Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

«МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ СТВОРЕННЯ MINECRAFT PIXEL ART»

Виконав: студент 4 курсу, групи 4ПІ-20б
спеціальності 121 – Інженерії програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Грицишин В. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри ПЗ
Майданюк В. П.
(прізвище та ініціали)

Рецензент: к.т.н., доцент кафедри ОТ
Кожем'яко А. В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н.
«10» червня 2024р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«12» березня 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Грицишину Владиславу Олеговичу

1. Тема роботи – Методи та програмні засоби створення Minecraft Pixel Art. Керівник роботи: Майданюк Володимир Павлович, к.т.н., доцент, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.
2. Строк подання студентом роботи 3 червня 2024.
3. Вихідні дані до роботи: тип програми – веб-застосунок; модель розробки – ітеративна; засоби організації даних програми – Typescript, блоки гри Minecraft; вихідні дані: створення видозміненого зображення у піксельноблоковому форматі; середовище розробки – Visual Studio Code; редактор коду – Visual Studio Code; мова програмування – Typescript, JavaScript; використані бібліотеки та фреймворки – React JS, CSS, HTML.
4. Зміст текстової частини: вступ, обґрунтування вибору методу розробки та постановка задач дослідження; розробка програмного забезпечення; тестування програмного забезпечення; висновки; список використаних джерел; додатки.
5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність одержаних результатів; порівняльний аналіз аналогів; структура застосунку; інтерфейс програмного забезпечення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Майданюк В.П. к.т.н., доцент	13.03.2024	03.06.202

7. Дата видачі завдання 13 березня 2024

КАЛЕНДАРНИЙ ПЛАН

№ п\п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз підходів до створення Minecraft Pixel Art	14.03.24 – 21.03.24	Виконано
2	Розробка методів створення Minecraft Pixel Art	11.03.24 – 22.03.24	Виконано
3	Розробка алгоритмів роботи програмних методів	26.03.24 – 20.04.24	Виконано
4	Розробка програмних методів	20.04.24 – 10.05.24	Виконано
5	Тестування програмних методів	12.05.24 – 25.05.24	Виконано
6	Оформлення матеріалів до захисту БДР	26.05.24 – 30.05.24	Виконано

Студент _____ **В.О. Грицишин**
(підпис) (ініціали та прізвище

Керівник бакалаврської дипломної роботи _____ **В. П. Майданюк**
(підпис) (ініціали та прізвище

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 55 сторінок формату А4, на яких є 31 рисунок, 3 таблиці, список використаних джерел містить 22 найменувань.

У бакалаврській дипломній роботі було визначено доцільність, практичну цінність та мету розробки, яка полягає у покращені систем перетворенню зображень в Minecraft Pixel Art, що додатково буде використано різні методи для кращої взаємодії користувачів.

Запропоновано модифікацію алгоритму дій програмного застосунку. Розроблено алгоритм взаємодії системи із методами та способами створення піксельних зображень. Удосконалено спосіб перетворення зображення у 2D-масив і порівняння та заміна кольорів на блоки з гри Minecraft.

Програмне забезпечення розроблено із використанням середовища розробки Visual Studio Code та мови програмування Typescript, зокрема з використанням фреймворку Javascript, CSS та React. У результаті виконання бакалаврської дипломної роботи розроблено програмний веб-застосунок, працездатність та коректність виконання якого було перевірено за допомогою впровадження модульного тестування, також було підготовлено інструкцію користувача та визначено системні вимоги.

Отримані в бакалаврській дипломній роботі результати можна використати для ефективного застосування у сфері маркетингу та покращення досвіду серверобудування.

Ключові слова: Pixel Art, перетворення зображення, гра Minecraft, веб-застосунок, креатив.

ABSTRACT

The bachelor's thesis consists of 55 A4 pages, which include 31 figures, 3 tables, and a list of references containing 22 titles.

In the bachelor's thesis, the expediency, practical value and purpose of the development were determined, which is to improve the image conversion system in Minecraft Pixel Art, which will additionally use various methods for better user interaction.

A modification of the algorithm of the software application is proposed. An algorithm for the system's interaction with methods and techniques for creating pixel images has been developed. The method of converting an image into a 2D array and comparing and replacing colors with blocks from the game Minecraft has been improved.

The software was developed using the Visual Studio Code development environment and the Typescript programming language, in particular, using the Javascript, CSS, and React framework. As a result of the bachelor's thesis, a web application was developed, the operability and correctness of which was checked by implementing unit testing, a user manual was also prepared and system requirements were determined.

The results obtained in the bachelor's thesis can be used for effective application in the field of marketing and improvement of the server building experience.

Keywords: Pixel Art, image conversion, Minecraft game, web application, creativity.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ потребує змін. Великих змін. 21 сторіччя наповнене новими цікавими та креативними професіями, які передбачають незвичайний підхід до подачі і обробки контенту. Тому, розробка програмних засобів створення Minecraft Pixel Art набуває неабиякого значення. Тема є досить актуальною для тих, хто працює чи пробує себе у сфері маркетингу і намагається просунути свій контент під новим виглядом. Це спонукає людей використовувати більш новітні методи та засоби по обробці зображень, адже піксель арт оцінюється доволі високо [1].

Однією з ключових причин актуальності створення таких програмних засобів слугує недосконаліть перетворення звичних нам зображень у піксель арт. Веб-застосунки, що пропонує мережа Інтернет є і можуть бути неефективними і частіше всього не виправдовують результат, на який очікує користувач. А ті застосунки, що працюють більш-менш коректно, зачасту є платними або використовуються по підпискам чи кодам запрошення.

Окрім цього, існують потреби у різноманітності таких програмних засобах та веб-застосунках, оскільки кожна програма може бачити ідею створення Minecraft Pixel Art з власних зображень. Така система дозволить кожному користувачеві, незалежно від виду діяльності отримати досвід та нові знімки відповідно до їх вподобань.

Розвиток створення такого роду систем відбувався з кожним виходом нової глобальної версії гри Minecraft. Різноманіття блоків, що було додано у гру дозволяє створювати та використовувати різні комбінації для кращого відтворення оригінального зображення, що в свою чергу допомагає у розробці більш ефективних та точних передач інформації у нову фотографію. Розробка таких програмних засобів має великий потенціал у створенні креативів для будь-яких цілей. Вони допоможуть покращити загальне сприйняття піксель арту, а коли він створений незвичним способом з використанням незвичних пікселів – блоків, то з’являється неабиякий інтерес у людей [2].

Також важливо враховувати сучасний стан ринку програмних модулів і веб-застосунків, що займаються створенням піксель арту із зображень. Наразі існує багато застосунків, які містять різноманітні функції, від стандартного перетворення зображення до редагування таких створених зображень. Однак, не дивлячись на широкий вибір існуючих рішень, багато з них все ще мають певні обмеження щодо ефективності, гнучкості та підходу до створення [3].

Актуальність покращення існуючих систем полягає в неефективності створення піксель арту з зображень, що може не виправдати очікувань користувача. Це стосується як створення зображення та його завантаження, так і розширеного функціоналу, такого як редагування створеного зображення та його імпорт у існуючі системи, що часто базуються на загальних засадах, а не на персоналізованих потребах окремого користувача. Розвиток методів, особистого підходу та цікавих оновлень блокової палітри дозволить створити цікавий продукт.

Тому реалізація програмних засобів створення Minecraft Pixel Art є актуальною розробкою.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення зручності та швидкості формування Minecraft Pixel Art з використанням додаткових інструментів щодо імпорту та редагування.

Основними задачами дослідження є:

- провести аналіз стану питання та методи розв’язання задачі;
- спроєктувати модель даних та структуру програмних засобів;
- розробити загальний алгоритм роботи веб-застосунку;
- розробити алгоритм обробки зображень;
- виконати програмну реалізацію головного екрану застосунку;
- запровадити взаємодію між клієнтом та системою веб-застосунку;
- провести тестування програмних засобів;

- розробити інструкцію користувача та описати системні вимоги для програмних засобів.

Об’єкт дослідження – процес розробки програмних модулів для реалізації засобів створення Minecraft Pixel Art.

Предмет дослідження – методи і програмні засоби формування Minecraft Pixel Art.

Методи дослідження. У процесі досліджень використовувались: порівняльний аналіз функціональних характеристик програм-аналогів для виявлення їх недоліків та постановки задач розробки; теорія алгоритмів для розробки та візуалізації алгоритмів роботи програмного забезпечення; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Удосконалено функцію редагування створеного зображення, яка відрізняється від відомих наявністю повноцінного редактора зображення, що дозволило більш точну передачу колірної гами для задоволення потреб користувача.

2. Подальшого розвитку отримав метод створення пакетів даних для гри Minecraft, у якому на відміну від відомих додано масив для ідентифікації, що забезпечує повну сумісність з будь-якими версіями гри Minecraft.

3. Запропоновано метод перетворення зображення у 2D-масив, у якому на відміну від відомих виконується порівняння колірної гами оригінального зображення з колірною гамою палітри блоків, що дозволяє більш точно відобразити новостворене зображення.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для вдосконалення та популяризації програмних засобів створення Minecraft Pixel Art**.**

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській дипломній роботі, отримано автором самостійно. У наукових працях, опублікованих у співавторстві, автору належать такі результати: виконав аналіз інструментів піксельної графіки [4].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Міжнародній науково-практичній інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)»;

Публікації. За тематикою дослідження опубліковано 1 наукова праця у збірниках матеріалів конференцій.

1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 **Аналіз стану програмних засобів створення піксель арту**

Розвиток піксель артів – досить цікава тема для обговорення, адже саме така гілка малюнків привабила левову частку користувачів Інтернету. Піксель арт або ж піксельна графіка – тип цифрового зображення, яке створюється на комп'ютері за допомогою растрового графічного редактора. У цьому типі графіки зображення редагується на рівні окремих пікселів (точок), і роздільна здатність зображення настільки низька, що окремі пікселі чітко видно [5]. Цей виразний стиль має цілу низку переваг.

- Ретро-шарм: Піксельне мистецтво викликає почуття ностальгії, нагадуючи користувачам про класичні відеоігри 80-х і 90-х років.

- Ефективність використання ресурсів: Низька роздільна здатність означає, що ці зображення швидко завантажуються і вимагають мінімальної обчислювальної потужності.

- Простота створення: Редагувати піксельний арт дуже просто, оскільки він передбачає маніпуляції з окремими пікселями, не потребуючи складних інструментів або багато часу.

- Експерименти: Простота дозволяє легко експериментувати, заохочуючи творчість і швидкі ітерації.

- Адаптивність: Піксельні ілюстрації можна легко масштабувати до різних роздільних здатностей без втрати якості та деталізації.

Створення піксельного мистецтва не обов'язково є складним процесом, але коли впроваджуються зовнішні інструменти – ось де часто криється виклик. Розробка спеціалізованого програмного забезпечення вимагає глибокого розуміння як технік піксельного мистецтва, так і технічних аспектів програмування таких інструментів.

Розробники та користувачі часто співпрацюють для подолання цих викликів, звертаючись до експертів у галузі піксельного мистецтва за порадою.

Ці фахівці мають великий досвід і можуть надати цінні поради щодо створення ефективних додатків.

Однією з цікавих ідей є створення Minecraft Pixel Art, де пікселі замінюються різними ігровими блоками. Це виводить піксельне мистецтво на новий рівень, пропонуючи більші творчі можливості та розширюючи способи створення зображень.

Розробляючи таке програмне забезпечення, користувачі отримують доступ до інтригуючих можливостей і функцій:

- Завантаження зображень: Дозволяє користувачам завантажувати та конвертувати свої зображення в піксельний арт у Minecraft.

- Інтеграція: Безперешкодне включення цих зображень в ігрове середовище.

- Редагування: Дозволяє вносити зміни та покращення у світ Minecraft.

- Створення команд: Створення шаблонів команд для автоматизації та кастомізації процесу створення піксель арту.

Такі функціональні можливості не лише розширюють творчі можливості користувачів, але й підвищують задоволеність та попит на ці інструменти.

Отже, розробка програмних засобів Minecraft Pixel Art є можливо не критичним, але важливим завданням для створення сучасного цифрового контенту для різних галузей. З використанням досвіду людей та контентмейкерів по створенню таких зображень, розробники можуть створити цікаві рішення, які покращуватимуть досвід пересічного користувача та будуть конкурентоспроможними в такій галузі. Беручи до уваги постійний розвиток технологій по створенню таких піксель артів, важливо слідкувати за оновленнями як самої гри Minecraft, так і створення стандартних піксель артів, аби залишатись в тренді та бути актуальними.

1.2 Порівняльний аналіз аналогів

Використання технологій по створенню Minecraft Pixel Art все більше стає популярним протягом останніх 3 років, і це доволі часто інтегрують у власні

роботи контентмейкери, геймдизайнери, звичайні дизайнери, власники серверів. Розробка програмних засобів по створенню Minecraft Pixel Art може стати захоплюючий досвід, створення унікального контенту, а також дати нові можливості для креативних людей. До найбільш відомих таких рішень відносять:

- minecraftart;
- Pixel art maker for minecraft;
- spritecraft;
- mcstacker murals.

Одним із цікавих рішень по створенню Minecraft Pixel Art є веб-застосунок minecraftart (рис 1.1). Даний застосунок використовує принцип переробки завантажуваного зображення на нове, але з використанням блоків. Таке рішення можна використовувати для різних стартових цілей. Але воно не є ідеальним через повільні алгоритми опрацювання отриманого зображення та перетворення його [6].

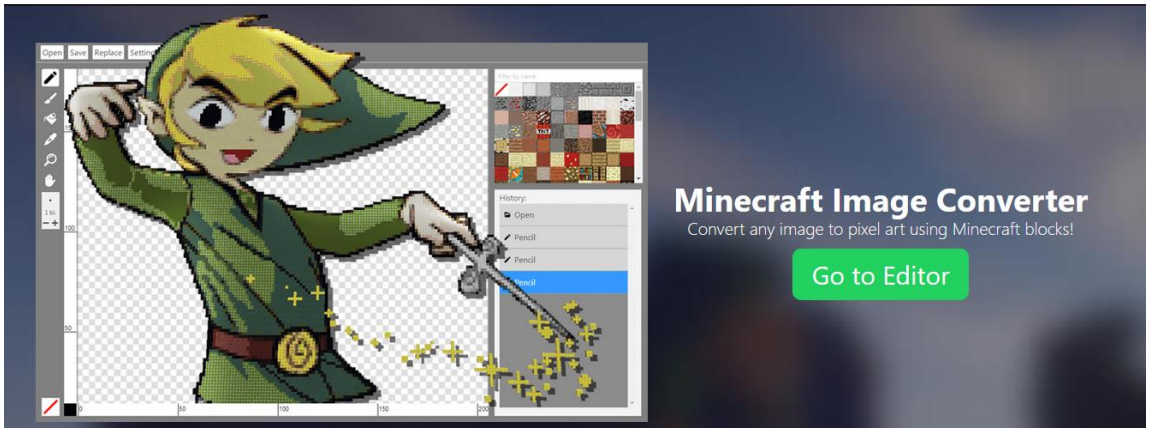


Рисунок 1.1 – Приклад роботи застосунку minecraftart

Іншим прикладом може слугувати застосунок Pixel art maker for minecraft (рис. 1.2), який уже має у використанні більш прості та швидкі алгоритми, але все ще не так ідеально. Додаток дозволяє створити картинку, після чого її відредагувати напряму у редакторі. Але не дозволяє інтеграцію напряму з грою чи іншими сервісами, що робить його не досить конкурентоспоможним на фоні інших застосунків [7].

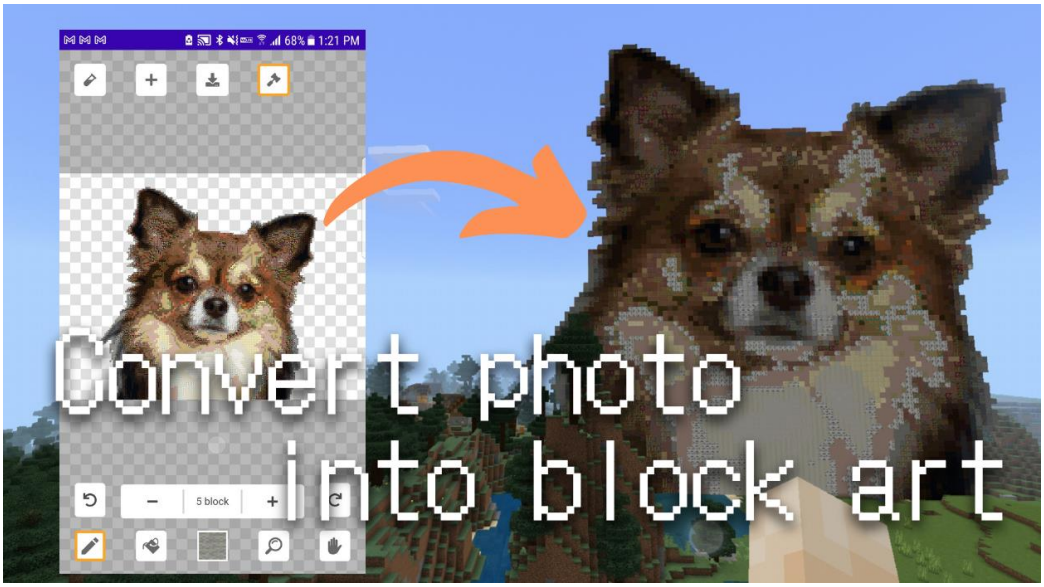


Рисунок 1.2 – Приклад роботи додатку Pixel art maker for Minecraft

Наступним застосунком може слугувати spritecraft (рис 1.3). Додаток має широкі можливості з редагування, але обмежений вибором колірної схеми та текстур для створення Minecraft Pixel Art – лише різнокольорова вовна. Це робить створюваний піксель арт не таким цікавим на фоні інших рішень, але більш простим і стриманим у використанні та редагуванні [8].

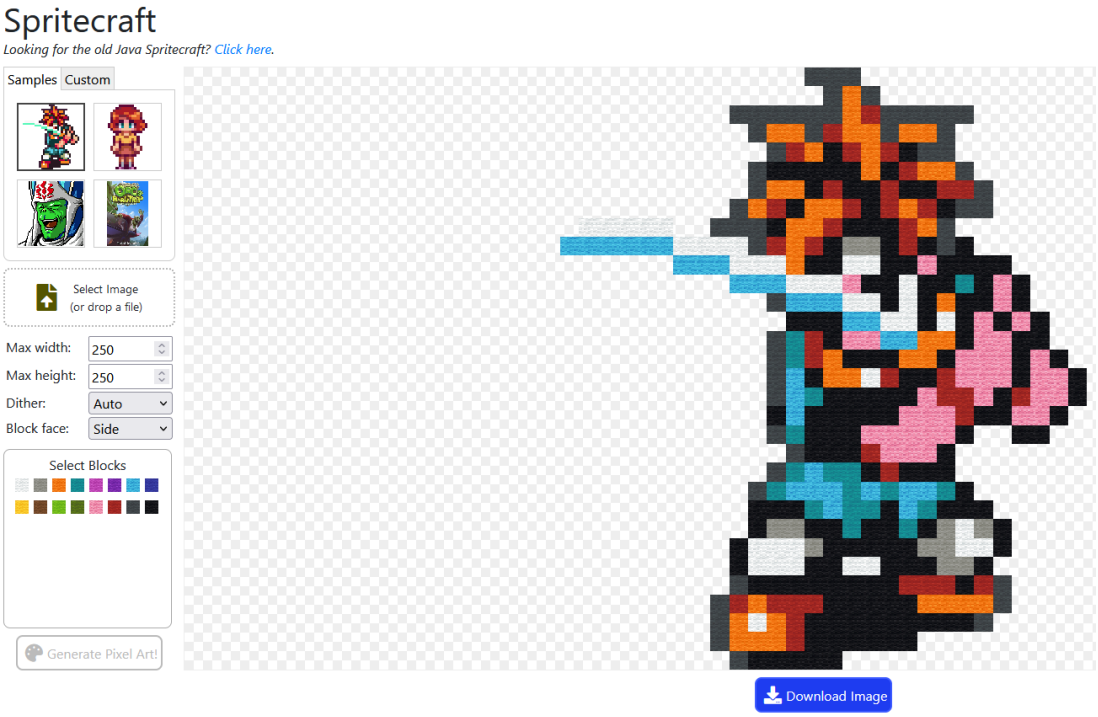


Рисунок 1.3 – Приклад роботи застосунку spritecraft

Mcstacker murals (рис. 1.4) доволі стара розробка, але й одночасно розширена, з великою палітрою блоків та текстур маючи арсенал різних версій гри Minecraft, починаючи з 1.8 та закінчуючи останньою версією 1.20. З цікавих особливостей варто виділити покрокову інструкцію як створити зображення використовуючи такий Minecraft Pixel Art генератор. Мало сказати, що даний застосунок містить абсолютно ВСІ блоки та предмети з гри, а також він дозволяє імпортувати згенероване зображення у датапак (пакети даних, що додають у гру різні можливості та корисності), функцію Minecraft (дозволяють гравцям запускати списки команд, використовуючи текстові файли з розширенням .mcfunction та при цьому не навантажувати гру, як це роблять командні блоки), самостійне створення такого піксель арту у режимі виживання, через командний блок у грі. Маючи такий широкий функціональний набір, даний застосунок користується популярністю у багатьох дизайнерів, контентмейкерів, власників серверів [9].

