З. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Розробка загальної структури програмного забезпечення

Виходячи із поставленої задачі, програма має мати такі логічні модулі:

1. Модуль візуалізації
2. Модуль інтерфейсу
3. Модуль створення карти
4. Модуль валідації даних
5. Модуль введення розмірів карти
6. Модуль обробки прискорювачей
7. Модуль обробки сповільнювачей
8. Модуль поширення хвилі
9. Модуль повернення хвилі
10. Модуль загрузки JSON – файлу
11. Модуль формування JSON – файлу
12. Модуль збереження JSON – файлу

Після того як було визначено логічних модулей треба розробити схему їх взаємодії.

Розглянемо модуль візуалізації. Через цей модуль користувач буде бачити матрицю елементів із іконками, які відображують різні типи елементів (стіни, старт, фініш, прискорювач, сповільнювач). Також цей модуль буде відображати індекси хвилі у процесі її поширення, та шлях у процесі зворотного трасування.

Розглянемо модуль інтерфейсу. Через цей модуль користувач буде використовувати такі модулі як модуль загрузки, модуль збереження та запускати сам алгоритм. У свою чергу на інтерфейс отримує данні про наявність старту та фінішу, данні щодо можливості трасування маршруту, та інші повідомлення.

Модуль створення карти буде мати доступ до елементів інтерфейсу, які відповідають за визначення поточного режиму зміни елементів. Також до цього модуля відноситься кнопка «create new map» яка надає можливість користувачу створити нову, чисту карту. Користувач вводить розміри цієї карти. Уся інформація отримана цим модулем проходить валідацію у відповідному модулі.

Модуль валідації даних – це той модуль, який не дозволяє створити два фінішних елемента, та не дозволяє створити карту розміром, який буде потребувати великого часу обчислення чи «це\_точно\_не\_число». Потрібність у цьому модулю також обумовлена обраною мовою програмування, оскільки у Javascript відсутня типізація.

Модуль введення розмірів карти є модальним вікном, і отримує розміри по вертикалі та горизонталі від користувача. Ця інформація також валідуєтся у відповідному модулі.

Модуль обробки прискорювачей як і сповільнювачей використовується через модуль поширення хвилі. Ці два модуля визначають поточні індекси, та додають їх до індексу хвилі при проходженні через елемент відповідного типу.

Модуль поширення хвилі отримує від модуля інтерфейсу команду, а від пам’яті програми – координати старту та масив елементів із типами цих елементів. Він оброблює кожен елемент сусідній до стартової точки, передаючи до нього індекс хвилі. Якщо елемент по типу відноситься до прискорювачей, чи сповільнювачей – цей елемент обробляється у відповідних модулях, та результат повертається назад. Якщо сусідній елемент по типу відноситься до фінішу – то модуль припиняє свою дію.

Модуль повернення хвилі реалізує алгоритм оберненого трасування, а для цього йому потрібно отримати із пам’яті програми координати фінішу та масив елементів. Він знаходить сусідній елемент до фінішної із найменшим індексом хвилі, передаючи до нього іконку стрілки, яка вказує шлях від старту до фінішу.

Усі останні модулі позволяють використовувати заповнену карту знову, зберігаючи на пристрої користувача JSON, у який вказується координати старту, фінішу та масив елементів, уникаючи елементи знайденого шляху. Таким чином ми можемо знаходити різні маршрути на старих картах, змінюючи їх після знайденого шляху.

Приблизний вигляд структури можна побачити на цьому зображенні:

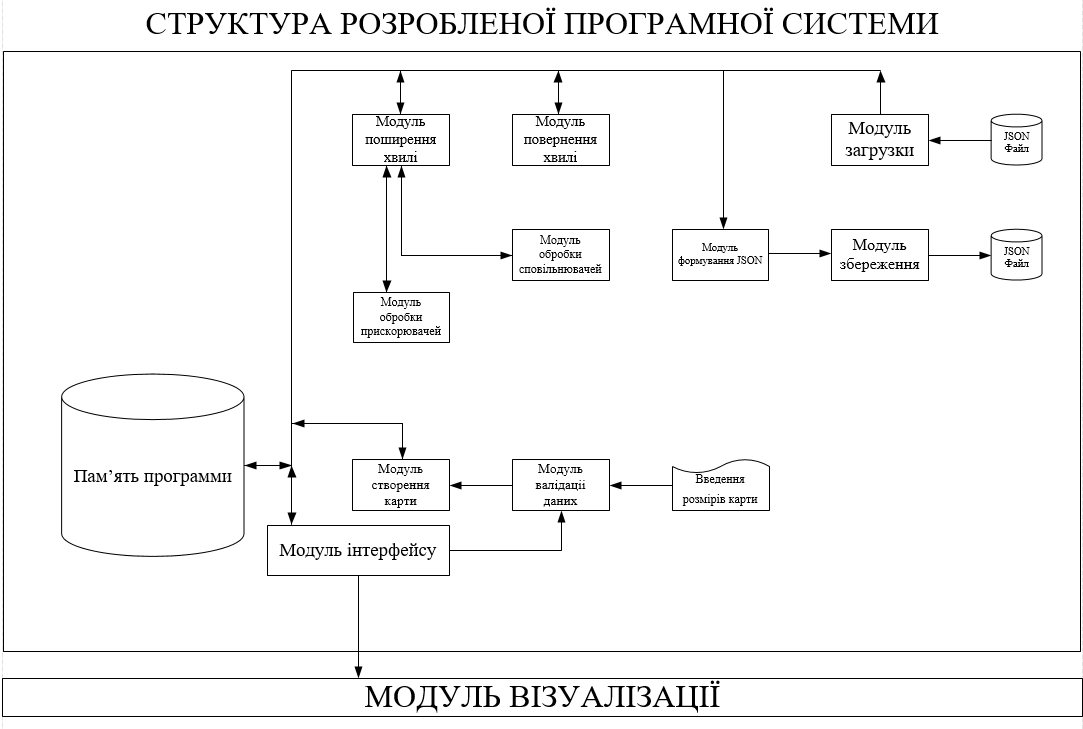


Рис. НОМЕР

3.2 Розробка інтерфейсу користувача

Для візуалізації програми було використано HTML із елементами CSS. HTML у цій схемі використовується для розмітки сторінки та для створення елементів інтерфейсу. CSS було використано для надання належного стилю. А тепер розглянемо як було реалізовано інтерфейс.

По-перше створимо 3 контейнера для карти, селектору режиму редагування і інтерфейс трасування.

У першому контейнері ми маємо таблицю, яка буде відображати карту, меню редагування масштабу, меню збереження, створення та експорту карти. Таблиця має елементи, які змінюють свій колір та зовнішній вигляд в залежності від режиму, індексу хвилі та наявності шляху який пролягає через цей елемент. Усі елементи сторінки мають однакові параметри розміру, чорну рамку в 1 піксель, відстань до внутрішніх даних є також однаковою та становить 5 пікселів.

Другий контейнер має у собі радіо кнопки, із надписами та іконками які відображують той тип елементу який вмикається при вибору одного з даних режимів. Режими прискорювача та сповільнювача також мають модулі регулювання потужності, для цього було використано класичні html input-и, які були налаштовані на діапазон 0.01 – 0.9 із кроком у 0.01, початкове значення дорівнює 0.5. Ці інпути мають тип number що не дозволяє користувачеві ввести туди щось окрім цифр.

У третьому контейнері знаходяться кнопки запуску алгоритму. Також чекбокси ортоганально-діаганального режиму та режиму крок-за-кроком із кнопкою, що вмикає наступний крок.

За допомогою CSS-стилів усім контейнерам було надано чорну рамку у 1 піксель та відступи до краю елементу у 5 пікселів та від краю елементу до іншого 5 пікселів. Таким чином елементи інтерфейсу логічно поділені на розділи. Також за допомогою CSS інпути мають сталу ширину у розмірі 45 пікселів, яка не дає їм розповсюджуватись в ширину, що псує загальний вид.

3.3 Розробка схем алгоритмів

Виходячи із описаних вище структури та інтерфейсу можна зазначити основні алгоритми програми, а саме алгоритм поширення та повернення хвилі.

Алгоритм поширення хвилі повинен починатися з того, що навколо зазначеної точки старту розповсюджується хвиля, що надає усім сусідам (в залежності чи включено ортоганально-діагональний режим) цієї точки, які можуть містити шлях, індекс хвилі 1, а якщо ці сусідні елементи по типу дорівнюють прискорювачу, чи сповільнювачу то індекс змінюється відповідно до типу та значенні бонусу, або пенальті відповідного елементу. Після чого візуалізація масиву оновлюється, якщо режим «крок за кроком» було включено. Усі сусіди, що отримали індекс потрапляють до масиву nextElements, елементи якого потім передаються до черги elementsQueue. Ця черга працює на алгоритмі FIFO. Кожен крок цієї черги наповнює масив nextElements сусідами, подібно до першої функції цього алгоритму, але додаючи до індексу хвилі 1 на кожному повторі черги. Таким чином ми отримуємо цикл, який продовжується доки уся карта не буде заповнена індексами хвилі. Залишилось тільки добавити розрив у випадку знайдення фінішу, що захистить алгоритм від зайвих дій. Після того як фініш було успішно знайдено залишається тільки оновити візуалізацію.

Алгоритм повернення хвилі починається з того, що навколо зазначеної точки фінішу визначається сусідній (залежно від ортоганально-діагонального режиму) елемент із найменшим індексом хвилі. Після цього, якщо режим «крок-за-кроком» активовано, оновлюється візуалізація карти. Таким чином визначається напрям найкоротшого шляху до фінішу. Але цей напрям потрібно продовжити до самого старту. Ось чому далі, взявши за середину попередній сусідній елемент ми шукаємо його «найменшого» сусіда. І так далі, поки не прокладемо шлях до старту. Після чого оновлюємо візуалізацію, та показуємо вікно із повідомленням «шлях було знайдено».

3.4 Розробка програмного коду програмної системи