**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ**
Фізико-математичний факультет
Кафедра інформатики та прикладної математики

«Допущено до захисту»
Завідувач кафедри
_____ Моісеєнко Н. В.
«___» _____ 2024 р.

Реєстраційний № _____
«___» _____ 2024 р.

# НАВЧАЛЬНО-РОЗВИВАЛЬНА МАТЕМАТИЧНА ГРА ДЛЯ УЧНІВ МОЛОДШОГО ШКІЛЬНОГО ВІКУ

Кваліфікаційна робота студента групи І-20
ступінь вищої освіти «бакалавр»
спеціальності
014.09 Середня освіта (Інформатика)
**Кобзєва Богдана Олександровича**

Керівник
**к. пед. н., доцент Шокалюк С. В.**

Оцінка:
Національна шкала _____
Шкала ECTS ___ Кількість балів _____

Голова ЕК _____ _____
Члени ЕК _____ _____
_____ _____
_____ _____

Кривий Ріг – 2024

# ЗАПЕВНЕННЯ

Я, *Кобзєв Богдан Олександрович*, розумію і підтримую політику Криворізького державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений(а). Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

**ЗМІСТ**

# ВСТУП

У сучасній освіті дедалі більша увага приділяється використанню інноваційних методів і засобів навчання, які сприяють розвитку учнів на всіх етапах навчання. Одним із таких методів є використання ігрових елементів в освітньому процесі. Гра – це не лише спосіб розваги та відпочинку, а й ефективний інструмент для засвоєння знань і розвитку навичок. Ігри в освітньому процесі знижують рівень тривожності учнів, сприяють розвитку логічного мислення та вміння приймати рішення.

У даній роботі розроблено та створено навчально-розвивальну математичну гру для учнів молодшого шкільного віку. Такий підхід дає змогу зробити процес навчання математики цікавим і захопливим, а також сприяє розвитку логічного мислення, засвоєнню математичних понять і навичок. Ігрові методи навчання покращують успішність учнів, роблячи освітній процес більш захоплюючим та ефективним.

Актуальність дослідження «Навчально-розвивальна математична гра для учнів молодшого шкільного віку» зумовлена необхідністю ефективного та цікавого навчання математики в ранньому віці. Математичні навички є основою для подальшого засвоєння більш складних предметів і розвитку аналітичного мислення. Традиційні методи навчання математики можуть бути нудними та мало мотивуючими для дітей. Тому розробка навчально-розвивальної математичної гри, яка поєднує в собі навчальні завдання та ігрові елементи, може значно підвищити інтерес дітей до вивчення математики та сприяти більш ефективному засвоєнню матеріалу.

*Метою* дослідження є розробка та створення навчально-розвивальної математичної гри для учнів молодшого шкільного віку. Ця гра має не тільки допомогти дітям засвоїти базові математичні знання, а й розвинути їх логічне мислення, креативність та навички вирішення проблем.

*Завдання* дослідження:

1) вивчити педагогічні та психологічні аспекти використання ігрових елементів у навчанні**.** Проаналізувати, як ігрові методи впливають на мотивацію, успішність та когнітивні навички учнів;

2) проаналізувати наявні навчальні ігри**.** Вивчити існуючі приклади освітніх ігор, визначити їх переваги та недоліки, і на основі цього сформувати вимоги до нової гри;

3) розробити концепцію навчально-розвивальної математичної гри**.** Визначити цілі та завдання гри, розробити ігрові механіки, сценарій, дизайн персонажів та інтерфейс;

4) технічно реалізувати гру на основі розробленої концепції**.** Використати програмні засоби для створення гри, протестувати її на цільовій аудиторії, внести необхідні корективи;

5) провести тестування гри та оцінити її ефективність у навчальному процесі**.** Зібрати відгуки від учнів та вчителів, оцінити, наскільки гра допомагає у вивченні математики та розвитку необхідних навичок.

*Об'єктом* дослідження є учні молодшого шкільного віку, які перебувають у процесі вивчення математики. Ця вікова група обрана через важливість формування базових математичних навичок та інтересу до навчання на ранніх етапах освітнього процесу.

*Предметом* дослідження є розробка та апробація навчально-розвивальної математичної гри, яка сприятиме ефективному та цікавому засвоєнню математичних знань і навичок у дітей. Гра повинна поєднувати в собі навчальні завдання та ігрові елементи, бути адаптованою до вікових особливостей учнів та відповідати вимогам сучасної освіти.

*Методи дослідження*:

– теоретичний аналіз літератури з педагогіки та психології. Вивчення наукових праць, статей та досліджень, присвячених використанню ігрових методів у навчанні;

– емпіричні методи: спостереження, анкетування, тестування. Збір даних про ефективність використання гри в освітньому процесі, аналіз відгуків учнів та вчителів;

– методологія розробки ігор: проєктування, програмування, тестування. Використання сучасних програмних засобів та підходів для створення інтерактивної навчальної гри.

*Практичне значення одержаних результатів* полягає у створенні навчально-розвивальної математичної гри, яка може бути використана в освітньому процесі для підвищення мотивації та покращення математичних знань учнів молодшого шкільного віку. Гра забезпечує інтерактивний та захоплюючий спосіб вивчення математики, сприяє розвитку логічного мислення та навичок вирішення проблем. Вона може бути впроваджена в навчальні плани шкіл, використовуватись на уроках та в позакласній роботі.

*Структура роботи*. Робота складається з вступу, трьох розділів, висновків та списку використаних джерел.

Перший розділ присвячений теоретичним засадам розроблення навчально-розвивальної математичної гри. У ньому розглянуто основні педагогічні та психологічні аспекти, які необхідно враховувати під час створення такої гри, а також переваги використання ігрових елементів в освітньому процесі.

Другий розділ присвячений проєктуванню навчально-розвивальної математичної гри. У ньому розглянуто етапи проєктування гри: визначення цілей і завдань, вибір тематики та контенту, розробка ігрових механік і правил, створення інтерфейсу та графіки.

Третій розділ присвячений розробці та тестуванню прототипу навчально-розвивальної математичної гри. Описано процес розробки гри, включно з вибором програмного забезпечення, створенням ігрових елементів і контенту, програмуванням та тестуванням гри.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ РОЗРОБЛЕННЯ НАВЧАЛЬНО-РОЗВИВАЛЬНОЇ МАТЕМАТИЧНОЇ ГРИ

## 1.1. Роль ігор в освітньому процесі

Ігрові методи навчання набувають все більшого поширення в сучасній освіті завдяки своїй ефективності у стимулюванні інтересу до навчання та розвитку когнітивних навичок учнів. Використання ігрових методів має суттєвий позитивний вплив на різні аспекти освітнього процесу, включаючи мотивацію, розвиток мислення та емоційний стан учнів.

Вплив ігрових методів на навчання

Однією з основних переваг ігрових методів є підвищення мотивації до навчання. Діти природно схильні до гри, тому інтеграція ігрових елементів у навчальний процес робить його більш захоплюючим і привабливим для них. Ігрова діяльність створює умови для активного навчання, де учні залучені в процес, проявляють ініціативу та самостійно шукають рішення.

З психологічної точки зору, ігрові методи допомагають знизити рівень тривожності та страху перед помилками, що часто супроводжують традиційні форми навчання. Граючи, діти можуть експериментувати, робити помилки та вчитися на них без відчуття стресу. Це створює безпечне навчальне середовище, де учні можуть вільно виражати свої ідеї та досліджувати нові концепції.

Ігрові методи також сприяють розвитку когнітивних навичок, таких як логічне мислення, планування та прийняття рішень. В ігрових ситуаціях учні стикаються з завданнями, що вимагають аналітичного підходу, вирішення проблем та творчого мислення. Наприклад, освітні ігри часто містять завдання на побудову стратегії, вирішення логічних головоломок та аналіз ситуацій, що сприяє всебічному розвитку інтелектуальних здібностей.

Приклади успішних освітніх ігор


Рис. 1.1. Скріншот з гри «Math Blaster»

"Math Blaster" [17] є чудовим прикладом інтеграції математичних завдань у захоплюючий ігровий сюжет. Розроблена та видана компанією Davidson & Associates у США в 1983 році, ця навчальна гра в жанрі аркади дозволяє учням молодшого та середнього шкільного віку виступати в ролі космічних героїв, які рятують галактику від небезпек, вирішуючи математичні задачі.

Завдання на додавання, віднімання, множення та ділення подаються у формі космічних місій, що робить процес навчання динамічним та цікавим. Гра не підтримує українську мову, але завдяки своїй популярності її використовували мільйони дітей у всьому світі на різних платформах, від персональних комп'ютерів до мобільних пристроїв. "Math Blaster" допомагає учням закріплювати математичні навички в ігровій формі, підвищуючи їх мотивацію до навчання. Крім того, гра розвиває логічне мислення, здатність до швидкого прийняття рішень та креативність. Використання яскравих візуальних ефектів та захоплюючого сюжету робить гру популярною серед учнів.

Завдяки структурованим ігровим завданням, учні можуть покращувати свої навички вирішення проблем і вдосконалювати математичні розрахунки. "Math

Blaster" також сприяє розвитку моторики, оскільки учні повинні швидко реагувати та взаємодіяти з ігровим середовищем. Це допомагає їм розвивати координацію рухів та здатність швидко приймати рішення, що є важливими навичками для успішного навчання.