Рисунок 1.4 – Приклад роботи додатку mcstacker murals

Розробка програмних засобів для створення Minecraft Pixel Art вимагає не лише знань над тонкощами піксель арту, а й також правильної розробки програмного забезпечення та створення відповідного дизайну для досвіду користувача. Це означає, що створення такої системи програмних засобів буде задовольняти потреби користувача та швидко обробляти команди і створені алгоритми шляхом різних оптимізацій, а також безперешкодної інтеграції у саму гру Minecraft чи сервіси дизайну. Інтерфейс передбачає розробку простого та інтуїтивно зрозумілого як для звичайного користувача, так і для досвідченого контентмейкера чи дизайнера у ігровій сфері.

Результати порівняння аналогів наведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристика платформ створення Minecraft Pixel Art

Критерії	minecraft art	Pixel art maker for minecraft	spritecraft	mcstacker murals	Власна розробка
Функції імпорту	+	+	+	+	+
Різноманітність експорту	+	-	-	+	+
Палітра блоків і текстур	-	+	-	+	+
Зручність інтерфейсу	+	-	-	+	+
Швидкодія застосунку	-	-	-	-	+
Загальна оцінка	60%	40%	20%	80%	100%

Відповідно до таблиці порівняльних характеристик аналогів (табл. 1.1) розробка власних програмних засобів створення Minecraft Pixel Art є доцільною.

Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити новий досвід та можливості створення піксель артів.

Отже, розробка програмних засобів створення Minecraft Pixel Art відкриває перед користувачами широкі можливості для взаємодії із зображеннями, створення унікального контенту для різних цілей, а також підвищення рівня залученості та утримання аудиторії. Розуміючи потенційні можливості створення Minecraft Pixel Art та вимоги до розробки програмного забезпечення, користувачі можуть створювати доволі цікаві та привабливі зображення, які були створені з використанням технологій піксель арту для власних цілей та успіху.

1.3 Аналіз напрямків удосконалення програмного забезпечення для створення Minecraft Pixel Art

Зважаючи на результати аналізу, можна сформулювати програмне забезпечення для створення Minecraft Pixel Art, яке буде допомагати користувачам швидко та зручно створювати малюнки. Ось опис можливих напрямків удосконалення такого програмного засобу:

Розширення бази даних блоків: Основою для створення Minecraft pixel art є блоки, тому важливо мати широку базу даних різних типів блоків і предметів. Окрім стандартних блоків, таких як земля, камінь і дерево, варто додати також спеціальні блоки, які відображають особливості Minecraft, наприклад, блоки Redstone або грибні блоки.

Інтуїтивний інтерфейс: Важливо мати зручний інтерфейс, який дозволить користувачам легко знаходити необхідні блоки та швидко створювати свої шедеври. Меню вибору блоків може бути організоване за категоріями або кольорами для зручності.

Можливість імпорту та експорту: Додавання функції імпорту та експорту зображень дозволить користувачам використовувати свої власні шаблони або ділитися своїми творіннями з іншими користувачами.

Розширений набір інструментів: Покращення інструментів для роботи з блоками, таких як інструмент для видалення, зміни розміру, обертання і копіювання блоків, дозволить користувачам створювати більш складні та деталізовані шедеври.

Модуль для навчання та натхнення: Включення розділу з навчальними матеріалами та прикладами Minecraft pixel art може бути корисним для початківців, які тільки починають вивчати цю техніку малювання.

Підтримка спільноти: Додавання функцій спільноти, таких як можливість ділитися твореннями у власному профілі, коментування і лайкання інших творінь, а також участь у конкурсах та подіях, сприятиме взаємодії користувачів та створенню активної спільноти навкого програми.

Такий програмний засіб для створення Minecraft pixel art, якщо його розробляти та удосконалювати, може стати потужним інструментом для творчості та спілкування у спільноті фанатів Minecraft і не тільки.

1.4 Аналіз методів розв’язання поставленої задачі

Розробка програмних модулів для реалізації програмних засобів Minecraft Pixel Art вимагає ретельного розгляду найкращих підходів до вирішення проблеми. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних модулів.

Одним з варіантів вирішення проблеми розробки програмних модулів для реалізації веб-застосунку Minecraft Pixel Art є використання окремих бібліотек та функцій, що дозволять інтегрувати модуль в існуючі рішення. Проте цей метод має свої недоліки. Незважаючи на те, що це швидкий спосіб розробки модуля, він потребує додаткових зусиль від існуючих онлайн-платформ. Цей підхід також може призвести до значного обсягу роботи, який потрібно виконати команді розробників онлайн-платформи, що бажає інтегрувати це рішення. Тому рекомендується утриматися від застосування цього підходу [10].

Інший варіант передбачає створення окремого сервісу для інтеграції модуля в існуючі системи. Цей підхід дозволяє розробникам фокусуватися на створенні модуля, не витрачаючи часу на розробку повномасштабної онлайн-платформи. Користувачі цього сервісу можуть впроваджувати готове рішення у свої системи, що сприяє зменшенню затрат, порівняно з попереднім методом. Крім того, цей підхід передбачає розробку внутрішньої системи авторизації для захисту даних онлайн-платформ, які планують інтегрувати модуль. Однак для інтеграції з багатьма сервісами цей сервіс може вимагати значних апаратних ресурсів. Тому цей підхід вважається ефективним лише у випадку, якщо його можна швидко та стабільно масштабувати [11].

Найбільш поширеним варіантом є створення власної онлайн-платформи зі вбудованим модулем для реалізації програмних засобів Minecraft Pixel Art. У цьому випадку немає потреби адаптувати систему до різних потреб зовнішніх платформ. Крім того, такий підхід призводить до створення унікальної платформи з власним функціоналом. Проте цей метод має свої недоліки, оскільки вимагає повноцінної розробки платформи. Розглянувши переваги та недоліки кожного методу, було вирішено використати метод розробки власної онлайн-платформи з вбудованим модулем для реалізації програмних засобів Minecraft Pixel Art. Такий підхід дозволить отримати повноцінний програмний продукт без необхідності використання потужного апаратного забезпечення. Більше того, такий метод розробки дозволить створити унікальну платформу з унікальним функціоналом.

1.5 Постановка задач для розробки веб-системи створення Minecraft pixel art

Проаналізувавши переваги та недоліки існуючих програмних засобів Minecraft Pixel Art було визначено наступні завдання, які необхідно виконати для розробки веб-застосунку:

- здійснити аналіз стану питання та методів розв’язання задачі;

- спроектувати модель даних та структуру мобільного застосунку;
- розробити загальний алгоритм роботи програми;
- визначити найбільш ефективний підхід для перетворення зображення;
- розробити алгоритм розпізнавання кольорів;
- розробити зручний та адаптивний графічний інтерфейс;
- розробити веб-застосунок з імплементацією перетворення зображення в піксель арт;
- провести тестування веб-застосунку згідно поставлених задач.

Технічне завдання на розробку наведено в додатку А.

1.6 Висновки

У першому розділі було розглянуто стан програмних засобів Minecraft Pixel Art. Було проведено порівняння відомих платформ для перетворення зображень в піксель арт з використанням блоків з гри Minecraft, таких як: mcstacker murals, spritecraft, Pixel art maker for Minecraft, minecraftart.

У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності серед учасників по створенню піксель артів завдяки своїй зручності та доступності. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що вони не надають можливості навігації та мультиплатформеності. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного веб-застосунку.

2. РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних та вихідних даних програмного застосунку

Вхідні та вихідні дані - це своєрідний "алфавіт" програмного забезпечення, за допомогою якого воно спілкується зі світом. Вхідні дані - це "повідомлення", які програма отримує, а вихідні дані - це її "відповіді". Цей обмін інформацією визначає, як програма працює, які завдання вона може виконувати та як взаємодіє з користувачами чи іншими програмами.

Вхідними даними від користувача для веб-застосунку слугують його звичайне зображення та обрана одна чи декілька палітр блоків.

Вихідними даними можуть слугувати новостворений піксель арт, команди та архіви для додавання його у гру та статистичні дані.

Для реалізації веб-застосунку з створення Minecraft Pixel Art будуть використані мови програмування TypeScript, JavaScript та CSS. TypeScript — це мова програмування, розроблена Microsoft у 2012 році як надбудова JavaScript. Це означає, що будь-який код JavaScript є валідним TypeScript кодом. Натомість TypeScript привносить деякі нові функції та синтаксис до JavaScript, що робить код більш надійними, зручним в обслуговуванні та масштабуванні [12].

Інша відмінність від JavaScript полягає в тому, що TypeScript є строго типізованою мовою програмування. Це означає, що змінні, функції, аргументи та інші конструкції мають бути визначені за допомогою конкретного типу даних [13].

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки [14].

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу [15].

CSS (Cascading Style Sheets або каскадні таблиці стилів) називається мова візуального представлення вмісту HTML-документа. Він має на увазі додавання кольору, редагування відступів, шрифтів, розташування кожного блоку на сторінці.

HTML організовує інформацію і надає структуру документу, а CSS оформляє його, прописуючи унікальні властивості кожному з елементів.

CSS – це файл, в якому прописується правила оформлення всіх елементів на сторінці. Завдяки цьому код містить менше повторюваних елементів, його простіше читати. Можна поширити одну таблицю стилів на безліч сторінок, істотно знизивши час на верстку. Кожне правило в файлі складається з селектора і блоку оформлення. Функція селектора -визначити, до якої саме частини документа застосувати конкретне правило. Блок оформлення складається з пар «властивість: значення». Вони пишуться в фігурних дужках і закінчуються крапкою з комою.

При наявності на сторінці CSS, браузер кешує її, завдяки чому знижується час завантаження сайту при наступному вході [16].

Поєднання TypeScript, JavaScript та CSS дозволить створити веб-застосунок, який надасть користувачам цікавий досвід створення піксель артів, використовуючи блоки з гри Minecraft.

Розробка програмних модулів для реалізації створення Minecraft Pixel Art передбачає кілька етапів. Спершу необхідно створити модуль, який буде перетворювати бажане зображення користувача в масив. Цей модуль буде відповідати за конвертацію зображення у 2d-масив та подальше надання даних

для системи створення Minecraft Pixel Art. Далі буде розроблений модуль для обробки отриманих даних і виявлення кольорів. Цей модуль буде використовувати власні функції для розпізнавання кольорів і подальшу заміну їх на блоки схожого кольору.

Після конвертації та обробки зображення буде розроблено модуль для відображення видозміненого зображення з підібраними блоками по кольорам з гри. Цей модуль використовуватиме дані, які надав користувач, завантаживши картинку та функціями перетворення зображення та розпізнавання кольорів для створення веб-застосунку Minecraft Pixel Art. Наступний буде модуль редагування створеного піксель арту. Даний модуль дозволить користувачам відредагувати створене зображення або ж зробити власне.

Розробка цих програмних модулів вимагає розуміння колірних блоків гри Minecraft, креатив та створення верстки функцій на мовах програмування. Для реалізації таких модулів будуть використані різноманітні інструменти, зокрема VS Code, гра Minecraft та бібліотеки програмування. Програмні модулі будуть доопрацьовані та протестовані, аби забезпечити надійну роботу щодо створення Minecraft Pixel Art для різних користувачів.

Отже, розробка програмних модулів для реалізації веб-застосунку Minecraft Pixel Art з використанням TypeScript, JavaScript та CSS є комплексним завданням, що включає в себе кілька етапів. Модулі необхідно ретельно спроектувати, реалізувати та протестувати, аби забезпечити якісний досвід роботи користувача з веб-застосунком. Однак, маючи правильні інструменти та досвід, можна створити юзерфреднлі застосунок для користувачів і контентмейкерів.

2.2 Розробка структури програмного застосунку

Структура розробки веб-застосунку для реалізації програмних засобів Minecraft pixel art передбачає організацію та компонування всіх функцій та взаємозв’язки між ними для більш точної роботи над зображенням та надання необхідних даних користувачеві. Структура має містити всіх необхідні

компоненти веб-застосунку, які дозволять взаємодіяти з користувачем, отримувати палітру блоків, змінене зображення, команди та файли тощо.

Для візуального представлення структури веб-застосунку та зв’язків між функціями та компонентами було створено діаграму варіантів використання (рисунок 2.1). Дана діаграма відображає дії користувача, які можуть використовуватись під час роботи, це може бути:

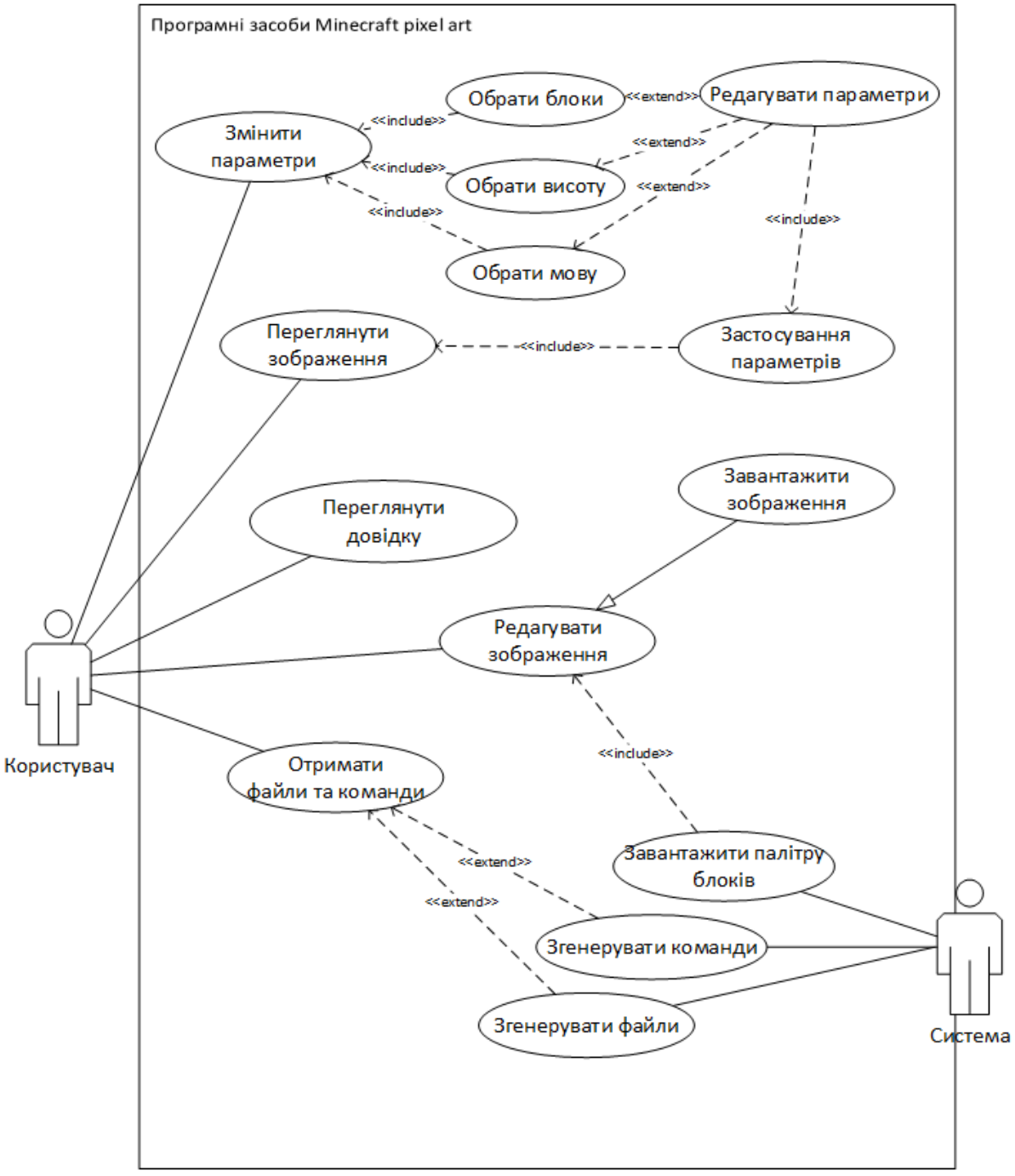


Рисунок 2.1 – Зображення діаграми варіантів використання

2.3 Розробка моделі системи

Для кращого розуміння роботи програмних модулів веб-застосунку створення Minecraft Pixel Art було вирішено розробити модель роботи системи на прикладі UML діаграми діяльності (рис. 2.11, 2.12). Згідно вказаної діаграми першим ділом користувач повинен обрати зображення. Наступним кроком користувач розпочинає перегляд обраного зображення. Перед початком конвертації зображення у піксель арт, застосунок перетворює його у 2d-масив і аналізує палітру кольорів. Якщо результат перетворення виявився невдалим, користувачу пропонується обрати інше зображення. Далі користувач: конвертує зображення, після чого він може обрати одну з чотирьох дій: завантажити зображення, завантажити csv, згенерувати команду, редагувати.

Рисунок 2.11 – Частина діаграми діяльності веб-застосунку

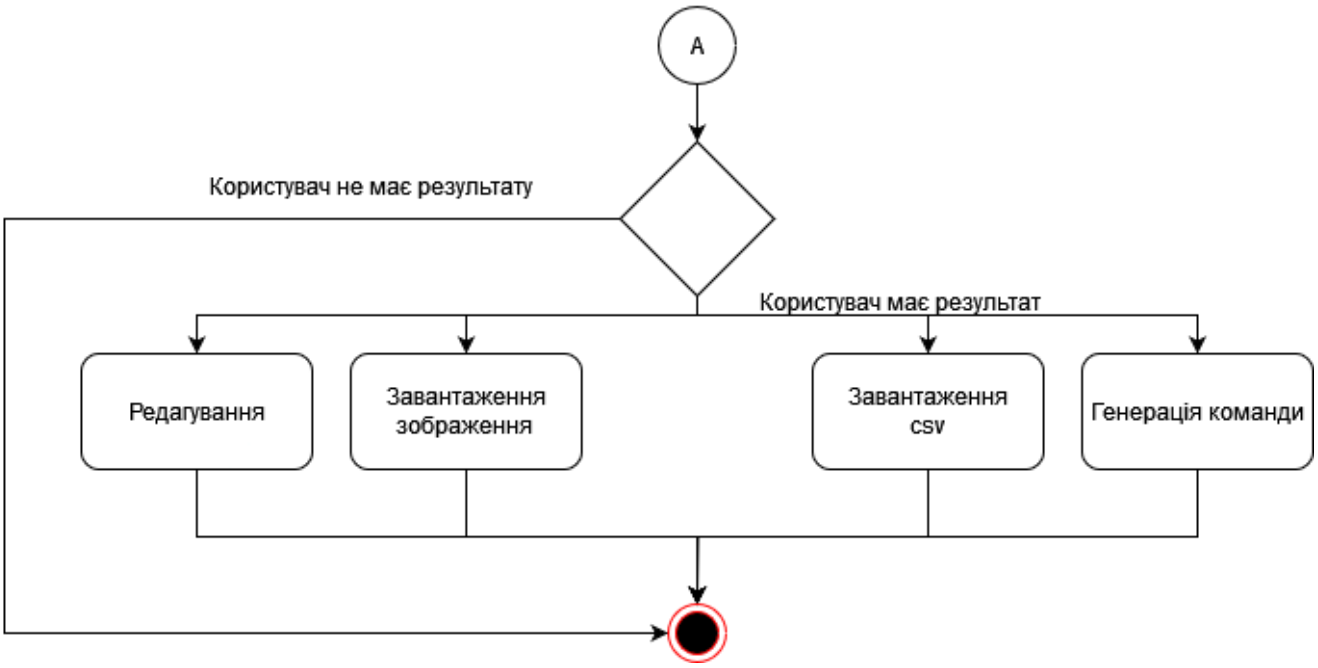


Рисунок 2.12 – Продовження діаграми діяльності веб-застосунку

2.4 Розробка загального алгоритму роботи програми

Для розробки веб-застосунку по створенню Minecraft Pixel Art необхідно розробити загальний алгоритм, згідно якого застосунок буде працювати. Згідно даного алгоритму користувач спершу отримує екран робочого простору, де може вибрати потрібне зображення та потрібну палітру кольорів з блоків. Обравши фотографію, необхідно впевнитись, що зображення імпортувалось правильно, це можна побачити на передперегляді обраного зображення. Якщо все добре, користувач далі мусить обрати хоча б одну палітру блоків із доступних, аби у подальшому можна було створити піксель арт. Доки користувач не вибере зображення чи палітру блоків, він далі не зможе працювати у застосунку. Отже, після запуску веб-застосунку, запускається модуль головного вікна, що відображає передперегляд обраного чи ще не вибраного зображення та палітру блоків, що є доступною для вибору (рис. 2.13).

