МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

(повна назва інституту/факультету)

Кафедра комп'ютеризованих систем керування

(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри КСУ

_____ _Петро ЛЕОНТЬЄВ

(підпис) (Ім'я та ПРІЗВИЩЕ)

____ _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

(бакалавр / магістр)

зі спеціальності 153 – «Мікро- та наносистемна техніка»

(код та назва)

освітньо-наукової програми Нанотехнології та біомедичні системи

(освітньо-професійної / освітньо-наукової) (назва програми)

на тему «*Використання алгоритмів штучного інтелекту при проектуванні систем навігації у симуляторах надання невідкладної допомоги*»

Здобувача групи ФЕ.м-21 Сичова Дениса Миколайовича

(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Денис СИЧОВ

(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент каф. КСУ, канд. фіз.-мат. Наук Вадим БОРИСЮК _____

(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

_____*Сумський Державний Університет*_____
(назва вузу)

Факультет ___*ЕлІТ*__ Кафедра *комп'ютеризованих систем управління*

Спеціальність_____*153 «Мікро- та наносистемна техніка»*_____

ЗАТВЕРДЖУЮ:

Зав. Кафедрою_____

_____

«__»_____20__р.

ЗАВДАННЯ

НА____*магістерську*____ РОБОТУ СТУДЕНТОВІ
магістерська/ бакалаврську

_____*Сичова Дениса Миколайовича*_____
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Використання алгоритмів штучного інтелекту при проектуванні систем навігації у симуляторах надання невідкладної допомоги*

2. затверджені наказом по університуту від "___"_____20__року №_____

3. Термін здачі студентом закінченого проекту (роботи) _20.12.2023 р.

4. Вхідні дані до проекту (роботи) *Провести аналіз алгоритмів пошку для інтеграції та використання їх у веб додатку. Створити веб додаток з функціоналом для пошкук коротких шляхів та покращення досвіду користування. Провести аналіз вихідних данних роботи алгоритму.*

5. Перелік графічного матеріалу (з точним зазначення обов'язкових креслень) *Презентація із наведеними результатами проведеного аналізу*, що складається із 29 змістовних слайдів (титульний слайд , актуальність роботи, досліджувані теоретичні моделі і їх характеристики, зображення інтерфейсу, графіки.

6. Дата видачі завдання: 16.11.2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

РЕФЕРАТ

Записка: 56 сторінок, 31 рисунків, 30 джерел.

Об’єкт дослідження — штучний інтелект, алгоритми, розробка інтерфейсу.

Мета роботи — метою цього проекту є створення веб-програми, яка візуалізує дії алгоритму пошуку вшир, щоб визначити найкоротший шлях на графіку. Додаток полегшить взаємодію користувача з алгоритмом і дозволить спостерігати за процесом в реальному часі. Візуалізація полегшить розуміння різних етапів алгоритму, його призначення та ефективності в пошуку найкоротшого шляху на графі.

Результати —результатом цієї роботи є функціональний веб-додаток, який дозволяє користувачам відслідковувати та досліджувати прогрес алгоритму пошуку вшир, щоб визначити найкоротший шлях на двовимірному полі.

Зміст

ВСТУП

У сучасному світі технологій, що швидко розвиваються, використання алгоритмів штучного інтелекту стає ключовим елементом у проектуванні та розробці навігаційних систем у симуляторах, які використовуються для надання невідкладної допомоги. Одним із важливих аспектів таких систем є взаємодія з пошуковими алгоритмами, які відіграють вирішальну роль у визначенні найкращого маршруту та ефективної навігації в навколишньому середовищі.

Ця робота спрямована на вивчення та реалізацію алгоритму пошуку в ширину в області проектування навігаційних систем для тренажерів, які використовуються для моделювання аварійних ситуацій. У цьому випадку алгоритми штучного інтелекту допомагають оптимізувати просторове переміщення та приймати зважені рішення, щоб допомогти в критичних ситуаціях найшвидше та найефективніше.

У цій роботі досліджується використання метода пошуку в ширину в мові програмування JavaScript з метою візуалізації та оцінки його ефективності в гіпотетичному сценарії. Розроблена програма дозволяє дослідникам спостерігати за роботою алгоритмів у реальному часі, що сприяє кращому розумінню впливу алгоритму на навігацію та вибір найбільш підходящого маршруту.

Описана розробка має потенціал для розширення можливостей навігаторів і покращення діагностики в тренажерах, які б надавали невідкладну допомогу. Впровадження штучного інтелекту в ці системи може призвести до створення більш ефективних та інтелектуальних рішень, які підвищать якість обслуговування та допоможуть у надзвичайних ситуаціях.

РОЗДІЛ 1

ШТУЧНИЙ ІНТЕЛЕКТ ТА ЙОГО СУТНІСТЬ

Штучний інтелект — це галузь інформатики, яка стосується створення машин, здатних виконувати завдання, що потребують людського інтелекту. У цій галузі створюються алгоритми та моделі, які дозволяють системам аналізувати інформацію, використовувати дані для навчання, вирішувати проблеми та приймати рішення.

Штучний інтелект використовує кілька підходів для досягнення своїх цілей:

- машинне навчання: цей метод дозволяє машинам навчатися без додаткового програмування, використовуючи дані для автоматизації та підвищення результатів.

- глибоке навчання: це більш просунута форма машинного навчання, яка використовує нейронні мережі для вирішення таких складних проблем, як розпізнавання шаблонів і розуміння мови.

- еволюційні алгоритми: процедура, яка моделює процес еволюції, щоб покращити вирішення проблем і створити оптимізацію.

1.1 Штучний інтелект у сучасному світі

У сучасну еру технологічного прогресу штучний інтелект (ШІ) відіграє значну роль у різних аспектах життя. Ця спеціальна галузь дослідження зосереджена на розробці систем і програм, які можуть виконувати завдання, які зазвичай вимагають людського інтелекту [1].

У сучасному суспільстві є кілька фундаментальних компонентів штучного інтелекту, на які варто звернути увагу:

Машинне навчання — це спеціалізована область у сфері штучного інтелекту, яка дозволяє комп’ютерам навчатися на основі даних, не вимагаючи явного програмування. Алгоритми, які використовуються в машинному

навчанні, призначені для визначення закономірностей, передбачення тенденцій і автоматизації широкого кола завдань.

Область обробки природної мови (NLP) спрямована на те, щоб комп'ютери могли розуміти, інтерпретувати та створювати людську мову. НЛП має різноманітні програми, включаючи переклад, розпізнавання мови, створення чат-ботів та інші завдання, пов'язані з обробкою мови.

Впровадження штучного інтелекту в області комп'ютерного зору виявилося цінним інструментом для аналізу [1] та ідентифікації зображень і відео. Наслідки цієї технології виходять далеко за рамки комп'ютерних наук і можуть бути використані в медицині, автономному керуванні автомобілем, безпеці та багатьох інших сферах.

Застосування штучного інтелекту в багатьох галузях промисловості призвело до автоматизації повсякденних і повторюваних завдань. Це особливо очевидно у сферах виробництва та бізнес-процесів, де зазвичай використовуються автоматизація та робототехніка [2]. Штучний інтелект має можливість ретельно досліджувати величезні масиви даних і надавати пропозиції та допомогу для прийняття рішень у різних сферах, зокрема в науці, бізнесі тощо, за допомогою систем підтримки прийняття рішень.

Медицина використовує штучний інтелект різними способами. Одним із найбільш помітних застосувань є аналіз медичних зображень, а також ідентифікація різних захворювань і станів. Крім того, ШІ бере участь у створенні інноваційних ліків та інших медичних технологій.

Використання штучного інтелекту викликає етичні проблеми та проблеми безпеки. Ці питання включають керування алгоритмами, запобігання зловживанням і захист конфіденційності даних.

1.2 Штучний інтелект у пошуку шляхів

Використання штучного інтелекту (ШІ) відіграє важливу роль у вдосконаленні та оптимізації процедур визначення шляху.

Штучний інтелект (ШІ) покладається на ряд алгоритмів пошуку, щоб ефективно знаходити шляхи в різних доменах, включаючи транспортні системи та мережеву маршрутизацію [3]. Ці алгоритми включають алгоритм Дейкстри і генетичні алгоритми, які можна налаштувати відповідно до різних функцій пошуку шляху.

Удосконалення транспортних систем є одним із багатьох застосувань штучного інтелекту. Беручи до уваги низку факторів, як-от потік транспорту, погодні умови та стан доріг, ШІ може підвищити ефективність транспортних мереж. Приклади таких удосконалень включають використання навігаційних систем в автомобілях та оптимізацію планування маршрутів у громадському транспорті. Технології штучного інтелекту в логістиці в основному зосереджено на оптимізації процесів. Це включає в себе вибір оптимального розташування складу, оптимізацію управління ланцюгом поставок і оптимізацію маршрутів доставки.

Процес розробки безпілотних транспортних засобів передбачає впровадження штучного інтелекту для створення складних систем, здатних самостійно планувати маршрут, уникати перешкод і оперативно реагувати на будь-які зміни [4] в водійському середовищі.

Використання штучного інтелекту значно вплинуло на сферу географії та суміжних дисциплін через геопросторовий аналіз. Цей аналітичний процес передбачає вивчення геопросторових даних для визначення найбільш підходящого розташування для різних об’єктів та інфраструктури. Крім того, він також використовується для планування та виконання завдань у цій сфері. Досліджувати незнайомі та потенційно небезпечні території можна за допомогою роботів та дронів зі штучним інтелектом. Ці пристрої мають здатність працювати автономно, що дозволяє їм заходити в райони, які невідомі або вважаються небезпечними для дослідження людиною.

Об’єднання геодезичних даних, датчиків та інших інформаційних джерел зі штучним інтелектом призводить до вмілого та точного планування шляхів у багатьох областях.

1.3 Використання штучного інтелекту в двовимірному просторі комп'ютерних ігор

У двовимірних іграх штучний інтелект має значний вплив на поведінку комп’ютерних персонажів, що має на меті відобразити реалістичну поведінку комп’ютерів, незалежно від того, є вони друзями чи ворогами. Основною метою впровадження штучного інтелекту є надання гравцеві досвіду змагання з розумними та стратегічними віртуальними супротивниками. Штучний інтелект робить це за допомогою алгоритмів, які шукають і вибирають найвигідніші шляхи, якими слід скористатися або уникнути.

Дизайн 2D-ігор обертається навколо створення систем поведінки персонажів, які адаптуються до змінних станів гри. Такі дії, як атака, ухилення та захист, розпізнаються та реагують на них, тоді як ворожий ШІ використовує тактичні сильні та слабкі сторони гравця для реалізації різних ігрових стратегій [5]. У той час як штучний інтелект деяких ігор використовує повторювані передбачувані сценарії, інші складні ігри включають алгоритми навчання, які спостерігають за поведінкою гравців і відповідно приймають рішення в грі. Це створює захоплюючий досвід для гравців.

У грі "Super Mario Bros." Інші ігри серії «Супер Маріо» використовують штучний інтелект, щоб регулювати гру та створювати труднощі для гравця. Ось головні пункти використання ШІ в Super Mario:

1) Рух і поведінка супротивників: вороги в "Супер Маріо" мають свої методи пересування і взаємодії з гравцем. Наприклад, черепахи можуть подорожувати по платформах і змінювати напрямок, коли наближаються до краю. Це створює труднощі для гравця та вимагає стратегічного розгляду.
2) Поведінка боса та його ворогів до завершення рівня: після завершення кожного рівня гравець зазвичай стикається з босом або особливо могутнім супротивником. ШІ впливає на їх поведінку, атаки та вразливі зони. Це збільшує складність останніх етапів рівнів.

3) Взаємодія з навколишнім середовищем: іншим критичним елементом є взаємодія між персонажами та їхнім оточенням. Наприклад, вороги можуть реагувати на перешкоди, змінюючи свій рух або намагаючись обійти перешкоду.
4) Складність негараздів: у рідкісних випадках ШІ адаптується до рівня досвіду гравця. Якщо гравець схильний легко проходити рівні, ШІ може збільшити складність, створивши більше ворогів або змінивши їхню поведінку.
5) Імітація емоцій: деякі вороги могли прикинутися емоційними або в стані хаосу. Наприклад, деякі вороги можуть сприймати гравця як загрозу і дистанціюватися від нього, а інші які проаналізувавши слабкості гравця намагатися атакувати його.

На основі вище перерахованих пунктів можна зробити висновок що штучний інтелект використовували ще у минулому, для задання цікавості іграм, надаючи можливість ігровим некерованим персонажам надавати відчуття живих істот, які можуть за допомогою алгоритмів будувати короткі шляхи, аналізувати дії гравця та адаптуватися під всі дії у віртуальному середовищі.

РОЗДІЛ 2

АЛГОРИТМИ ПОШУКУ ТА ЇХ ВИКОРИСТАННЯ

Алгоритми, похідні від арабської назви перського математика Аль-Хорезмі, складаються з серії кроків, призначених для вирішення проблеми за обмежену кількість кроків. Ці набори правил забезпечують структуру для виконання певної процедури з конкретною метою, яка буде досягнута за певний проміжок часу. Крім того, блок-схеми часто використовуються для візуалізації алгоритмів.

2.1 Що являє собою поняття алгоритм

Алгоритм — це ряд конкретних інструкцій, які керують обчислювальним процесом комп'ютера. Він починається зі стану, який є логічним, а потім прогресує через низку станів, перш ніж досягти кінцевого стану. Перехід між станами не завжди може бути прямим, а певні методи можуть містити випадкові компоненти. Математика популяризувала ідею алгоритмів як важливого компонента. Багатовікові знання людства щодо обчислювальних процесів включають алгоритми, ці процеси включають знаходження залишку числа та витягання квадратного кореня з числа [6].

Як передумова успіху, кожен алгоритм передбачає наявність вихідної інформації, а звідти слідує курсу дій, який у кінцевому підсумку призводить до певного результату. Послідовність окремих основних кроків, які виконуються під час процесу, називають «кроками», а виконання цієї серії дій називають «алгоритмічним процесом». Таким чином, можна зробити висновок, що алгоритм працює на основі окремої, дискретної властивості.

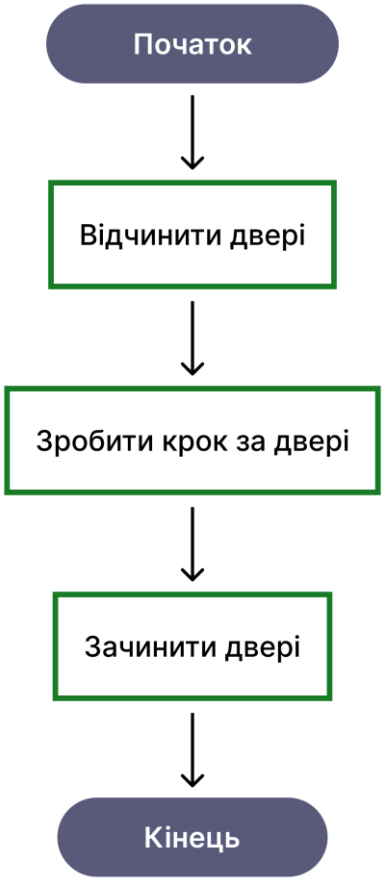

Рис. 2.1. Проста концепція алгоритму.

