fg="#333").pack(pady=40)

            tk.Button(frame, text="Почати гру", command=lambda:
self._show_frame("difficulty_select"),    font=("Arial",    16),
width=20,        height=2,        bg="#4CAF50",        fg="white",
relief="raised").pack(pady=10)
            tk.Button(frame,                    text="Статистика",
command=self._show_stats, font=("Arial", 16), width=20, height=2,
bg="#2196F3", fg="white", relief="raised").pack(pady=10)
            tk.Button(frame, text="Правила гри", command=lambda:
self._show_frame("rules"),    font=("Arial",    16),    width=20,
height=2,                bg="#FFC107",                fg="black",
relief="raised").pack(pady=10)
            tk.Button(frame,                         text="Вийти",
command=self._exit_game, font=("Arial", 16), width=20, height=2,
bg="#F44336", fg="white", relief="raised").pack(pady=10)

        def _create_difficulty_frame(self):
            frame = self._create_frame("difficulty_select")

            tk.Label(frame,  text="Оберіть  рівень  складності",
font=("Arial",         20,         "bold"),         bg="#F0F0F0",
fg="#333").pack(pady=30)

            tk.Button(frame,    text="Легкий",    command=lambda:
self._start_game_with_difficulty(Difficulty.EASY), font=("Arial",
16),     width=15,     height=2,     bg="#8BC34A",     fg="white",
relief="raised").pack(pady=8)
            tk.Button(frame,   text="Середній",   command=lambda:
self._start_game_with_difficulty(Difficulty.MEDIUM),
font=("Arial", 16), width=15, height=2, bg="#FF9800", fg="white",
relief="raised").pack(pady=8)
            tk.Button(frame,   text="Складний",   command=lambda:
self._start_game_with_difficulty(Difficulty.HARD), font=("Arial",
16),     width=15,     height=2,     bg="#F44336",     fg="white",
relief="raised").pack(pady=8)
            tk.Button(frame,    text="Назад",    command=lambda:
self._show_frame("main_menu"),   font=("Arial",   14),   width=10,
height=1,                bg="#9E9E9E",                fg="white",
relief="raised").pack(pady=20)
```

```
        def _start_game_with_difficulty(self, difficulty):
            if
self.game_manager.select_difficulty(difficulty.value):
                self._load_next_cryptogram()
                if self.game_manager.current_cryptogram: # Only
switch if a cryptogram was loaded
                    self._show_frame("game_play")
                else: # All cryptograms for this difficulty are
solved
                    messagebox.showinfo("Криптограма",
f"Вітаємо!    Ви    розгадали    всі    криптограми    на    рівні
'{difficulty.name}'!")
                    self._show_frame("main_menu")
            else:
                messagebox.showerror("Помилка",    "Не    вдалося
обрати рівень складності.")


        def _create_game_frame(self):
            frame = self._create_frame("game_play")
            frame.columnconfigure(0, weight=1) # Center content

            tk.Label(frame,      text="Зашифрований      текст:",
font=("Arial",         16,         "bold"),         bg="#F0F0F0",
fg="#555").pack(pady=(10, 0))
            self.encrypted_text_label = tk.Label(frame, text="",
font=("Courier     New",     18,     "bold"),     wraplength=700,
justify="center",    bg="#E0E0E0",    fg="#000",    borderwidth=2,
relief="groove", padx=10, pady=10)
            self.encrypted_text_label.pack(pady=5)

            tk.Label(frame,        text="Ваша        розгадка:",
font=("Arial",         16,         "bold"),         bg="#F0F0F0",
fg="#555").pack(pady=(10, 0))
            self.current_guess_label = tk.Label(frame, text="",
font=("Courier     New",     18,     "bold"),     wraplength=700,
justify="center",    bg="#E0E0E0",    fg="#000",    borderwidth=2,
relief="groove", padx=10, pady=10)
            self.current_guess_label.pack(pady=5)