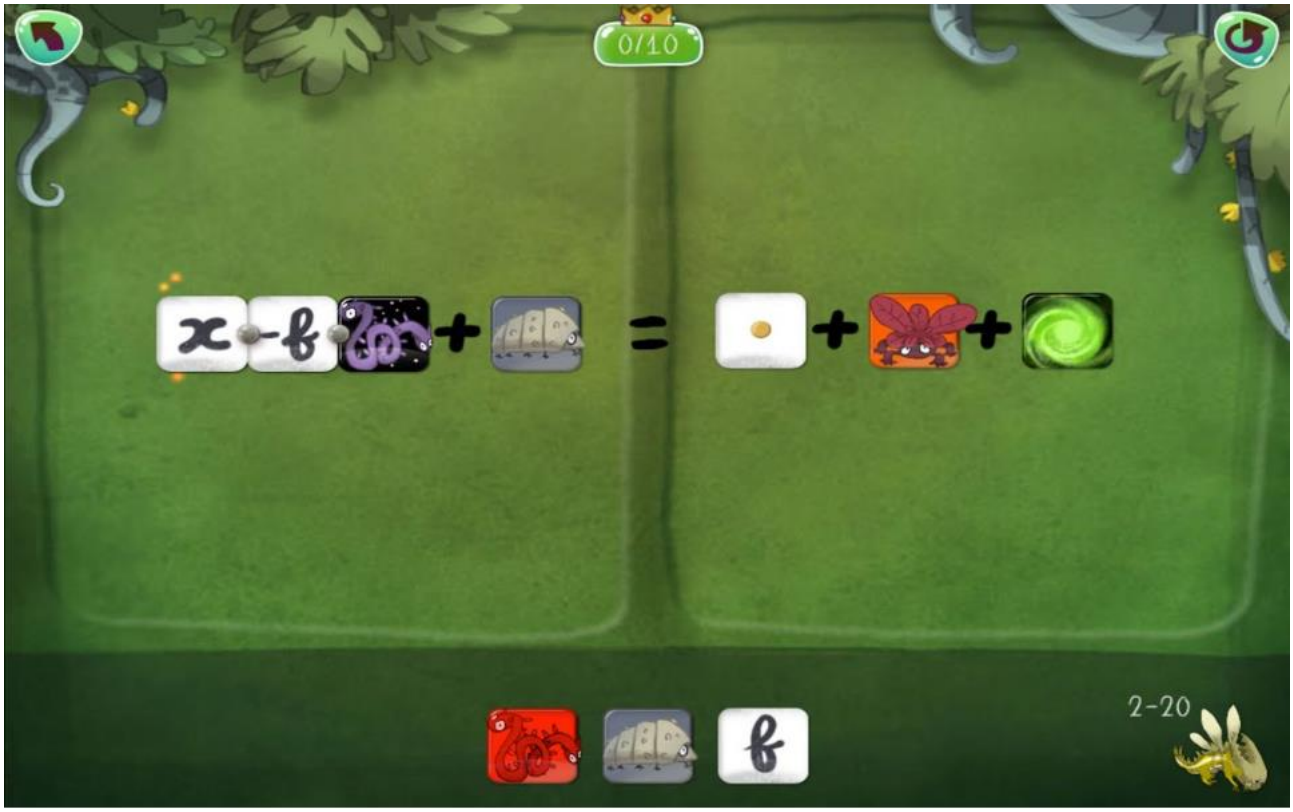


Рис. 1.2. Скріншот з гри «DragonBox»

Іншим успішним прикладом є "DragonBox" [18], розроблена компанією WeWantToKnow AS у Норвегії та випущена у 2012 році. Ця мобільна головоломка та навчальна гра спрямована на навчання основам алгебри для дітей та дорослих, використовуючи унікальний підхід до пояснення абстрактних математичних понять через візуальні образи та метафори. Замість традиційних алгебраїчних символів у грі використовуються зображення драконів та коробок, які необхідно упорядковувати за певними правилами. Такий підхід значно полегшує розуміння складних концепцій і робить процес навчання більш інтуїтивним.

Учні можуть візуально спостерігати, як змінюються об'єкти під час виконання алгебраїчних операцій, що допомагає їм краще засвоювати матеріал. Хоча гра не підтримує українську мову, її завантажили сотні тисяч користувачів

по всьому світу. "DragonBox" робить навчання цікавим і захоплюючим, формуючи позитивне ставлення до математики у дітей. Ця гра допомагає учням розвивати аналітичні навички та вміння знаходити закономірності, що є важливими для успішного вивчення математики. Використовуючи візуальні метафори, гра спрощує розуміння абстрактних понять і робить їх більш доступними для дітей. "DragonBox" також сприяє розвитку критичного мислення, оскільки учні повинні знаходити правильні рішення та впорядковувати об'єкти за певними правилами. Це допомагає їм покращувати навички логічного мислення та розвивати здатність до аналізу та синтезу інформації.

Обидві гри, "Math Blaster" та "DragonBox", можна охарактеризувати як розвиваючі та освітні, з інтерактивним стилем та елементами гейміфікації. Вони використовують дидактичний підхід навчання через гру та візуалізацію абстрактних понять, що робить їх ефективними інструментами для вивчення математики.

Додаткові переваги ігрових методів

Ігрові методи також сприяють поліпшенню пам'яті та уваги учнів. В іграх діти часто запам'ятовують правила, послідовності дій та деталі, що сприяє тренуванню їхньої короткочасної та довготривалої пам'яті. Крім того, ігрові завдання вимагають концентрації та уважності, що допомагає розвивати ці важливі когнітивні навички.

Ігри дозволяють учням повторювати і закріплювати навчальні навички в приємній і невимушеній обстановці. Це особливо важливо для молодших школярів, які часто потребують багаторазового повторення матеріалу для його повного засвоєння. Граючи, діти можуть багаторазово виконувати завдання, не відчуваючи нудьги чи втоми.

Вплив ігрових методів на соціалізацію

Ігрові методи навчання – це не просто розвага, а потужний інструмент для розвитку соціальних навичок учнів. Коли діти грають разом, вони вчаться спілкуватися, слухати один одного, домовлятися та шукати компроміси.

У командних іграх кожен учасник має свою роль, свою відповідальність. Діти вчаться працювати разом, допомагати один одному, долати труднощі та радіти спільним перемогам. Це зміцнює їхні стосунки, вчить цінувати внесок кожного члена команди.

Ігрові ситуації часто моделюють різні життєві обставини, що дозволяє учням краще зрозуміти почуття та мотиви інших людей. Це сприяє розвитку емпатії, толерантності та вмінню ставити себе на місце іншого.

Завдяки іграм, атмосфера в класі стає більш позитивною та невимушеною. Учні забувають про стрес та тривогу, вільно висловлюють свої думки та активно беруть участь у навчальному процесі.

У процесі гри діти вчаться довіряти один одному, підтримувати та покладатися на своїх товаришів. Це зміцнює дружні зв'язки та створює атмосферу взаємоповаги та довіри в колективі.

Отже, ігрові методи навчання – це не лише шлях до знань, а й важливий крок у доросле життя, де вміння спілкуватися, співпрацювати та розуміти інших є запорукою успіху.

### 1.2. Методичні основи розробки навчальних ігор

У сучасній освіті розробка навчальних ігор є ключовим напрямком, що дозволяє впроваджувати інноваційні методи у навчальний процес. Використання ігор у навчанні підвищує зацікавленість учнів, покращує засвоєння матеріалу та розвиває різні навички. У цьому розділі ми розглянемо основні принципи створення освітнього контенту та вплив інтерактивних елементів на навчальний процес.

Принципи розробки освітнього контенту

Створення навчальних ігор вимагає дотримання кількох ключових принципів, які забезпечують ефективність та результативність навчання.

**Відповідність віковим особливостям**. Один з основних принципів полягає у відповідності гри віковим особливостям учнів. Кожна вікова група має свої психолого-педагогічні характеристики, які необхідно враховувати при

створенні контенту. Для молодших школярів важливо забезпечити простоту і зрозумілість завдань, використовуючи яскраві візуальні образи та доступну мову.

**Інтерактивність та залученість.** Інтерактивність є одним з ключових елементів навчальних ігор. Ігри повинні заохочувати активну участь учнів, надаючи можливість взаємодії з контентом через різноманітні завдання, головоломки та ситуаційні задачі, які потребують прийняття рішень та виконання дій. Учні повинні бути активними учасниками навчального процесу.

**Доступність та адаптивність.** Навчальні ігри мають бути доступними для всіх учнів, незалежно від їхніх початкових знань та вмінь. Адаптивність гри дозволяє підлаштовуватися під індивідуальні потреби та рівень підготовки кожного учня. Це можна досягти через різні рівні складності, підказки та інструкції, що допомагають учням поступово освоювати новий матеріал.

**Мотивація та зворотний зв'язок.** Ефективна навчальна гра повинна містити мотиваційні елементи, такі як бали, нагороди та рейтинги, що стимулюють учнів до активної участі та досягнення цілей. Важливим є також зворотний зв'язок, який допомагає учням зрозуміти свої помилки та отримати рекомендації для покращення результатів.