Кожен алгоритм створено для вирішення проблем певного типу, передаючи його важливу особливість — бути масивним у своїх можливостях вирішення проблем. Щоб алгоритм вважався дійсним, він повинен задовольняти життєво важливу передумову: детермінізм. Це передбачає те, що всі кроки в алгоритмі виконуються послідовно й без відхилень, створюючи єдиний вихід з ідентичних вхідних даних.

Виконання алгоритму може зіткнутися з тупиковою блокуванням або стати нескінченним, коли застосовується до недійсних вхідних даних, підкреслюючи важливість обмеження вхідних даних визначеним прийнятним набором.

Математична валідація комп’ютерних систем є одним із методів уникнення програмних помилок у ситуаціях із високим ризиком відмови через поширення інформаційних технологій.

Процес визначення та вивчення алгоритмів, а також їх реалізації за допомогою математичних методів зазвичай називають формальними методами. У цих практиках використовуються формальні правила аналізу та набір інструментів перевірки. Докладаючи зусиль для реалізації речей окремо, властивості системи можуть бути ізольовані від реалізації. Крім того, точність і ясність математичних тверджень усуває неоднозначність і неточність, присутню в природних мовах.

«Перевірка програми вирішує питання правильності, документації та сумісності», згідно з теорією Гоара. Забезпечення ефективності програм дозволяє нам розпізнавати їхні властивості в повному діапазоні вхідних параметрів, філософію, яку підтримує Річард Мейс. Для доведення коректності програм, поняття коректності було розширене на два типи – часткова коректність та повна коректність.

Часткова коректність - це коректність програми, яка дає правильний результат лише для обмеженого діапазону вхідних даних, але їй все ще потрібно виконати своє завдання. Ця властивість полегшує взаємодію програми з конкретними областями вхідних даних і забезпечує певний ступінь впевненості в тому, що після завершення програми результат буде точним. Однак програма не вимагає висновків щодо будь-яких можливих вхідних даних.

Повна коректність є ширшою, ніж часткова коректність, вона вимагає від програми висновку та отримання відповідного результату для кожного компонента можливого діапазону вхідних даних. Це означає, що програма повинна мати можливість правильно обробляти будь-які потенційні вхідні дані та завжди повертати відповідний результат.

В результаті формальні методи, зокрема методи Гоара, полегшують математичну перевірку правильності програм і систем. Фундаментальна концепція полягає у встановленні передумов і постумов за допомогою термінів,

які описують стан системи до і після виконання програми. Трійки Гоара, у цьому контексті має вигляд :

$$P\{Q\}R \quad (2.1)$$

У цій формулі:

- P — це передумова, яка виражається як умова, яка має бути виконана перед початком програми Q.
- Q - програма або частина програми, яка буде вивчатися.
- R є постумовою, яка виражається як умова, яка має бути досягнута після завершення виконання програми Q.

Це може включати такі аспекти, як точність результату, точність виконання завдань, обмеження часу, який можна використати, тощо.

2.2 Алгоритми пошуку

Алгоритм пошуку — це ряд дій, призначених для вирішення певної проблеми пошуку. Для цього можуть використовуватися різні підходи та методи. Структура даних, про яку йде мова, може бути виражена різними способами, включаючи дерева пошуку, хеш-таблиці, масиви, пов’язані списки та інші методи зберігання інформації. Залежно від конкретного складу даних, алгоритм пошуку може використовувати різні методи [8]. Його мета - знайти конкретні деталі.

Знання того, що шукати, має вирішальне значення для розробки алгоритму пошуку. Ефективні методи характеризуються проведенням часу зі структурою даних. Використання лінійного алгоритму пошуку підходить для пов’язаних списків, але двійковий пошук ефективний для відсортованих масивів. Команди, специфічні для структури даних або її операцій, часто включаються в алгоритми пошуку. Ці команди сприяють створенню, зміні або знищенню елементів, які можуть бути використані для спеціального пристосування процедури до конкретних вимог і контексту завдання пошуку.

Рис. 2.2. Приклад розширеного алгоритму пошуку

На практиці зазвичай використовується лінійний пошук (зображенний на рисунку «2.1») , незважаючи на його нижчу швидкість [8], ніж двійковий пошук. Класифікація пошукових алгоритмів базується на їхньому механізмі пошуку. Алгоритм лінійного пошуку перевіряє кожен запис у структурі на наявність потрібного значення, але є більш ефективний метод так званого бінарного пошуку або пошуку напівінтервалів. Цей алгоритм ефективний лише з елементами, які впорядковані, але він все одно є одним із найшвидших і завжди порівнюватиме пошуковий елемент із центральним елементом даної структури. Процес пошуку або зупиняється, або продовжується залежно від компонентів, при цьому пошук розгалужується ліворуч або праворуч від структури, щоб знайти правильний результат. Ефективні алгоритми пошуку хеш-таблиць мають велику складніть. Компоненти [9] структури пов'язані з ключами, які впорядковують дані. Щоб знайти елемент за його ключем, хеш-функція має бути першокласною, тобто має бути макимально чистою за своєю струкутрою. Алгоритм пошуку знайде розташування ключа, а потім дослідить значення ключа.

Бінарний пошук має максимальну складність (або логарифмічний час виконання задачі) яка виражаєтеся $O(log(n))$. Це веде за собою те що максимальною кількістю операцій для виконання пошуку має вигляд:

$$Cmp_{max} = log_2 n \qquad 2.2$$

У цьому рівнянні n – кількість елементів.

2.3 Типи алгоритмів пошуку

Алгоритми пошуку діляться на кілька основних категорій залежно від стратегії та методу, який вони використовують для пошуку елемента в структурі даних.

2.3.1 Лінійний пошук

Лінійний пошук — це метод послідовного дослідження простору пошуку заданого значення певної функції на певному його сегменті. Цей алгоритм є найпростішим з алгоритмів пошуку, на відміну, наприклад, від бінарного пошуку, він не має обмежень по функціях і простий у реалізації. Пошук значення функції здійснюється шляхом простого порівняння наступного розглянутого значення (як правило, це робиться від меншого значення аргументу до більшого значення аргументу), і якщо значення збігаються, пошук вважається завершеним.

Коли кількість елементів у сегменті невелика, найефективнішим є лінійний пошук. Звичайно, він не такий ефективний, як інші методи, але він не вимагає додаткового зберігання чи обробки на додаток до джерела, що робить його ідеальним для потокового передавання. Крім того, лінійний пошук зазвичай використовується як форма максимального чи мінімального пошуку, яка працює на лінійному графіку[10].

2.3.2 Інтерполяційний пошук

Алгоритм інтерполяційного пошуку — це алгоритм, який використовує заданий ключ для пошуку заданого значення в масиві, упорядкованому за значенням ключів. Це схоже на те, як люди шукають певний номер телефону в книзі. На кожному кроці позиція елемента обчислюється в полі пошуку на основі граничних значень цього поля та ключа, пов'язаного з шуканим елементом, зазвичай за допомогою лінійної інтерполяції [11].

Ключ у вибраній позиції порівнюється з ключем пошуку, якщо вони не ідентичні, то простір для пошуку скорочується до частини, що передує або слідує даному ключу [12]. Такий підхід буде ефективним лише за умови виправданого контрасту між клавішами.

Наприклад, двійковий пошук завжди вибиратиме середню точку в полі пошуку заданого розміру та пропускатиме одну з половин, середнє значення порівнюватиметься зі значенням пошуку. А лінійний пошук оцінює кожен елемент окремо, без урахування їх порядку.

2.3.3 Дерево пошуку

В інформатиці дерево пошуку — це структура даних, яка нагадує дерево. Його мета полягає в тому, щоб полегшити пошук конкретних ключів у певному наборі. Щоб функціонувати як дерево пошуку, ключ кожного вузла має бути більшим за ключі у піддеревах ліворуч від вузла та меншим за ключі у піддеревах праворуч.

Дерево пошуку має перевагу щодо ефективності пошуку, оскільки дерево збалансовано до достатньої міри, що листя на обох кінцях мають порівнянну глибину. Дерева пошуку вважаються різними типами структур даних [13], деякі з яких також мають швидкий спосіб додавання або видалення компонентів. Однак операції з деревами повинні гарантувати, що зберігається їх нейтральність.

Дерева пошуку часто використовуються як засіб реалізації асоціативного масиву. Це досягається за допомогою алгоритму дерева пошуку, який

використовує ключ із пари ключ-значення для визначення конкретного розташування.

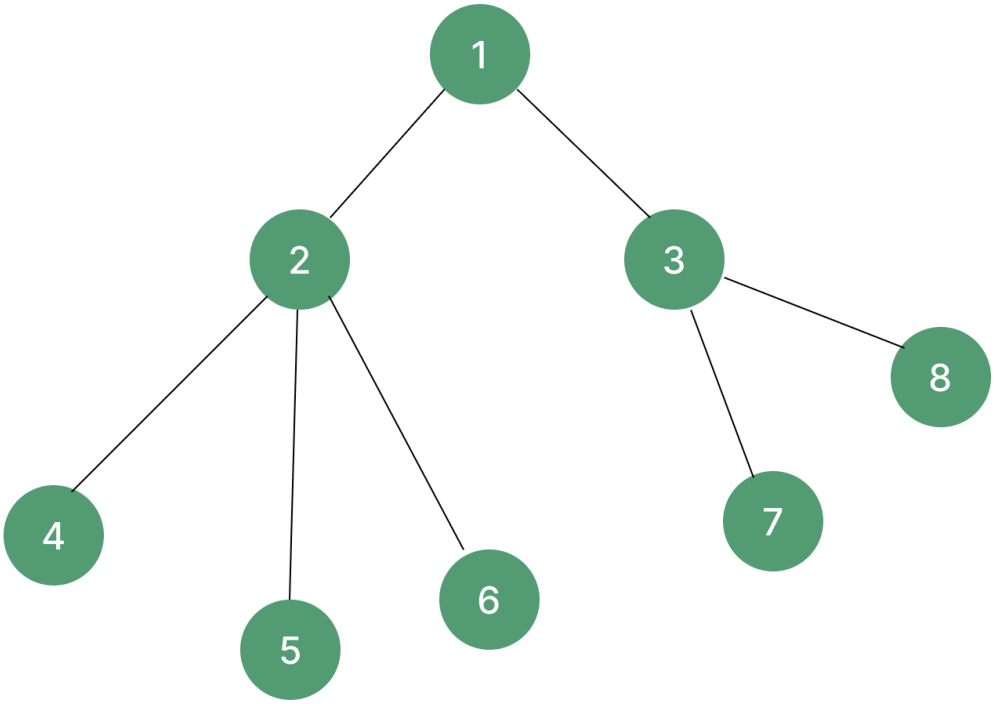

Рис. 2.3. Вигляд дерева пошуку

2.3.4 Обхід графу

В інформатиці пошук графа (або обхід графа) — це обхід (перевірка або оновлення) кожної з вершин графа. Ці алгоритми пошуку класифікуються за порядком, у якому вони проходять через вершини [14]. Пошук по дереву є незвичайним випадком пошуку по графах.

Під час використання пошуку на графі вам може знадобитися повторно перевірити певні вершини кілька разів, на відміну від пошуку в дереві. Це пояснюється тим, що немає гарантії, що вершина була оцінена під час нового відвідування. У міру збільшення заповненості графа ця надмірність стає більш очевидною, що призводить до збільшення часу обчислення.

Як правило, більш вигідно зберігати інформацію про вершини, які вже були перевірені алгоритмом, щоб зменшити частоту надлишкових перевірок або запобігти нескінченному пошуку. Щоб досягти цього, кожна вершина може бути позначена як така, що має «колір» або «стан трафіку», який алгоритм розпізнає та змінює під час проходження кожної вершини графа. Якщо вершина вже зустрічалася, алгоритм ігнорує її та припиняє дослідження в цьому напрямку. Якщо вершина ще не була відвідана, алгоритм оцінить і перегляне можливість руху в цьому напрямку, перш ніж продовжити.

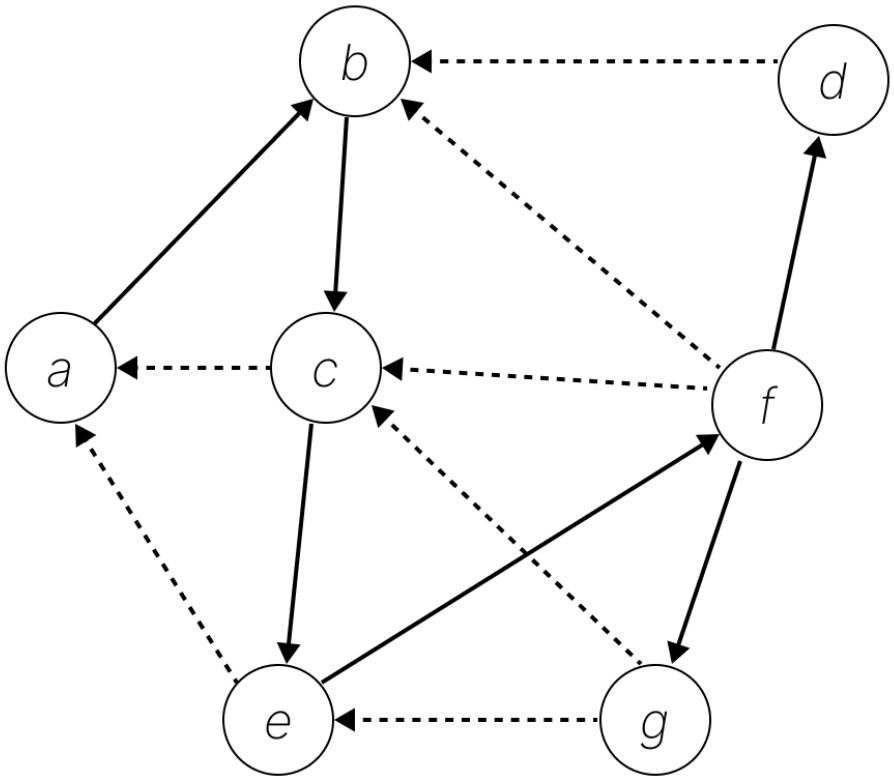

Рис 2.4. Вигляд обходу графів

2.4 Алгоритми обходів графів

Алгоритми обходів графів ділитися на декілька головних підтипів які мають свою систему аналізу даних та виводу інформації :

1. Пошук в глибину (Depth-First Search)
2. Пошук в ширину (Breadth-First Search)

3. Топологічне сортування (Topological Sort)

4. Алгоритм Беллмана-Форда (Bellman-Ford Algorithm)

2.4.1 Пошук в глибину

Пошук у глибину (Depth-First Search) — це метод обходу дерева, деревоподібної структури або графа. Алгоритм починається з кореня дерева (або іншої визначеної вершини в графі) і досягає максимально можливої глибини перед переходом до наступної вершини.