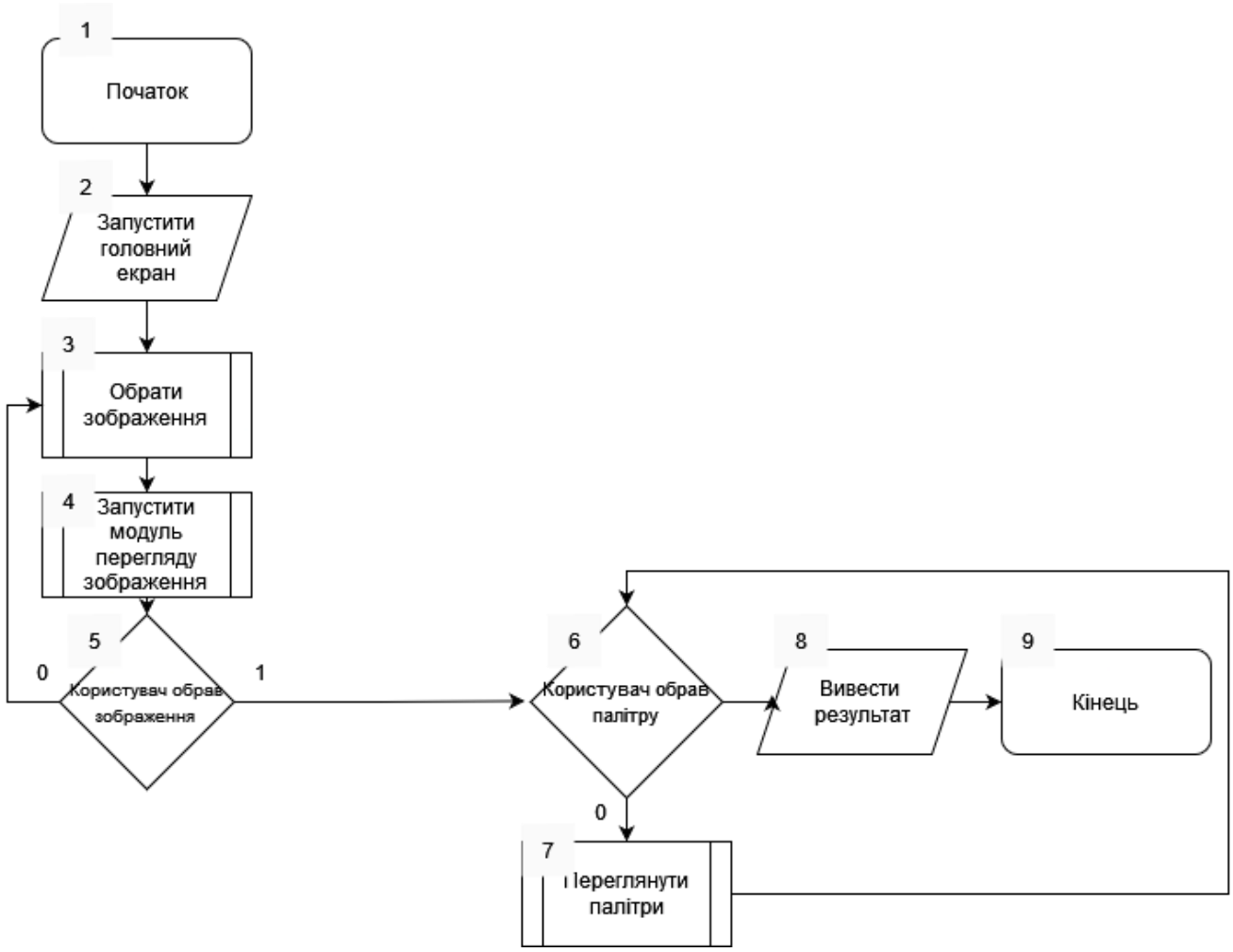


Рисунок 2.13 – Головний екран веб-застосунку

Після натискання кнопки конвертації або ж перетворення вибраного зображення та палітри користувачем у Minecraft Pixel Art веб-застосунок, його переносить на екран передперегляду уже створеного піксель арту функціями завантаження зображення, файлу csv, генерації команди та редагування новоствореного піксель арту. Користувач може обирати одну з наявних функцій або ж просто перегенерувати зображення. Завантаження зображення відбувається з збереженням розміру, csv файл генерує замість блоків текст (корисно для імпорту в інші інструменти), створення команди для гри (у подальшому для імпорту у гру Minecraft) та редагування новоствореного зображення (рис. 2.14).

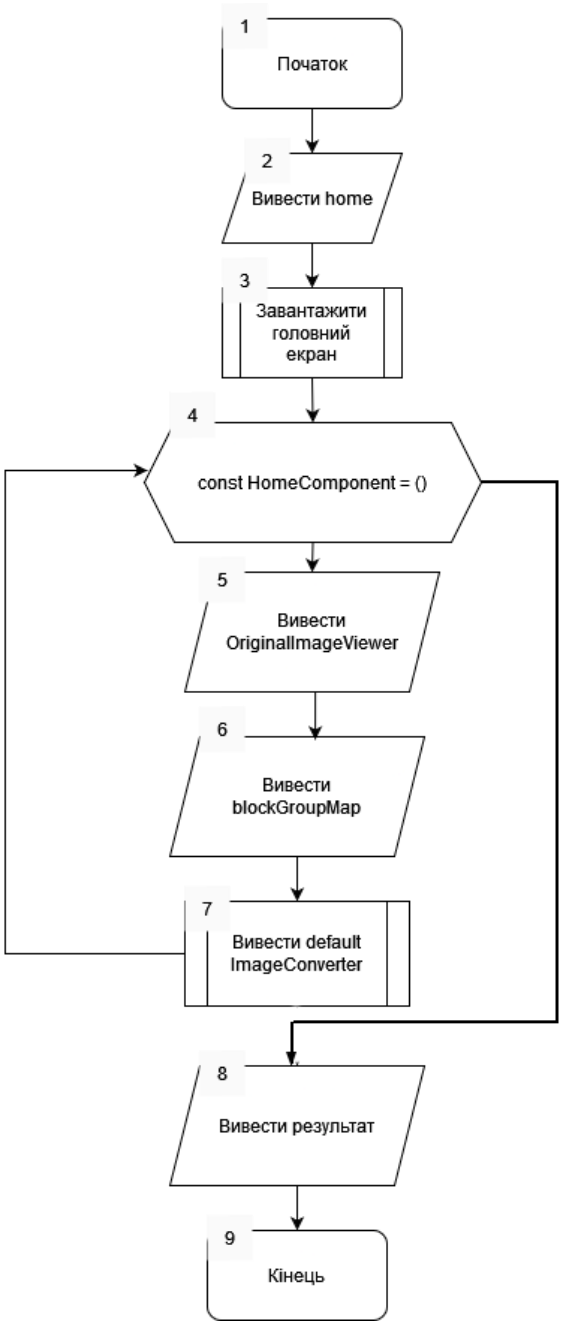


Рисунок 2.14 – Алгоритм роботи веб-застосунку

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмного продукту. Також було розглянуто можливість покращення продуктивності веб-застосунку шляхом кешування вхідних даних. Було розроблено основний алгоритм роботи веб-застосунку та алгоритм перетворення зображення. Також було розроблену модель роботи програмного продукту за допомогою UML діаграм [17].

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розробляючи програмні модулі для веб-застосунку створення Minecraft Pixel Art, важливо ретельно обирати технології, що використовуються.

Мови програмування:

TypeScript є розширенням JavaScript, яке надає статичну типізацію, що допомагає у виявленні помилок на етапі розробки. Використання TypeScript підвищує стабільність програми та спрощує роботу в команді завдяки автодокументації коду.

Основні особливості TypeScript:

Статична типізація: TypeScript дозволяє визначати типи для змінних, параметрів функцій, об'єктів, масивів та інших елементів коду. Це допомагає виявити помилки під час розробки, такі як помилки типізації, ще до того, як код буде виконаний.

Класи та інтерфейси: TypeScript підтримує об'єктно-орієнтовану парадигму програмування. Ви можете використовувати класи для створення об'єктів та інтерфейси для визначення форми об'єктів.

Генеріки: TypeScript дозволяє створювати загальні типи, які можуть використовуватися для різних типів даних. Це дозволяє створювати більш загальні та повторно використовувані функції та класи.

Екосистема: TypeScript має широку екосистему інструментів, таких як компілятор, який перетворює TypeScript код в JavaScript, і різноманітні інструменти розробки, які підтримують автодоповнення, перевірку типів та інші функції.

Сумісність з JavaScript: TypeScript код може бути інтегрований з існуючим JavaScript кодом. Ви можете почати писати новий код TypeScript, додавати типи до існуючих JavaScript файлів та поступово перетворювати їх на TypeScript.

Транспіляція: TypeScript код компілюється в JavaScript за допомогою TypeScript компілятора. Це дозволяє писати код на TypeScript, який може бути виконаний в будь-якому середовищі, яке підтримує JavaScript.

Переваги: Забезпечує безпеку типів, покращує продуктивність розробника, підтримує сучасні можливості JavaScript.

Недоліки: Вимагає додаткового часу для ознайомлення з типізацією та впровадженням її у проект.

JavaScript є основною мовою для веб-розробки і є важливим компонентом у всіх сучасних веб-програмах.

Детальний огляд основних рис JavaScript:

Веб-програмування: JavaScript є основною мовою для реалізації клієнтської логіки веб-додатків. Вона дозволяє взаємодіяти з HTML та CSS, змінювати вміст сторінки, обробляти події користувача та виконувати анімацію.

Легкість вивчення: JavaScript має простий синтаксис, що дозволяє швидко вивчити основи мови. Вона використовується для розв'язання різноманітних задач, починаючи від простих скриптів на сторінках до складних веб-додатків.

Об'єктно-орієнтована мова: JavaScript підтримує об'єктно-орієнтовану парадигму програмування, що дозволяє організовувати код у вигляді об'єктів, які містять дані та функції.

Динамічна типізація: У JavaScript не потрібно явно визначати типи змінних. Замість цього, тип змінної визначається автоматично під час виконання програми.

Події та обробники подій: JavaScript дозволяє додавати обробники подій до елементів HTML та реагувати на користувацькі події, такі як натискання кнопок, рух миші тощо.

Багатопотоковість: JavaScript виконується в одному потоці, що означає, що вона не підтримує багатопотоковість. Однак, за допомогою асинхронних функцій та об'єктів Promise, можна створювати асинхронний код, який виконується паралельно з основним потоком.

Платформонезалежність: JavaScript може виконуватися на будь-якій платформі, що підтримує веб-браузер, а також на багатьох серверних платформах, таких як Node.js.

Розвинена екосистема: У JavaScript є велика кількість бібліотек та фреймворків, які допомагають в розробці веб-додатків, таких як React.js, Angular, Vue.js тощо.

Переваги: Широка підтримка, велика кількість ресурсів та бібліотек, простота вивчення.

Недоліки: Відсутність статичної типізації може призвести до помилок на етапі розробки, менша стабільність у великих проектах.

Технології для користувацького інтерфейсу:

React – це потужна бібліотека для створення користувацьких інтерфейсів. Вона дозволяє розробляти компоненти, які легко керуються, підтримуються і перевикористовуються, що сприяє швидкому розвитку програми [18].

Детальний огляд основних рис React:

Компонентна архітектура: React базується на ідеї компонентів. Все в React є компонентами - від простих кнопок до складних веб-сторінок. Це дозволяє створювати код, який легко розуміти та підтримувати, розділяючи логіку інтерфейсу на невеликі незалежні блоки.

Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення вмісту сторінки. Замість безпосереднього маніпулювання DOM, React зберігає внутрішнє представлення сторінки у вигляді віртуального DOM та оновлює реальний DOM лише в разі потрібних змін. Це дозволяє покращити продуктивність та ефективність додатків.

JSX: JSX - це розширення синтаксису JavaScript, що дозволяє писати HTML-подібні розмітки прямо в коді React. Це полегшує розробку та читання коду, оскільки компоненти React виглядають більш декларативно та схоже на HTML.

Односторінкові додатки (SPA): React чудово підходить для створення односторінкових додатків, де весь інтерфейс відображається на одній сторінці. Він дозволяє динамічно оновлювати контент на сторінці без перезавантаження сторінки, що робить додатки більш інтерактивними та зручними для користувачів.

Реактивність: React дозволяє легко реагувати на події користувача та змінювати стан додатку відповідно до цих подій. Він забезпечує механізми для оновлення інтерфейсу користувача на основі змін стану додатку.

Широка екосистема: У React є велика кількість допоміжних бібліотек, компонентів та інструментів, які полегшують розробку веб-додатків. Деякі з них включають Redux для управління станом додатку, React Router для маршрутизації, Material-UI для готових дизайнів тощо.

Підтримка серверного рендерингу: React підтримує серверний рендеринг, що дозволяє відображати React компоненти на стороні сервера та відправляти готовий HTML клієнту. Це може покращити швидкодію завантаження сторінок та забезпечити кращий досвід користувача з точки зору SEO.

Недоліки: Потребує додаткового вивчення концепцій, таких як JSX та життєвий цикл компонентів.

Мови розмітки та стилізації:

CSS – це основний механізм для стилізації веб-сторінок. Він простий у вивченні та дозволяє створювати привабливий дизайн для користувацьких інтерфейсів.

Детальний огляд основних концепцій та функціональності CSS:

Селектори: Селектори в CSS використовуються для вибору елементів веб-сторінки, до яких будуть застосовані стилі. Наприклад, ви можете використовувати селектори тегів (p, div), класів (.class-name), ідентифікаторів (#element-id), атрибутів та інших методів вибору.

Властивості: CSS має широкий набір властивостей, які дозволяють контролювати вигляд елементів. Деякі з найпоширеніших властивостей

включають color (колір тексту), font-size (розмір шрифту), margin (відступи), padding (відступи всередині елементу), background-color (колір фону) та багато інших.

Позиціонування: CSS надає різні способи позиціонування елементів на веб-сторінці. Це може бути звичайний потік документа (normal flow), абсолютне позиціонування, фіксоване позиціонування, відносне позиціонування та інші методи.

Адаптивний та респонсивний дизайн: CSS дозволяє створювати адаптивний та респонсивний дизайн, який пристосовується до різних розмірів екрану та пристроїв. Це може бути досягнуто за допомогою медіазапитів (media queries), гнучких блоків (flexbox), сіток (grid) та інших технік.

Анімація та переходи: CSS дозволяє створювати анімацію та переходи для покращення візуального вигляду веб-сторінок. Це може бути зроблено за допомогою ключових кадрів (keyframes), трансформацій, переходів та інших ефектів

Переваги: Простота вивчення та використання, широкі можливості для стилізації, велика підтримка.

Недоліки: Проблеми з управлінням стилями у великих проектах, низька зручність перевикористання коду, можливість виникнення конфліктів.

Використання TypeScript разом з JavaScript дозволить покращити стабільність програми та продуктивність розробників. React є потужним інструментом для реалізації користувацьких інтерфейсів, який дозволяє створювати ефективні та масштабовані компоненти. CSS використовується для стилізації веб-сторінок і дозволяє створювати привабливий та функціональний дизайн.

Отже, вибір технологій для розробки програмних модулів для веб-застосунку створення Minecraft Pixel Art в TypeScript, JavaScript, React і CSS, має вирішальне значення для створення надійного і захоплюючого досвіду. Поєднання цих технологій надає необхідні інструменти та фреймворки для розробки ефективного, надійного та простого в обслуговуванні веб-застосунку.

3.2 Розробка користувацької частини

Розробляючи інтерфейс веб-застосунку вирішено зробити його максимально простим та цікавим для використання користувачем.

Основним кольором інтерфейсу обрано сірий або ж нічний гонщик, оскільки такий колір – нейтральний. Допоміжним кольором обрано чорний, аби не порушувати колірну гаму веб-застосунку.

При відкритті застосунку, користувача зустрічає екран з початком роботи по створенню піксель артів (рис. 3.1). Через це користувача не буде перенавантажено незрозумілими стоп-екранами завантаження та лишніх дій, через що він не буде почувати себе роздратованим, а одразу розпочне роботу з веб-застосунком.

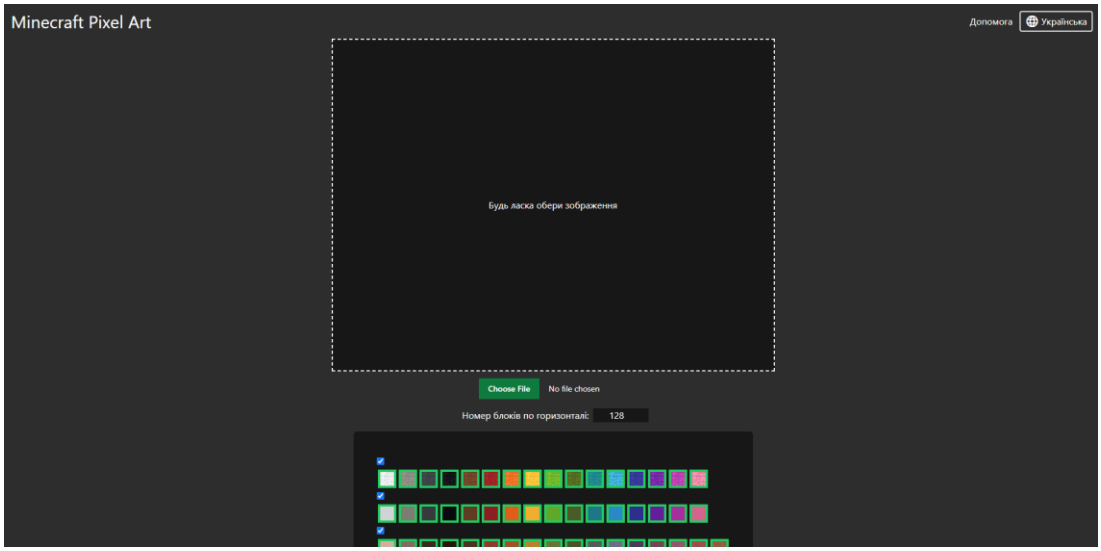


Рисунок 3.1 – Екран застосунку

Оскільки веб-застосунок передбачає вибір зображення, було спроектовано та реалізовано екран з вибором зображення користувачем (рис. 3.2). На цьому екрані користувач повинен спершу натиснути на кнопку «Обрати файл», після чого йому відкриється модальне вікно з вибором необхідного файлу. Після відкриття файлу, веб-застосунок його завантажить у власну пам’ять (рис. 3.3). Після вибору зображення, користувач зможе обрати палітру блоків для використання в генерації Minecraft Pixel Art.

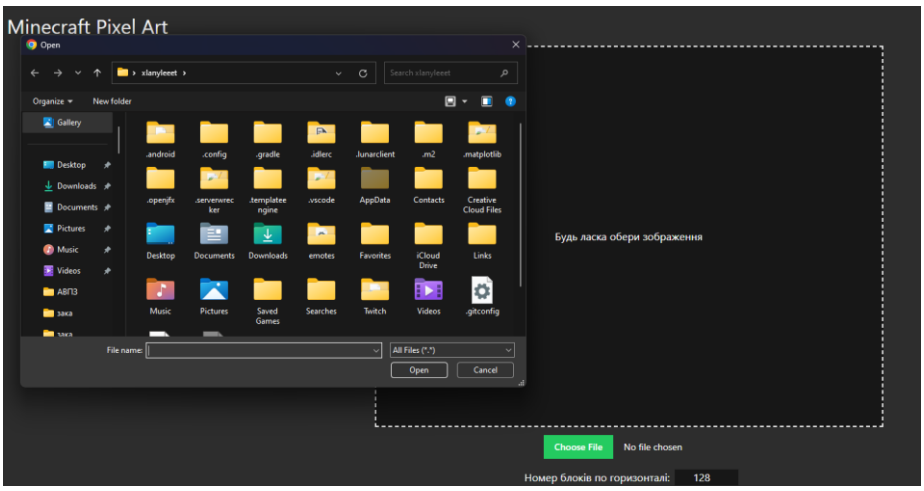


Рисунок 3.2 – Екран вибору зображення

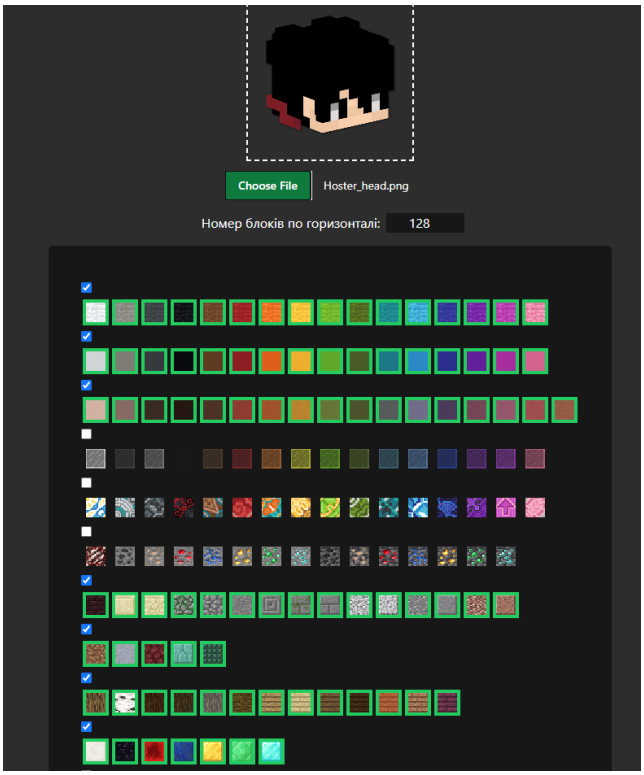


Рисунок 3.3 – Вибране зображення та палітра блоків

На екрані перетворення зображення у піксель арт з обраними блоками, користувач може побачити передперегляд створеного зображення та завантажити його, відредагувати, створити команду та завантажити csv файл (рис. 3.4). Якщо створене зображення не подобається, то можна спробувати використати різні взаємодії з зображеннями та блоками, допоки результат не сподобається.