**Інтеграція навчального контенту.** Навчальні ігри мають бути інтегровані з навчальним контентом, щоб забезпечити цілісність та послідовність освітнього процесу. Ігрові завдання мають відповідати навчальним цілям та програмі, допомагаючи учням закріплювати та застосовувати знання на практиці.

Вплив інтерактивних елементів на процес навчання

Інтерактивні елементи відіграють важливу роль у навчальних іграх та значно впливають на навчальний процес. Основні аспекти їхнього впливу включають:

| Аспекти впливу | Опис |
|---|---|
| **Активізація навчальної діяльності** | Інтерактивні елементи стимулюють активну участь учнів у навчанні, допомагаючи їм краще засвоювати матеріал через виконання завдань та прийняття рішень [8]. |
| **Підвищення мотивації** | Ігрові завдання, головоломки та ситуаційні задачі роблять навчання більш захоплюючим, стимулюючи учнів до подальших зусиль через зворотний зв'язок та нагороди [2]. |

| | |
|---|---|
| **Розвиток критичного мислення** | Інтерактивні елементи сприяють розвитку логічного мислення та аналітичних навичок учнів, допомагаючи їм у вирішенні проблемних ситуацій та прийнятті обґрунтованих рішень [4]. |

Переваги інтерактивних навчальних ігор

Інтерактивні навчальні ігри мають низку переваг, що роблять їх ефективним інструментом навчання. Однією з головних переваг є можливість надавати миттєвий зворотний зв'язок, що дозволяє учням швидко коригувати свої дії та покращувати результати [9].

Інтерактивні ігри також створюють умови для індивідуального підходу до навчання. Учні можуть самостійно обирати темп навчання, рівень складності завдань та отримувати підказки відповідно до своїх потреб, що забезпечує більш ефективне засвоєння матеріалу [10].

Використання мультимедійних елементів

Мультимедійні елементи, такі як відео, аудіо, анімації та інтерактивні графіки, є важливими компонентами інтерактивних навчальних ігор. Вони допомагають зробити навчальний процес більш захоплюючим та наочним, сприяючи кращому розумінню та запам'ятовуванню матеріалу.

Мультимедійні елементи також дозволяють створювати реалістичні ситуації та моделі для дослідження, що сприяє розвитку практичних навичок та здатності застосовувати отримані знання в реальних умовах.

Інтерактивні завдання та оцінювання

Інтерактивні навчальні ігри можуть містити різноманітні завдання та вправи, що дозволяють учням перевіряти свої знання та навички. Оцінювання може здійснюватися автоматично, забезпечуючи оперативний зворотний зв'язок та можливість швидкого коригування навчальної діяльності. Автоматичне оцінювання також знімає частину навантаження з викладачів, дозволяючи їм зосередитися на інших аспектах навчання [11].

**Висновки до розділу 1**

У цьому розділі розглянуто значення ігрових методів в освітньому процесі, зокрема їхній вплив на мотивацію учнів, розвиток когнітивних навичок та

емоційний стан. Теоретичний аналіз літератури з педагогіки та психології показав, що ігрові методи є ефективним інструментом для підвищення інтересу учнів до навчання та покращення засвоєння матеріалу. Особлива увага була приділена принципам, таким як відповідність віковим особливостям, інтерактивність, доступність та адаптивність, які є критично важливими при розробці навчальних ігор. Аналіз наявних освітніх ігор, таких як "Math Blaster" та "DragonBox", підтвердив ефективність використання ігор для навчання математики молодших школярів.

# РОЗДІЛ 2. РОЗРОБКА КОНЦЕПЦІЇ ГРИ "МАТЕМАТИЧНИЙ ЛАБІРИНТ"

## 2.1. Опис гри та її елементів

### Головний герой та його здібності

Головним героєм гри є маленький школяр, який потрапив у магічний математичний лабіринт. Цей персонаж символізує кожного молодшого школяра, що починає свою подорож у світ математики. Герой має базові здібності, такі як рух у чотирьох напрямках (вперед, назад, вліво, вправо), що дозволяє гравцеві досліджувати лабіринт. Крім того, головний герой має здатність вступати у математичні дуелі з ворогами, що робить гру інтерактивною та освітньою.

Гра орієнтована на учнів 1-4 класів, тому персонаж має бути візуально привабливим і зрозумілим для цієї вікової групи. Зовнішній вигляд героя, його здібності та поведінка мають сприяти позитивному сприйняттю гри дітьми. Наприклад, герой може мати яскравий костюм, який символізує знання та силу, отримані через навчання.

### Вороги та їх функції

Основними ворогами у грі є Слизні, які являють собою метафору математичних проблем, що виникають на шляху учня. Слизні хаотично переміщуються по лабіринту, створюючи додаткові перешкоди для гравця. Коли герой наближається до ворога, починається математична дуель. Цей елемент гри робить її динамічною та стимулює швидке мислення.

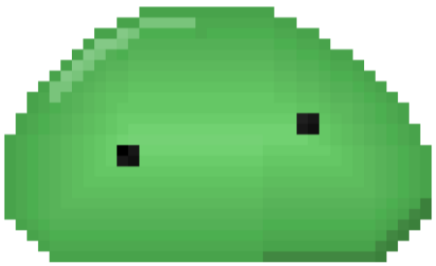

Рис. 2.1. Приклад ворога «Слайм»

Функції Слизнів включають затримування героя та створення викликів, які необхідно подолати через вирішення математичних задач. Це дозволяє інтегрувати навчальний елемент у ігровий процес. Наприклад, при зустрічі з ворогом гравцеві пропонується вирішити задачу на додавання чи віднімання. Якщо відповідь правильна, ворог втрачає частину здоров'я, якщо ні – здоров'я втрачає герой.

Структура рівнів та лабіринтів

Лабіринт у грі автоматично генерується для кожного рівня, що забезпечує унікальність кожного проходження. Це означає, що кожен новий рівень має різну конфігурацію, що робить гру цікавою та непередбачуваною. Початкова точка (старт) і кінцева точка (фініш) завжди розташовані у різних місцях лабіринту, що додає елемент пошуку та стратегії.

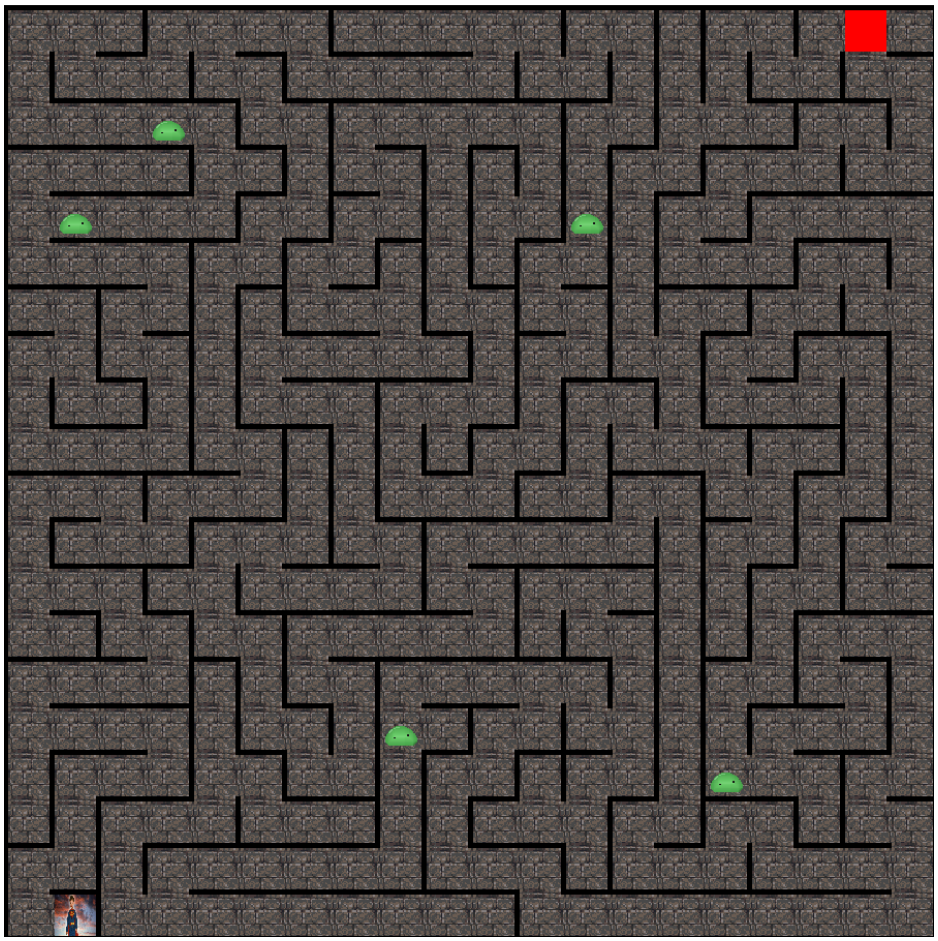

Рис. 2.2. Приклад лабіринту

Особливістю кожного рівня є те, що з кожним новим рівнем складність підвищується за рахунок збільшення кількості ворогів. Це означає, що гравець

повинен розробляти нові стратегії для досягнення фінішу. Наприклад, на початкових рівнях гравець може зустріти одного чи двох Слизнів, тоді як на вищих рівнях їх кількість значно збільшується, що вимагає більшої концентрації та швидкості мислення.