Припустимо, що G = (V, E) — це простий зв’язний граф, і кожна вершина позначена парами унікальних символів [15]. Оскільки вершини в G досліджуються за допомогою пошуку в глибину, їм надають номери DFS і зберігають у стеку. Цей тип зберігання даних дотримується принципу "останній прийшов - перший вийшов" (LIFO), тобто можна видалити лише найновіший елемент, доданий до стеку. Верхній кінець стека – це місце, де додаються та видаляються елементи. Номери DFS, призначені вершині x, називаються DFS(x).

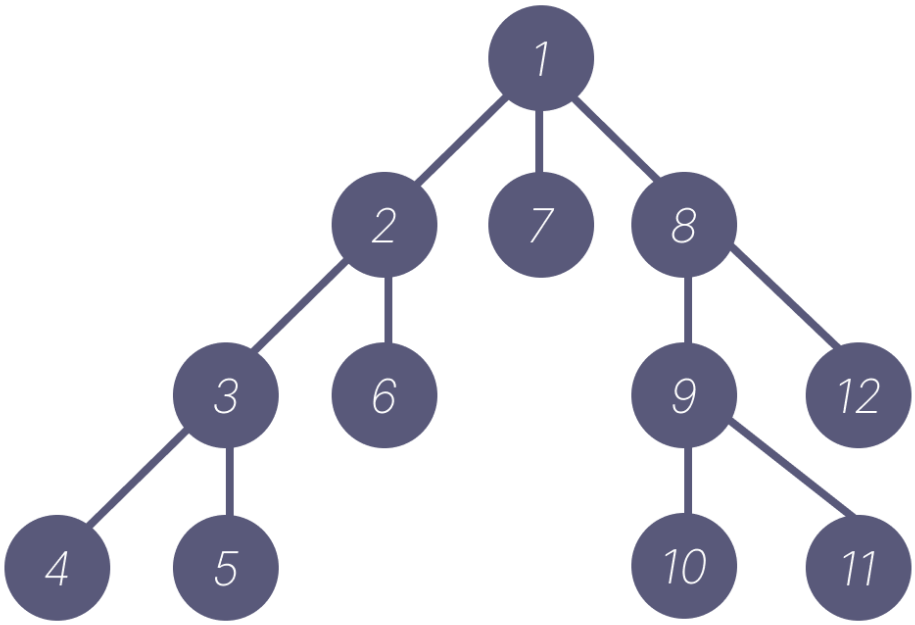

Рис. 2.5. Порядок обходу вершин у алгоритмі пошуку в глибину

У певних випадках пошук у глибину може призвести до повного дослідження всіх гілок графа до того, як буде знайдено або побудовано рішення. Це може статися, коли структура графа є великою та різнобарвною, або коли немає єдиного шляху розв'язання, а алгоритм продовжується в усіх можливих напрямках. В деяких випадках, коли графік має кілька незалежних гілок, пошук у глибину може досліджувати всі можливі шляхи, лише тоді він знайде вирішальний шлях або шляхи [16]. Цей підхід корисний у ситуаціях, коли необхідно розглянути кілька можливостей або коли вирішення завдання передбачає ітерацію різних варіантів.

2.4.2 Пошук в ширину

Пошук в ширину теж охоплює обхід графів. Маючи граф G = (V, E) і початкову вершину S, алгоритм пошуку в ширину спочатку відвідує всі вершини, до яких можна дістатися з S, без відвідування інших проміжних вершин. На кожному наступному кроці всі вершини в поточному списку позначаються як пройдені, а новий список складається з решти вершин, які є суміжними з поточним списком вершин, але ще не були пройдені. Для відстеження вершин часто використовуються черги та правило «перший прийшов – перший вийшов» [17]. Алгоритм триватиме, доки не буде досягнуто бажаної вершини або поки певний крок не створить нові вершини, які слід додати до списку.

Останній сценарій передбачає, що всі початкові вершини доступні з початкової позиції, а шлях до передбачуваної цільової вершини неможливо знайти.

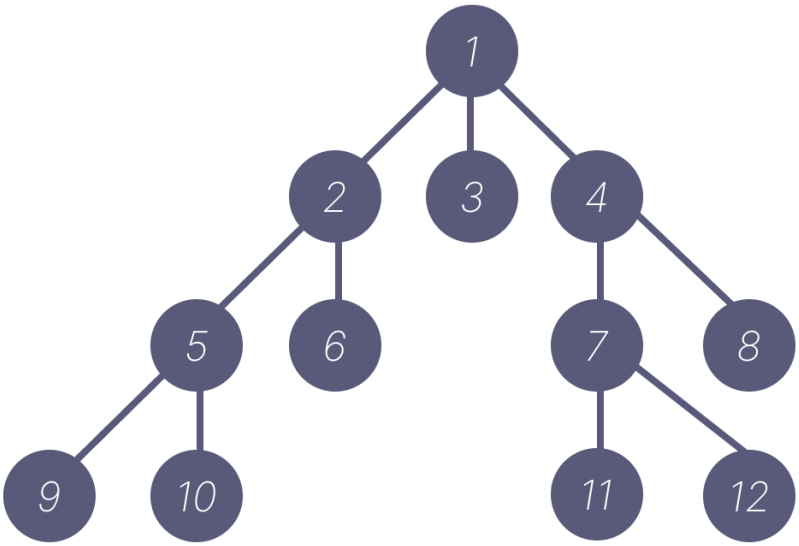

Рис. 2.6. Демонстрація перебору у ширину

2.4.3 Топологічне сортування

Топологічне сортування — це впорядкування вершин орієнтованого ациклічного графа (DAG), який дотримується того самого шаблону, що й ребро:кожне ребро вказує на наступну вершину, яка з’єднана з попередньою вершиною шляхом. Це впорядкування присутнє лише для DAG, оскільки інші графи можуть мати цикл, і створення топологічного порядку неможливо [18].

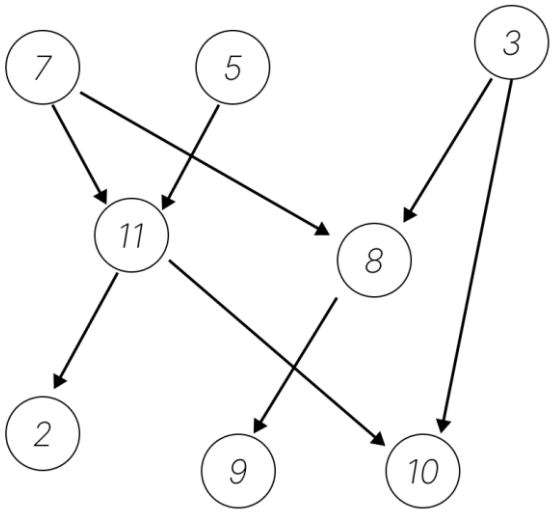

Рис. 2.7. Безконтурний арієнтований граф

2.4.4 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда — це метод пошуку найкоротшого шляху в графі, включаючи графи з від'ємними вагами ребер. Основна відмінність між алгоритмом Беллмана-Форда та іншими алгоритмами на основі графів, такими як Дейкстра, полягає в тому, що останній може обробляти графи, які мають негативні ваги ребер [19].

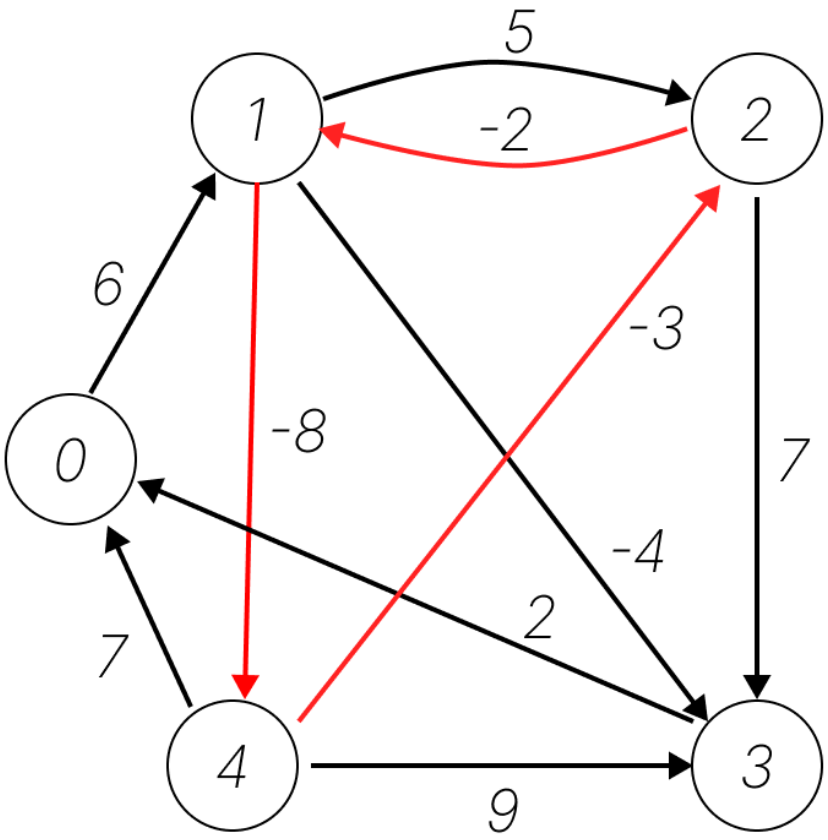

Рис. 2.7. Алгоритм Беллмана-Форда на графах

Алгоритм Беллмана-Форда також корисний для вирішення проблем у деяких сценаріях, але його обчислювальна складність становить O (V * E), де V – кількість вершин, E – кількість ребер, і він менш ефективний, ніж алгоритми для незважені графи або графи з інтегральними вагами на своїх стрічках [20].

2.5 Алгоритми в іграх

Алгоритми відіграють значну роль у розробці ігор, ці особливості включають кілька ключових принципів, які визначають якість і різноманітність ігрового процесу. Одним із основних застосувань алгоритмів є створення штучного інтелекту (ШІ), який використовує ШІ, щоб забезпечити персонажам розумну поведінку та реакцію на різні ситуації. Крім того, управління персонажами в іграх обертається навколо створення алгоритмів, які керують рухом, стратегією та взаємодією гравців у віртуальному просторі.

Крім того, генерація рівнів в іграх використовує алгоритми, які створюють цікаві унікальні локації, якими гравці зайняті. Іншими важливими функціями алгоритмів є оптимізація продуктивності гри та запобігання затримкам і зависанням у грі.

2.5.1 Генерація рівнів

Створення рівнів у відеоіграх – це складна процедура, яка потребує ретельного планування та виконання. Створення цих рівнів передбачає низку кроків, які потрібно виконати, щоб гарантувати, що гра буде інтригуючою та складною для гравців. Процедура передбачає розробку розташування кожного рівня, вибір розташування перешкод, ворогів і переваг, а також збалансування складності кожного етапу [21]. Життєво важливо знайти баланс між достатньою складністю, щоб підтримувати гравців, а також уникати надмірного розчарування. Зрештою, успіх відеоігри залежить від якості створених рівнів.

Сьогодні створення рівнів є важливою складовою досвіду гри, яка намагається створити веселе та різноманітне середовище для гравців. Реалізація алгоритмів генерації рівнів приносить переваги як універсалізації, так і оптимізації процесу створення рівнів, це гарантує, що кожен етап буде різним і привабливим для гравців.

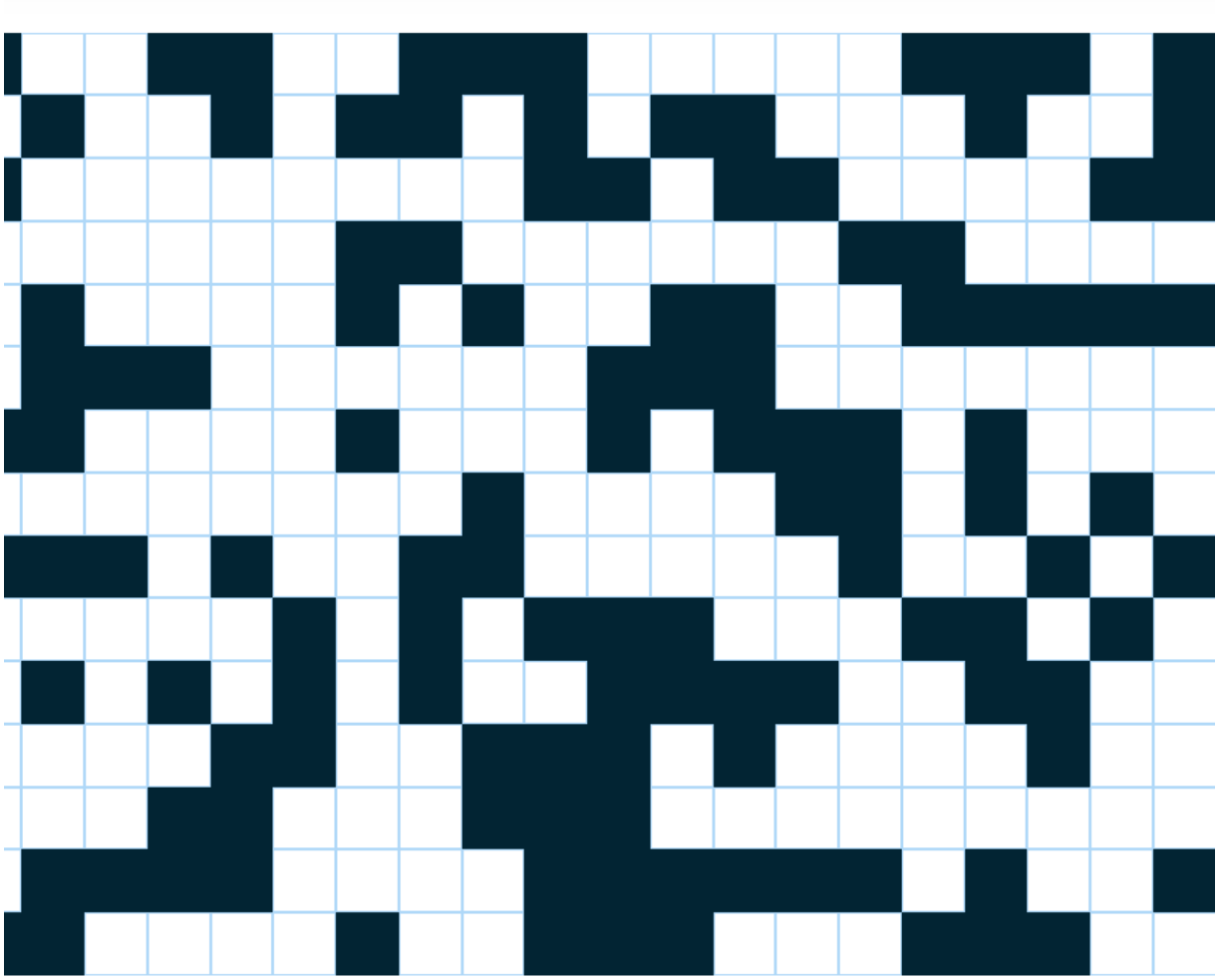

Рис. 2.8. Збудований рівень з допомогою рекурсивного алгоритму

2.5.2 Керування персонажами

У розробці ігор керування персонажами має вирішальне значення. Комп'ютери, які відповідають за регулювання персонажа, мають значний вплив на забезпечення цікавих і справжніх ігрових вражень. Основними компонентами управління персонажами є методи навігації, обчислення шляху, методи взаємодії з іншими персонажами та оптимізація ефективності персонажів у реальному світі.