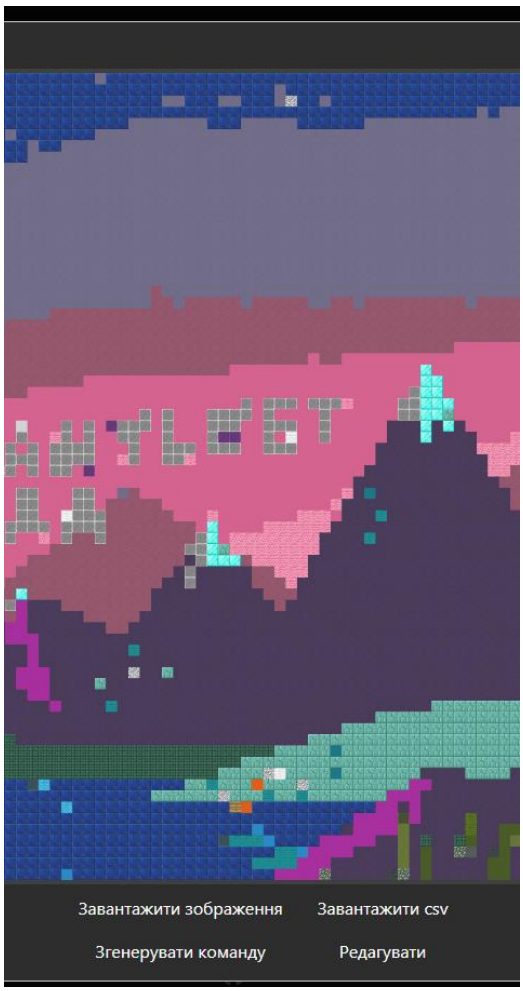


Рисунок 3.4 – Передперегляд створеного зображення

3.3 Розробка частини веб-застосунку

При натисканні на кнопку «Завантажити зображення», буде відкрито модальне вікно куди зберегти файл і буде завантажено картинку у форматі .png (рис. 3.5).

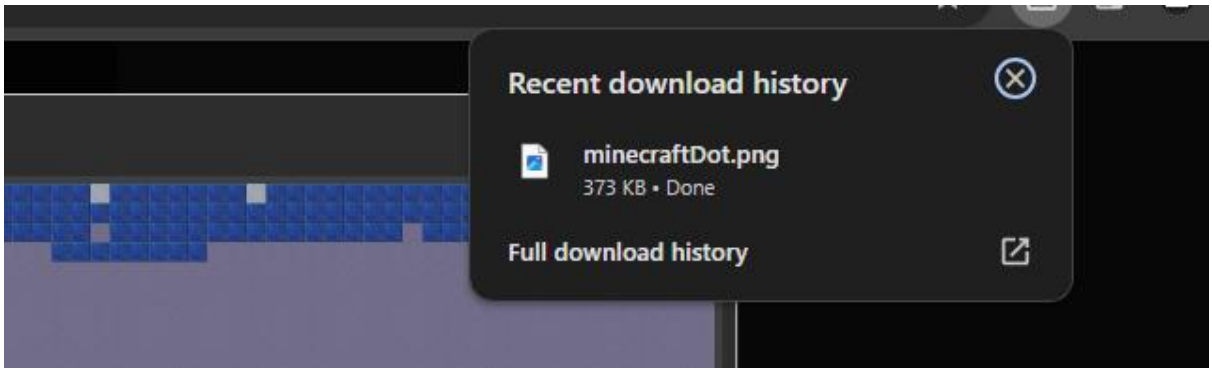


Рисунок 3.5 – Завантаження зображення

Кнопка «Завантажити csv» аналогічна до кнопки «Завантажити зображення», але користувач отримує не зображення, а csv файл, який можна відкрити за допомогою Microsoft Excel, де кожна комірка – окремий піксель (рис. 3.6).

Рисунок 3.6 – Структура csv файлу

Також внизу буде підрахунок використаних блоків у створеного зображення Minecraft Pixel Art (рис. 3.7). Це невелика загальна статистика блоків. Буде цікаво для тих, хто будує у грі Minecraft в режимі «Виживання» (для збору і підготовки блоків для створення такого піксель арту у самій грі).

block list	
minecraft:lapis_block	1326
minecraft:light_blue_terracotta	2298
minecraft:clay	5
minecraft:diorite	9
minecraft:magenta_terracotta	1200
minecraft:pink_concrete	737
minecraft:orange_stained_glass	40
minecraft:pink_wool	390
minecraft:yellow_terracotta	31
minecraft:diamond_block	36
minecraft:white_stained_glass	189
minecraft:prismarine_bricks	244
minecraft:white_concrete	5
minecraft:magenta_stained_glass	9
minecraft:blue_terracotta	1850
minecraft:white_wool	4
minecraft:magenta_wool	1
minecraft:magenta_concrete	378
minecraft:cyan_concrete	7
minecraft:cyan_wool	36
minecraft:cyan_stained_glass	5
minecraft:dark_prismarine	223
minecraft:lime_terracotta	42
minecraft:green_stained_glass	62

Рисунок 3.7 – Використання блоків

Кнопка «Згенерувати команду» містить в собі модальне вікно (рис. 3.8), де необхідно вписати координати вершин, завдяки яким потім виконвши команду у гру буде вставлений новостворений Minecraft Pixel Art. Слід зауважити, що користувачу слід ретельно підібрати координати, аби новостворений піксель арт «вбудувався» у гру правильно.

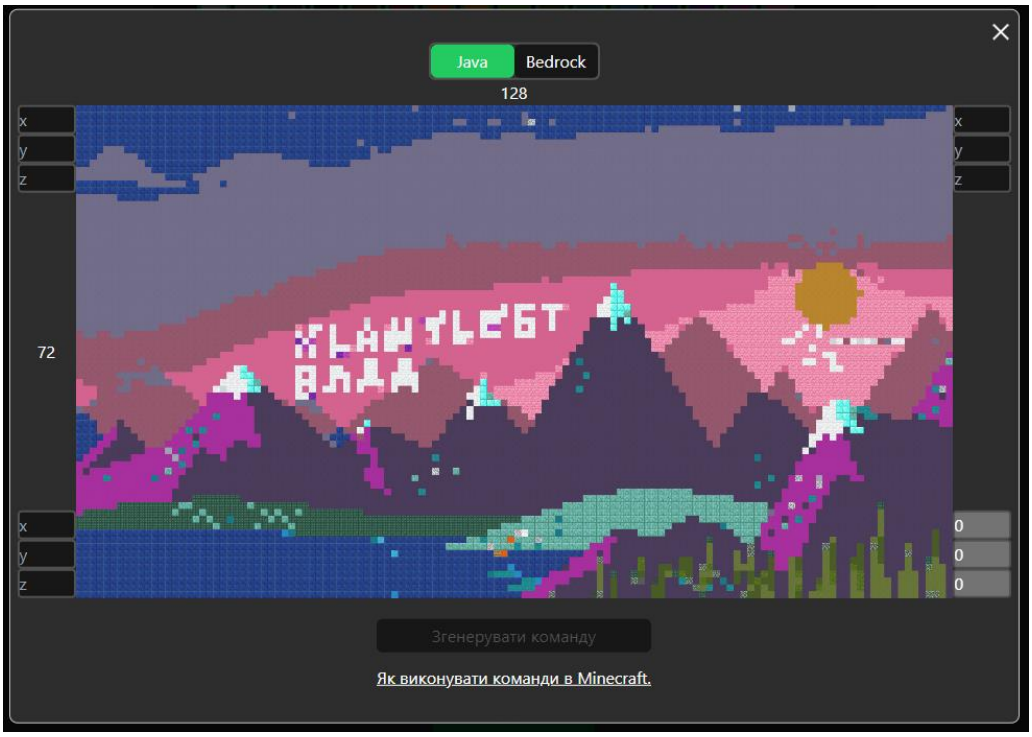


Рисунок 3.8 – Вікно генерації команди

Кнопка «Редагувати» відкриє вікно редагування новостворено зображення (рис. 3.9).

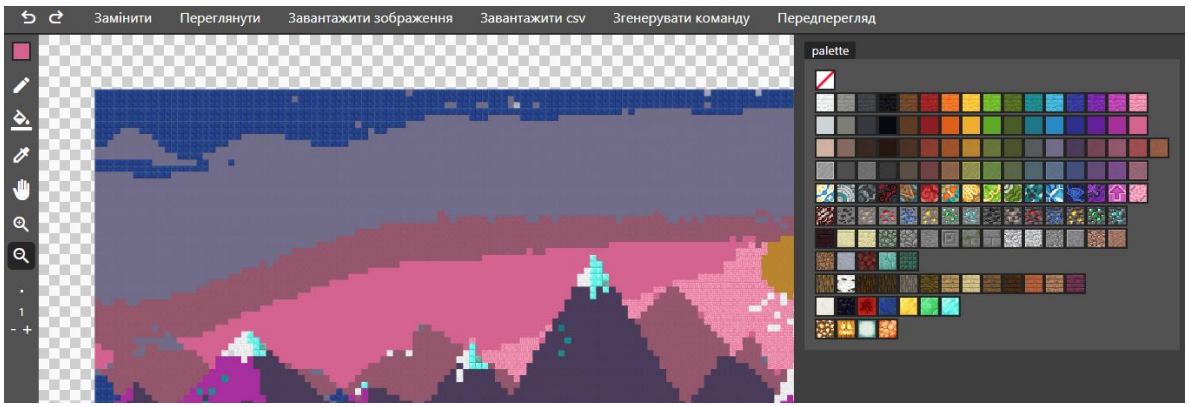


Рисунок 3.9 – Вікно редагування

Завдяки вбудованому редактору, користувач зможе сам відредагувати певні блоки чи об’єкти, що йому по певним причинам не сподобались. Саме вікно редагування містить палітру блоків, що використовувались для генерації зображення (рис. 3.10).

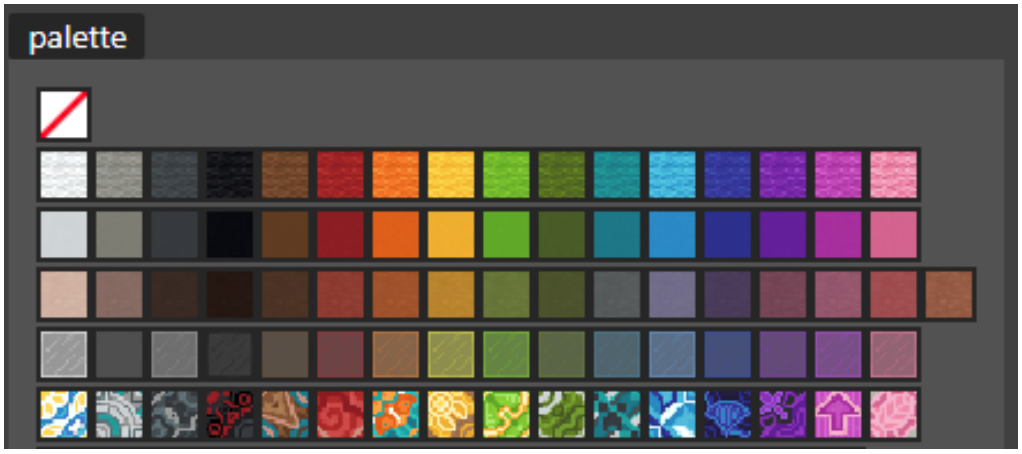


Рисунок 3.10 – Вікно генерації команди

Також наявний мінімальний набір будь-якого графічного редактора: пензлик, визначення кольору, масштабування, розмір пензлику, заливка (рис. 3.11).

Рисунок 3.11 – Панель малювання

Розробка графічного інтерфейсу веб-застосунку був проведений у середовищі розробки VS Code з використанням МП Typescript та React.

3.4 Висновки

У третьому розділі було обґрунтовано вибір мов програмування та технологій для розробки веб-застосунку. Проаналізувавши потреби програмного продукту було обрано мови програмування TypeScript та JavaScript. Для зручності розробки графічного інтерфейсу було обрано React з CSS через можливість написання графічного інтерфейсу з використанням мови TypeScript.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

Тестування веб-застосунку – процес перевірки створеного функціоналу, безпеки та коректності роботи програмного застосунку не лише на десктопних платформах Windows, Linux чи MacOS, а також на мобільних – Android чи iOS. Одним з найважливіших критеріїв успішного тестування є перевірка сумісності веб-застосунку у браузерах на різних платформах.

Функціональне тестування:

Функціональне тестування перевіряє, чи виконує веб-застосунок очікувані функції та дії. Це включає тестування різних функціональних можливостей, введення даних, обробку та виведення результатів.

Тестування інтерфейсу користувача (UI тестування):

UI тестування перевіряє, чи відповідає інтерфейс користувача веб-застосунку специфікаціям дизайну та чи працюють всі елементи інтерфейсу коректно.

Тестування взаємодії користувача (UX тестування):

UX тестування оцінює зручність використання веб-застосунку користувачами, їхні очікування та сприйняття. Це може включати тестування швидкості завантаження, навігації, зручності введення даних тощо.

Тестування сумісності:

Тестування сумісності перевіряє, як веб-застосунок працює на різних веб-браузерах, операційних системах та пристроях. Це важливо для того, щоб забезпечити, що веб-застосунок працює на всіх платформах із зручним та коректним відображенням.

Тестування продуктивності:

Тестування продуктивності оцінює швидкодію та продуктивність веб-застосунку під час роботи з ним. Це може включати тестування завантаження сторінок, часу відповіді сервера, оптимізацію ресурсів тощо.

Тестування безпеки:

Тестування безпеки оцінює вразливості веб-застосунку та виявляє можливі проблеми безпеки, такі як вразливості XSS (Cross-Site Scripting), SQL Injection, незахищені точки входу тощо.

Тестування відновлення після збоїв:

Це тип тестування перевіряє, як веб-застосунок поводиться у випадку збоїв або відмов. Відновлення після збоїв може включати автоматичне відновлення роботи після збою, відновлення даних тощо.

Тестування масштабованості:

Тестування масштабованості оцінює, як добре веб-застосунок впорається зі зростанням обсягів даних та користувачів. Це включає тестування на великих об'ємах даних та великих навантаженнях.

Інструменти розробника (Developer Tools) веб-браузера – це потужний набір інструментів, які допомагають розробникам аналізувати, відлагоджувати та тестувати веб-застосунки безпосередньо в браузері. Нижче наведено кілька способів, які можна використовувати Developer Tools для тестування веб-застосунків:

Інспектування елементів (Element Inspection):

Використовуючи інструмент інспектування елементів, можна детально аналізувати HTML та CSS код сторінки. Це допоможе перевірити, як елементи взаємодіють між собою та як вони відображаються на сторінці.

Аналіз мережевих запитів (Network Analysis):

Можна використовувати інструмент мережевого аналізу для перегляду всіх запитів, які відправляються та отримуються під час завантаження сторінки. Це допоможе виявити будь-які проблеми зі завантаженням ресурсів, такі як повільні або помилкові запити.

Відлагодження JavaScript (JavaScript Debugging):

Інструменти розробника дозволяють встановлювати точки зупинки (breakpoints) у JavaScript коді та аналізувати його в режимі відлагодження. Це дозволяє крок за кроком відслідковувати виконання коду та виявляти можливі помилки.

Тестування респонсивності (Responsive Testing):

Інструменти розробника мають можливості для тестування респонсивного дизайну, що дозволяє переглядати, як сторінка виглядає на різних розмірах екрану. Ви можете симулювати різні пристрої та розміри екранів, щоб перевірити, як веб-застосунок адаптується до них.

Тестування швидкодії (Performance Testing):

За допомогою інструментів розробника можна аналізувати швидкодію веб-застосунку, включаючи час завантаження сторінок, оптимізацію ресурсів та інші параметри продуктивності.

Перевірка безпеки (Security Checking):

Можна використовувати інструменти розробника для перевірки безпеки веб-застосунку, виявлення можливих вразливостей та забезпечення безпеки даних користувачів.

Тестування кешування та локального сховища (Cache and Local Storage Testing):

Можна перевірити, як веб-застосунок використовує кешування та локальне сховище браузера, а також як він взаємодіє з ними під час роботи.

Для початку тестування необхідно скласти проєкт у середовищі розробки (підійде будь-який що містить підтримку різних мов програмування) використовуючи десктопні чи мобільні платформи. Проєкт необхідно збирати, аби всі зміни, що вносились у нього були застосовані. Після чого проєкт необхідна запустити. Далі проводяться функціональні тести для перевірки правильності роботи веб-застосунку, що включають введення і виведення даних користувачеві, тести реакції програми на негативні формати файлів, а також тести бізнес логіки.

Потім у веб-застосунку проводяться тести на безпеку, включно з тестами на проникнення, конфіденційність і злом. Для цього можуть використовуватися спеціальні програмні інструменти, що дають змогу імітувати атаки на застосунок і виявляти його вразливості.

Після успішного завершення всіх тестів оцінюється продуктивність застосунку, включно з тестами швидкості роботи застосунку, тестами використання ресурсів мобільного пристрою та тестами життєвого циклу застосунку.

Усі результати тестування фіксуються у відповідній документації та використовуються для подальшого вдосконалення застосунку та забезпечення якості його роботи на всіх програмних платформах [19].

Результат тестування зображено на рисунку 4.1.

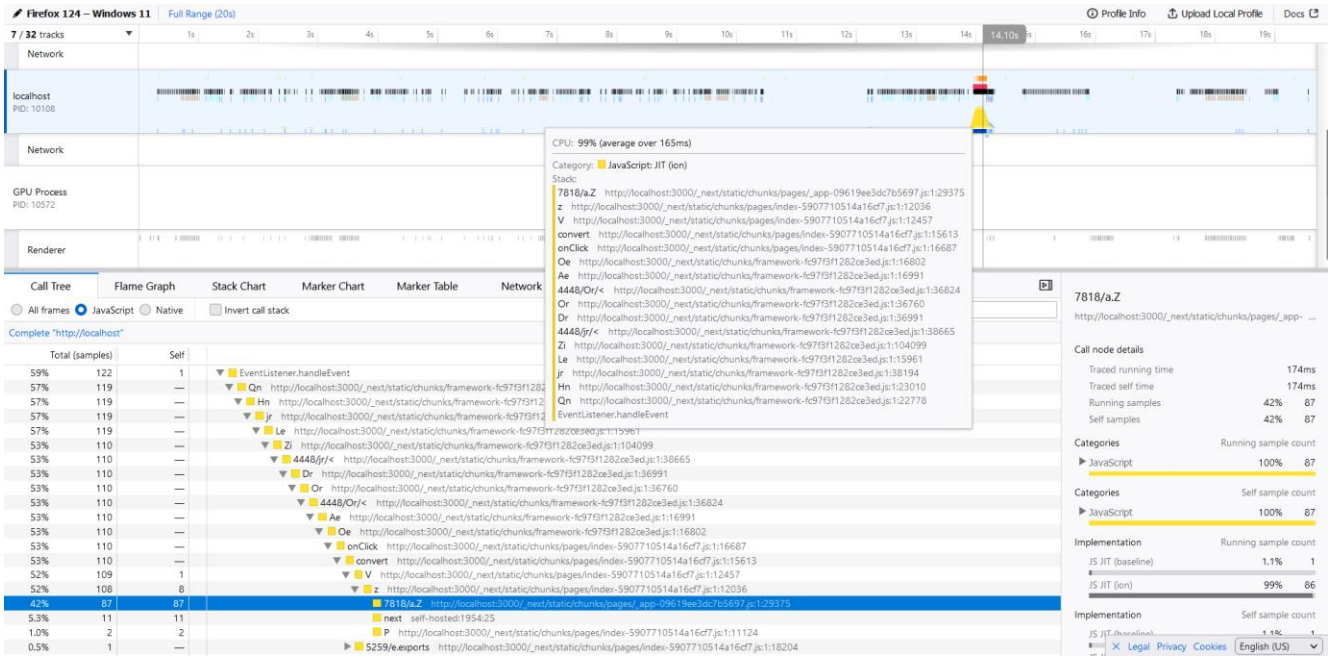


Рисунок 4.1 – Тестування продуктивності веб-застосунку

4.2 Тестування валідації даних форм

Наступним кроком буде перевірка головного екрану, що відображається користувачеві, спробувати згенерувати піксель арт без обрання користувачем зображення. У разі спроби згенерувати піксель арт без зображення користувач просто побачить пустий екран. Таким чином веб-застосунок повідомляє про помилку, і користувач змушений буде повернутись назад і обрати потрібне зображення. Результат тестування зображено на рисунку 4.2.