### 2.2. Розробка сценарію та ігрового процесу

Опис покрокового ігрового процесу

**Початок гри** - герой починає гру на старті лабіринту. На цьому етапі гравець знайомиться з основними механіками гри, такими як рух та взаємодія з ворогами.

**Дослідження лабіринту** - гравець керує рухом героя, досліджуючи лабіринт у пошуках виходу. Важливою частиною цього етапу є уникання ворогів або підготовка до дуелей з ними.

**Математичні дуелі -** під час зустрічі з ворогом гравець має 6 секунд на вирішення математичної задачі, наприклад, 124 + 49. Цей елемент гри стимулює швидке мислення та прийняття рішень. Питання генеруються випадковим чином, що забезпечує різноманітність.

**Вирішення задач** - за правильну відповідь ворог втрачає частину здоров'я. За неправильну – здоров'я втрачає герой. Це створює баланс між ризиком та винагородою, що робить гру захоплюючою.

**Завершення рівня** - герой продовжує рух до фінішу, подолавши всі перешкоди. На кожному новому рівні гравець стикається з новими викликами, що підвищує складність гри та стимулює подальший розвиток навичок.


Рис. 2.3. Скріншот математичної дуелі

1. Коли герой підходить до ворога, з'являється математична задача.

2. Гравець має 6 секунд на відповідь. Питання генеруються випадковим чином, що забезпечує різноманітність і постійний виклик.

3. Відповідь оцінюється негайно: правильна відповідь завдає шкоди ворогу, неправильна – герою.

4. Мета дуелі – повністю знизити здоров'я ворога, зберігаючи своє. Кожна перемога над ворогом наближає гравця до кінцевої мети – виходу з лабіринту.

**Висновки до розділу 2**

У цьому розділі була розроблена концепція навчально-розвивальної гри "Математичний лабіринт". Детально описано головного героя, ворогів та їхні

функції, а також структуру рівнів та лабіринтів. Особливу увагу приділено створенню динамічного ігрового процесу, який стимулює розвиток логічного мислення та навичок вирішення проблем у дітей. Використання автоматичної генерації лабіринтів забезпечує унікальність кожного проходження гри, що підвищує інтерес і мотивацію учнів.

# РОЗДІЛ 3. ТЕХНІЧНА РЕАЛІЗАЦІЯ ГРИ "МАТЕМАТИЧНИЙ ЛАБІРИНТ"

## 3.1. Вибір платформи та інструментів розробки

Гра "Математичний лабіринт" розроблена для роботи на платформі Windows. Вибір цієї платформи обумовлений її широким розповсюдженням серед користувачів, зручністю у використанні та підтримкою багатьох необхідних інструментів для розробки. Windows також забезпечує високу продуктивність і стабільність, що є критично важливим для забезпечення плавного та безперебійного ігрового процесу.

Програмне забезпечення та інструменти

Чому саме обрано Windows?

Windows, як домінуюча операційна система на ринку персональних комп'ютерів, забезпечує найширший доступ до потенційних гравців, що дозволяє на першому етапі залучити максимальну кількість користувачів та отримати зворотний зв'язок для подальшого розвитку проекту. Розробка під Windows надає доступ до широкого спектру спеціалізованих інструментів та бібліотек, що суттєво спрощує та прискорює процес створення гри, даючи змогу сконцентрувати зусилля на ключових аспектах проекту, таких як геймплей, дизайн та контент, замість вирішення технічних проблем. Попри початковий фокус на Windows, проект має довгострокову перспективу розширення на мобільні платформи, що дозволить у майбутньому охопити ще більшу аудиторію та зробити гру доступною для користувачів смартфонів та планшетів.

Чому саме обрано Python?

Python [19] вирізняється простотою та лаконічністю синтаксису, що сприяє швидкому написанню та легкому розумінню коду, що особливо важливо для проектів, де задіяна команда розробників, оскільки забезпечує узгодженість та полегшує процес внесення змін. Величезна кількість готових бібліотек та фреймворків, доступних для Python, дозволяє значно скоротити час розробки та зосередитися на унікальних аспектах проекту, що робить Python ідеальним вибором для проектів різного масштабу та складності. Python має велику та

активну спільноту розробників, що забезпечує постійний розвиток мови та доступ до величезної кількості навчальних матеріалів, документації та прикладів коду, що значно спрощує процес навчання та вирішення проблем, що можуть виникнути під час розробки.

Чому саме обрано Pygame?

Pygame [20] – це бібліотека, спеціально створена для розробки 2D-ігор на Python. Вона надає повний набір інструментів для роботи з графікою, звуком, обробкою подій та управлінням введенням користувача, що дозволяє створювати повноцінні ігрові проекти без необхідності залучення додаткових бібліотек. Pygame має інтуїтивно зрозумілий інтерфейс та добре документовані функції, що робить її доступною навіть для початківців. Завдяки цьому, розробники можуть швидко розпочати роботу над проектом та зосередитися на його творчій складовій. Pygame має велику спільноту розробників, які активно діляться своїм досвідом, знаннями та готовими рішеннями, що дозволяє швидко вирішувати проблеми, що виникають під час розробки, та отримувати підтримку від більш досвідчених колег.

### 3.2. Процес розробки гри

Проектування гри

Процес проектування рівнів у грі "Математичний лабіринт" був здійснений з нуля, без використання готових редакторів. Це дозволило мені повністю контролювати всі аспекти гри та забезпечити унікальність кожного рівня.

В основі розробки гри «Математичний лабіринт» лежить об'єктно-орієнтований підхід (ООП), що дає змогу уявити кожен елемент гри (персонажа, ворогів, лабіринт) як окремий об'єкт із власними характеристиками (положення, здоров'я) і діями (рух, атака). Такий підхід робить код більш структурованим, читабельним і зручним у підтримці.

Для забезпечення гнучкості та розширюваності гри було використано абстрактні класи. Абстрактний клас - це свого роду шаблон, який визначає загальні властивості та методи для групи об'єктів, але не може бути використаний для створення конкретних об'єктів. Наприклад, у грі є абстрактний клас «Ворог»,

який визначає загальні характеристики всіх ворогів (здоров'я, шкоди), а також методи, які мають бути реалізовані в кожному конкретному типі ворога (Слизень).

Генерація ігрового поля

Генерація лабіринту - для створення унікального лабіринту для кожного рівня використовується спеціальний алгоритм. Цей алгоритм випадковим чином генерує конфігурацію лабіринту, забезпечуючи різноманітність ігрового досвіду.

```
def generate(self):
     """Генерує випадковий лабіринт, використовуючи алгоритм пошуку
     в глибину."""
     # Визначення розмірів лабіринту
     n, m = self.width, self.height
     # Ініціалізація матриці досяжності клітинок
     for i in range(n):
          self.reach_matrix.append([])
          for j in range(m):
               self.reach_matrix[i].append(False)
     # Ініціалізація матриці переходів (стінок)
     for i in range(n * 2 - 1):
          self.transition_matrix.append([])
          for j in range(m * 2 - 1):
          # Початково всі стінки присутні, крім переходів між
     клітинками
               if i % 2 == 0 and j % 2 == 0:
                    self.transition_matrix[i].append(True)
               else:
                    self.transition_matrix[i].append(False)

          # Визначення початкової та кінцевої точок лабіринту
          self.start = start_point_generate(n, m)
          self.finish = finish_point_generate(self.start, n, m)
          # Список переходів для відстеження шляху
          list_transition = [self.start]
          # Початкові координати
          x, y = self.start
          # Початкова клітинка відмічається як досяжна
          self.reach_matrix[x][y] = True
          # Вибір першого переходу
          x, y, tx, ty = transition_choice(x, y, self.reach_matrix)
          # Основний цикл генерації лабіринту
          for i in range(1, m * n):
               #Повернення до попередньої клітинки, якщо поточна
          недосяжна
               while not (x >= 0 and y >= 0):
                    x, y = list_transition[-1]
                    list_transition.pop()
```

```
                x, y, tx, ty = transition_choice(x, y,
            self.reach_matrix)
            # Відмітка поточної клітинки як досяжної
            self.reach_matrix[x][y] = True
            # Додавання поточної клітинки до списку переходів
            list_transition.append((x, y))
            # Видалення стінки (створення переходу)
            self.transition_matrix[tx][ty] = True
            # Вибір наступного переходу
            x, y, tx, ty = transition_choice(x, y,
        self.reach_matrix)
```