Рух і взаємодія між персонажами в іграх здійснюється за допомогою складних рівнянь. Ці алгоритми включають навігаційні програми, які визначають оптимальний маршрут, а також стратегії взаємодії, які керують поведінкою персонажів на основі умов гри та події. Автентичність і правдивість світу гри

випливає з того, як реалізовано ці аспекти керування персонажами, що забезпечує гравцеві активний і цікавий досвід [22].

2.5.3 Динамічний алгоритм

Одним з важливих компонентів, який може посилити участь і ентузіазм гравців, є динамічна зміна ігрового процесу. Реалізація цього принципу в першу чергу залежить від реалізації алгоритму гри, зокрема щодо використання алгоритмів прийняття рішень.

Одним із прикладів використання алгоритмів прийняття рішень є зміна рівня складності гри в середині гри. Ці алгоритми враховують кілька факторів, у тому числі зміни зовнішніх факторів або дії гравців [23]. Таким чином, вони можуть ефективно змінювати механіку гри, щоб забезпечити більш ефективний рівень складності. Це включає в себе автоматичну зміну сили ворогів, доступність ресурсів або структуру гри.

Практика використання алгоритмів в іграх не тільки підвищує динамічність і гнучкість, але також підвищує вольове почуття гравця і впливає на хід гри. Це значно збільшує участь гравця в грі та підвищує емоційну цінність досвіду [30].

РОЗДІЛ 3

РОЗРОБКА ДОДАТКУ

Для реалізації задуманого веб-додатку було обрано мову програмування JavaScript, яка є однією з найпоширеніших мов для розробки веб-застосунків. Для створення графічного інтерфейсу використовувалася програма Figma, яка дозволяє створювати і прототипувати інтерфейси перед переходом до реалізації. У функціональній частині додатку були використані різні алгоритми, зокрема алгоритм для перебору в ширину та рекурсивний алгоритм.

3.1 Мова програмування

JavaScript, HTML і CSS є частиною сучасного процесу веб-розробки. Кожна з цих технологій відіграє значну роль у створенні веб-сайтів, які не тільки інформативні, але й мають динамічний та естетичний вигляд. Разом вони створюють потужне об’єднання, яке дозволяє розробникам створювати веб-сайти з різними можливостями, водночас надаючи їм естетичну привабливість для споживачів.

3.1.1 JavaScript

JavaScript часто називають абревіатурою JS, це динамічна об’єктно-орієнтована мова програмування на основі прототипу. Синтаксис реалізований за допомогою стандарту ECMAScript. Він часто використовується для створення веб-сторінок, які дозволяють користувачам взаємодіяти з браузером, взаємодіяти з користувачем, асинхронно обмінюватися інформацією з сервером і змінювати структуру та візуальний вигляд веб-сторінок.

Основними характеристиками JavaScript є:

1. Об’єктно-орієнтований підхід: JavaScript походить від концепції об’єктно-орієнтованого програмування (ООП), це дозволяє організувати код як об’єкти, які мають властивості та методи.

2. Динамічний тип: у JavaScript типи змінних можуть змінюватися під час виконання програми, що забезпечує гнучкість роботи з даними.
3. Події: JavaScript дозволяє реагувати на такі події, як клацання мишею або введення тексту, і викликати відповідні функції (обробники) у взаємодії з користувачем.
4. Асинхронний код: мова підтримує асинхронні операції, наприклад, завантаження даних із сервера без повторного завантаження сторінки, що робить веб-додатки ефективнішими та ефективнішими.
5. DOM: JavaScript здатний змінювати DOM, який є структурою сторінки HTML, елементи можна додавати, видаляти та змінювати.

JavaScript став найпопулярнішою мовою програмування [25] для створення динамічного вмісту в Інтернеті. Завдяки широкій сумісності браузерів у всьому спектрі розробники можуть покладатися на JavaScript для створення програм, які динамічно взаємодіють з клієнтом.

3.1.2 HTML

HTML — поширена мова, яка використовується для створення та представлення веб-сторінок. Його призначення полягає в тому, щоб упорядкувати та класифікувати такі елементи на веб-сайті, як заголовки, параграфи, таблиці, зображення, посилання тощо. HTML описує компоненти сторінки та способи їх взаємодії один з одним. Кожен елемент у HTML має унікальний тег, який вказує на початок і кінець його вмісту.

HTML дозволяє розміщувати різноманітний вміст на веб-сторінках, цей вміст допомагає створювати інформативні та зрозумілі документи для користувачів. Однією з головних переваг HTML є те, що він є основою для інших методів веб-розробки, таких як CS), який використовується для стилізації та відображення на сторінках, і JavaScript, який використовується для забезпечення динамічності та взаємодії.

3.1.3 CSS

CSS або каскадні таблиці стилів, — це мова таблиць стилів, яка використовується для створення та впорядкування документів HTML. Основна мета CSS полягає в диференціації представлення та вмісту веб-сторінки, що забезпечує більш ефективний і швидкий контроль над зовнішнім виглядом сторінки.

CSS дозволяє вказати зовнішній вигляд, розмір, відтінок і розташування компонентів на веб-сайті. За допомогою CSS ви можете визначити кілька варіантів дизайну шрифту, кольору фону, відступу, рамок, анімації та інших елементів дизайну. Ці стилі можуть бути реалізовані в HTML веб-сторінки або на зовнішніх ресурсах.

Каскадний розвиток CSS є одним із його найвидатніших атрибутів. Це означає, що стилі можна передавати від одного компонента до іншого [27], що дає змогу вичерпно та процедурно описати візуальні атрибути різних частин веб-сторінки.

3.2 Вибір алгоритмів

Вибираючи алгоритми для конкретної мети, важливо враховувати їх ефективність, складність і зручність для користувача в системі, для якої вони розроблені. Із низки доступних алгоритмів мною були обрані два алгоритми:

1. Пошук у ширину (BFS) — це стратегія обходу графа, яка відвідує всі суміжні вершини поточної вершини перед переходом до наступного рівня. Цей підхід особливо ефективний у вирішенні проблем, пов’язаних із пошуком найкоротших шляхів або ідентифікацією найближчих друзів.
2. Рекурсивний алгоритм — це метод вирішення проблеми, який передбачає рекурсивне вирішення менших частин проблеми. Цей підхід особливо корисний для вирішення проблем, які можна розділити на менші компоненти. Рекурсивний алгоритм ефективний і читабельний, він забезпечує стисле і просте програмування.

У разі знаходження найкоротшого шляху на сітці застосовується алгоритм «по ширині». Ефективність цього алгоритму у поставленій задачі дуже висока тому що його здатність систематично обходити граф, досліджуючи всі суміжні вершини поточної вершини перед переходом до наступного рівня, реалізується дуже вправно. Цей метод особливо корисний, коли пошук найкоротшого шляху між двома точками на сітці є надзвичайно важливим.

У поставленій задачі найефективнішим чином розміщення перешкод у випадковому порядку – це використання рекурсивного алгоритму. Із-за своєї особливості, алгоритму зручно ділити інтерактивну сітку на секції і тому реалізація буде дуже чіткою та не буде виникати помилок.

3.3 Графічний інтерфейс

Сьогодні графічний інтерфейс користувача (GUI) має велике значення в програмних і технологічних системах. Його головна мета — максимізувати взаємодію користувача з програмним забезпеченням, зробити його більш зрозумілим і практичним. Кілька критичних компонентів графічного інтерфейсу мають велике значення для користувача. Доступ до інформації можна отримати через графічний інтерфейс, який дозволяє користувачам візуалізувати інформацію за допомогою символів, графіків та інших компонентів графіки. Це спрощує складні для розуміння дані та покращує взаємодію користувача з системою чи програмою [28].

Коли інтерфейс програми більш зручний та інтуїтивно зрозумілий - це дозволяє користувачам легко працювати з програмою, навіть якщо вони мають обмежені технічні знання. Графічний інтерфейс полегшує навігацію. Інтерфейс призначений для зручності користувача [29] та містить численні елементи керування, такі як меню, кнопки та вкладки, які полегшують швидкий перехід між різними частинами програми чи системи.

Розробники можуть віддати перевагу інтуїтивно зрозумілому дизайну інтерфейсу. Завдяки цьому вони можуть враховувати фізіологічні та психологічні

властивості користувачів, щоб створити зручний інтерфейс. Це важливо для забезпечення позитивної взаємодії з користувачем.

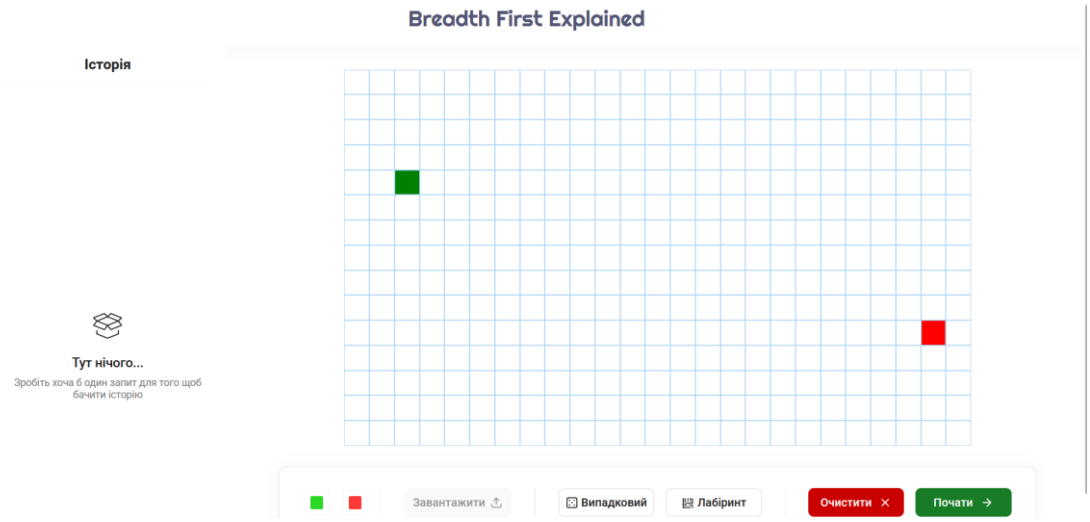

Рис. 3.1. Графічний інтерфейс додатку

3.3.1 Історія

Включення історії запитів в інтерфейс є важливим компонентом, який може значно підвищити ефективність програм, веб-сайтів і систем. Функція історії запитів сприяє ефективному доступу до попередніх запитів. У даній реалізації історія відіграє місце де зберігаються результати запитів.

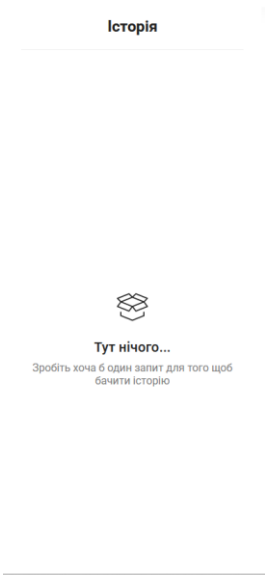

Рис. 3.2. Вид історії при відсутності запитів

Після завершення хоча б одної сесії, до історії додається картка, на якій зображено тип генерації, успішна вона чи ні.

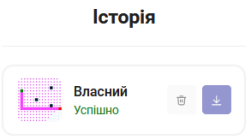

Рис 3.3. Вид історії при наявності одного запиту

Після того як картка була збережена до історії, її можна переглянути у самій історії або при натисканні, відкриєтеся модальне вікно, де можна переглянути зображення у збільшеному розмірі. Додатково користувач може видалити чи завантажити результат на будь якому з цих кроків.

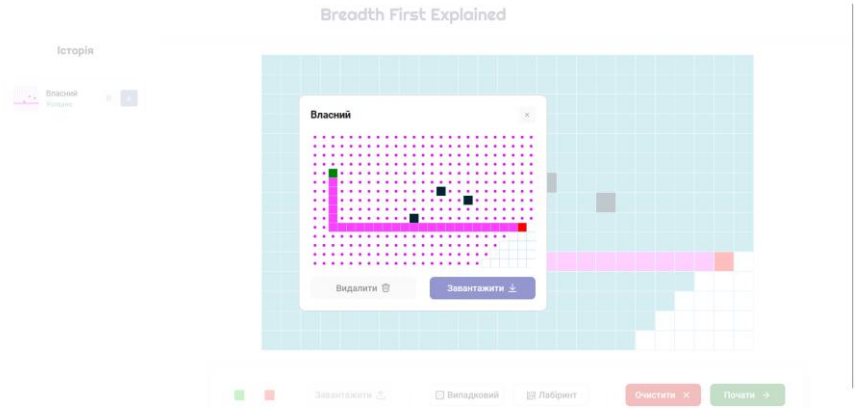

Рис 3.4. Вид модального вікна результату

3.3.2 Навігація

Навігація є невід'ємною частиною любого додатку, сайту чи програми. Вона дуже сильно спрощують користування продуктом і до цього своєю зручність та зрозумілістю спонукають користувача залишитись у продукті на більш великий час. Із-за зручного інтерфейсу та навігації користувач захоче повернутися та користуватись продуктом. Першочергово важливо враховувати зручність використання. З точки зору доступності, навігація має бути простою для пошуку та використання. Компоненти, які складають навігацію, такі як кнопки, меню та підменю, мають бути виділеними та легко доступними.

Щоб зробити систему більш зручною для користувача, важливо переконатися, що користувач може досягти своїх цілей за найменшу можливу кількість кроків. Це можна досягти шляхом видалення зайвих дій, що призведе до оптимізації процесу та зменшення складності системи.

Говорячи про поняття інтуїції, немає точного, загальноприйнятого визначення. Однак загальновизнано, що інтуїція — це легкість розуміння та використання певної системи чи процесу. Це може бути будь-що: від комп'ютеризованого процесу до фізичного представлення. Чим більш інтуїтивно зрозумілою є система, тим менше для користувача є її розуміння та навігація.