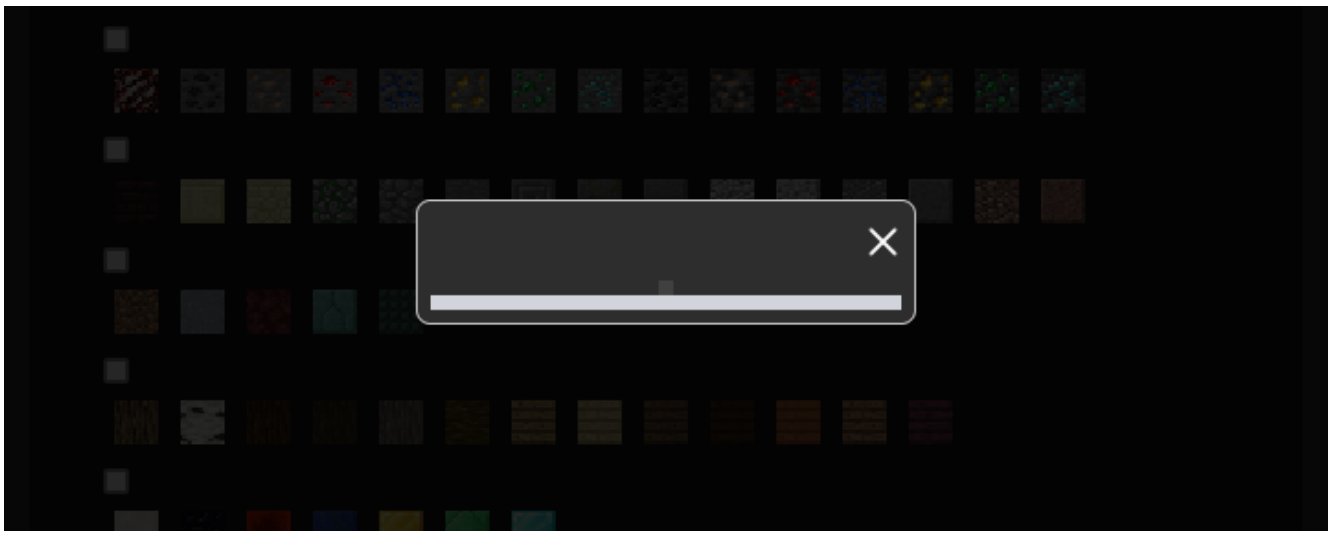


Рисунок 4.2 – Створення піксель арту без зображення

Обравши зображення, але не вибравши ніяку палітру кольорів блоків, веб застосунок відкриє просто пусте вікно з функціональними кнопками. Результат тестування можна побачити на рисунку 4.3.

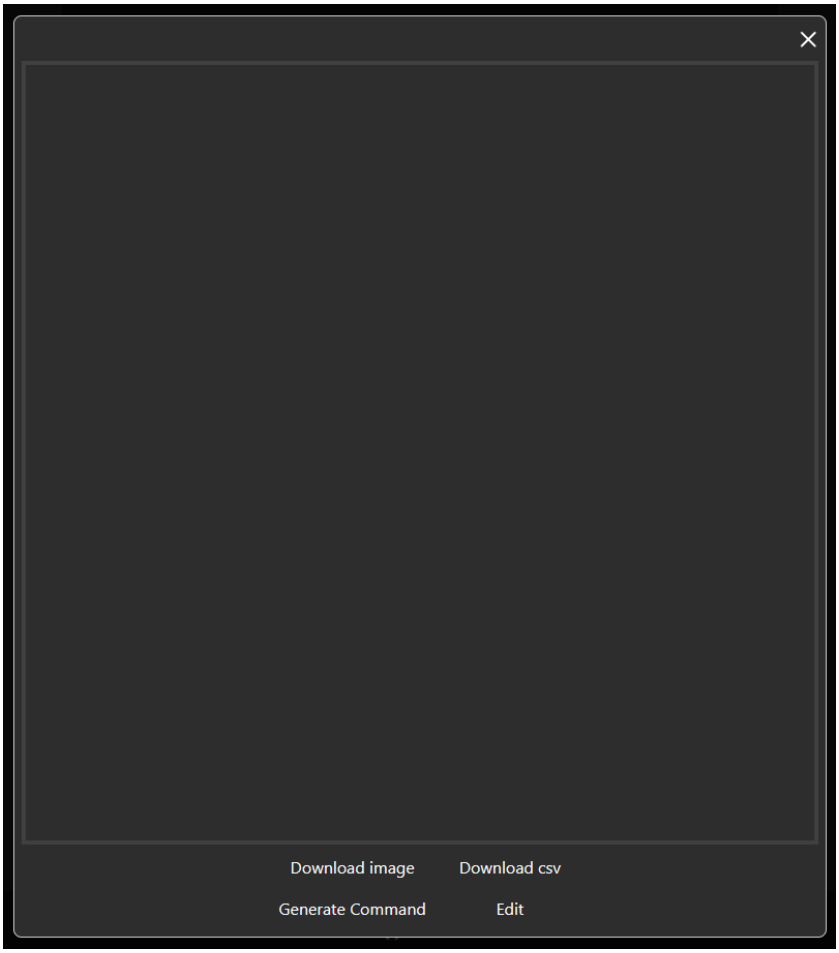


Рисунок 4.3 – Створення піксель арту без палітри

4.3 Розробка інструкції користувача

Створення інструкції користувача для програмного забезпечення є важливою частиною процесу розробки, оскільки вона допомагає користувачам зрозуміти, як працювати з програмою та вирішувати потенційні проблеми. Нижче наведено кілька кроків, які можуть допомогти створити ефективну інструкцію користувача:

Визначення аудиторії:

Розуміння, хто буде користуватися програмним забезпеченням, допоможе вам створити інструкцію, яка відповідає їхнім потребам та рівню експертизи. Наприклад, інструкція для початківців може бути менш технічною, ніж для досвідчених користувачів.

Структурування інструкції:

Розділення інструкції на логічні розділи та підрозділи, щоб користувачі могли швидко знаходити необхідну інформацію. Наприклад, розділи можуть бути "Початок роботи", "Основні функції", "Налаштування" та "Відомі проблеми та рішення".

Просте та зрозуміле мовлення:

Потрібно використовувати простий та зрозумілий мовний стиль, уникати складних технічних термінів, які можуть бути незрозумілими для аудиторії.

Використання графіків та зображень:

Вставляти скріншоти екрану, діаграми та ілюстрації, щоб візуалізувати кроки та процеси, що описуються в інструкції.

Крок-за-кроком інструкції:

Проводити користувача через кроки з одного етапу до іншого, надаючи точні інструкції та вказівки для кожного кроку. Використовувати нумерацію кроків для легкого відслідковування.

Налаштування та управління проблемами:

Надати інформацію про те, як користувачі можуть налаштувати програмне забезпечення для своїх потреб та як вони можуть вирішити проблеми, які можуть виникнути під час використання програми.

Перевірка та оновлення:

Періодично перевіряти та оновлювати інструкцію, особливо після випуску нових версій програмного забезпечення.

Наприклад, якщо розробляється інструкція для веб-додатка, можна включити розділи про вхід в систему, створення нового запису, редагування даних, виходу з системи та контактні дані для підтримки користувачів.

Першим кроком є ознайомлення з сайтом або ж з початковою сторінкою роботи (рис. 4.4).

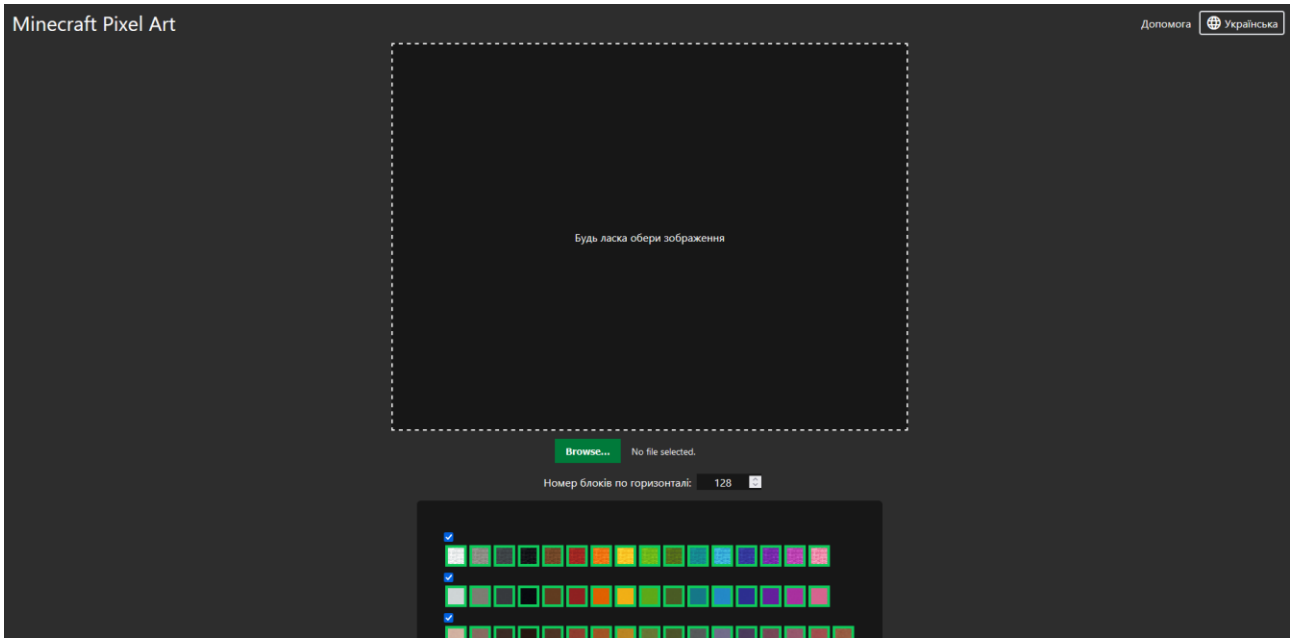


Рисунок 4.4 – Зображення головного екрану

Для подальшої роботи користувача з системою йому необхідно обрати зображення, яке буде використовуватись для створення піксель арту (див. рис. 4.5). Варто звернути увагу що, сильно грає роль розмір зображення у пікселях, не потрібно брати занадто малі чи великі зображення, адже це може дуже сильно вплинути на швидкодію вашої системи.

Коли зображення було обрано, далі необхідно вибрати кількість блоків по горизонталі та палітру блоків, що буде використовуватись при створенні нового піксель арту (див. рис. 4.6). Примітка: палітра блоків може бути обрана одна або

декілька, це залежить від дій користувача і що він в кінцевому результаті хоче побачити.

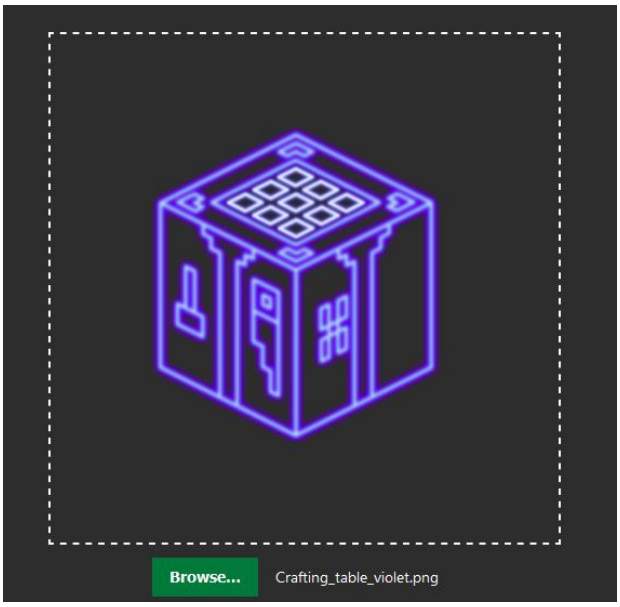


Рисунок 4.5 – Обране користувачем зображення

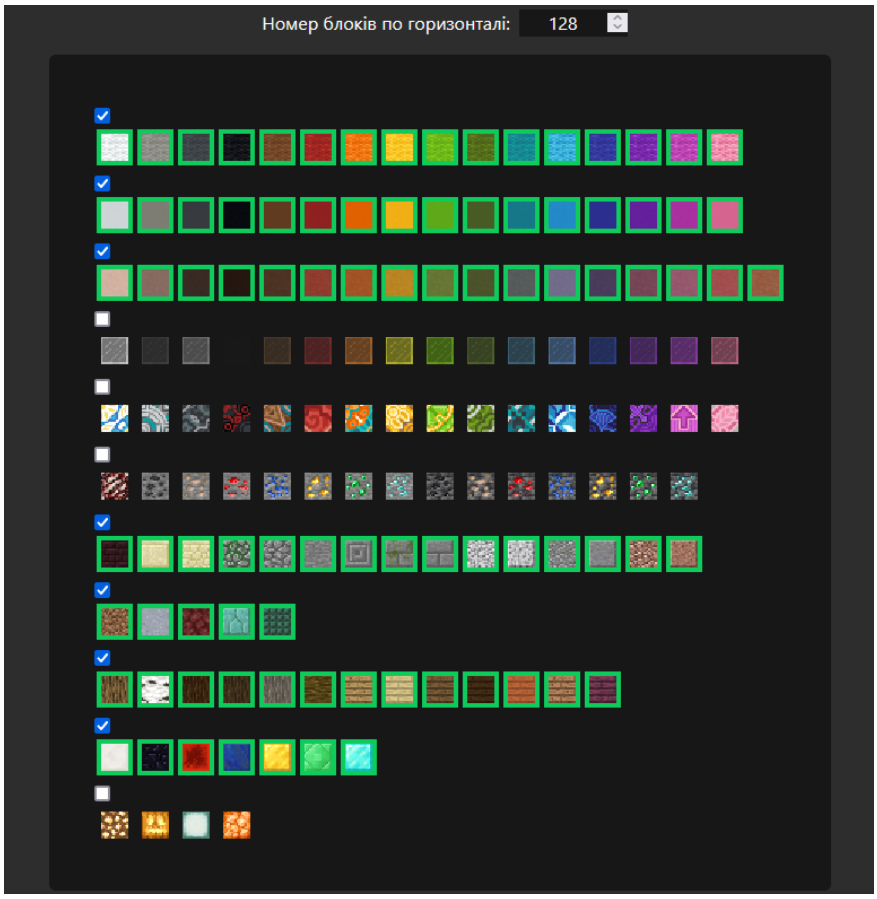


Рисунок 4.6 – Обрана кількість блоків та палітра

Після натискання кнопки конвертації користувач зможе побачити вікно (див. рис. 4.7) уже згенерованого зображення з різним набором функцій.

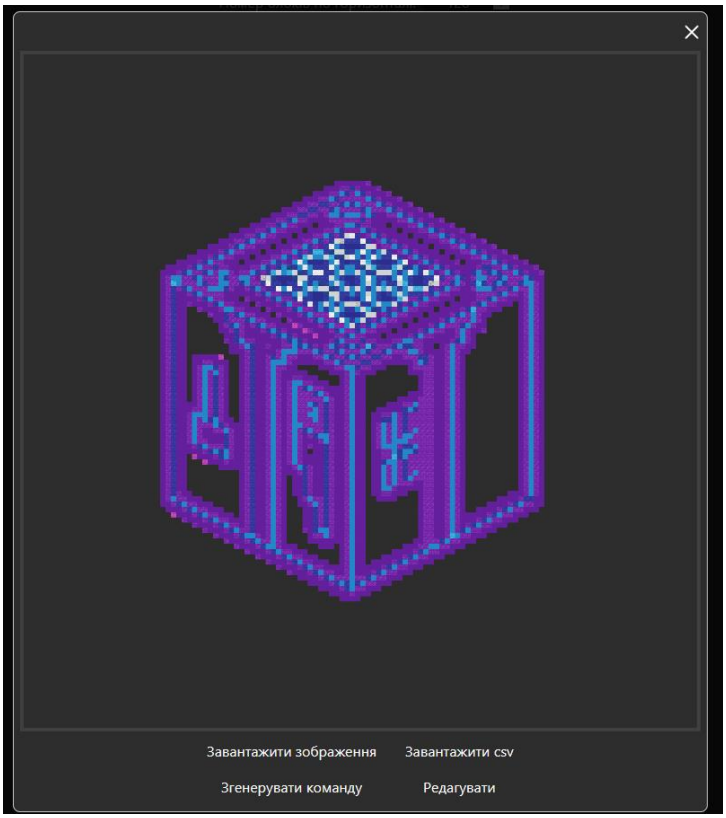


Рисунок 4.7 – Результат конвертування

При натисканні користувачем кнопки «Завантажити зображення», система відравить запит браузеру на зберігання новоствореного піксель арту (рис. 4.8).

Рисунок 4.8 – Завантажене зображення веб-застосунком

Кнопка «Завантажити csv» (корисно для збору статистичних даних і реалізації власної програмної компоненти) дасть запит системі згенерувати файл, що буде

містити використану кількість різних блоків та заповнені комірки в точності як на новоствореному зображенні (рис. 4.9).

block list	
minecraft:air	12528
minecraft:purple_concrete	1480
minecraft:purple_wool	1219
minecraft:blue_wool	460
minecraft:light_blue_concrete	531
minecraft:white_concrete	54
minecraft:white_wool	29
minecraft:polished_diorite	4
minecraft:blue_concrete	34
minecraft:light_blue_wool	22
minecraft:lapis_block	16
minecraft:magenta_wool	7

Рисунок 4.9 – Кількість використаних блоків

Кнопка «Згенерувати команду» надішле запит веб-застосунку для створення команди, яка в подальшому може бути використана у грі для вставки нового піксель арту (рис. 4.10).

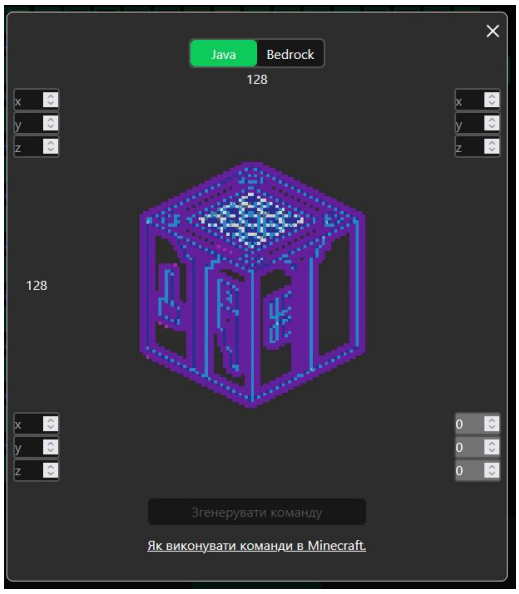


Рисунок 4.10 – Створення команди

Кнопка «Редагувати» дозволяє відкрити редактор новоствореного зображення

аби підправити або ж перемалювати сворений піксель арт користувачеві (рис. 4.11).

Рисунок 4.11 – Редактор зображення

4.4 Системні вимоги

Мінімальна та рекомендована конфігурація для запуску програмного забезпечення (ПЗ) визначається залежно від вимог самого ПЗ, його функціональності та навантаження на систему.

Мінімальна конфігурація:

Мінімальна конфігурація – це найнижча можлива конфігурація, яка необхідна для запуску програмного забезпечення без суттєвих проблем з продуктивністю та функціональністю. Вона може бути визначена розробниками програмного забезпечення і може включати мінімальні вимоги щодо обсягу оперативної пам'яті, швидкодії процесора, версії операційної системи тощо. Зазвичай мінімальна конфігурація дозволяє запустити ПЗ, але може призвести до обмеженого функціоналу або повільної роботи.

Рекомендована конфігурація:

Рекомендована конфігурація – це оптимальна конфігурація, яка забезпечує найкращу продуктивність та функціональність програмного забезпечення. Вона може включати високопродуктивні компоненти, такі як швидкий процесор, достатньо оперативної пам'яті, достатньо простору на диску та відповідну версію операційної системи. Рекомендована конфігурація забезпечує найкращий досвід користувача та забезпечує оптимальну продуктивність програми.