            # --- Section for guessing the full phrase ---
            tk.Label(frame,  text="Введіть  повну  відповідь:",
font=("Arial", 14), bg="#F0F0F0").pack(pady=(15, 5))
            self.answer_entry   =   tk.Entry(frame,   width=60,
font=("Arial",     14),     justify="center",     relief="solid",
borderwidth=1)
            self.answer_entry.pack(pady=5)
            tk.Button(frame, text="Перевірити повну відповідь",
command=self._submit_answer,  font=("Arial",  14),  bg="#4CAF50",
fg="white", relief="raised").pack(pady=10)

            # --- New section for guessing letter by letter ---
```

```
            tk.Label(frame, text="Або вгадайте по літерам:",
font=("Arial", 14, "bold"), bg="#F0F0F0").pack(pady=(20, 5))

            letter_guess_frame = tk.Frame(frame, bg="#F0F0F0")
            letter_guess_frame.pack(pady=5)

            tk.Label(letter_guess_frame,       text="Зашифрована
літера:", font=("Arial", 12), bg="#F0F0F0").grid(row=0, column=0,
padx=5, pady=5)
            self.encrypted_char_entry                          =
tk.Entry(letter_guess_frame,    width=5,    font=("Arial",    12),
justify="center", relief="solid", borderwidth=1)
            self.encrypted_char_entry.grid(row=0,      column=1,
padx=5, pady=5)

            tk.Label(letter_guess_frame,   text="Ваша   здогадка
(оригінал):",   font=("Arial",   12),   bg="#F0F0F0").grid(row=0,
column=2, padx=5, pady=5)
            self.guessed_char_entry                            =
tk.Entry(letter_guess_frame,    width=5,    font=("Arial",    12),
justify="center", relief="solid", borderwidth=1)
            self.guessed_char_entry.grid(row=0,        column=3,
padx=5, pady=5)

            tk.Button(letter_guess_frame,      text="Застосувати
здогадку       літер",       command=self._submit_letter_mapping,
font=("Arial",        12),        bg="#673AB7",        fg="white",
relief="raised").grid(row=0, column=4, padx=10, pady=5)


            # Hints section (remains the same)
            hint_frame = tk.Frame(frame, bg="#F0F0F0")
            hint_frame.pack(pady=10)

            tk.Button(hint_frame,    text="Підказка:    Відкрити
літеру",                                          command=lambda:
self._request_hint(HintType.REVEAL_LETTER),  font=("Arial",  12),
bg="#FFEB3B", fg="black", relief="raised").grid(row=0, column=0,
padx=5, pady=5)
            tk.Button(hint_frame,   text="Підказка:   Інфо   про
автора",                                          command=lambda:
self._request_hint(HintType.AUTHOR_INFO),   font=("Arial",   12),
bg="#FFEB3B", fg="black", relief="raised").grid(row=0, column=1,
padx=5, pady=5)
            tk.Button(hint_frame,     text="Підказка:     Аналіз
частоти",                                         command=lambda:
self._request_hint(HintType.FREQUENCY_ANALYSIS),   font=("Arial",
12),   bg="#FFEB3B",   fg="black",   relief="raised").grid(row=0,
column=2, padx=5, pady=5)

            self.hints_left_label       =       tk.Label(frame,
text="Підказок залишилось:", font=("Arial", 12), bg="#F0F0F0")
            self.hints_left_label.pack(pady=5)
```

```
            self.message_label    =    tk.Label(frame,    text="",
font=("Arial", 12), fg="blue", bg="#F0F0F0")
            self.message_label.pack(pady=10)

            tk.Button(frame,  text="Вийти  до  головного  меню",
command=self._back_to_main_menu,        font=("Arial",        14),
bg="#9E9E9E", fg="white", relief="raised").pack(pady=20)


        def _load_next_cryptogram(self):
            cryptogram = self.game_manager.get_next_cryptogram()
            if cryptogram:

self.encrypted_text_label.config(text=cryptogram.encrypted_text)