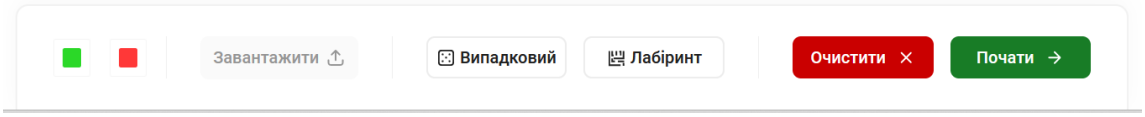

Рис.3.5. Зовнішній вигляд навігації

За місце розташування було обрано нижню частину користувацького інтерфейсу, оскільки в центрі додатку знаходиться інтерактивне поле, то користуватися навігацією легше коли вона прибита до низу, симулюючи інтерфейс мобільних додатків.

На рисунку 3.5 можна побачити інтуїтивно зрозумілий набір функцій яку несе навігація. Починаючи с вибору точок, закінчуючи кнопками які взаємодіють з полем.

3.4 Функціонал

Забезпечення позитивної взаємодії з веб-програмою в першу чергу залежить від її функціональності та чіткості. Наявність функцій має бути очевидною для користувача, без необхідності вникати в складні інтерфейси. Взаємодія в програмі має бути зрозумілою та послідовною, щоб користувачі могли легко зрозуміти, як взаємодіяти з різними функціями та досягати своїх цілей.

3.4.1 Інтерактивне поле

Поле являє собою набір клітинок які зв’язані між собою. Якщо його порівнювати з графами, то це буде поле у якому кожна частинка сполучена з іншою.

Рис. 3.6. Зовнішній вигляд поля

Це поле для більшого розуміння можна зобразити у вигляді графів. І у цьому моменті буде зрозуміло, як у полі працює алгоритм перебору в ширину.

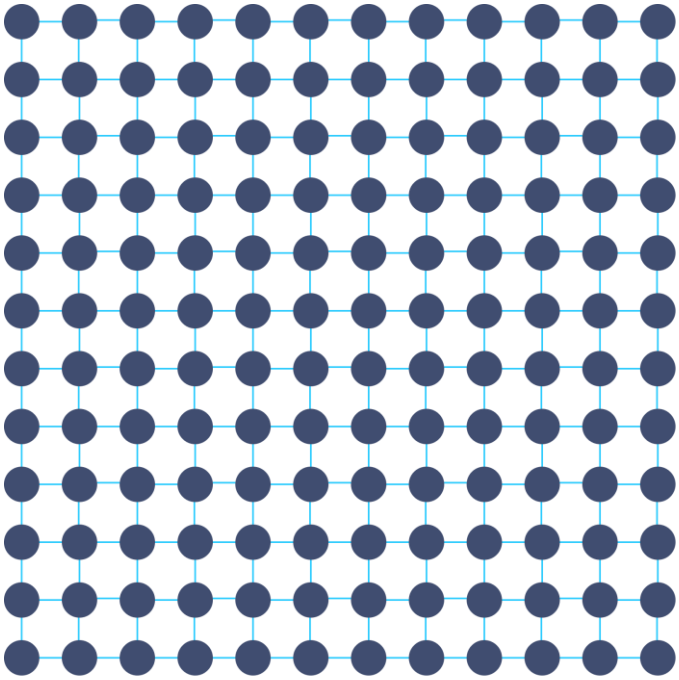

Рис 3.7. Вид поля у вигляді графів

Користувач взаємодіяючи з полем може ставити у клітинки чорні блоки – вимикаючи з поля клітинку. Повторне натискання на «заблоковану» клітинку претворює її на початковий вигляд.

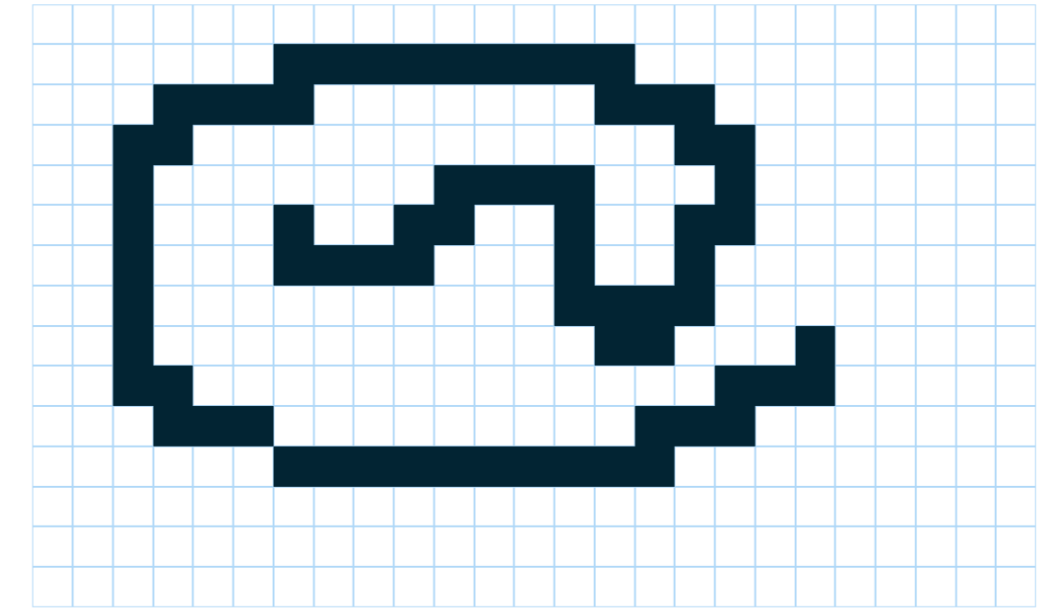

Рис. 3.8. Вигляд поля разом с заблокованими клітинками

Таким чином на графовій структурі даний елемент буде виглядати подібним чином:

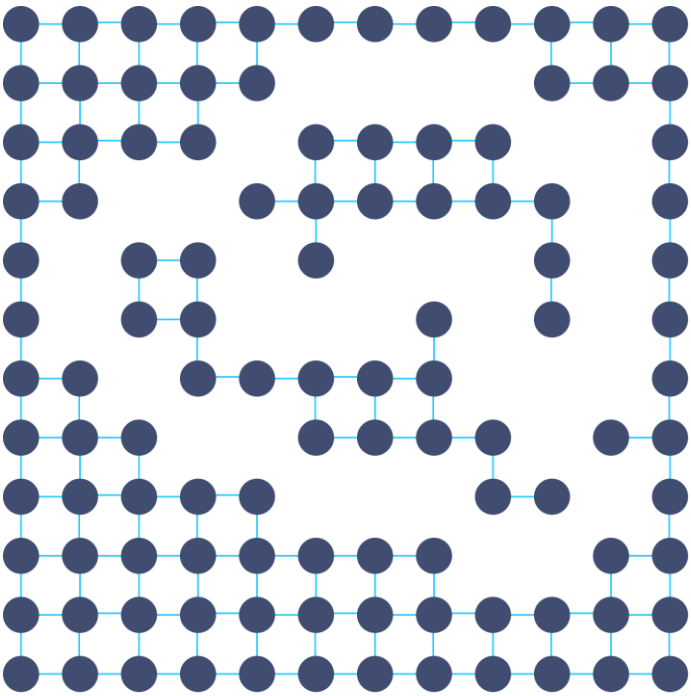

Рис. 3.9. Графова структура с заблокованими частинами

3.4.2 Навігація та взаємодія

Нижня панель навігації містить всі можливі функції які напряму пов’язані з полем. Навігація (Рисунок 3.5) в першу чергу дозволяє розмістити початкову точку – з якої буде починатися пошук найкоротшого шляху. При першому запуску точки розміщаюся у відведених зонах у випадковому місці, це зроблено для мінімізуваня випадків точки будуть максимально наближені один до одного. Користувач може натиснути на кнопку с зеленим (початковим) квадратом та розмістити його де завгодно по своєму бажанню.

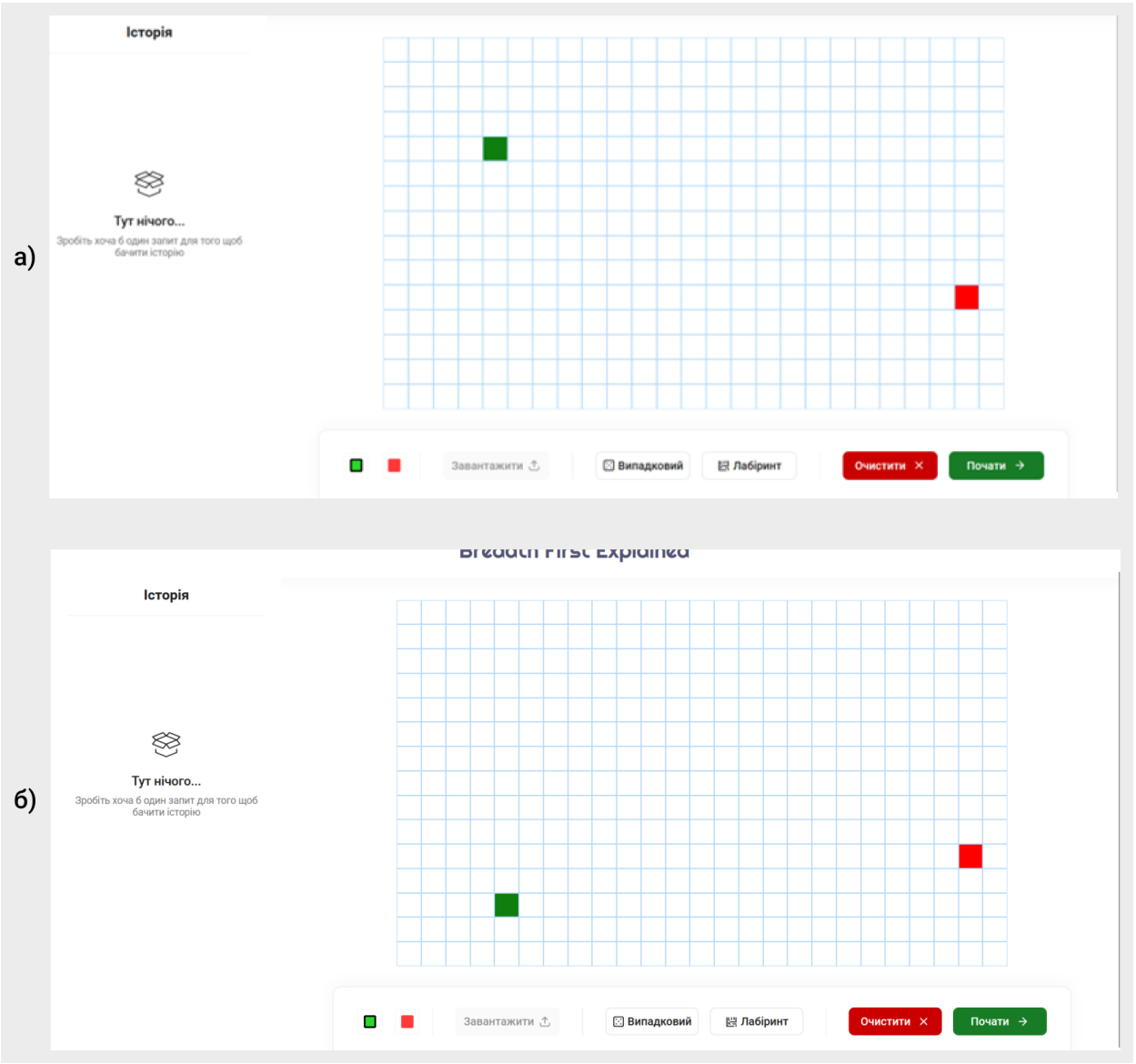

Рис. 3.10. Вид поля з переміщенням точки у сітці

Як показано на рисунку (3.10), включивши функцію можна перемістити точку на будь яку вільну клітинку. Наступним кроком по навігації іде розташування другої точки яка відіграє кінцеву точку куди повинен бути побудований шлях.

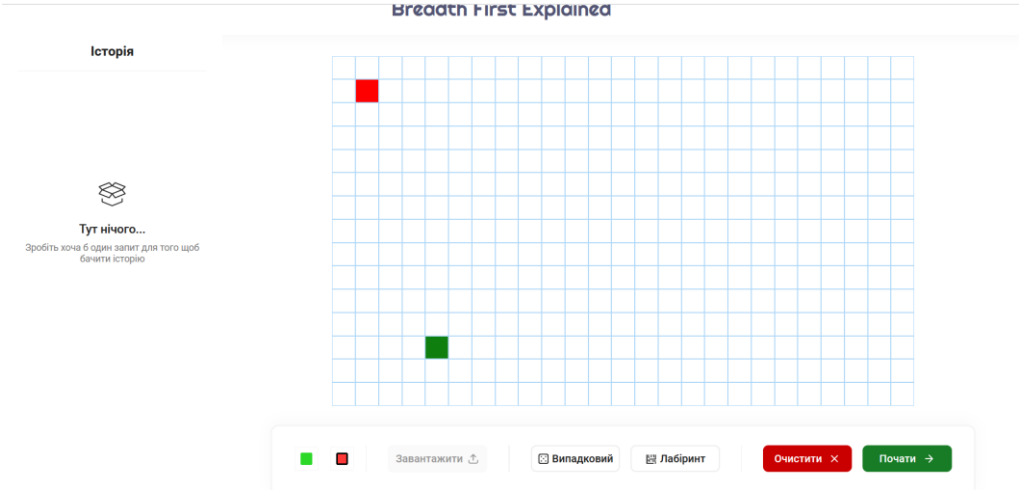

Рис. 3.11. Вид поля після переміщенні кінцевої точки

Далі після розташування точок, зустрічає набір опціональних функцій. Першою розташована кнопка «завантажити». Вона дозволяє завантажити будь яке зображення, після цього воно стане «підкладкою»

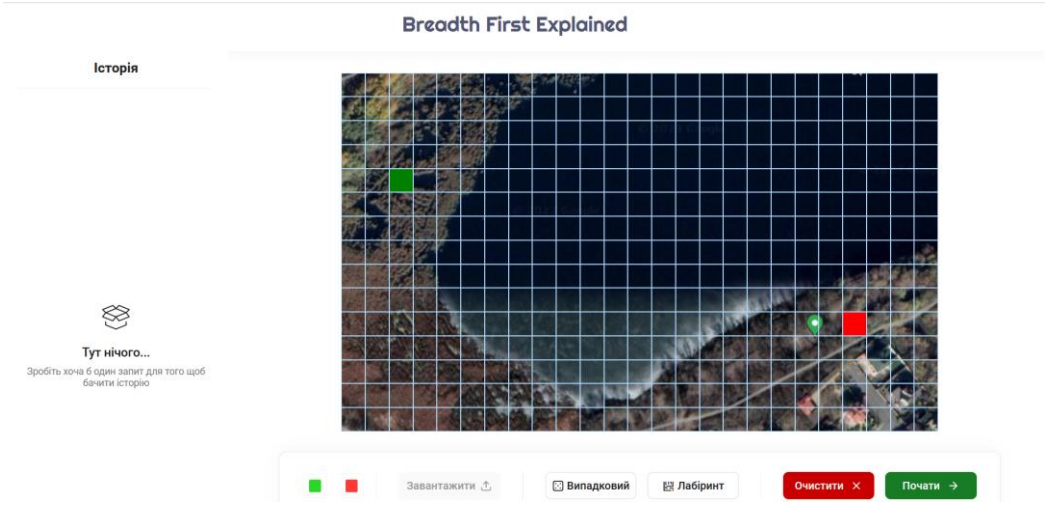

Рис. 3.12. Вид поля після завантаження зображення

Після цієї процедури функціональність поля залишається незмінною. І при цьому дає великий аспект можливостей та реалізацій на цьому кроці.