У посібнику користувача визначено технічні вимоги до роботи програмного продукту. Детальна інформація про мінімальну та рекомендовану конфігурацію серверного комп'ютера наведена в таблицях 4.1 і 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	AMD Ryzen 3 1300
Об’єм оперативної пам’яті	2 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Windows, Linux

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	AMD Ryzen 5 3600
Об’єм оперативної пам’яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	4+ ГБ
Операційна система	Windows, Linux

Після обрання потрібної конфігурації, користувач має встановити особисте або ж серверне програмне забезпечення для збірки та запуску проєкту. Для цього йому необхідно встановити Node.js [20] та Visual Studio Code [21]. Після встановлення та налаштування цих програм користувачеві необхідно встановити залежності. Залежності проекту – це компоненти або бібліотеки, які використовуються у вашому проекті для вирішення різних завдань або функцій. Це можуть бути сторонні бібліотеки, модулі або інші ресурси, які використовуються для розробки, тестування або розгортання програмного забезпечення. Після встановлення залежностей треба зібрати за запустити веб-застосунок. Після запуску веб-застосунку користувачеві буде надано адресу, за якою необхідно перейти, аби почати роботу (рис. 4.3). Коли перехід на сайт було виконано, користувача зустрічає домашня сторінка, де для створення Minecraft Pixel Art необхідно обрати зображення та палітру кольорів блоків і натиснути

кнопку конвертувати. Після обробки зображення користувач побачить вікно передперегляду створеного піксель арту і зможе з ним взаємодіяти. Коли зображення чи палітра блоків не була вибрана, користувач не зможе згенерувати власний піксель арт.

4.5 Висновки

У четвертому розділі було проведено тестування програмних модулів веб-застосунку Minecraft Pixel Art. Було перевірено швидкість виконання генерації зображення, валідацію полів введення інформації. Також було розроблено інструкцію користувача по встановленню та початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській дипломній роботі було проведено дослідження програмних засобів створення піксель арту з використанням палітри блоків гри Minecraft, у результаті якого створено програмне забезпечення, що виконує роботу по створенню Minecraft Pixel Art.

Основні результати роботи такі:

1. У першому розділі було розглянуто стан програмних засобів Minecraft Pixel Art. Було проведено порівняння відомих платформ для перетворення зображень в піксель арт з використанням блоків з гри Minecraft, таких як: mcstacker murals, spritecraft, Pixel art maker for Minecraft, minecraftart. У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності серед учасників по створенню піксель артів завдяки своїй зручності та доступності. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що вони не надають можливості навігації та мультиплатформеності. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного веб-застосунку.

2. У другому розділі було проведено аналіз вхідних та вихідних даних програмного продукту. Також було розглянуто можливість покращення продуктивності веб-застосунку шляхом кешування вхідних даних. Було розроблено основний алгоритм роботи веб-застосунку та алгоритм перетворення зображення. Також було розроблену модель роботи програмного продукту за допомогою UML діаграм.

3. У третьому розділі було обґрунтовано вибір мов програмування та технологій для розробки веб-застосунку. Проаналізувавши потреби програмного продукту було обрано мови програмування TypeScript та JavaScript. Для зручності розробки графічного інтерфейсу було обрано React з CSS через можливість написання графічного інтерфейсу з використанням мови TypeScript.

4. У четвертому розділі було проведено тестування програмних модулів веб-застосунку Minecraft Pixel Art. Було перевірено швидкість виконання генерації зображення, валідацію полів введення інформації. Також було розроблено інструкцію користувача по встановленню та початковій експлуатації програмного продукту.

Таким чином можна дійти до висновку, що поставлені задачі та цілі бакалаврської дипломної роботи виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pixel Art Tutorial: How to Make Pixel Art by Adobe Creative Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/eg_en/creativecloud/design/discover/pixel-art.html
2. Make It Pixel! - Make pixel art from any image by dev.to [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/miguelmj/make-it-pixel-make-pixel-art-from-any-image-2o4n
3. Історія браузерів: легендарна боротьба за владу, яка принесла нам сучасні браузери by Mozilla [Електронний ресурс] – Режим доступу до ресурсу: https://www.mozilla.org/uk/firefox/browsers/browser-history/
4. Грицишин В.О. Майданюк В.П. Покращення технологій програмного забезпечення для ефективного створення піксельних зображень / Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21722
5. Learn How to Make Pixel Art: Tutorial with Tips & Tools | Adobe [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/creativecloud/design/discover/pixel-art.html
6. How did you guy's learn pixel art? [Електронний ресурс] – Режим доступу до ресурсу: https://www.reddit.com/r/gamedev/comments/13338yu/how_did_you_guys_learn_pixel_art/
7. Minecraft Image Converter. Convert any image to pixel art using Minecraft blocks! [Електронний ресурс] – Режим доступу до ресурсу: https://minecraftart.netlify.app/
8. Pixel art maker for Minecraft [Електронний ресурс] – Режим доступу до ресурсу:https://play.google.com/store/apps/details?id=com.bikaba.pixelartbuilder4minecraft&hl=uk

9. Spritecraft can quickly and easily convert any image into Minecraft pixel art. Supports nearly all Minecraft blocks. [Електронний ресурс] – Режим доступу до ресурсу: https://autosaved.org/spritecraft
10. Minecraft Pixel Art Mural Generator. [Електронний ресурс] – Режим доступу до ресурсу: https://mcstacker.net/murals/
11. Creating your own modules | ITS Advanced Research Computing. [Електронний ресурс] – Режим доступу до ресурсу: https://arc.umich.edu/creating-modules/
12. How To Write Large Programs. Techniques for programming in the… | by Oleg Alexander | Medium [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@olegalexander/how-to-write-large-programs-628c90a70615
13. Stefan Baumgartner. TypeScript in 50 Lessons /Уклад. TypeScript за 50 уроків був написаний Стефаном Баумгартнером (Stefan Baumgartner) та рецензовано Шоном Вангом – Німеччина: 2020. – 424 с.
14. TypeScript це JavaScript на стероїдах | Друкарня [Електронний ресурс] – Режим доступу до ресурсу: https://drukarnia.com.ua/articles/typescript-ce-javascript-na-steroyidakh-uAxV2
15. Data Structures and Algorithms with JavaScript: Bringing classic computing approaches to the Web /Уклад. O'Reilly Media. Норвегія: 2014. – 370 с.
16. Що таке JavaScript і для чого він потрібен – GoIT Global Електронний ресурс] – Режим доступу до ресурсу: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/
17. What actually is CSS-in-JS?. CSS-in-JS refers to a collection of… | by Oleg Isonen | DailyJS | Medium [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/dailyjs/what-is-actually-css-in-js-f2f529a2757
18. Підручник з Umbrello UML Modeller. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html
19. React. [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/
20. Tools and testing - Learn web development | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing

21. Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
22. Documentation for Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs

ДОДАТКИ

ДОДАТОК А

(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Методи та програмні засоби створення Minecraft Pixel Art» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент В.П. Майданюк
«12» березня 2024 р.
Виконав:
_____ студент гр. 4ПІ-20б В.О. Грицишин
«12» березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Методи та програмні засоби створення Minecraft Pixel Art».

Галузь застосування – веб системи.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від 11березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є підвищення зручності та швидкості формування Minecraft Pixel Art з використанням додаткових інструментів щодо імпорту та редагування.

Основними задачами дослідження є:

- провести аналіз стану питання та методи розв’язання задачі;
- спроєктувати модель даних та структуру програмних засобів;
- розробити загальний алгоритм роботи веб-застосунку;
- розробити алгоритм обробки зображень;
- виконати програмну реалізацію головного екрану застосунку;
- запровадити взаємодію між клієнтом та системою веб-застосунку;
- провести тестування програмних засобів;
- розробити інструкцію користувача та описати системні вимоги для програмних засобів.

Призначення роботи – розробка та реалізація програмних засобів створення Minecraft Pixel Art.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. FF DOT: The Pixel Art of Final Fantasy, Square Enix 2020. 280 с.

2. Романюк О.Н. Романюк О.В., Чехместрук Р. Ю. Комп’ютерна графіка : ВНТУ, 2022. 73 с.

3. Learn How to Make Pixel Art: Tutorial with Tips & Tools | Adobe [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/creativecloud/design/discover/pixel-art.html

4. How did you guy's learn pixel art? [Електронний ресурс] – Режим доступу до ресурсу: https://www.reddit.com/r/gamedev/comments/13338yu/how_did_you_guys_learn_pixel_art/

5. Технічні вимоги

Тип програми – веб система; модель розробки – ітеративна; засоби організації даних програми – фреймворк React; вхідні дані: зображення користувача; вихідні дані: видозмінене піксельне зображення; середовище розробки – Visual Studio Code; мова програмування – TypeScript.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз підходів до створення Minecraft Pixel Art	14.03.24 – 21.03.24
2	Розробка методів створення Minecraft Pixel Art	11.03.24 – 22.03.24
3	Розробка алгоритмів роботи програмних методів	26.03.24 – 20.04.24
4	Розробка програмних методів	20.04.24 – 10.05.24
5	Тестування програмних методів	12.05.24 – 25.05.24
6	Оформлення матеріалів до захисту БДР	26.05.24 – 30.05.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

61

ДОДАТОК Б

(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Методи та програмні засоби створення Minecraft Pixel Art

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Майданюк В.П.

Оригінальність	**86,1%**
Схожість	13,9%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Грицишин В. О.

Керівник роботи Майданюк В.П.

ДОДАТОК В

(обов’язковий)