self.current_guess_label.config(text="".join(self.game_manager.c
urrent_guess))
                self.answer_entry.delete(0, tk.END)
                self.encrypted_char_entry.delete(0,   tk.END)   #
Clear new entry fields
                self.guessed_char_entry.delete(0, tk.END)
                self.message_label.config(text="")
                self._update_hints_label()
            else:
                messagebox.showinfo("Криптограма", f"Вітаємо! Ви
розгадали         всі         криптограми         на         рівні
'{self.game_manager.current_difficulty.name}'!")
                self._show_frame("main_menu")

        def _submit_answer(self):
            answer = self.answer_entry.get()
            if not answer:
                self.message_label.config(text="Будь      ласка,
введіть відповідь.", fg="red")
                return

            if self.game_manager.submit_answer(answer):
                self.message_label.config(text="!!!     ВІТАЄМО!
ВІДПОВІДЬ ПРАВИЛЬНА !!!", fg="green")
                self._update_hints_label() # Update hints label
as a result of potential score change
                self.master.after(1500,
self._load_next_cryptogram) # Load next after 1.5 seconds
            else:
                self.message_label.config(text="Невірна
відповідь. Спробуйте ще раз.", fg="red")

        def _submit_letter_mapping(self):
            encrypted_char                                      =
self.encrypted_char_entry.get().strip()
            guessed_char = self.guessed_char_entry.get().strip()
```

```
            # Input validation
            if not encrypted_char or not guessed_char:
                self.message_label.config(text="Введіть обидві
літери.", fg="red")
                return
            if len(encrypted_char) != 1 or len(guessed_char) !=
1:
                self.message_label.config(text="Введіть лише
одну літеру для кожного поля.", fg="red")
                return
            if encrypted_char.upper() not in UKRAINIAN_ALPHABET
or guessed_char.upper() not in UKRAINIAN_ALPHABET:
                self.message_label.config(text="Обидва символи
мають бути літерами українського алфавіту.", fg="red")
                return

            message, is_solved =
self.game_manager.process_letter_mapping(encrypted_char,
guessed_char)

            self.message_label.config(text=message, fg="blue" if
is_solved or "успішно" in message else "red")

self.current_guess_label.config(text="".join(self.game_manager.c
urrent_guess)) # Update guess display
            self.encrypted_char_entry.delete(0, tk.END) # Clear
entries after guess
            self.guessed_char_entry.delete(0, tk.END)
            self._update_hints_label() # Update hints label for
score or other changes

            if is_solved:
                self.master.after(1500,
self._load_next_cryptogram)


        def _request_hint(self, hint_type):
            if hint_type == HintType.REVEAL_LETTER:
                # For REVEAL_LETTER, ask for position via
simpledialog
                pos_str = simpledialog.askstring("Підказка:
Відкрити літеру",
                                                 f"Введіть
позицію літери (1-
{len(self.game_manager.current_cryptogram.encrypted_text)}) або
'скасувати':",

parent=self.master)
                if pos_str and pos_str.lower() != 'скасувати':
                    try:
                        pos = int(pos_str) - 1
                        if not (0 <= pos <
len(self.game_manager.current_cryptogram.encrypted_text)):
```

```
                                messagebox.showerror("Помилка",
"Невірна позиція. Введіть число в межах довжини криптограми.")
                                return  #  Don't  consume  hint  if
position is invalid
                            hint_message                                =
self.game_manager.request_hint(hint_type, pos)
                            messagebox.showinfo("Підказка",
hint_message)

self.current_guess_label.config(text="".join(self.game_manager.c
urrent_guess)) # Update guess display
                            self._update_hints_label()
                        except ValueError:
                            messagebox.showerror("Помилка",
"Невірний ввід. Будь ласка, введіть число.")
                    elif pos_str and pos_str.lower() == 'скасувати':
                        messagebox.showinfo("Підказка",
"Використання підказки скасовано.")
                        return
                    else: # User clicked cancel or closed dialog
                        return

            else: # For other hint types, no position needed
                hint_message                                        =
self.game_manager.request_hint(hint_type)
                messagebox.showinfo("Підказка", hint_message)
                self._update_hints_label()