Рис. 3.13. Вид поля після нанесення перешкод

Якщо завантажене зображення є недоречним або його потрібно замінити то потрібно звернутися до функції очистити. Після взаємодії с кнопкою «очистити» положення початкової та кінцевої точки не зміняться, а зображення буде видалено. Для повторного завантаження потрібно звернутися до функції «завантажити».

Наступною функцією є створення випадкового паттерну, який генерується за допомогою рекурсивного алгоритму. Цей інноваційний метод створення візерунків дозволяє користувачеві створювати унікальні цікаві структури. Рекурсивність алгоритму полегшує створення складних і змінних візерунків, що веде до безлічі творчих можливостей . Це функціональне доповнення розширює спектр можливостей користувача, що дозволяє створювати унікальні візуальні компоненти з ефективними експериментами.

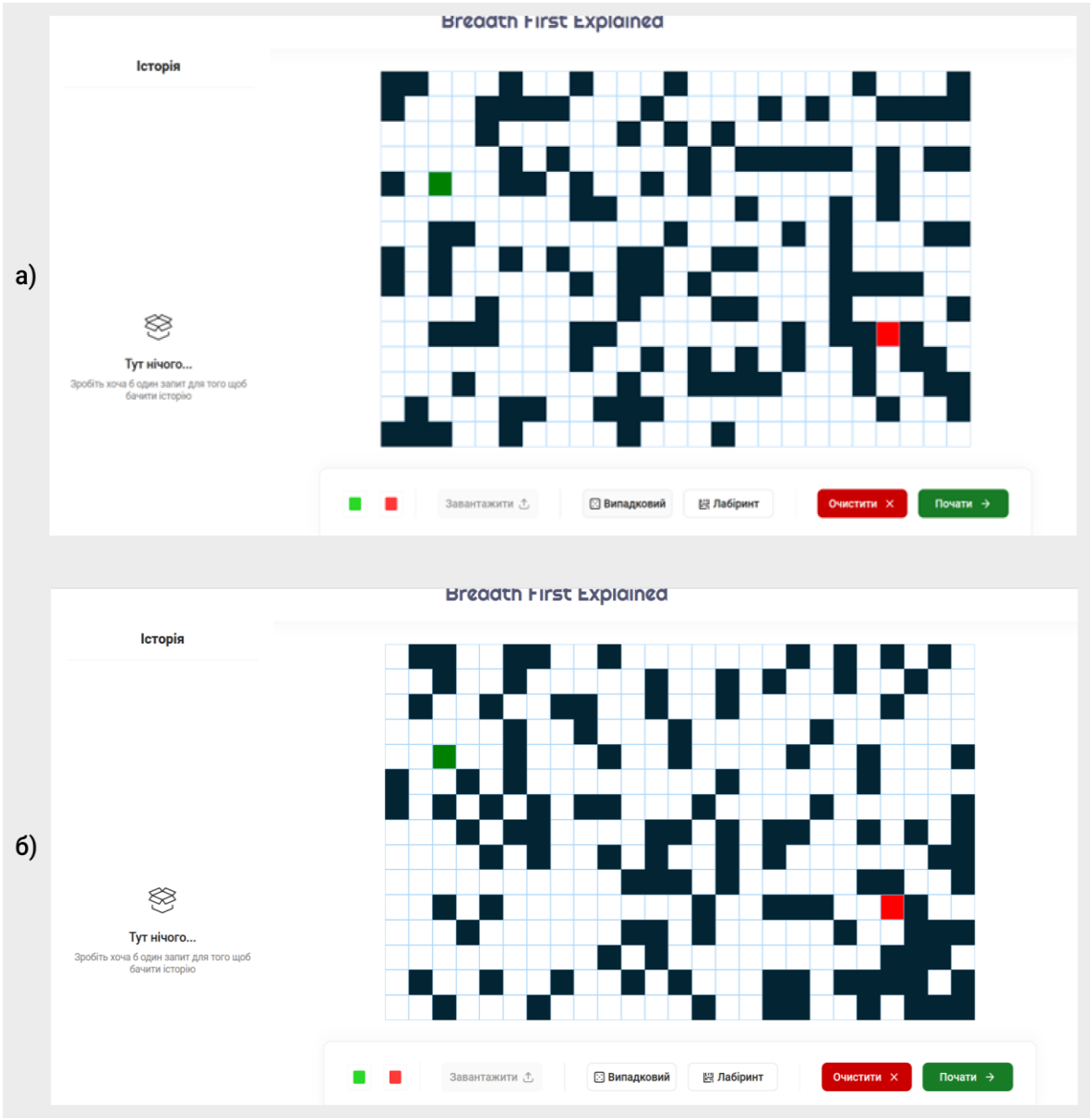

Рис. 3.13. Вид двох генерацій випадкових перешкод на рисунках а) і б)

Далі на блоці навігації знаходиться функція «Лабіринт», яка дозволяє створювати випадкові лабіринти, будуючи вертикальні та горизонтальні лінії з отворами. Цей захоплюючий атрибут використовується для створення лабіринтів, які мають унікальні структури на основі випадкових параметрів.

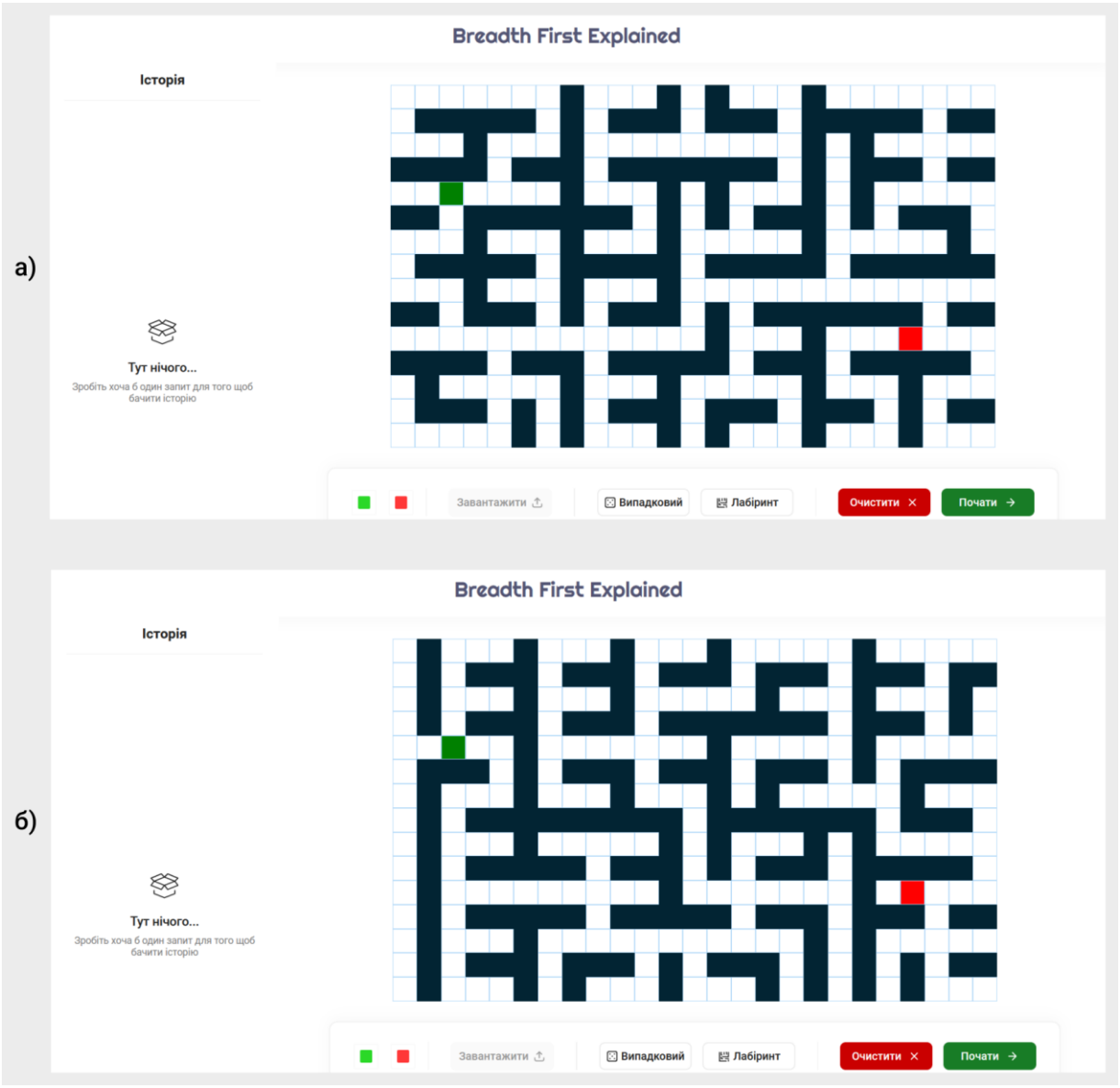

Рис. 3.14. Вид двох генерацій випадкових лабіринтів на рисунках а) і б)

Наступні кнопки для роботи – це «Очистити» та «Почати». Функція «Очистити» призначена для повного видалення всіх блоків на полі, додатково, як уже було сказано, вона має можливість видалення завантаженого зображення. Ця опція полегшує відновлення поля до початкового стану.

І навпаки, кнопка «Пуск» відповідає за ініціювання алгоритму, який відповідає обробці або відображенню інформації на полі. Ця функція ініціює вбудовані процеси та виконує обчислення, графічні та інші завдання. Кнопка «Пуск» забезпечує ефективний зв’язок із внутрішнім функціоналом і методами запуску завдань з максимальною легкістю для користувача.

3.4.3 Історія запитів

Історія відіграє другорядну, але значущу роль. Усі запити, зроблені користувачем, автоматично з’являються зліва у меню, включаючи зображення поля під час початку виконання алгоритму та тип запиту: лабіринт, випадковий або власний. Крім того, історія містить інформацію про успіх або невдачу певного запиту. Також якщо потрібно, запит можна подивитися поближче, відкривши його у модальному вікні, при натисканні на запит в історії. Для більш комфортного використання можна завантажити чи видалити запит з історії скориставшись відповідними кнопками.

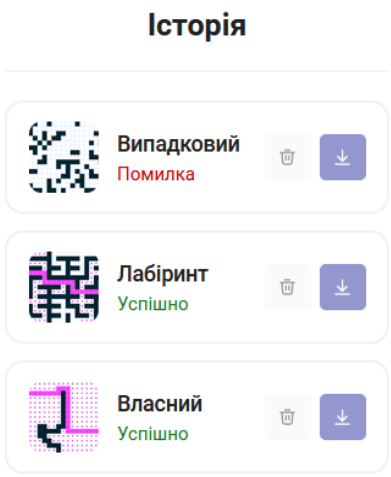

Рис. 3.15. Вид історії з успішним, помилковим та всіма видами запитів

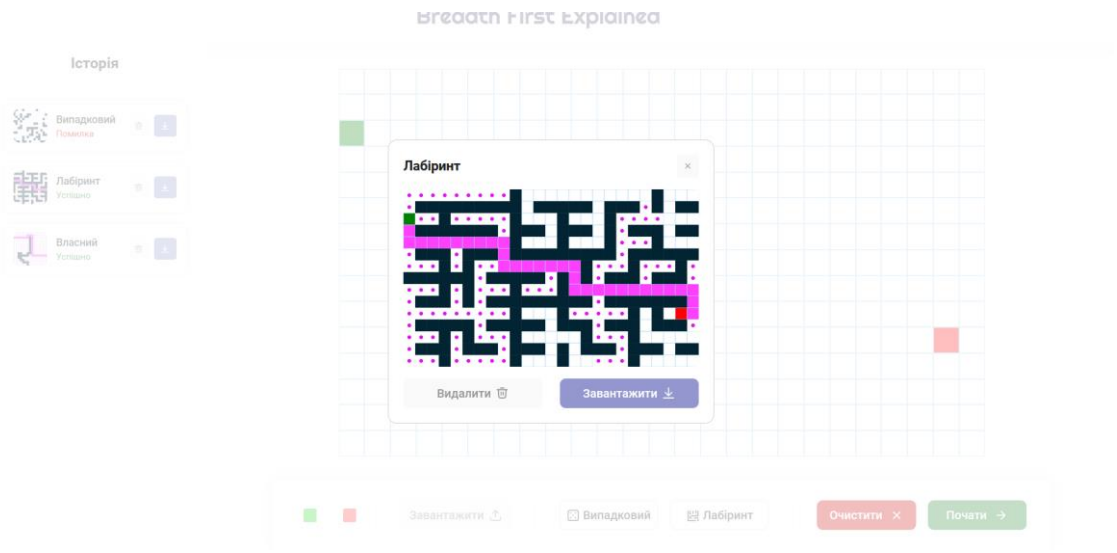

Рис. 3.16. Вид модального вікна одного с запитів з історії

Ця функціональність історії надає користувачеві документацію про його попередні дії та рішення, а також засіб для відстеження результатів і взаємодії з програмою. Одним з унікальних аспектів є здатність швидко розпізнавати та аналізувати помилки або проблеми під час процесу виконання запиту. Цей аспект програмного забезпечення сприяє покращенню взаємодії з користувачем та ефективності по відношенню до програмного середовища.

3.4.4 Тостові повідомлення

Тостові повідомлення є невід'ємною частиною інтерфейсу, який спілкується з користувачем, ці повідомлення надають інформацію про завершення операцій або статус програми. Ці спливаючі сповіщення відіграють важливу роль у сповіщенні користувача про важливі події чи результати.

У додатку реалізовані два типи таких повідомлень: «Успіх» та «Помилка». Перший у свою чергу сигналізує про успішне знаходження найкоротшого шляху, а другий – навпаки сигналізує про те що шлях не знайдений.