ЛІСТИНГ ПРОГРАМИ

```
/* eslint-disable @next/next/no-img-element */

interface Props {
  checked: boolean;
  blockBasic: BlockBasic;
  handleBlockClick: (javaId: string) => void;
}

const BlockButton: React.FC<Props> = ({ checked, blockBasic, handleBlockClick })
=> {
  return (
    <>
      <label htmlFor={blockBasic.javaId} className="" key={blockBasic.javaId +
"label"}>
        <input
          type="radio"
          id={blockBasic.javaId}
          onChange={() => {}}
          checked={checked}
          className="hidden peer"
        />

        <img
          className="inline cursor-pointer rendering-pixelated m-[2px] p-[0px]
sm:border-[4px]
          border-2 border-transparent peer-checked:border-m-green-light"
          src={blockBasic.imagePath}
          alt="paletteBlock"
          width={32}
          key={blockBasic.javaId}
          onClick={() => handleBlockClick(blockBasic.javaId)}
        ></img>
      </label>
    </>
  );
};

export default BlockButton;
import { useLocale } from "src/hooks/useLocale";
```

```
interface Props {
  blockBasic: BlockBasic;
}

const BlockNameLabel: React.FC<Props> = ({ blockBasic }) => {
  const { locale } = useLocale();
  const Ename = blockBasic.javaId.split(":")[1].replace("_", " ");

  const blockName = locale === "en" ? Ename : blockBasic.jname;

  if (blockBasic === undefined || BlockNameLabel === null) return <></>;
  return (
    <>
      <div className="bg-black text-white p-1">{blockName}</div>
    </>
  );
};

export default BlockNameLabel;
import { ReactNode } from "react";

interface Props {
  onChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  placeholder: string;
}

const CoordinateInput: React.FC<Props> = ({ onChange, placeholder }) => {
  return (
    <>
      <input
        placeholder={placeholder}
        type="number"
        onChange={onChange}
        className="border-2 border-neutral-600 bg-neutral-900 rounded w-14"
      />
    </>
  );
};

export default CoordinateInput;
import { ReactNode } from "react";

interface Props {
  selected: boolean;
```

```
  id: string;
  children: ReactNode;
  onChange: () => void;
}

const EditionButton: React.FC<Props> = ({ onChange, selected, id, children }) => {
  return (
    <>
      <label htmlFor={id} className="">
        <input type="radio" id={id} onChange={() => onChange()}
checked={selected} className="hidden peer" />
        <div
          className="w-20 h-8 justify-center flex items-center rounded-md
                bg-neutral-900 cursor-pointer peer-checked:bg-m-green-light"
        >
          {children}
        </div>
      </label>
    </>
  );
};

export default EditionButton;
import { useEffect, useRef, useState } from "react";
import geenrateImageFromBlueprint from
"src/Feature/Home/functions/generateImageFromBlueprint";
import { useBlockDBContext } from "src/context/useBlockDB";
import Modal from "react-modal";
import { useBlueprintContext } from "src/context/useBlueprint";
import getContext from "src/functions/getContext";
import getBufferCanvas from "src/functions/getBufferCanvas";
import CrossButton from "src/components/CrossButton.tsx";
import { calcRBCoordinate, validInput } from
"src/Feature/Editor/functions/cornerCoordinate";
import generateCommand from "src/Feature/Editor/functions/generateCommand";
import generatePackZip from "../../Feature/Editor/functions/generatePackZip";
import { saveAs } from "file-saver";
import EditionButton from "./EditionButton";
import CoordinateInput from "./CoordinateInput";
import { useLocale } from "src/hooks/useLocale";
import Link from "next/link";

interface Props {
  blueprint: string[][];
  isModalOpen: boolean;
```

```
  setIsModalOpen: (isModalOpen: boolean) => void;
}

interface CornerCoordinate {
  ltx: number;
  lty: number;
  ltz: number;
  rtx: number;
  rty: number;
  rtz: number;
  lbx: number;
  lby: number;
  lbz: number;
}
const modalStyles = {
  overlay: {
    backgroundColor: "rgba(0, 0, 0, 0.85)",
  },
  content: {
    top: "50%",
    left: "50%",
    right: "auto",
    bottom: "auto",
    padding: "7px",
    marginRight: "-50%",
    backgroundColor: "#2d2d2d",
    borderRadius: "8px",
    transform: "translate(-50%, -50%)",
  },
};

const CommandModal = ({ blueprint, isModalOpen, setIsModalOpen }: Props) => {
  const { t } = useLocale();
  const [isCommandReady, setIsCommandReady] = useState(false);
  const [editon, setEdition] = useState<Edition>("java");
  const [RBCoordinate, setRBCoordinate] = useState<Coordinate3D>({ x: 0, y: 0, z:
0 });
  const [cornerCoordinates, setCornerCoordinates] = useState<CornerCoordinate>({
    ltx: 0,
    lty: 0,
    ltz: 0,
    rtx: 0,
    rty: 0,
    rtz: 0,
    lbx: 0,
```

```
    lby: 0,
    lbz: 0,
  });
  // const [coodinateIsOk, setCoordinateIsOk] = useState(false);
  const { setInitBlueprint } = useBlueprintContext();
  const { blockImageDataDict, javaId2index, blockDB } = useBlockDBContext();
  const mainCanvas = useRef(null);

  const tryComplement = (coordinate: CornerCoordinate, expectedWidth: number,
expectedHeight: number) => {
    setIsCommandReady(false);
    const c1: Coordinate3D = { x: coordinate.ltx, y: coordinate.lty, z: coordinate.ltz };
    const c2: Coordinate3D = { x: coordinate.rtx, y: coordinate.rty, z: coordinate.rtz };
    const c3: Coordinate3D = { x: coordinate.lbx, y: coordinate.lby, z: coordinate.lbz
};

    if (validInput(c1, c2, c3, expectedWidth, expectedHeight)) {
      /* if coordinate is valid, show RB coordinate */
      setRBCoordinate(calcRBCoordinate(c1, c2, c3));
      setIsCommandReady(true);
    }
  };

  const closeModal = () => {
    setIsModalOpen(false);
  };

  const handleChangeCoordinate = (key: string, value: number) => {
    setCornerCoordinates({ ...cornerCoordinates, [key]: value });
  };

  useEffect(() => {
    if (blueprint === undefined || blueprint.length === 0) return;
    tryComplement(cornerCoordinates, blueprint[0].length, blueprint.length);
  }, [cornerCoordinates]);

  const afterOpenModal = () => {
    const minecraftImage = geenrateImageFromBlueprint(blueprint,
blockImageDataDict);

    /* set minecraftImage to buffer canvs */
    const bufferCanvas = getBufferCanvas(minecraftImage, minecraftImage.width,
minecraftImage.height);

    /* output image */
```

```
    const mainCanvasContext = getContext(mainCanvas.current,
minecraftImage.width, minecraftImage.height);
    mainCanvasContext.drawImage(bufferCanvas, 0, 0, minecraftImage.width,
minecraftImage.height);
    setInitBlueprint(blueprint);
  };

  const handleGenerateCommand = () => {
    const LT = { x: cornerCoordinates.ltx, y: cornerCoordinates.lty, z:
cornerCoordinates.ltz };
    const RT = { x: cornerCoordinates.rtx, y: cornerCoordinates.rty, z:
cornerCoordinates.rtz };
    const LB = { x: cornerCoordinates.lbx, y: cornerCoordinates.lby, z:
cornerCoordinates.lbz };

    const packZip = generatePackZip(
      generateCommand(blueprint, javaId2index, blockDB, editon, LT, RT, LB),
      editon
    );

    // zip download
    packZip.generateAsync({ type: "blob" }).then(
      function (blob: Blob) {
        saveAs(blob, "minecraftDot.zip");
      },
      function (err: any) {
        console.log("zip download error", err);
      }
    );
  };

  if (blueprint === undefined || blueprint.length === 0) return <></>;
  return (
    <>
      <Modal isOpen={isModalOpen} ariaHideApp={false}
onAfterOpen={afterOpenModal} style={modalStyles}>
        <div className="flex justify-end">
          <button onClick={closeModal}>
            <CrossButton />
          </button>
        </div>

        <div className="flex justify-center">
          <div className="w-fit border-2 border-neutral-600 rounded-md flex justify-
center">
```

```
          <div className="">
            <EditionButton id="javaButton" selected={editon === "java"}
onChange={() => setEdition("java")}>
              {t.JAVA_EDITION}
            </EditionButton>
          </div>
          <div className="">
            <EditionButton
              id="bedrockButton"
              selected={editon === "bedrock"}
              onChange={() => setEdition("bedrock")}
            >
              {t.BEDROCK_EDITION}
            </EditionButton>
          </div>
        </div>
      </div>

      <div className="flex justify-center">{blueprint[0].length}</div>

      <div className="flex justify-between">
        <div className="flex flex-col justify-between items-center">
          <div className="flex flex-col">
            <CoordinateInput
              onChange={(e) => handleChangeCoordinate("ltx",
Number(e.target.value))}
              placeholder="x"
            ></CoordinateInput>
            <CoordinateInput
              placeholder="y"
              onChange={(e) => handleChangeCoordinate("lty",
Number(e.target.value))}
            ></CoordinateInput>
            <CoordinateInput
              placeholder="z"
              onChange={(e) => handleChangeCoordinate("ltz",
Number(e.target.value))}
            ></CoordinateInput>
          </div>
          <div>{blueprint.length}</div>
          <div className="flex flex-col">
            <CoordinateInput
              placeholder="x"
              onChange={(e) => handleChangeCoordinate("lbx",
Number(e.target.value))}
```

```
            ></CoordinateInput>
            <CoordinateInput
              placeholder="y"
              onChange={(e) => handleChangeCoordinate("lby",
Number(e.target.value))}
            ></CoordinateInput>
            <CoordinateInput
              placeholder="z"
              onChange={(e) => handleChangeCoordinate("lbz",
Number(e.target.value))}
            ></CoordinateInput>
          </div>
        </div>

        <div className="flex justify-center">
          <canvas
            id="canvas-out"
            ref={mainCanvas}
            className="h-auto w-auto max-h-[50vh] max-w-[50vw]"
          ></canvas>
        </div>

        <div className="flex flex-col justify-between">
          <div className="flex flex-col">
            <CoordinateInput
              placeholder="x"
              onChange={(e) => handleChangeCoordinate("rtx",
Number(e.target.value))}
            ></CoordinateInput>
            <CoordinateInput
              placeholder="y"
              onChange={(e) => handleChangeCoordinate("rty",
Number(e.target.value))}
            ></CoordinateInput>
            <CoordinateInput
              placeholder="z"
              onChange={(e) => handleChangeCoordinate("rtz",
Number(e.target.value))}
            ></CoordinateInput>
          </div>
          {/* these input is read only */}
          <div className="flex flex-col">
            <input
              type="number"
              value={RBCoordinate?.x}
```

```
            readOnly={true}
            className="border-2 border-neutral-600 bg-neutral-500 rounded w-14"
           />
           <input
            type="number"
            value={RBCoordinate?.y}
            readOnly={true}
            className="border-2 border-neutral-600 bg-neutral-500 rounded w-14"
           />
           <input
            type="number"
            value={RBCoordinate?.z}
            readOnly={true}
            className="border-2 border-neutral-600 bg-neutral-500 rounded w-14"
           />
          </div>
         </div>
        </div>
        <div className="flex justify-center mt-2">
         <div className="my-3 flex flex-col justify-center">
          <button
           onClick={handleGenerateCommand}
           className="p-1 rounded-md disabled:bg-neutral-900 disabled:text-neutral-
600 bg-m-green-light text-white"
           disabled={!isCommandReady}
          >
           {t.COMMAND_GENERATION}
          </button>
          <div className="my-3">
           <Link href="/command-help">
            <a className="underline text-
decoration">{t.HOW_TO_RUN_COMMAND}</a>
           </Link>
          </div>
         </div>
        </div>
      </Modal>
    </>
  );
};

export default CommandModal;
const CrossButton: React.FC = () => {
  return (
    <>
```

```
      <span className="inline-block align-middle text-white bg-white leading-4 w-5
h-0.5 rounded-sm relative rotate-45 before:content-['] before:absolute before:top-0
before:left-0 before:w-full before:h-full before:rounded-sm before:bg-white
before:rotate-90"></span>
    </>
  );
};

export default CrossButton;
import Link from "next/link";
import { useState } from "react";
import { useLocale } from "src/hooks/useLocale";
import OutsideClickHandler from "react-outside-click-handler";

interface Props {
  path: string;
}

const LanguageSwitch = ({ path }: Props) => {
  const { t, locale } = useLocale();
  const [isMenuOpen, setIsMenuOpen] = useState(false);

  const handleMenuOpen = () => {
    setIsMenuOpen(true);
  };

  const handleMenuClose = () => {
    setIsMenuOpen(false);
  };

  return (
    <>
      <div className="">
        <button onClick={handleMenuOpen} className="border-2 px-2 py-1
rounded">
          <div className="flex whitespace-nowrap">
            <div className="material-symbols-outlined text-md mr-1">language</div>
            {locale === "uk" ? <div>Українська</div> : <div>English</div>}
          </div>
        </button>
      </div>
      <OutsideClickHandler
        onOutsideClick={() => {
          handleMenuClose();
        }}
```

```
    >
      <div className="relative">
        {isMenuOpen && (
          <div className="absolute">
            <div className="bg-neutral-900 hover:bg-m-green w-24 p-1 text-right
border-b border-neutral-600">
              <Link href={path} locale="en" passHref>
                <a onClick={handleMenuClose}>English</a>
              </Link>
            </div>
            <div className="bg-neutral-900 hover:bg-m-green w-24 p-1 text-right">
              <Link href={path} locale="uk" passHref>
                <a onClick={handleMenuClose}>Українська</a>
              </Link>
            </div>
          </div>
        )}
      </div>
    </OutsideClickHandler>
  </>
 );
};

export default LanguageSwitch;
import { createContext, useState, useContext, ReactNode } from "react";
import loadImage from "src/functions/loadImage";
import { t } from "src/functions/t";

/* block data import */
import wool from "src/json/wool.json";
import terracotta from "src/json/terracotta.json";
import glazedTerracotta from "src/json/glazedTerracotta.json";
import ore from "src/json/ore.json";
import concrete from "src/json/concrete.json";
import stone from "src/json/stone.json";
import soil from "src/json/soil.json";
import jewel from "src/json/jewel.json";
import wood from "src/json/wood.json";
import light from "src/json/light.json";
import air from "src/json/air.json";
import glass from "src/json/glass.json";
import rgb2Lab from "src/functions/rgb2lab";

const BlockDBContext = createContext(
 {} as {
```

```
    blockDB: Array<BlockBasic>;
    javaId2index: Map<string, number>;
    blockImageDataDict: Map<string, BlockImageData>;
    getBlockBasic: (javaId: string) => BlockBasic;
    initBlockImageDataDict: () => void;
  }
);

export function useBlockDBContext() {
  return useContext(BlockDBContext);
}

export function BlockDBProvider({ children }: { children?: ReactNode }) {
  const javaId2index: Map<string, number> = new Map();
  const blockDB: Array<BlockBasic> = air.blocks.concat(
    wool.blocks,
    concrete.blocks,
    terracotta.blocks,
    glass.blocks,
    glazedTerracotta.blocks,
    ore.blocks,
    stone.blocks,
    soil.blocks,
    wood.blocks,
    jewel.blocks,
    light.blocks
  );

  for (let index = 0; index < blockDB.length; index++) {
    javaId2index.set(blockDB[index].javaId, index);
  }

  const getBlockBasic = (javaId: string): BlockBasic => {
    return blockDB[javaId2index.get(javaId)!];
  };

  const [blockImageDataDict, setBlockImageDataDict] = useState<Map<string,
BlockImageData>>(new Map());

  const initBlockImageDataDict = async () => {
    const size = 16;
    const blockImageDataDict: Map<string, BlockImageData> = new Map();

    // image load
    for (const block of blockDB) {
```

```
      const mem_canvas = document.createElement("canvas");
      mem_canvas.width = size;
      mem_canvas.height = size;

      const context = mem_canvas.getContext("2d");
      if (context == null) continue;
      const image = await loadImage(block.imagePath);
      context.drawImage(image, 0, 0);
      const blockImageData = context.getImageData(0, 0, size, size);

      let R = 0;
      let G = 0;
      let B = 0;
      for (let i = 0; i < size; i++) {
        for (let j = 0; j < size; j++) {
          const srcImage = blockImageData.data;
          R += srcImage[t(j, i, 0, size)];
          G += srcImage[t(j, i, 1, size)];
          B += srcImage[t(j, i, 2, size)];
        }
      }
      R /= size * size;
      G /= size * size;
      B /= size * size;
      const { L, a, b } = rgb2Lab(R, G, B);

      blockImageDataDict.set(block.javaId, { imageData: blockImageData, R, G, B, L,
a, b });
      setBlockImageDataDict(blockImageDataDict);
    }
  };

  const value = {
    blockDB,
    javaId2index,
    blockImageDataDict,
    getBlockBasic,
    initBlockImageDataDict,
  };

  return <BlockDBContext.Provider
value={value}>{children}</BlockDBContext.Provider>;
}
import { createContext, useState, useContext, ReactNode } from "react";
```

```
const BlueprintContext = createContext(
  {} as {
    initBlueprint: Array<Array<string>>;
    setInitBlueprint: React.Dispatch<React.SetStateAction<Array<Array<string>>>>;
  }
);

export function useBlueprintContext() {
  return useContext(BlueprintContext);
}

export function BlueprintProvider({ children }: { children?: ReactNode }) {
  const [initBlueprint, setInitBlueprint] = useState<Array<Array<string>>>([[]]);
  const value = {
    initBlueprint,
    setInitBlueprint,
  };

  return <BlueprintContext.Provider
value={value}>{children}</BlueprintContext.Provider>;
}
import { createContext, useState, useContext, ReactNode } from "react";

const EditorContext = createContext(
  {} as {
    mode: Mode;
    inkBlockJavaId: string;
    replaceFromJavaId: string;
    replaceToJavaId: string;
    hoverBlockJavaId: string;
    penSizeIndex: number;
    setMode: React.Dispatch<React.SetStateAction<Mode>>;
    setInkBlockJavaId: React.Dispatch<React.SetStateAction<string>>;
    setReplaceFromJavaId: React.Dispatch<React.SetStateAction<string>>;
    setReplaceToJavaId: React.Dispatch<React.SetStateAction<string>>;
    setHoverBlockJavaId: React.Dispatch<React.SetStateAction<string>>;
    upPenSize: () => void;
    downPenSize: () => void;
    getPenSize: () => number;
  }
);

export function useEditorContext() {
  return useContext(EditorContext);
}
```

```
export function EditorProvider({ children }: { children?: ReactNode }) {
  /* parameters that need to be managed by useState */
  const [mode, setMode] = useState<Mode>("pen");
  const [inkBlockJavaId, setInkBlockJavaId] = useState("minecraft:white_wool");
  const [replaceFromJavaId, setReplaceFromJavaId] =
useState("minecraft:white_wool");
  const [replaceToJavaId, setReplaceToJavaId] = useState("minecraft:white_wool");
  const [hoverBlockJavaId, setHoverBlockJavaId] =
useState("minecraft:white_wool");
  const [penSizeIndex, setPenSizeIndex] = useState(1);

  const penSizeList = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25];

  const upPenSize = () => {
    if (penSizeIndex >= penSizeList.length - 1) return;
    setPenSizeIndex(penSizeIndex + 1);
  };
  const downPenSize = () => {
    if (penSizeIndex <= 0) return;
    setPenSizeIndex(penSizeIndex - 1);
  };

  const getPenSize = (): number => {
    return penSizeList[penSizeIndex];
  };

  const value = {
    mode,
    inkBlockJavaId,
    penSizeIndex,
    hoverBlockJavaId,
    replaceFromJavaId,
    replaceToJavaId,
    setMode,
    setInkBlockJavaId,
    setReplaceFromJavaId,
    setReplaceToJavaId,
    setHoverBlockJavaId,
    upPenSize,