        def _update_hints_label(self):
            if self.game_manager.current_difficulty:
                difficulty_name                                     =
self.game_manager.current_difficulty.name
                hints_left                                          =
self.game_manager.player.hints_available[difficulty_name]
                self.hints_left_label.config(text=f"Підказок
залишилось на рівні {difficulty_name}: {hints_left}")
            else:
                self.hints_left_label.config(text="Підказок
залишилось:") # Default state

        def _show_stats(self):
            stats_frame = self.frames["stats"]
            for widget in stats_frame.winfo_children(): # Clear
previous stats
                widget.destroy()

            tk.Label(stats_frame,   text="Статистика   гравця",
font=("Arial", 20, "bold"), bg="#F0F0F0").pack(pady=20)
            stats_text = self.game_manager.player.get_stats()
            tk.Label(stats_frame,              text=stats_text,
font=("Courier New", 14), justify="left", bg="#E0E0E0", padx=10,
pady=10, relief="groove").pack(pady=10)
```

```
            tk.Button(stats_frame, text="Назад", command=lambda:
self._show_frame("main_menu"),  font=("Arial",  14),  width=10,
height=1,               bg="#9E9E9E",               fg="white",
relief="raised").pack(pady=20)
            self._show_frame("stats")

        def _create_stats_frame(self):
            # Frame created without content, content filled by
_show_stats
            self._create_frame("stats")

        def _create_rules_frame(self):
            frame = self._create_frame("rules")

            tk.Label(frame,  text="Правила  гри  'Криптограф'",
font=("Arial", 20, "bold"), bg="#F0F0F0").pack(pady=20)
            rules_text = (
                "Розгадайте  зашифровані  цитати  українських
класиків та пісень.\n\n"
                "1. Оберіть рівень складності (Легкий, Середній,
Складний).\n"
                "2. **Варіант 1: Введіть повну відповідь** у
перше текстове поле та натисніть 'Перевірити повну відповідь'.\n"
                "3. **Варіант 2: Вгадайте по літерам.** Введіть
зашифровану літеру та вашу здогадку для неї (наприклад, 'А' та
'Б'), потім натисніть 'Застосувати здогадку літер'. Всі такі
зашифровані літери будуть замінені на вашу здогадку, якщо вона
правильна.\n"
                "4.  Регістр  літер  та  зайві  пробіли  не
враховуються при перевірці.\n"
                "5. Використовуйте кнопки підказок, якщо потрібна
допомога.\n"
                "    - 'Відкрити літеру': відкриває оригінальну
літеру на вказаній позиції.\n"
                "    - 'Інфо про автора': надає інформацію про
автора та джерело цитати.\n"
                "     - 'Аналіз частоти': показує найчастіші
зашифровані літери та підказки щодо української мови.\n"
                "6. Кількість підказок обмежена для кожного рівня
складності.\n"
                "7.  Прогрес  (рахунок,  розгадані  криптограми,
підказки) автоматично зберігається.\n"
                "8.  Ви  можете  вийти  з  поточної  криптограми,
натиснувши 'Вийти до головного меню'.\n\n"
                "Насолоджуйтесь грою та розгадуйте українські
шифри!"
            )
            tk.Label(frame, text=rules_text, font=("Arial", 12),
justify="left",       bg="#E0E0E0",       padx=15,       pady=15,
relief="groove").pack(pady=10)
            tk.Button(frame,    text="Назад",    command=lambda:
self._show_frame("main_menu"),  font=("Arial",  14),  width=10,
```

```
height=1,                bg="#9E9E9E",                fg="white",
relief="raised").pack(pady=20)

        def _back_to_main_menu(self):
            if  messagebox.askyesno("Вихід",  "Ви  впевнені,  що
хочете вийти до головного меню? Прогрес по поточній криптограмі
буде втрачено (якщо вона не розгадана)."):
                self._show_frame("main_menu")

        def _exit_game(self):
            if  messagebox.askyesno("Вихід",  "Ви  впевнені,  що
хочете вийти з гри?"):
                self.game_manager.player.save_progress()
                self.master.destroy()


    if __name__ == "__main__":
        root = tk.Tk()
        app = CryptographerApp(root)
        root.mainloop()
```