Заповнення блоками - після генерації лабіринту ігрове поле заповнюється блоками (стінами), які обмежують рух персонажа і ворогів. Це дає змогу створити різноманітні маршрути та перешкоди для гравця.

```
def draw(self, window):
"""Малює лабіринт у вікні Pygame."""
# Проходимося по матриці переходів
    for i in range(len(self.transition_matrix)):
        for j in range(len(self.transition_matrix[0])):
            # Розраховуємо координати для малювання стіни або
        шляху
            x = self.border + (i // 2) * (self.width_line +
            self.width_walls) + (i % 2) * self.width_line
            y = self.border + (j // 2) * (self.width_line +
            self.width_walls) + (j % 2) * self.width_line
            # Якщо в матриці transition_matrix є стінка, малюємо
            її
            if self.transition_matrix[i][j] == 1:
                window.blit(self.wall_image_resized, (x, y))
                #Якщо немає стінки, малюємо шлях
            else:
                pygame.draw.rect(window, color_wall, (x, y,
                self.width_line, self.width_line))
    # Малюємо початкову точку лабіринту
    pygame.draw.rect(window, color_start, (
            self.border + self.start[0] * (self.width_line +
        self.width_walls),
            self.border + self.start[1] * (self.width_line +
        self.width_walls),
            self.width_line, self.width_line))
    # Малюємо кінцеву точку лабіринту
    pygame.draw.rect(window, color_finish, (
            self.border + self.finish[0] * (self.width_line +
        self.width_walls),
            self.border + self.finish[1] * (self.width_line +
        self.width_walls),
            self.width_line, self.width_line))
```

Намальовування персонажа та ворогів: Персонаж гравця та вороги (Слизні) розміщуються на ігровому полі у випадкових позиціях. Це додає елемент несподіванки і робить кожне проходження гри унікальним.

```
def draw(self, window):
     """Відображає всі елементи гри на"""
     # Очищуємо попередні позиції гравця та ворогів
     self.labyrinth.clear(window, self.player.old_position)
     for enemy in self.enemies:
          self.labyrinth.clear(window, enemy.old_position)
          #Малюємо гравця та ворогів на нових позиціях
          self.player.draw(window, self.labyrinth)
          for enemy in self.enemies:
               enemy.draw(window, self.labyrinth)
     self.draw_game_info()
     pygame.display.flip()
```

Програмування ігрових механік

Контроль руху персонажа- гравець керує персонажем за допомогою клавіш управління, переміщаючи його по 2D-лабіринту по одному кроку за раз. Після кожного ходу персонажа вороги також здійснюють свій хід, що додає елемент стратегії та планування до гри.

```
def handle_movement(self, event, labyrinth):
     """Обробляє рух гравця на основі натискань клавіш."""
     dx, dy = 0, 0  # Зміни по осі X та Y (спочатку руху немає)
     # Визначаємо напрямок руху залежно від натиснутої клавіші
     if event.key == pygame.K_RIGHT or event.key == pygame.K_d:
          dx = 1   # Рух вправо
     elif event.key == pygame.K_LEFT or event.key == pygame.K_a:
          dx = -1  # Рух вліво
     elif event.key == pygame.K_DOWN or event.key == pygame.K_s:
          dy = 1   # Рух вниз
     elif event.key == pygame.K_UP or event.key == pygame.K_w:
          dy = -1  # Рух вгору
     # Розраховуємо нові координати в матриці переходів лабіринту
     new_x, new_y = self.position[0] * 2 + dx, self.position[1] * 2
+ dy

     # Перевіряємо, чи можна переміститися на нові координати:
     # 1. Чи не виходять вони за межі лабіринту
     # 2. Чи є прохід (True) у матриці переходів на нових
координатах
     if (0 <= new_x < len(labyrinth.transition_matrix) and
     0 <= new_y < len(labyrinth.transition_matrix[0]) and
     labyrinth.transition_matrix[new_x][new_y]):
```

```
            self.old_position = self.position.copy()#Зберігаємо
      попередню позицію ПЕРЕД оновленням
            self.position[0] += dx#Оновлюємо позицію гравця
            self.position[1] += dy
            self.moved = True#Позначаємо, що гравець зробив хід
      else:
            self.moved = False# Гравець не зміг зробити хід
```

Перевірка колізій

Перевірка колізій є важливим елементом для забезпечення коректної взаємодії між персонажем гравця та ворогами, а також між персонажем та іншими об'єктами в грі.

Колізії з ворогами - визначення зіткнень персонажа з ворогами здійснюється за допомогою алгоритмів перевірки перетинання координат. Якщо координати персонажа і ворога збігаються, ініціюється дуель.

Колізії з перешкодами - перевірка колізій з перешкодами здійснюється аналогічним чином. Якщо персонаж намагається переміститися на зайняту перешкодою клітинку, рух блокується.

Обробка колізій - після виявлення колізії запускається відповідний сценарій, такий як ініціація дуелі або блокування руху.

```
def check_proximity(self):
      """Перевіряє, чи знаходиться гравець поруч з ворогом
(враховуючи стіни) або стикається з ним."""

      # Перебираємо всіх ворогів
      for enemy in self.enemies:
            # Перевірка на безпосереднє зіткнення з ворогом
            if self.player.position == enemy.position:
                  enemy.position = enemy.old_position.copy()
            #Повертаємо ворога на попередню позицію
            return True, "collision", enemy  # Повертаємо True, тип
      події та ворога
            #Перевірка на сусідство з ворогом (без стіни між ними)
            if (abs(self.player.position[0] - enemy.position[0]) +
            abs(self.player.position[1] - enemy.position[1]) == 1 and
            not self.labyrinth.is_wall_between(self.player.position,
      enemy.position)):
            return True, "proximity", enemy# Повертаємо True, тип
      події та ворога

      # Якщо зіткнення чи сусідства не виявлено, повертаємо False та
None
      return False, None, None
```

Математичні дуелі

Математичні дуелі є ігровим механізмом, що активується при зустрічі персонажа з супротивником у суміжних клітинках за відсутності перешкод

```
     if is_near_or_collision:  # Якщо відбулося зіткнення або
наближення до ворога
          if type_of_interaction == "collision":  # Перевіряємо тип
     взаємодії
          print("Зіткнення!")  # Виводимо повідомлення про
     зіткнення
          elif type_of_interaction == "proximity":  # Якщо гравець
     близько до ворога
               fight = Fight(  # Створюємо об'єкт класу Fight для
          початку бою
               hero=self.player,  # Передаємо гравця як героя
               enemy=enemy)
               fight.start()  # Починаємо бій
```

Вибір математичного завдання - завдання генеруються випадковим чином із заздалегідь підготовленого набору питань. Це гарантує різноманітність завдань та запобігає повторенню одних і тих самих питань.

```
def generate_question(self):
     # Генерація випадкового питання для бою
     num1 = random.randint(1, 50)
     num2 = random.randint(1, 50)

     # Вибір випадкової арифметичної операції та відповідної
функції
     operations = {
          "+": operator.add,
          "-": operator.sub,
          "*": operator.mul,
          "/": operator.truediv  # Додали ділення
          }
     operation_symbol = random.choice(list(operations.keys()))
     operation_func = operations[operation_symbol]

     # Обчислення правильної відповіді з урахуванням типу операції
     if operation_symbol == "/":
          # Уникаємо ділення на нуль та забезпечуємо цілочисельний
     результат
          while num2 == 0:
               num2 = random.randint(1, 50)
          correct_answer = operation_func(num1, num2)
          # Округлюємо результат до найближчого цілого числа
          correct_answer = round(correct_answer)
     else:
          correct_answer = operation_func(num1, num2)
```

```
     self.question = f"Скільки буде {num1} {operation_symbol}
{num2}?"
     self.correct_answer = str(correct_answer)
```

Таймер - гравцеві надається 6 секунд на вирішення завдання. Таймер відображається на екрані, щоб стимулювати швидке мислення та реагування.

```
def display(self):
     """Відображає спливаюче вікно з питанням та обробляє відповідь
гравця."""
     # Цикл триває, поки залишається час і гравець не відповів
     while self.time_remaining > 0 and not self.answered:
          self.handle_events()  # Обробляємо події (натискання
          клавіш, тощо)
          # Оновлюємо таймер, якщо відповідь ще не дана
          if not self.answered:
               self.time_remaining -= self.clock.tick(60) / 1000
               # Віднімаємо час, що минув
               self.update_screen()
               #Оновлюємо вміст екрану (питання, таймер, здоров'я)

               # Якщо герой помер під час відповіді, виходимо з
          циклу
               if not self.fight.hero.is_alive:
                    return
               # Якщо час вийшов, а гравець не відповів, герой
          отримує пошкодження
               if not self.answered:
                    self.fight.hero.health -= 1
```

Перевірка відповідей - відповіді гравця перевіряються негайно після введення. Якщо відповідь правильна, ворог отримує пошкодження. Якщо відповідь неправильна, пошкодження отримує персонаж гравця.

Після завершення кожної математичної дуелі відбувається наступне:

– перемога гравця - якщо гравець правильно відповідає на запитання і здоров'я Слизню (ворога) падає до нуля, то Слизень зникає з лабіринту, а гравець продовжує свій шлях до фінішу з поточним рівнем здоров’я;

– поразка гравця - якщо гравець дає неправильну відповідь або час вичерпується, здоров'я його персонажа зменшується. Якщо здоров'я персонажа падає до нуля, гравець програє, і гра закінчується.

Таким чином, кожна дуель є важливим етапом гри, який вимагає від гравця швидкої реакції та правильних відповідей на математичні запитання. Перемога в дуелі наближає гравця до мети, а поразка може призвести до програшу.