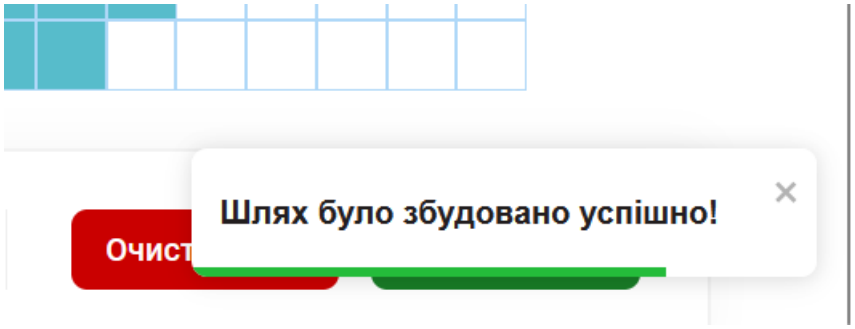

Рис. 3.15. Вид успішного повідомлення

Частіше за всього «Помилка» сигналізує про відсутність будь якого шляху до кінцевої точки.

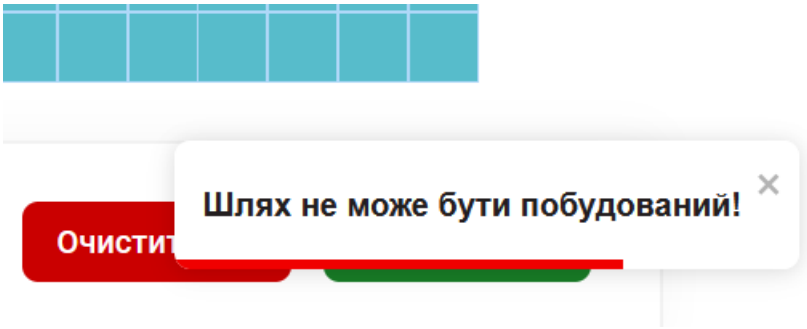

Рис. 3.16. Вид помилкового повідомлення

3.5 Процес пошуку

Після того як всі точки розташовані по місцях, можна почати пошук найкоротшого шляху. Натискаючи на кнопку «почати» починає працювати алгоритм, він перебирає клітинки, навколо себе, роблячи аналіз. Після того як алгоритм дійде до кінцевої точки – помічає клітинки які не входять до короткого шляху як негативні, а ті які входять до короткого шляху – позитивні. Перед візуалізуванням алгоритм засилаєтеся на варіативність клітинки і натомість вона зміню свій колір. Після цього запит у форматі картинки зберігається і відправляється до історії, де користувач вже може провести потрібні дії над нею.

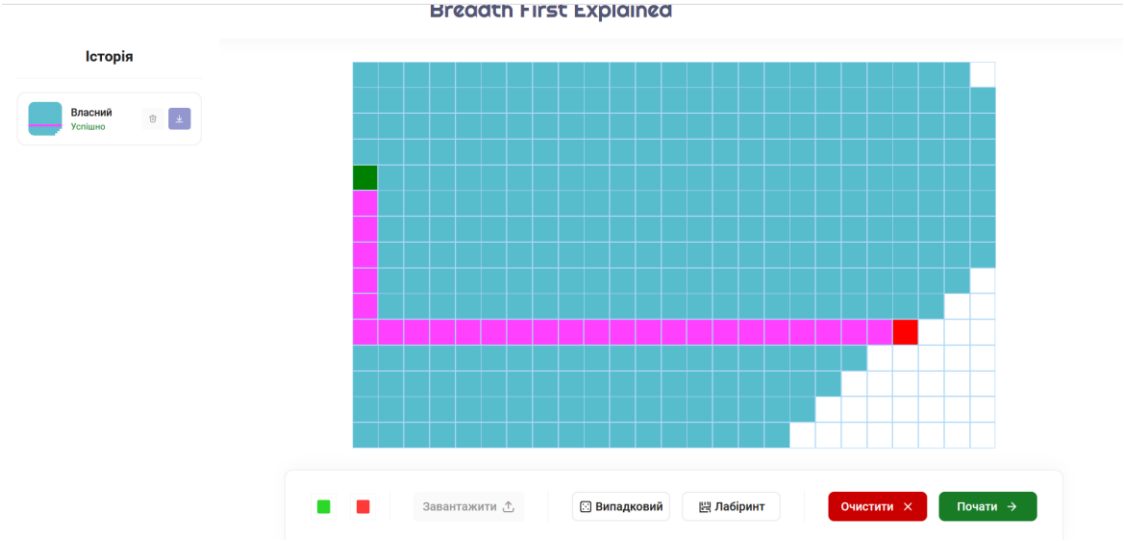

Рис. 3.17 Зображення роботи алгоритму та інтерфейсу

3.5.1 Розміри поля

Для забезпечення максимально комфортного користування додатком було обране конкретне поле був проведений аналіз з різними видами полів, був виконаний замір часу і навантаження, яке не тільки зменшує обсяг програми, але й покращує загальну взаємодію з користувачем, роблячи її плавною та приємною. Використовуючи це рішення, користувачі можуть насолоджуватися функціональністю програми без потреби мати справу з напругою чи затримкою, пов’язаною з цим, це також призведе до ефективної та швидкої взаємодії з інтерфейсом.

Основна область взаємодії з програмою має значний вплив на позитивне враження від додатку. Такий вибір призначення не тільки економить ресурси, але й підвищує ефективність програми, забезпечуючи приємну та ефективну роботу користувача.

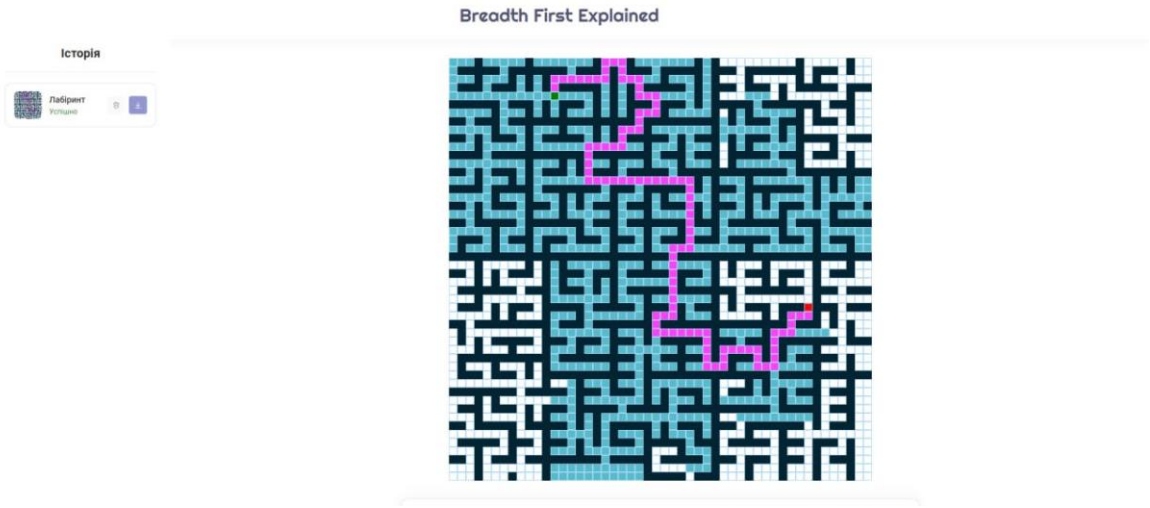

Рис. 3.18. Вид найбільшого поля

Дане поле дуже гарне для варіативності функцій які відображенні в навігації, оскільки має велику площу і відповідно генерація перешкод буде більш цікавою.

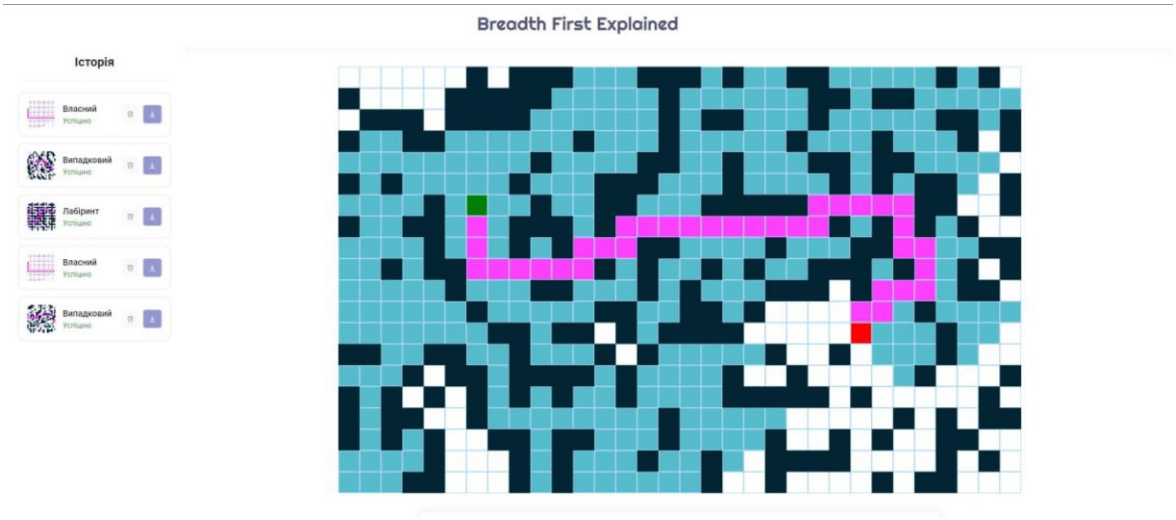

Рис. 3.19. Поле меншою площею

Дане поле є більш зручнішим для використання, але дуже не гнучке в використанні бо все ще потребує більше взаємодії від користувача.

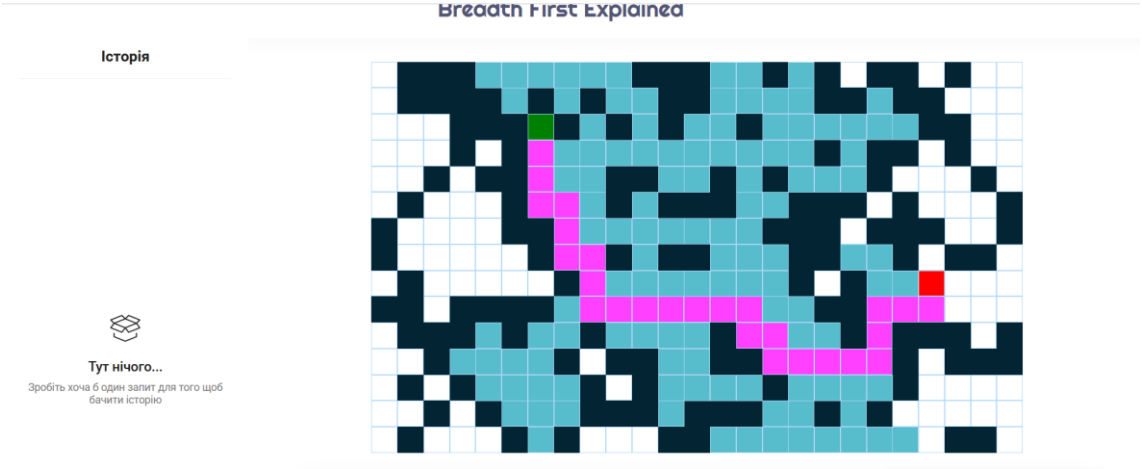

Рис. 3.20. Поле яке було обране для постійного використання

Дане поле буле обране як головне, оскільки воно може реалізувати потреби користувача – швидка взаемодія. На цьому полі дуже зручний маштаб, завдяки цьому курстувач може швидко розставити блоки і при цьому функції, які пропонує навігація, також справно працюють для демострації та отримання важливого досвіду.

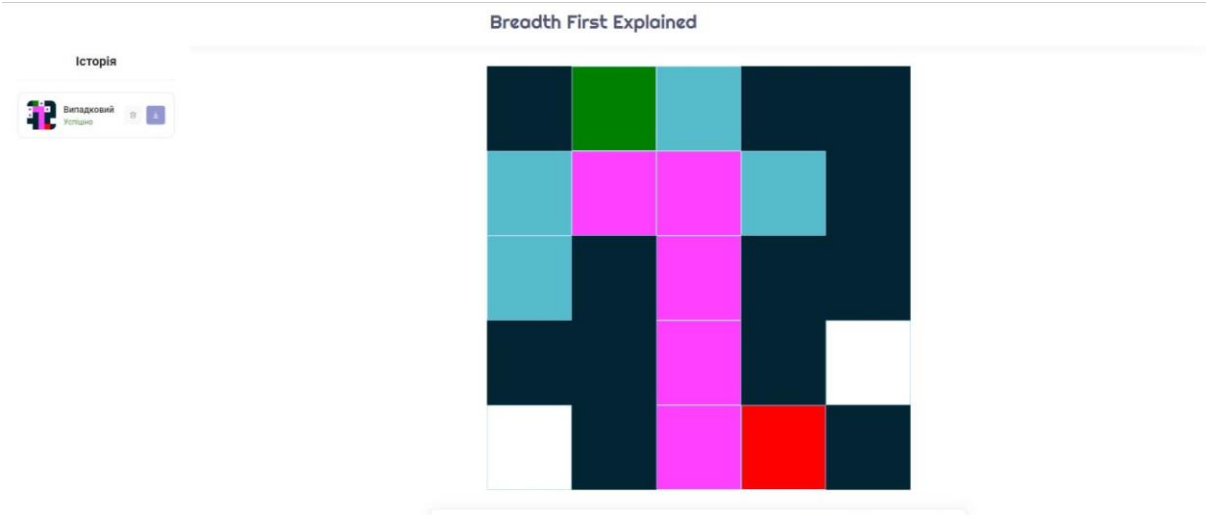

Рис. 3.21. Найменше поле на якому можна зробити пошук шляху

На найменшому полі неякісно працюють функції оскільки вибірка клітинок дуже мала. При цьому воно не дає важливого користувацького досвіду. На основі цих полів було обрано найзручніше.

3.5.2 Аналіз даних

Також окрім функціональних було проведено аналіз навантаження, швидкодії та затраченого часу. Оскільки алгоритм досить оптимізований та використовується невеличке поле, час, який буде затрачено на аналіз та побудову шляху в більшості випадків буде біля однієї мілісекунди. Тож для аналізу були обрані середні розміри полів, які не є надто великими або замалими.

Знаючи що алгоритм аналізує клітинки до кінцевої точки, він обходить не все поле, щоб не робити зайвих дій та не зменшувати швидкодію. Їх цього можна зробити висновок що можна зібрати інформацію пройдених клітинок по відношенню до проаналізованих. В рамках функціоналу було обрано поле розміром 15 x 25 та 25 x 32 клітинок для подальшого аналізу. Дві ситуації були проаналізовані: у першому випадку точки випадково розміщені у своїх діапазонах, а в другому випадку, крім випадкового розташування точок у своїх діапазонах, також був побудований лабіринт. Зібрані данні були переведені у графіки.