    downPenSize,
    getPenSize,
  };
```

```
  return <EditorContext.Provider
value={value}>{children}</EditorContext.Provider>;
}
const get2DArray = (h: number, w: number): Array<Array<boolean>> => {
  let array = new Array(h);
  for (let i = 0; i < h; i++) {
    array[i] = new Array<boolean>(w);
  }
  return array;
};

/*
const arounds: Array<Coordinate> = [
  { x: 0, y: 1 },
  { x: 0, y: -1 },
  { x: 1, y: 0 },
  { x: -1, y: 0 },
];

const dfs = (
  x: number,
  y: number,
  block: string,
  blueprint: Array<Array<string>>,
  visit: Array<Array<boolean>>
): void => {
  for (const around of arounds) {
    const next = { x: x + around.x, y: y + around.y };
    if (next.x < 0 || blueprint[0].length <= next.x) continue;
    if (next.y < 0 || blueprint.length <= next.y) continue;
    if (block != blueprint[next.y][next.x]) continue;
    if (visit[next.y][next.x]) continue;
    visit[next.y][next.x] = true;
    coordinates.push({ x: next.x, y: next.y });
    dfs(next.x, next.y, block, blueprint, visit);
  }
};
*/

const getSameBlockCoordinates = (x: number, y: number, blueprint:
Array<Array<string>>) => {
  const arounds: Array<Coordinate> = [
    { x: 0, y: 1 },
    { x: 0, y: -1 },
    { x: 1, y: 0 },
```

```
    { x: -1, y: 0 },
  ];
  const stack: Array<Coordinate> = [];
  const coordinates: Array<Coordinate> = [];
  stack.push({ x: x, y: y });
  const visit = get2DArray(blueprint.length, blueprint[0].length);
  visit[y][x] = true;
  const block = blueprint[y][x];

  while (stack.length != 0) {
    const c = stack.pop();
    coordinates.push(c!);

    for (const around of arounds) {
      const next = { x: c!.x + around.x, y: c!.y + around.y };
      if (next.x < 0 || blueprint[0].length <= next.x) continue;
      if (next.y < 0 || blueprint.length <= next.y) continue;
      if (block != blueprint[next.y][next.x]) continue;
      if (visit[next.y][next.x]) continue;
      visit[next.y][next.x] = true;
      stack.push({ x: next.x, y: next.y });
    }
  }

  return coordinates;
};

export default getSameBlockCoordinates;
const getTargetCoordinates = (targetJavaId: string, blueprint: Array<Array<string>>)
=> {
  const coordinates: Array<Coordinate> = [];
  for (let i = 0; i < blueprint.length; i++) {
    for (let j = 0; j < blueprint[0].length; j++) {
      if (blueprint[i][j] == targetJavaId) coordinates.push({ x: j, y: i });
    }
  }

  return coordinates;
};

export default getTargetCoordinates;
import { createContext, useContext, ReactNode } from "react";
import geenrateImageFromBlueprint from
"src/Feature/Home/functions/generateImageFromBlueprint";
```

```
import getSameBlockCoordinates from
"src/context/useEditorCanvas/functions/getSameBlockCoordinates";
import getTargetCoordinates from
"src/context/useEditorCanvas/functions/getTargetCoordinates";
import getBufferCanvas from "src/functions/getBufferCanvas";
import { useBlockDBContext } from "../useBlockDB";
import { useHistoryContext } from "../useHistory";

const EditorCanvasContext = createContext(
  {} as {
    init: (
      mainCanvas_: HTMLCanvasElement,
      naviCanvas_: HTMLCanvasElement,
      canvas: ImageData,
      widthBlockNumber_: number,
      heightBlockNumber_: number,
      blueprint_: string[][]
    ) => void;
    isShowGrid: boolean;
    getBlueprint: () => string[][];
    getMinecraftImage: () => HTMLCanvasElement;
    translate: (x: number, y: number) => void;
    zoomIn: (x: number, y: number, isWheel: boolean) => void;
    zoomOut: (x: number, y: number, isWheel: boolean) => void;
    showNaviBox: (x: number, y: number, size: number) => void;
    putBlock: (x: number, y: number, size: number, javaId: string) => void;
    bucket: (x: number, y: number, javaId: string) => void;
    replace: (from: string, to: string) => void;
    pickBlock: (x: number, y: number) => string | undefined;
    undo: () => void;
    redo: () => void;
    switchGrid: (isShow: boolean) => void;
    clearNaviCanvas: () => void;
    render: () => void;
  }
);

export function useEditorCanvasContext() {
  return useContext(EditorCanvasContext);
}

export function EditorCanvasProvider({ children }: { children?: ReactNode }) {
  const canvasSize = 1000;
  const zoomStepByWheel = 0.5;
  const zoomStepByButton = 1.5;
```

```
let minecraftImageX = 0;
let minecraftImageY = 0;
let minecraftImageWidth = 0;
let minecraftImageHeight = 0;
let longerEdge = 0;
let magnification = 0.5;
let widthBlockNumber = 0;
let heightBlockNumber = 0;
let isShowGrid = false;
let mainCanvas: HTMLCanvasElement;
let naviCanvas: HTMLCanvasElement;
let mainCanvasContext: CanvasRenderingContext2D;
let naviCanvasContext: CanvasRenderingContext2D;
let minecraftImage: HTMLCanvasElement;
let minecraftImageContext: CanvasRenderingContext2D;
const { blockImageDataDict } = useBlockDBContext();
const { backward, forward, addHistory } = useHistoryContext();
let blueprint: Array<Array<string>> = [];

console.log("EditorCanvasProvider render");

const init = (
  mainCanvas_: HTMLCanvasElement,
  naviCanvas_: HTMLCanvasElement,
  image: ImageData,
  widthBlockNumber_: number,
  heightBlockNumber_: number,
  blueprint_: Array<Array<string>>
) => {
  console.log("EditorCanvasProvider init");
  blueprint = blueprint_;
  mainCanvas = mainCanvas_;
  naviCanvas = naviCanvas_;

  mainCanvasContext = mainCanvas.getContext("2d")!;
  naviCanvasContext = naviCanvas.getContext("2d")!;

  /* canvas preference */
  mainCanvas.width = canvasSize;
  mainCanvas.height = canvasSize;
  naviCanvas.width = canvasSize;
  naviCanvas.height = canvasSize;
  mainCanvasContext.imageSmoothingEnabled = false;
  naviCanvasContext.imageSmoothingEnabled = false;
  naviCanvasContext.lineWidth = 4;
```

```
    /* iamge set */
    minecraftImage = getBufferCanvas(image, image.width, image.height);
    minecraftImageContext = minecraftImage.getContext("2d")!;

    /* minecraft image initial size and position */
    longerEdge = minecraftImage.width > minecraftImage.height ?
minecraftImage.width : minecraftImage.height;
    minecraftImageX = canvasSize / 6;
    minecraftImageY = canvasSize / 6;
    magnification = ((canvasSize / longerEdge) * 2) / 3;

    /* block number */
    widthBlockNumber = widthBlockNumber_;
    heightBlockNumber = heightBlockNumber_;
  };

  const toPixelCoordinate = (x: number, y: number) => {
    const px = x * canvasSize;
    const py = y * canvasSize;
    return { px, py };
  };

  const getBlockCoordinate = (x: number, y: number) => {
    const { px, py } = toPixelCoordinate(x, y);
    const blockSize = minecraftImageWidth / widthBlockNumber;
    const dx = px - minecraftImageX;
    const dy = py - minecraftImageY;
    const blockX = Math.floor(dx / blockSize);
    const blockY = Math.floor(dy / blockSize);

    return { blockX, blockY, blockSize };
  };

  const insideBlueprint = (i: number, j: number, blueprint: string[][]) => {
    if (i < 0 || i >= blueprint.length) return false;
    if (j < 0 || j >= blueprint[0].length) return false;

    return true;
  };

  const clearNaviCanvas = () => {
    naviCanvasContext.clearRect(0, 0, canvasSize, canvasSize);
  };
```

```
  const showNaviBox = (x: number, y: number, size: number) => {
    clearNaviCanvas();
    const { blockX, blockY, blockSize } = getBlockCoordinate(x, y);

    const beginBlockX = blockX - Math.floor(size / 2);
    const beginBlockY = blockY - Math.floor(size / 2);
    const rectX = minecraftImageX + beginBlockX * blockSize;
    const rectY = minecraftImageY + beginBlockY * blockSize;

    naviCanvasContext.strokeStyle = "#000";
    naviCanvasContext.strokeRect(rectX, rectY, blockSize * size, blockSize * size);
    naviCanvasContext.strokeStyle = "#fff";
    naviCanvasContext.strokeRect(rectX - 5, rectY - 5, blockSize * size + 10,
blockSize * size + 10);
  };

  const switchGrid = (isShow: boolean) => {
    isShowGrid = isShow;
  };

  const showGrid = () => {
    clearNaviCanvas();
    /* vertical grid */
    for (let i = 0; i <= widthBlockNumber; i++) {
      const x = minecraftImageX + (minecraftImageWidth / widthBlockNumber) * i;
      mainCanvasContext.beginPath();
      mainCanvasContext.moveTo(x, minecraftImageY);
      mainCanvasContext.lineTo(x, minecraftImageY + minecraftImageHeight);
      mainCanvasContext.stroke();
    }
    /* horizontal grid */
    for (let i = 0; i <= heightBlockNumber; i++) {
      const y = minecraftImageY + (minecraftImageHeight / heightBlockNumber) * i;
      mainCanvasContext.beginPath();
      mainCanvasContext.moveTo(minecraftImageX, y);
      mainCanvasContext.lineTo(minecraftImageX + minecraftImageWidth, y);
      mainCanvasContext.stroke();
    }
  };

  const putBlock = (x: number, y: number, size: number, javaId: string) => {
    const { blockX, blockY } = getBlockCoordinate(x, y);

    const beginX = blockX - Math.floor(size / 2);
    const endX = blockX + Math.floor(size / 2);
```

```
    const beginY = blockY - Math.floor(size / 2);
    const endY = blockY + Math.floor(size / 2);

    for (let i = beginY; i <= endY; i++) {
      for (let j = beginX; j <= endX; j++) {
        if (insideBlueprint(i, j, blueprint)) {
          /* change canvas data */

minecraftImageContext.putImageData(blockImageDataDict.get(javaId)?.imageData!,
j * 16, i * 16);
          /* change blueprint */
          blueprint[i][j] = javaId;
        }
      }
    }

    render();
  };

  const pickBlock = (x: number, y: number): string | undefined => {
    const { blockX, blockY } = getBlockCoordinate(x, y);
    if (!insideBlueprint(blockY, blockX, blueprint)) return;
    return blueprint[blockY][blockX];
  };

  const translate = (x: number, y: number) => {
    const { px, py } = toPixelCoordinate(x, y);
    minecraftImageX += px;
    minecraftImageY += py;
    render();
  };

  const bucket = (x: number, y: number, javaId: string) => {
    const { blockX, blockY } = getBlockCoordinate(x, y);
    if (!insideBlueprint(blockY, blockX, blueprint)) return;
    const coordinates = getSameBlockCoordinates(blockX, blockY, blueprint);
    for (const coordinate of coordinates) {
      /* change canvas data */
      minecraftImageContext.putImageData(
        blockImageDataDict.get(javaId)?.imageData!,
        coordinate.x * 16,
        coordinate.y * 16
      );
      /* change blueprint */
      blueprint[coordinate.y][coordinate.x] = javaId;
```

```
    }

    render();
  };

  const replace = (from: string, to: string) => {
    const coordinates = getTargetCoordinates(from, blueprint);

    for (const coordinate of coordinates) {
      /* change canvas data */
      minecraftImageContext.putImageData(
        blockImageDataDict.get(to)?.imageData!,
        coordinate.x * 16,
        coordinate.y * 16
      );
      /* change blueprint */
      blueprint[coordinate.y][coordinate.x] = to;
    }
    render();
  };

  const zoom = (px: number, py: number, zoomStep: number, isZoomOut: boolean)
=> {
    if (isZoomOut) zoomStep *= -1;

    const minecraftImageTopLeftX = px - minecraftImageX;
    const minecraftImageTopLeftY = py - minecraftImageY;
    const deltaScale = (magnification + zoomStep) / magnification;
    const deltaX = (deltaScale - 1) * minecraftImageTopLeftX;
    const deltaY = (deltaScale - 1) * minecraftImageTopLeftY;
    minecraftImageX -= deltaX;
    minecraftImageY -= deltaY;
    magnification += zoomStep;
    render();
  };
  /* zoom in around (x, y) */
  const zoomIn = (x: number, y: number, isWheel: boolean) => {
    if (magnification >= 16) return;
    let zoomStep = zoomStepByButton;
    if (isWheel) zoomStep = zoomStepByWheel;
    if (magnification <= 1) zoomStep = 0.04;
    const { px, py } = toPixelCoordinate(x, y);
    zoom(px, py, zoomStep, false);
  };
  /* zoom out around (x, y) */
```

```
const zoomOut = (x: number, y: number, isWheel: boolean) => {
  let zoomStep = zoomStepByButton;
  if (isWheel) zoomStep = zoomStepByWheel;
  /* change zoom step when magnification is small */
  if (magnification <= 1) zoomStep = 0.04;
  /* stop zoom out when image size is half of canvas */
  if ((magnification - zoomStep) * longerEdge < canvasSize / 2) return;

  const { px, py } = toPixelCoordinate(x, y);
  zoom(px, py, zoomStep, true);
};

const getBlueprint = () => {
  return blueprint;
};

const getMinecraftImage = () => {
  return minecraftImage;
};

const undo = () => {
  const b = backward();
  if (b === undefined) return;
  const image = geenrateImageFromBlueprint(b, blockImageDataDict);
  minecraftImageContext.putImageData(image, 0, 0);
  blueprint = b;
  render();
};

const redo = () => {
  const b = forward();
  if (b === undefined) return;
  const image = geenrateImageFromBlueprint(b, blockImageDataDict);
  minecraftImageContext.putImageData(image, 0, 0);
  blueprint = b;
  render();
};

const render = () => {
  mainCanvasContext.clearRect(0, 0, canvasSize, canvasSize);
  minecraftImageWidth = minecraftImage.width * magnification;
  minecraftImageHeight = minecraftImage.height * magnification;
  mainCanvasContext.drawImage(
    minecraftImage,
    minecraftImageX,
```

```
      minecraftImageY,
      minecraftImageWidth,
      minecraftImageHeight
    );
    if (isShowGrid) showGrid();
  };

  const value = {
    isShowGrid,
    getMinecraftImage,
    getBlueprint,
    init,
    translate,
    bucket,
    replace,
    render,
    zoomIn,
    zoomOut,
    pickBlock,
    showNaviBox,
    putBlock,
    clearNaviCanvas,
    undo,
    switchGrid,
    redo,
  };

  return <EditorCanvasContext.Provider
value={value}>{children}</EditorCanvasContext.Provider>;
}
import SideToolBar from "./components/SideToolBar";
import EditorBoard from "./components/EditorBoard";
import BlockPalette from "./components/BlockPalette";
import TopToolBar from "./components/TopToolBar";

const EditorPageComponent = () => {
  return (
    <>
      <div className="bg-neutral-700">
        <div className="flex justify-start">
          <TopToolBar />
        </div>
        <div className="h-[2px] 1-[100wv] bg-neutral-800"></div>
        <div className="flex flex-wrap justify-start">
          <SideToolBar />
```

```
          <EditorBoard />
          <div className="mx-3 my-2">
            <BlockPalette />
          </div>
        </div>
      </div>
    </>
  );
};

export default EditorPageComponent;
const copy2DArray = <T>(twoDArray: Array<Array<T>>): Array<Array<T>> => {
  const result: Array<Array<T>> = [];
  for (let i = 0; i < twoDArray.length; i++) {
    result.push([]);
    for (let j = 0; j < twoDArray[0].length; j++) {
      result[i].push(twoDArray[i][j]);
    }
  }

  return result;
};

export default copy2DArray;
const createNmaeTable = (
  blueprint: string[][],
  blockDB: BlockBasic[],
  javaId2index: Map<string, number>,
  locale: Locale
) => {
  const nameTable = [];

  for (let y = 0; y < blueprint.length; y += 1) {
    const row = [];
    for (let x = 0; x < blueprint[0].length; x += 1) {
      const blockName =
        locale === "ja"
          ? blockDB[javaId2index.get(blueprint[y][x])!].jname
          : blockDB[javaId2index.get(blueprint[y][x])!].javaId;

      row.push(blockName);
    }
    nameTable.push(row);
  }
```

```
  return nameTable;
};

const totalUpBlock = (blueprint: string[][]) => {
  const blockAmount: Map<string, number> = new Map();
  for (let i = 0; i < blueprint.length; i++) {
    for (let j = 0; j < blueprint[0].length; j++) {
      const javaId = blueprint[i][j];
      if (blockAmount.get(javaId) !== undefined) {
        blockAmount.set(javaId, blockAmount.get(javaId)! + 1);
      } else {
        blockAmount.set(javaId, 1);
      }
    }
  }

  return blockAmount;
};

const createCsv = (
  blueprint: string[][],
  blockData: BlockBasic[],
  javaId2index: Map<string, number>,
  locale: Locale,
  useBlockTitle: string
) => {
  const nameTable = createNmaeTable(blueprint, blockData, javaId2index, locale);
  const blockAmount = totalUpBlock(blueprint);

  // top row
  let csv = "\ufeff";
  for (let i = 0; i < nameTable[0].length; i++) {
    csv += ",";
    csv += String(i + 1);
  }
  csv += "\n";

  // blueprint csv
  nameTable.forEach((row, index) => {
    let line = String(index + 1);
    row.forEach((name) => {
      line += ",";
      line += name;
    });
    line += "\n";
```

```
    csv += line;
  });

  csv += "\n";
  csv += " ";
  csv += ",";
  csv += useBlockTitle;
  csv += "\n";
  // block amount csv
  for (const [javaId, amount] of blockAmount.entries()) {
    const blockName =
      locale === "ja"
        ? blockData[javaId2index.get(javaId)!].jname
        : blockData[javaId2index.get(javaId)!].javaId;
    csv += " ";
    csv += ",";
    csv += blockName;
    csv += ",";
    csv += amount;
    csv += "\n";
  }

  return csv;
};

/*
function createUseBlockCsv (blockData) {
  let csv = '\ufeff'
  let line = 'ブロック名' + ',' + '個数'
  line += '\n'
  blockData.forEach((block, index) => {
    if (block.number > 0) {
      line += block.jname
      line += ','
      line += block.number
      line += '\n'
    }
  })
  csv += line
  return csv
}
*/

export default createCsv;
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ СТВОРЕННЯ MINECRAFT PIXEL ART

Бакалаврська дипломна робота «Методи та програмні засоби створення Minecraft Pixel Art»

Виконав студент 4 курсу
Групи 4ПІ-20Б Грицишин В.О.
Керівник: Майданюк В.П.

Рисунок Г.1 – Титульний слайд

Мета і задачі роботи

Мета роботи:

- процес покращення ефективності програмних засобів Minecraft Pixel Art з використанням додаткових інструментів щодо імпорту та редагування.

Основні задачі дослідження:

- проведення аналізу стану питання та методи розв'язання задачі;
- спроєктування моделі даних та структури програмних засобів;
- розробка загального алгоритму роботи веб-застосунку;
- розробка алгоритму обробки зображень;
- виконання програмної реалізації головного екрану застосунку;
- запровадження взаємодії між клієнтом та системою веб-застосунку;
- проведення тестування програмних засобів;

Об'єкт дослідження – процес розробки програмних модулів для реалізації засобів створення Minecraft Pixel Art.

Предмет дослідження – методи і програмні засоби формування Minecraft Pixel Art.

Рисунок Г.2 – Мета, об'єкт та предмет дослідження

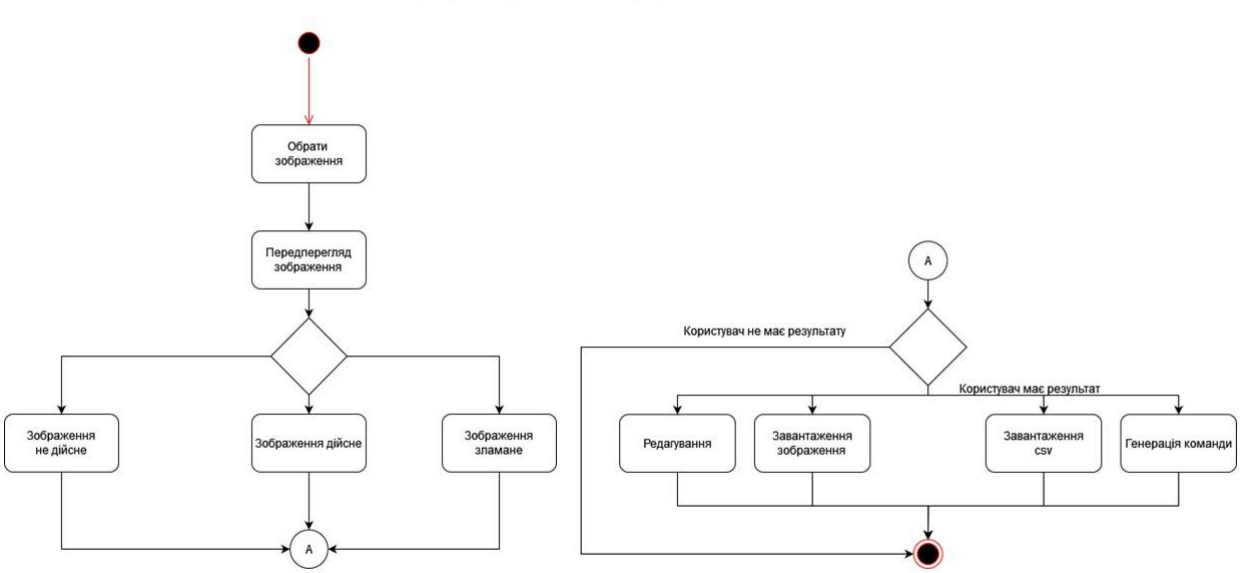


Рисунок Г.3 – Діаграма діяльності

Рисунок Г.4 – Алгоритм роботи програмних методів

Вигляд головного екрану

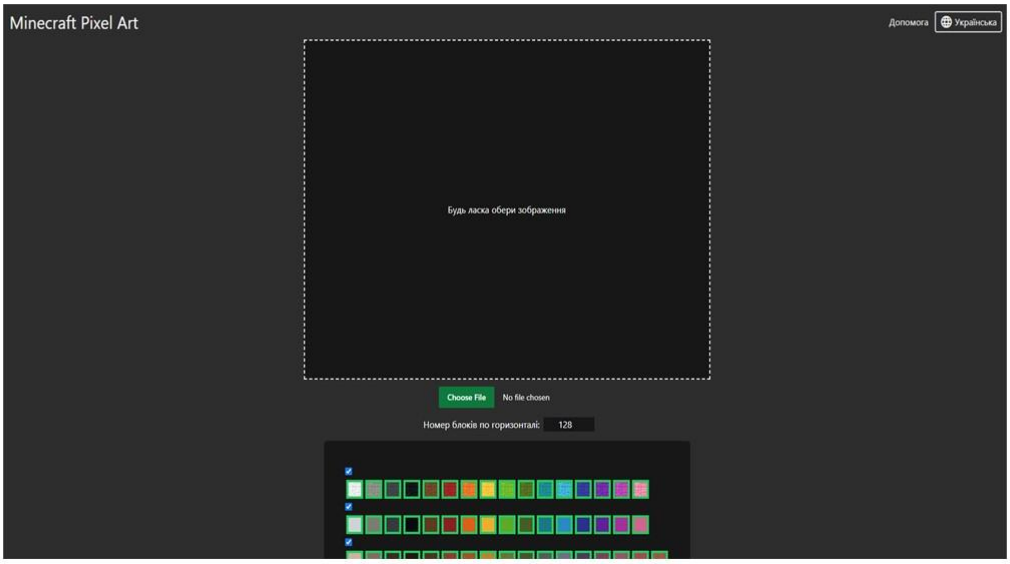
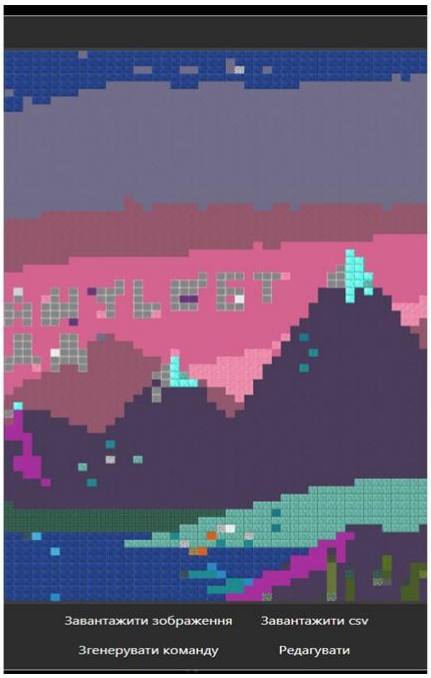


Рисунок Г.5 – Вигляд головного екрану

Створене зображення з палітри блоків



Зображення створюється з блоків гри
Де можна побачити текстуру блоку і
схожий колір початкового зображення

Рисунок Г.6 – Створене зображення з палітри блоків

Інші знімки програмних методів

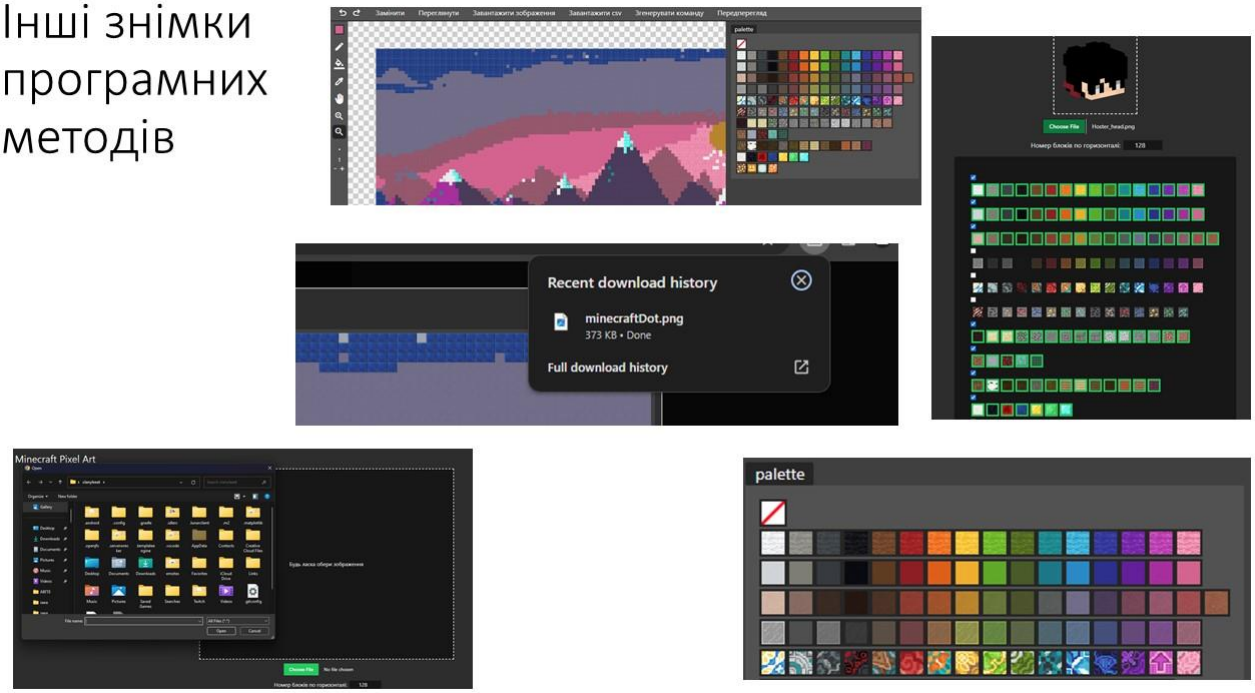


Рисунок Г.7 – Інші знімки програмних методів

Структура програмних методів

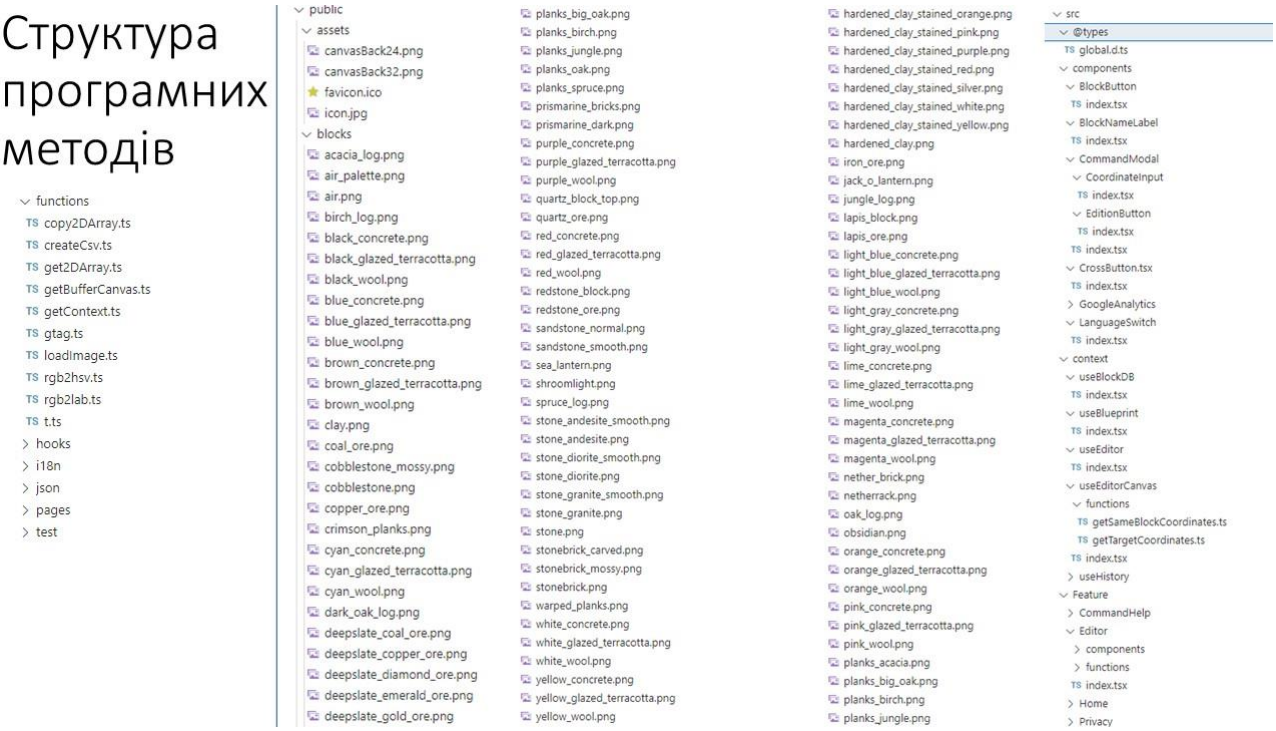


Рисунок Г.8 – Структура програмних методів

Тестування програмних засобів

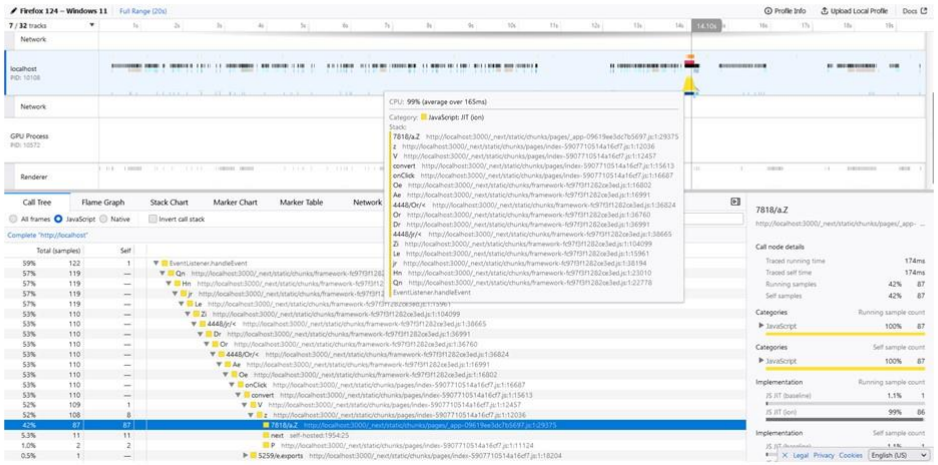


Рисунок Г.9 – Тестування програмних засобів

Основні результати роботи

Новизна отриманих результатів.

1. Удосконалено функцію редагування створеного зображення, яка відрізняється від відомих наявністю повноцінного редактора зображення, що дозволило більш точну передачу колірної гами для задоволення потреб користувача.
2. Подальшого розвитку отримав метод створення пакетів даних для гри Minecraft, у якому на відміну від відомих додано масив для ідентифікації, що забезпечує повну сумісність з будь-якими версіями гри Minecraft.
3. Запропоновано метод перетворення зображення у 2D-масив, у якому на відміну від відомих виконується порівнянн колірної гами оригінального зображення з колірною гамою палітри блоків, що дозволяє більш точно відобразити новостворене зображення.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для вдосконалення та популяризації програмних засобів створення Minecraft Pixel Art.

Рисунок Г.10 – Основні результати роботи

Рисунок Г.11 – Фінальний слайд