**Висновки до розділу 3**

У цьому розділі було розглянуто технічні аспекти розробки гри "Математичний лабіринт". Вибір платформи Windows та мови програмування Python з використанням бібліотеки Pygame обґрунтовано з огляду на їхню популярність та зручність. Описано процес проектування рівнів, програмування ігрових механік, реалізації системи колізій та тестування гри. В результаті була створена стабільна та функціональна гра, яка відповідає педагогічним вимогам і є ефективним інструментом для навчання математики молодших школярів.

.

# ВИСНОВКИ

У ході дослідження було розглянуто теоретичні основи та практичні аспекти розроблення навчально-розвивальної математичної гри для учнів молодшого шкільного віку. Вивчено педагогічні та психологічні аспекти, які слід враховувати при створенні таких ігор, а також їхню роль у підвищенні мотивації та когнітивних здібностей дітей. На основі аналізу існуючих освітніх ігор, таких як "Math Blaster" і "DragonBox", були визначені ключові принципи та методи розробки ефективних навчальних ігор, що сприяють кращому засвоєнню математичних знань.

Основна увага в роботі приділялася проектуванню та розробці гри "Математичний лабіринт". Ця гра орієнтована на учнів 1-4 класів і включає елементи, що розвивають логічне мислення, швидке прийняття рішень та навички вирішення математичних задач. Головним героєм гри є школяр, який подорожує магічним лабіринтом і вступає в математичні дуелі з ворогами. Структура гри, що включає автоматичну генерацію лабіринтів і математичних завдань, забезпечує унікальність кожного проходження, підвищуючи інтерес і мотивацію учнів до навчання.

Технічна реалізація гри "Математичний лабіринт" здійснювалася на платформі Windows з використанням мови програмування Python та бібліотеки pygame. Процес розробки включав проектування рівнів, програмування ігрових механік, реалізацію системи колізій та тестування гри. В результаті тестування були виявлені та виправлені баги, оптимізований код та покращено користувацький досвід.

Розроблена гра "Математичний лабіринт" має значний потенціал для використання в освітньому процесі. Вона дозволяє поєднувати навчання та розвагу, що сприяє підвищенню інтересу дітей до математики та ефективному засвоєнню матеріалу. Гра забезпечує розвиток ключових когнітивних навичок, таких як логічне мислення, планування та прийняття рішень, а також сприяє зниженню рівня тривожності та страху перед помилками.

Подальший розвиток гри може включати розширення набору математичних завдань, додавання нових рівнів і персонажів, а також інтеграцію

з іншими освітніми платформами. Перспективи використання "Математичного лабіринту" в освіті включають можливість його застосування як додаткового інструменту для закріплення знань, проведення інтерактивних уроків та індивідуальних занять з учнями.

Таким чином, розробка та впровадження навчально-розвивальних ігор, таких як "Математичний лабіринт", є важливим напрямком у сучасній освіті, що дозволяє підвищити ефективність та привабливість навчального процесу для дітей молодшого шкільного віку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Білик В. М. Інноваційні методи навчання: теорія і практика. Львів: ЛНУ імені Івана Франка, 2020. С. 12-23.
2. Василенко А. П. Переваги та недоліки інтеграції ігрових методів у навчальний процес. Журнал педагогічних досліджень, 15(2), 2018. С. 85-92.
3. Волошин В. М. Розвиток креативності у школярів за допомогою ігор. Вісник освіти, 16(2), 2017. С. 34-41.
4. Гончаренко С. У. Інтерактивні технології в сучасній освіті. Харків: ХНУ імені В. Н. Каразіна, 2016. С. 45-50.
5. DragonBox URL: https://dragonbox.com/
6. Зубарєва І. А. Ігрові технології в початковій школі. Освіта і суспільство, 3(1), 2019. С. 45-59.
7. Карпенко О. В. Психологічні особливості використання ігрових методів в освіті. Психологічний журнал, 19(2), 2017. С. 78-85.
8. Коваленко І. В. Освітні ігри як засіб розвитку критичного мислення учнів. Науковий вісник Миколаївського національного університету, 11, 2019. С. 45-54.
9. Кузьменко А. І. Роль ігрових технологій у формуванні мотивації до навчання. Журнал освіти та науки, 28(3), 2019. С. 99-106.
10. Литвиненко Ю. С. Інтерактивні методи навчання та їх вплив на успішність учнів. Журнал педагогічних інновацій, 7(1), 2020. С. 56-63.
11. Мельник Л. І. Використання інформаційних технологій в освітніх іграх. Вісник КНУ імені Тараса Шевченка, 21(3), 2018. С. 66-72.
12. Петренко М. С. Психолого-педагогічні аспекти використання ігор у навчальному процесі. Наукові записки НаУКМА, 18(1), 2017. С. 123-131.
13. Пономаренко О. Інтерактивні методи навчання в сучасній школі. Київ: Освіта України, 2015. С. 10-20.
14. Савченко В. П. Методологія розробки інтерактивних навчальних ігор. Наукові записки ВДПУ, 10(1), 2016. С. 89-97.

15. Ткаченко Р. О. Вплив ігрових методів на розвиток когнітивних навичок учнів. Педагогічний журнал, 12(4), 2018. С. 111-118.
16. Шевченко Л. В. Впровадження інтерактивних методів навчання в систему освіти України. Педагогічний альманах, 2, 2018. С. 10-19.
17. Math Blaster. URL: https://playclassic.games/games/educational-dos-games-online/play-math-blaster-plus-online/play/
18. DragonBox. URL: https://dragonbox.com/
19. Python. URL: https://www.python.org/doc/.
20. Pygame. URL: https://www.pygame.org/docs

# ДОДАТКИ

## *main.py*

```
import pygame
from loader import window
import time
from labirint import Labyrinth
from config import display, width, height, width_line,
width_walls, border
from Entity import Player,WeekEnemy
from fight import Fight
import random

class Game:
    def __init__(self,
                 labyrinth: Labyrinth,
                 player: Player,
                 enemies: list[WeekEnemy]):

        self.labyrinth = labyrinth
        self.player = player
        self.enemies = enemies
        self.running = True
        self.trace = False
        self.start_time = time.time()
        self.record_time = 9999
        self.score = 0

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                self.player.handle_movement(event, self.labyrinth)
                if event.key == pygame.K_q:
                    self.toggle_trace()

    def check_proximity(self):
        for enemy in self.enemies:
            if self.player.position == enemy.position:
                enemy.position = enemy.old_position.copy()
                return True, "collision", enemy
            if (abs(self.player.position[0] - enemy.position[0]) +
                abs(self.player.position[1] - enemy.position[1])
== 1 and
                not
self.labyrinth.is_wall_between(self.player.position,
enemy.position)):
                return True, "proximity", enemy
        return False, None, None

    def toggle_trace(self):
        self.trace = not self.trace
```

```
    def draw_game_info(self):
        font = pygame.font.Font(None, 25)
        heart_color = (255, 0, 0)
        empty_heart_color = (128, 128, 128)
        heart_width = 20
        heart_height = 15
        x_offset = 5
        y_offset = 5
        info_rect = pygame.Rect(0, window.get_height() - 70,
display.width_window, 70)
        pygame.draw.rect(window, (0, 0, 0), info_rect)
        time_text = f"Час проходження лабіринту: {int(time.time()
- self.start_time)}"
        text = font.render(time_text, True, (255, 255, 255))
        text_rect = text.get_rect(bottomleft=(5, info_rect.bottom
- 5))
        pygame.draw.rect(window, (0, 0, 0), text_rect)
        window.blit(text, text_rect)
        for i in range(self.player.max_health):
            x = info_rect.right - (i + 1) * (heart_width +
x_offset)
            y = info_rect.top + y_offset

            if i < self.player.health:
                color = heart_color
            else:
                color = empty_heart_color

            pygame.draw.polygon(window, color, [
                (x, y + heart_height // 2),
                (x + heart_width // 2, y + heart_height),
                (x + heart_width, y + heart_height // 2),
                (x + heart_width // 2, y)
            ])
    def update(self):
        if self.player.moved:
            for enemy in self.enemies:
                enemy.move(self.labyrinth)
            self.player.moved = False
        is_near_or_collision, type_of_interaction, enemy =
self.check_proximity()
        if is_near_or_collision:
            if type_of_interaction == "collision":
                print("Зіткнення!")
            elif type_of_interaction == "proximity":
                fight = Fight(
                    hero=self.player,
                    enemy=enemy
                )
                fight.start()
                print("FIGHT!")

        for enemy in self.enemies:
            if not enemy.is_alive:
```

```
                self.enemies.remove(enemy)
                self.labyrinth.clear(window, enemy.position)

    def draw(self, window):
        self.labyrinth.clear(window, self.player.old_position)
        for enemy in self.enemies:
            self.labyrinth.clear(window, enemy.old_position)

        self.player.draw(window, self.labyrinth)
        for enemy in self.enemies:
            enemy.draw(window, self.labyrinth)

        self.draw_game_info()
        pygame.display.flip()

    def run(self, window):
        self.labyrinth.draw(window)
        while self.running:
            self.handle_events()
            self.update()
            self.draw(window)
            pygame.display.flip()



labyrinth =
Labyrinth(width=width,height=height,width_line=width_line,width_wa
lls=width_walls, border=border)
labyrinth.generate()

player = Player(list(labyrinth.start))

enemy_count = 5
enemies = []
for _ in range(enemy_count):
    while True:
        enemy_x = random.randint(0, width - 1)
        enemy_y = random.randint(0, height - 1)
        if labyrinth.transition_matrix[enemy_x][enemy_y]:
            enemies.append(WeekEnemy([enemy_x,enemy_y]))
            break
game = Game(labyrinth, player, enemies)
game.run(window)
```