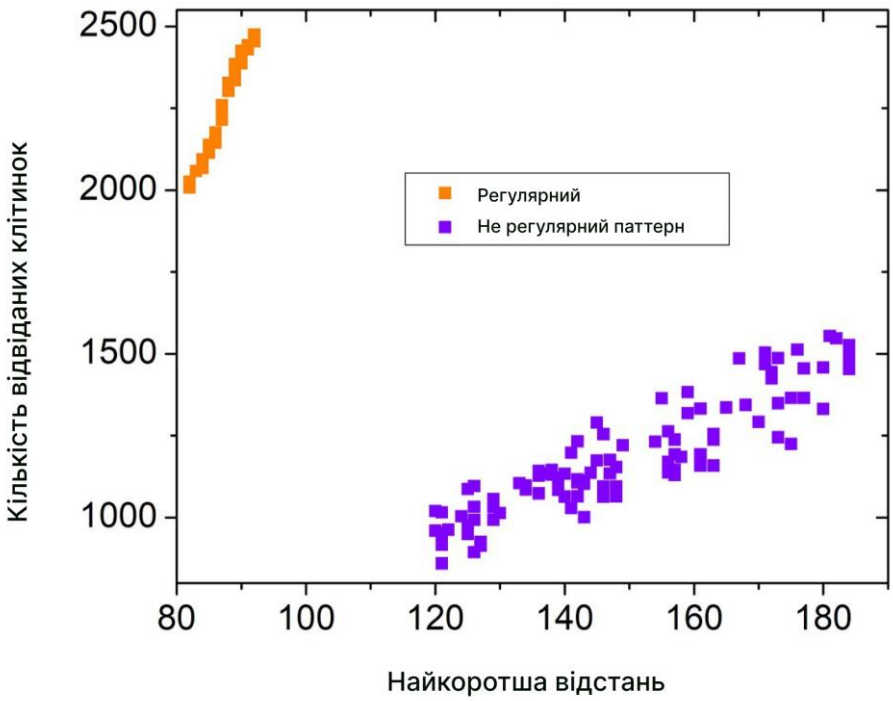

Рис 3.22. Графік відношення регулярного графа та не регулярного графа

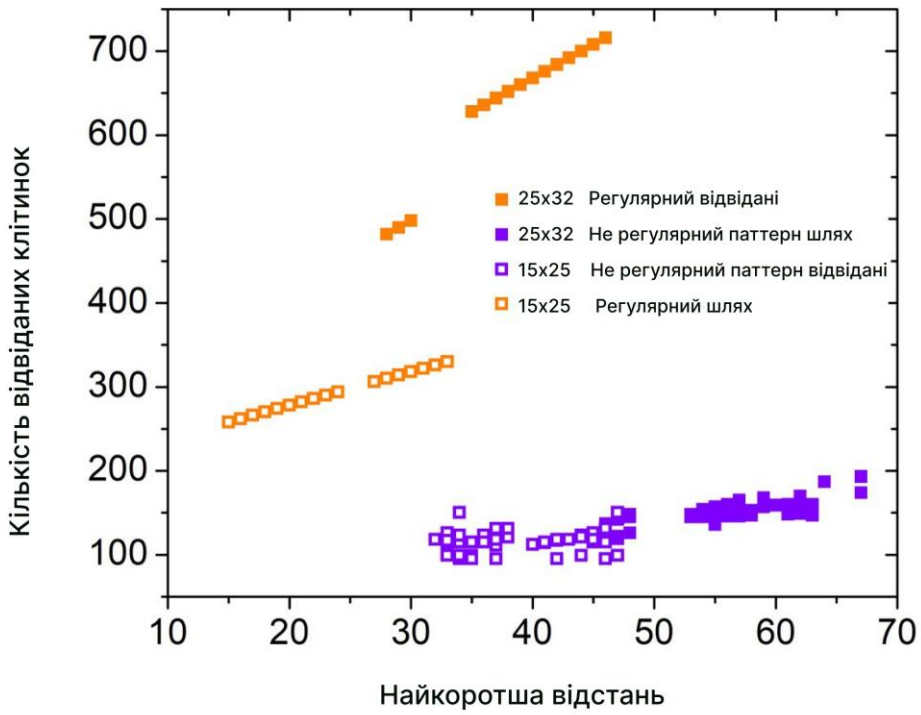

Рис 3.23. Графік відношення регулярного графа та не регулярного графа різних величин

Велика різниця результатів пов’язана з тим що при побудові лабіринтів, частина клітинок перетворюється на перешкоди, а також ці перешкоди іноді відрізають частину поля, тому алгоритм їх не рахує. Додатково до цього шлях у регулярному графі більш коротша із-за відсутності перешкод

РОЗДІЛ 4

ПРАВИЛА ТЕХНІКИ БЕЗПЕКИ ПРИ РОБОТІ ЗА КОМП’ЮТЕРОМ

Користуючись комп’ютером, важливо дотримуватися кількох важливих правил. Важливо забезпечити мінімум 5 см додаткового простору між стільцем і столом. Крім того, життєво важливо підтримувати правильну поставу тіла, з прямою спиною та розпущеними плечами. Ноги повинні мати рівну поверхню або розташовуватися на визначеній підставці під кутом 90 градусів, клавіатура має бути розташована на оптимальній відстані та під кутом. Крім того, важливо стежити за відстанню між очима та екраном, регулярно виконувати вправи для очей і не покладатися на телевізор як на заміну монітора. Крім того, під час роботи за комп’ютером рекомендується кожні дві години робити перерви, текст на екрані має бути легко читаним, темного кольору на світлому фоні.

До важливих правил безпеки відносяться підтримка чистоти і вентиляції приміщення, контроль за екраном і окулярами, уникнення контакту з лініями електропередач і негайне вимкнення ПК у разі виникнення надзвичайної ситуації. Освітлення кімнати повинно бути достатнім для виконання візуального завдання, забезпечуючи відповідне освітлення та уникаючи відблисків монітора.

Крім того, важливо враховувати, що використовувати телевізор замість монітора недоцільно через високе випромінювання. Робота за комп'ютером вимагає дотримання як фізичних, так і гігієнічних правил. Місце навколо комп’ютера слід регулярно прибирати, а в кімнаті має бути вентиляційна система, яка вмикається щогодини, щоб створити оптимальні умови для роботи.

У контексті безпеки ПК також слід звернути увагу на стан екрану, який має бути без пилу, плям і бути гігієнічним. Постійна перевірка та чищення окулярів, незалежно від того, чи це окуляри для роботи з комп’ютером, чи звичайні окуляри, також є важливими для збереження зору.

Важливо пам'ятати, що штучне освітлення в кімнаті повинно бути достатнім і уникати тіней і відблисків. Люмінесцентні лампи є найефективнішим видом лампочок. Постійне дотримання цих простих правил допоможе

забезпечити безпеку і комфорт під час використання комп'ютера. Важливо пам'ятати, що штучне освітлення в кімнаті повинно бути достатнім і уникати тіней і відблисків. Люмінесцентні лампи є найефективнішим видом лампочок. Постійне дотримання цих простих правил допоможе забезпечити безпеку і комфорт під час використання комп'ютера.

ВИСНОВОК

Завдяки науковому дослідженню було ретельно оцінено значення штучного інтелекту в розвитку суспільства. Крім того, це дослідження передбачало проведення перевірки різних алгоритмів для визначення найбільш ефективних для навігації найкоротшими маршрутами на двовимірній місцевості. Під час цього аналізу були враховані різні алгоритми, починаючи від фундаментальних і закінчуючи складнішими варіантами.

Окрім технічних компонентів, велика частина уваги була приділена оцінці та підвищенню практичності веб-додатку. Під час перегляду інтерфейсу пріоритет був на перспективі користувача, з наголосом на тому, щоб зробити його гладким та інтуїтивно зрозумілим. Були внесені необхідні вдосконалення для забезпечення позитивної взаємодії з користувачем. Розслідування включає в себе кілька компонентів, включаючи технічні рішення, аналітичні висновки та вдосконалення інтерфейсу користувача, усі з яких включено у функціональну веб-програму, призначену для ефективного вирішення проблем, пов’язаних із визначенням найкоротших шляхів.

СПИСОК ЛІТЕРАТУРИ

1. Artificial intelligence A modern approach : підручник. 3-тє вид. New Jersey : Pearson Education, 2010. 1-45.
2. Deep learning. MIT Press, 2017. 800 с.
3. Teja T. R. Autonomous robot motion path planning using shortest path planning algorithms. *IOSR Journal of Engineering*. 2013. Т. 3, № 01. С. 65–69. URL: https://doi.org/10.9790/3021-03116569
4. Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning / B. Wang та ін. *IEEE Robotics and Automation Letters*. 2020. Т. 5, № 4. С. 6932–6939. URL: https://doi.org/10.1109/lra.2020.3026638
5. Millington I., Funge J. AI-Based game genres. *Artificial intelligence for games*. 2018. С. 831–840. URL: https://doi.org/10.1201/9781315375229-13
6. Юскович-Жуковська В. І. Варіативне електронне середовище з вивчення навчальної дисципліни "Алгоритми та структури даних". *Психолого-педагогічні основи гуманізації навчально-виховного процесу в школі та ВНЗ*. 2019. Вип. 1 (21). С. 156–166.
7. Анастасія Харченко. Тема: що таке алгоритм?, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=k5l3lGzyTBM
8. Comparative research of random search algorithms and evolutionary algorithms for the optimal control problem of the mobile robot / S. V. Konstantinov та ін. *Procedia computer science*. 2019. Т. 150. С. 462–470. URL: https://doi.org/10.1016/j.procs.2019.02.080
9. Martelli A. On the complexity of admissible search algorithms. *Artificial intelligence*. 1977. Т. 8, № 1. С. 1–13. URL: https://doi.org/10.1016/0004-3702(77)90002-9
10. Davila C. E. Line search algorithms for adaptive filtering. *IEEE transactions on signal processing*. 1993. Т. 41, № 7. С. 2490–2494. URL: https://doi.org/10.1109/78.224257

11. Karane M., Panteleev A. Hybrid multi-agent optimization method of interpolation search. *Computational mechanics and modern applied software systems (cmmass'2019)*, м. 2019. URL: https://doi.org/10.1063/1.5135688
12. Perl Y., Itai A., Avni H. Interpolation search–a log log N search. *Communications of the ACM*. 1978. Т. 21, № 7. С. 550–553. URL: https://doi.org/10.1145/359545.359557
13. Dakin R. J. A tree-search algorithm for mixed integer programming problems. *The computer journal*. 1965. Т. 8, № 3. С. 250–255. URL: https://doi.org/10.1093/comjnl/8.3.250
14. Traversing large graphs on GPUs with unified memory / P. Gera та ін. *Proceedings of the VLDB Endowment*. 2020. Т. 13, № 7. С. 1119–1133. URL: https://doi.org/10.14778/3384345.3384358
15. Photphanloet C., Lipikorn R. PM10 concentration forecast using modified depth-first search and supervised learning neural network. *Science of the total environment*. 2020. Т. 727. С. 138507. URL: https://doi.org/10.1016/j.scitotenv.2020.138507
16. Mayorova A., Bratchenko I. Foreword to the Special Issue on Laser and optical technologies in biomedicine and ecology. *Laser and optical technologies in biomedicine and ecology*. 2019. Т. 5, № 2. URL: https://doi.org/10.18287/020101
17. Zhou R., Hansen E. A. Breadth-first heuristic search. *Artificial Intelligence*. 2006. Т. 170, № 4-5. С. 385–408. URL: https://doi.org/10.1016/j.artint.2005.12.002).
18. Ahammad T., Hasan M., Zahid Hassan M. A New Topological Sorting Algorithm with Reduced Time Complexity. *Advances in Intelligent Systems and Computing*. Cham, 2021. С. 418–429. URL: https://doi.org/10.1007/978-3-030-68154-8_38
19. Сравнительный анализ алгоритмов Беллмана-Форда и алгоритма Дейкстры : thesis / О. В. Бережная та ін. 2010. URL: http://essuir.sumdu.edu.ua/handle/123456789/4008

20. A General Tensor Prediction Framework Based on Graph Neural Networks / Y. Zhong та ін. *The Journal of Physical Chemistry Letters*. 2023. С. 6339–6348. URL: https://doi.org/10.1021/acs.jpclett.3c01200
21. Carmona R., Laurière M. Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games I: The Ergodic Case. *SIAM Journal on Numerical Analysis*. 2021. Т. 59, № 3. С. 1455–1485. URL: https://doi.org/10.1137/19m1274377
22. Rafiq A., Asmawaty Abdul Kadir T., Normaziah Ihsan S. Pathfinding algorithms in game development. *IOP conference series: materials science and engineering*. 2020. Т. 769. С. 012021. URL: https://doi.org/10.1088/1757-899x/769/1/012021
23. Papadimitriou C. Algorithms, games, and the internet. *The thirty-third annual ACM symposium*, м. Hersonissos, Greece. New York, New York, USA, 2001. URL: https://doi.org/10.1145/380752.380883
24. Hunicke R. The case for dynamic difficulty adjustment in games. *The 2005 ACM SIGCHI international conference*, м. Valencia, Spain, 15–17 черв. 2005 р. New York, New York, USA, 2005. URL: https://doi.org/10.1145/1178477.1178573
25. An analysis of the dynamic behavior of JavaScript programs / G. Richards та ін. *The 2010 ACM SIGPLAN conference*, м. Toronto, Ontario, Canada, 5–10 черв. 2010 р. New York, New York, USA, 2010. URL: https://doi.org/10.1145/1806596.1806598
26. Levering R., Cutler M. The portrait of a common HTML web page. *The 2006 ACM symposium*, м. Amsterdam, The Netherlands, 10–13 жовт. 2006 р. New York, New York, USA, 2006. URL: https://doi.org/10.1145/1166160.1166213
27. Schultz D., Cook C. Beginning HTML with CSS and XHTML: Modern Guide and Reference. Springer London, Limited, 2007.
28. Graphical user interface (GUI) testing: systematic mapping and repository / I. Banerjee та ін. *Information and software technology*.

2013. T. 55, № 10. C. 1679–1694. URL: https://doi.org/10.1016/j.infsof.2013.03.004

29. The evaluation of multiplex PCR and DNA profiling methods (DGGE and SSCP) for the detection of mycotoxigenic Fusarium species / by Karen Jordaan : thesis. URL: http://hdl.handle.net/10394/2117 (дата звернення: 12.12.2023).

30. Kientza H Graph K-means Based on Leader Identification, Dynamic Game, and Opinion Dynamics. *Notes académiques de l'académie d'agriculture de france / academic notes of the french academy of agriculture*. 2017. Vol. 4. P. 1–14. URL: https://doi.org/10.58630/pubac.not.a372706