***utils.py***

```
def start_point_generate(n, m):

    if random.choice([True, False]):
        if random.choice([True, False]):
            start = (0, random.randint(0, m - 1))
        else:
            start = (n - 1, random.randint(0, m - 1))
    else:
```

```
        if random.choice([True, False]):
            start = (random.randint(0, n - 1), 0)
        else:
            start = (random.randint(0, n - 1), m - 1)
    return start

def finish_point_generate(start, n, m):
    return n - 1 - start[0], m - 1 - start[1]

def transition_choice(x, y, rm):
    choice_list = []
    if x > 0 and not rm[x - 1][y]:
        choice_list.append((x - 1, y))
    if x < len(rm) - 1 and not rm[x + 1][y]:
        choice_list.append((x + 1, y))
    if y > 0 and not rm[x][y - 1]:
        choice_list.append((x, y - 1))
    if y < len(rm[0]) - 1 and not rm[x][y + 1]:
        choice_list.append((x, y + 1))

    if choice_list:
        nx, ny = random.choice(choice_list)
        if x == nx:
            tx, ty = x * 2, ny * 2 - 1 if ny > y else ny * 2 + 1
        else:
            tx, ty = nx * 2 - 1 if nx > x else nx * 2 + 1, y * 2
        return nx, ny, tx, ty
    else:
        return -1, -1, -1, -1
```

***loader.py***

```
from config import display
pygame.init()
print(display.width_window)
window       =       pygame.display.set_mode((display.width_window,
display.height_window))

pygame.display.set_caption("Лабіринт")
```

***config.py***

```
width = 20
height = 20
border = 5
width_line = 40
width_walls = 5
info_height = 70
color_way = (255, 255, 255)
color_wall = (0, 0, 0)
color_player = (0, 0, 255)
color_start = (0, 255, 0)
color_finish = (255, 0, 0)
```

```
class SettingsDisplay:

    @property
    def width_window(self):
        return ((width * 2 - 1) // 2 + 1) * width_line + ((width *
2 - 1) // 2) * width_walls + border * 2

    @property
    def height_window(self):
        return ((height * 2 - 1) // 2 + 1) * width_line + ((height
* 2 - 1) // 2) * width_walls + border * 2 + info_height


display = SettingsDisplay()
```

***entity.py***

```
from pygame import image, transform
from config import width_line
from labirint import Labyrinth
from abc import ABC
import pygame
import random

class EntityObject(ABC):
    path_to_image:image
    max_health:int
    health:int

    def __init__(self, position:list[int,int]) -> None:
        self.position = position
        self.image = image.load(self.path_to_image)
        self.image_resize       =       transform.scale(self.image,
(width_line,width_line))
        self.old_position = self.position.copy()

    @property
    def is_alive(self):
        return self.health > 0

    def draw(self, window, labyrinth:Labyrinth):
        x     =     labyrinth.border     +     self.position[0]     *
(labyrinth.width_line       +       labyrinth.width_walls)       +
labyrinth.width_line // 2 - self.image_resize.get_width() // 2
        y     =     labyrinth.border     +     self.position[1]     *
(labyrinth.width_line       +       labyrinth.width_walls)       +
labyrinth.width_line // 2 - self.image_resize.get_height() // 2

        window.blit(self.image_resize, (x, y))



class WeekEnemy(EntityObject):
```

```
    max_health = 3
    health = 3

    path_to_image = "static/week_enemy.png"

    def __init__(self, position: list[int]) -> None:
        super().__init__(position)

    def move(self, labyrinth):
        x, y = self.position
        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

        random.shuffle(directions)

        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if (0 <= new_x < labyrinth.width and 0 <= new_y <
labyrinth.height and
                    labyrinth.transition_matrix[new_x * 2][new_y *
2] and
                    labyrinth.transition_matrix[x * 2 + dx][y * 2 +
dy]):
                self.old_position = self.position.copy()
                self.position[0] += dx
                self.position[1] += dy
                return

class Player(EntityObject):
    max_health = 5         # Максимальний запас здоров'я гравця
    health = 5             # Поточне здоров'я гравця
    path_to_image = "static/hero-image.jpeg"  # Шлях до зображення
героя
    moved = False          # Чи гравець зробив хід під час поточного
оновлення

    def __init__(self, position: list[int]) -> None:
        super().__init__(position)    # Ініціалізуємо батьківський
клас (EntityObject)

    def handle_movement(self, event, labyrinth):
        """Обробляє рух гравця на основі натискань клавіш."""

        dx, dy = 0, 0  # Зміни по осі X та Y (спочатку руху немає)

        # Визначаємо напрямок руху залежно від натиснутої клавіші
        if event.key == pygame.K_RIGHT or event.key == pygame.K_d:
            dx = 1   # Рух вправо
        elif event.key == pygame.K_LEFT or event.key == pygame.K_a:
            dx = -1  # Рух вліво
        elif event.key == pygame.K_DOWN or event.key == pygame.K_s:
            dy = 1   # Рух вниз
        elif event.key == pygame.K_UP or event.key == pygame.K_w:
            dy = -1  # Рух вгору

        # Розраховуємо нові координати в матриці переходів лабіринту
```

```
        new_x, new_y = self.position[0] * 2 + dx, self.position[1]
* 2 + dy

        # Перевіряємо, чи можна переміститися на нові координати:
        # 1. Чи не виходять вони за межі лабіринту
        #  2.  Чи  є  прохід  (True)  у  матриці  переходів  на  нових
координатах
        if (0 <= new_x < len(labyrinth.transition_matrix) and
            0 <= new_y < len(labyrinth.transition_matrix[0]) and
            labyrinth.transition_matrix[new_x][new_y]):

            self.old_position = self.position.copy()  # Зберігаємо
попередню позицію ПЕРЕД оновленням
            self.position[0] += dx  # Оновлюємо позицію гравця
            self.position[1] += dy
            self.moved = True       # Позначаємо, що гравець зробив
хід
        else:
            self.moved = False      # Гравець не зміг зробити хід
```

### *fight.py*

```
import pygame
from pygame.locals import *
from typing import Tuple
import random
from config import display
import operator

class Fight:
    def __init__(self, hero, enemy):
        # Ініціалізація бою: встановлення учасників, екрану, шрифтів
та кольорів
        self.hero = hero
        self.enemy = enemy
        self.screen = pygame.display.get_surface()
        self.font = pygame.font.SysFont(None, 48)
        self.input_font = pygame.font.SysFont(None, 36)
        self.timer_font = pygame.font.SysFont(None, 24)
        self.color_inactive = pygame.Color('lightskyblue3')
        self.color_active = pygame.Color('dodgerblue2')

        # Завантаження та масштабування фонового зображення
        self.background_image                                    =
pygame.image.load("static/back_fight2.png")
        self.background_image                                    =
pygame.transform.scale(self.background_image,
(display.width_window, display.height_window))

        # Генерація першого питання
        self.generate_question()

    def start(self):
```

```
        # Збереження копії початкового екрану та встановлення режиму
відображення
        self.original_screen = self.screen.copy()
        pygame.display.set_mode((display.width_window,
display.height_window))

        # Запуск основного циклу бою
        self.fight_loop()

        # Відновлення початкового екрану після завершення бою
        self.screen.blit(self.original_screen, (0, 0))
        pygame.display.flip()

    def generate_question(self):
        # Генерація випадкового питання для бою
        num1 = random.randint(1, 50)
        num2 = random.randint(1, 50)

        # Вибір випадкової арифметичної операції та відповідної
функції
        operations = {
            "+": operator.add,
            "-": operator.sub,
            "*": operator.mul,
            "/": operator.truediv  # Додали ділення
        }
        operation_symbol = random.choice(list(operations.keys()))
        operation_func = operations[operation_symbol]

        # Обчислення правильної відповіді з урахуванням типу
операції
        if operation_symbol == "/":
            # Уникаємо ділення на нуль та забезпечуємо цілочисельний
результат
            while num2 == 0:
                num2 = random.randint(1, 50)
            correct_answer = operation_func(num1, num2)
            # Округлюємо результат до найближчого цілого числа
            correct_answer = round(correct_answer)
        else:
            correct_answer = operation_func(num1, num2)

        self.question = f"Скільки буде {num1} {operation_symbol}
{num2}?"
        self.correct_answer = str(correct_answer)

    def fight_loop(self):
        # Основний цикл бою: триває, поки обидва учасники живі
        while self.hero.is_alive and self.enemy.is_alive:
            # Отрисовка фону на початку кожного циклу
            self.screen.blit(self.background_image, (0, 0))

            # Створення та відображення спливаючого вікна з питанням
            self.popup = Popup(self)
            self.popup.display()
```

```
            # Перевірка умови завершення бою (поразка героя)
            if not self.hero.is_alive:
                self.show_end_message(False)
                break
            # Перевірка умови завершення бою (перемога героя)
            if not self.enemy.is_alive:
                self.show_end_message(True)
                break

            # Генерація нового питання для наступного раунду
            self.generate_question()

    def show_end_message(self, status:bool):
        # Відображення повідомлення про завершення бою (перемога або
поразка)
        if status:
            victory_image                                          =
pygame.image.load("static/victory_popup.jpg")
            victory_image  =  pygame.transform.scale(victory_image,
(display.width_window, display.height_window))
            self.screen.blit(victory_image, (0, 0))

        pygame.display.flip()
        pygame.time.wait(1500)

class Popup:
    def __init__(self, fight):
        self.fight = fight
        panel_width = 400
        panel_height = 90
        print(self.fight.screen.get_width())
        print(self.fight.screen.get_height())
        panel_x = (self.fight.screen.get_width() - panel_width) //
2
        panel_y = 680
        self.input_box = pygame.Rect(panel_x, panel_y, panel_width,
panel_height)

        self.active = False
        self.text_input = ''
        self.time_remaining = 6
        self.clock = pygame.time.Clock()
        self.answered = False

    def display(self):
        """Відображає спливаюче вікно з питанням та обробляє
відповідь гравця."""

        # Цикл триває, поки залишається час і гравець не відповів
        while self.time_remaining > 0 and not self.answered:
            self.handle_events()   # Обробляємо події (натискання
клавіш, тощо)

            # Оновлюємо таймер, якщо відповідь ще не дана
```

```
            if not self.answered:
                self.time_remaining -= self.clock.tick(60) / 1000  # Віднімаємо час, що минув

            # Оновлюємо вміст екрану (питання, таймер, здоров'я)
            self.update_screen()

            # Якщо герой помер під час відповіді, виходимо з циклу
            if not self.fight.hero.is_alive:
                return

        # Якщо час вийшов, а гравець не відповів, герой отримує пошкодження
        if not self.answered:
            self.fight.hero.health -= 1

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                exit()

            if event.type == MOUSEBUTTONDOWN:
                self.active = True

            if event.type == KEYDOWN:
                    if event.key == K_RETURN:
                        if self.text_input == self.fight.correct_answer:
                            self.fight.enemy.health -= 1
                        else:
                            self.fight.hero.health -= 1
                        self.answered = True

                    elif event.key == K_BACKSPACE:
                        self.text_input = self.text_input[:-1]
                    else:
                        self.text_input += event.unicode

    def update_screen(self):
        color = self.fight.color_active if self.active else self.fight.color_inactive
        popup = pygame.Surface((self.fight.screen.get_width(), self.fight.screen.get_height()), pygame.SRCALPHA)
        question_text = self.fight.font.render(self.fight.question, True, (255, 0, 0))
        question_rect = question_text.get_rect(center=((self.fight.screen.get_width() // 2), (self.fight.screen.get_height() // 4 )+ 15))
        popup.blit(question_text, question_rect)


        timer_rect = self.fight.timer_font.render(f"Час на відповідь: {round(self.time_remaining, 1)} с", True, (0, 0, 0, 0)).get_rect()
```

```
        timer_rect.topright = (self.fight.screen.get_width() - 10,
10)
        popup.blit(self.fight.background_image,         timer_rect,
timer_rect)

        timer_text    =    self.fight.timer_font.render(f"Час    на
відповідь: {round(self.time_remaining, 1)} с", True, (255, 255,
255))
        popup.blit(timer_text,   (self.fight.screen.get_width()   -
timer_text.get_width() - 10, 10))


        self.draw_health_bars(popup)


        self.fight.screen.blit(popup, (0, 0))
        pygame.display.flip()

    def draw_health_bars(self, popup):
        self.heart_width = 20
        self.heart_spacing = 5
        self.x_offset = 50
        self.y_offset = 50
        total_width      =      self.fight.hero.max_health      *
(self.heart_width + self.heart_spacing) - self.heart_spacing
        start_x = self.x_offset + (200 - total_width) // 2
        for i in range(self.fight.hero.max_health):
            x    =    start_x    +    i    *    (self.heart_width    +
self.heart_spacing)
            color = (255, 0, 0) if i < self.fight.hero.health else
(128, 128, 128)
            self.draw_heart(popup, x, self.y_offset, color)
        if self.fight.enemy:
            start_x = self.x_offset + 500 + (200 - total_width) //
2
            for i in range(self.fight.enemy.max_health):
                x   =   start_x   +   i   *   (self.heart_width   +
self.heart_spacing)
                color = (255, 0, 0) if i < self.fight.enemy.health
else (128, 128, 128)
                self.draw_heart(popup, x, self.y_offset, color)

    def draw_heart(self, surface, x, y, color):
        points = [
            (x + self.heart_width // 2, y + self.heart_width // 4),
            (x + self.heart_width, y),
            (x + self.heart_width * 3 // 2, y + self.heart_width //
4),
            (x + self.heart_width, y + self.heart_width // 2),
            (x + self.heart_width // 2, y + self.heart_width),
            (x, y + self.heart_width // 2),
        ]
        pygame.draw.polygon(surface, color, points)
```

***labirint.py***

```
import pygame
import random
from utils import start_point_generate, finish_point_generate,
transition_choice
from config import color_finish, color_start, color_wall

class Labyrinth:
    def __init__(self, width, height, width_line, width_walls,
border,
                 color_way=(255, 255, 255), color_wall=(0, 0, 0),
                 color_start=(0, 255, 0), color_finish=(255, 0,
0)):

        self.width = width
        self.height = height
        self.width_line = width_line
        self.width_walls = width_walls
        self.border = border
        self.colors = {
            "way": color_way,
            "wall": color_wall,
            "start": color_start,
            "finish": color_finish
        }
        self.reach_matrix = []
        self.transition_matrix = []
        self.wall_image =
pygame.image.load("static/back_plate_1.png")
        self.wall_image_resized =
pygame.transform.scale(self.wall_image, (width_line, width_line))

    def generate(self):
        n, m = self.width, self.height
        for i in range(n):
            self.reach_matrix.append([])
            for j in range(m):
                self.reach_matrix[i].append(False)
        for i in range(n * 2 - 1):
            self.transition_matrix.append([])
            for j in range(m * 2 - 1):
                if i % 2 == 0 and j % 2 == 0:
                    self.transition_matrix[i].append(True)
                else:
                    self.transition_matrix[i].append(False)
        self.start = start_point_generate(n, m)
        self.finish = finish_point_generate(self.start, n, m)

        list_transition = [self.start]

        x, y = self.start


        self.reach_matrix[x][y] = True
```

```
        x, y, tx, ty = transition_choice(x, y, self.reach_matrix)

        # Основний цикл генерації лабіринту
        for i in range(1, m * n):
            # Повернення до попередньої клітинки, якщо поточна
недосяжна
            while not (x >= 0 and y >= 0):
                x, y = list_transition[-1]
                list_transition.pop()
                x, y, tx, ty = transition_choice(x, y,
self.reach_matrix)

            # Відмітка поточної клітинки як досяжної
            self.reach_matrix[x][y] = True

            # Додавання поточної клітинки до списку переходів
            list_transition.append((x, y))

            # Видалення стінки (створення переходу)
            self.transition_matrix[tx][ty] = True

            # Вибір наступного переходу
            x, y, tx, ty = transition_choice(x, y,
self.reach_matrix)

    def is_wall_between(self, pos1, pos2):
        x1, y1 = pos1
        x2, y2 = pos2
        dx = x2 - x1
        dy = y2 - y1

        if dx == 1:
            return not self.transition_matrix[x1 * 2 + 1][y1 * 2]
        elif dx == -1:
            return not self.transition_matrix[x1 * 2 - 1][y1 * 2]
        elif dy == 1:
            return not self.transition_matrix[x1 * 2][y1 * 2 + 1]
        elif dy == -1:
            return not self.transition_matrix[x1 * 2][y1 * 2 - 1]
        else:
            return False



    def draw(self, window):
        for i in range(len(self.transition_matrix)):
            for j in range(len(self.transition_matrix[0])):
                x = self.border + (i // 2) * (self.width_line +
self.width_walls) + (i % 2) * self.width_line
                y = self.border + (j // 2) * (self.width_line +
self.width_walls) + (j % 2) * self.width_line
                if self.transition_matrix[i][j] == 1:
```

```
                    window.blit(self.wall_image_resized, (x, y))
                else:
                    pygame.draw.rect(window, color_wall, (x, y,
self.width_line, self.width_line))

        pygame.draw.rect(window, color_start, (
            self.border + self.start[0] * (self.width_line +
self.width_walls), self.border + self.start[1] * (self.width_line
+ self.width_walls), self.width_line,
            self.width_line))
        pygame.draw.rect(window, color_finish, (
            self.border + self.finish[0] * (self.width_line +
self.width_walls), self.border + self.finish[1] * (self.width_line
+ self.width_walls), self.width_line,
            self.width_line))


    def clear(self, window, entity_position: list[int, int],
background_color=(0, 0, 0)):
        x = self.border + entity_position[0] * (self.width_line +
self.width_walls)
        y = self.border + entity_position[1] * (self.width_line +
self.width_walls)
        if self.transition_matrix[entity_position[0] *
2][entity_position[1] * 2]:
            window.blit(self.wall_image_resized, (x, y))
        else:
            pygame.draw.rect(window, background_color, (x, y,
self.width_line, self.width_line))
```