РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: комп’ютерна гра, яка на основі результатів тесту на дальтонізм враховує індивідуальні особливості колірного сприйняття.

Мета кваліфікаційної роботи полягає в створенні технології зміни колірних схем у додатках на прикладі гри в жанрі action-adventure, адаптовану для мобільних пристроїв та призначену для людей з особливостями колірного сприйняття.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано платформу для розробки, виконано проєктування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає у створенні технології зміни колірних схем у іграх, адаптованої для людей з особливими потребами. Додаток може бути запропонований розробникам програмного забезпечення як приклад програми, в якій створено комфортні умови гри для різних категорій користувачів.

Актуальність даного програмного продукту пов'язана з необхідністю надати доступ до освіти та ігор дітям з особливими потребами, враховуючи їх індивідуальні особливості.

Список ключових слів: КОЛІРНЕ СПРИЙНЯТТЯ, ДАЛЬТОНІЗМ, ПРОГРАМА, КОМП’ЮТЕРНА ГРА.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: a computer game, which based on the results of the test for color blindness takes into account the individual characteristics of color perception.

The purpose of the qualification work is to create a technology for changing color schemes in applications on the example of the game in the genre of action-adventure, adapted for mobile devices and designed for people with color perception.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, designs and develops the program, describes the algorithm and structure of the program, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance of the work is to create a technology for changing color schemes in games, adapted for people with special needs. The application can be offered to software developers as an example of a program that creates comfortable game conditions for different categories of users.

The relevance of this software product is related to the need to provide access to education and games for children with special needs, taking into account their individual characteristics.

List of keywords: COLOR PERCEPTION, DALTONISM, PROGRAM, COMPUTER GAME.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

HSL – англ. Hue, Saturation, Lightness, колірна модель з кольоровим тоном, насиченістю та світлотим.

ЗМІСТ

ВСТУП

Для сприйняття інформації про стан навколишнього світу людині служать складні й досконалі органи чуття. До них відносяться й очі. Дивитися на світ і бачити його красу – велике щастя. За допомогою очей людина бачить предмети, їх переміщення і колір. Це робить очі найціннішим органом в системі органів чуття.

За допомогою зору люди отримують близько 80% всієї інформації, що надходить ззовні. Завдяки ньому людина розрізняє дрібні деталі предметів і самі предмети, правильно визначає їхнє місце розташування в просторі, сприймає найбагатшу гаму колірних відтінків. Зір дозволяє нам займатися кваліфікованою виробничою працею, читати, писати, малювати, дивитися кінофільми, спектаклі та інші видовищні заходи. Людям з порушеннями зору важко займатися на перший погляд звичайними справами, для них потрібно адаптувати додатки, ігри та сайти.

У наш час ігри набули широкого поширення на гаджетах, а також на ігрових консолях, але найцікавіші з них не є безкоштовними й часто не мають такої функції, як зміна колірних схем. Тобто є дефіцит ігор, адаптованих для людей з вадами зору, як на ПК, так і на мобільних платформах.

Може виникнути питання: "Навіщо взагалі орієнтуватися на таку вузьку групу користувачів?" Справа в тому, що елементи дизайну, які є комфортними для користувачів з вадами зору, можна вважати добре спроектованими й зручними для більш широкої аудиторії. І якщо дизайн гри розроблений якісно, він стане в нагоді всім зацікавленим користувачів.

Актуальність цієї теми пов'язана з необхідністю надати доступ до освіти та ігор дітям з особливими потребами, враховуючи їх індивідуальні особливості.

Мета кваліфікаційної роботи полягає в створенні технології зміни колірних схем у додатках на прикладі гри в жанрі action-adventure, адаптовану

для мобільних пристроїв та призначену для людей з особливостями колірного сприйняття.

Для досягнення мети слід розв’язати такі задачі:

– проаналізувати сучасні тенденції розвитку програмного забезпечення для людей з дальтонізмом;

– проаналізувати особливості створення технології зміни кольору;

– порівняти наявні на ринку аналоги;

– висунути вимоги до колірних схем для людей з порушенням колірного сприйняття;

– спроектувати логіку гри;

– створити програмну реалізацію та здійснити її тестування;

– описати технологію зміни колірних схем.

Задачею роботи є дослідження та програмна реалізація способів створення комфортних умов гри для людей з колірною сліпотою, використовуючи технологію, що дозволяє змінити колірну схему на сприятливу із заданими параметрами у рушії Unreal Engine 4.

В рамках виконання даної роботи створено технологію, яка дозволяє адаптувати колірні схеми у грі індивідуально для кожного користувача. Це створює комфортні умови як для людей з особливими потребами, так і для гравців зі звичайним зором.

Практичне значення роботи полягає у створенні технології зміни колірних схем у іграх, адаптованої для людей з особливими потребами. Додаток може бути запропонований розробникам програмного забезпечення як приклад програми, в якій створено комфортні умови гри для різних категорій користувачів.

Запропонована версія гри, яка на основі результатів тесту на дальтонізм враховує індивідуальні особливості кожного, допоможе охопити більшу аудиторію гравців.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

1.1.1. Загальна характеристика технології зміни контрастних схем

Дальтонізм або колірна сліпота - спадкова, рідше набута особливість зору людини та приматів, що виражається в нездатності розрізняти один або декілька кольорів.

У людини в центральній частині сітківки розташовані кольорочутливі рецептори - нервові клітини, які називаються колбочками. Кожен з трьох видів колбочок має свій тип кольорочутливого пігменту білкового походження, який чутливий до:

– червоного кольору (з максимумом довжини хвилі світла 552-557 нм);
– зеленого (максимум близько 530 нм);
– синього (максимум близько 426 нм).

Люди з нормальним кольоровим зором (трихромати) мають в колбочках усі три пігменти (червоний, зелений і синій) у необхідній кількості.

Приблизно 8-10% чоловіків і 0.5% жінок мають ту чи іншу форму дальтонізму. Тобто, на кожні 100 гравців припадає 10 таких, які бачать колір інакше, по-своєму. І як же зробити інтерфейс, однаково доступним для будь-яких користувачів? Є багато суперечливої інформації з цієї теми. Під час виконання даної роботи були проаналізували й зробені висновки про основні принципи, які варто враховувати в дизайні, щоб домогтися дружнього кольорового інтерфейсу гри [20].

Виділяючи об'єкти, дальтоніки більше спираються на світловий контраст. Він так чи інакше присутній в будь-якому поєднанні кольорів, проте може бути різким або приглушеним [9]. Чим він нижчий, тим складніше людині розрізняти кольори. Тобто, постає потреба створити контрастну та приглушену

кольорову схему для того, щоб інтерфейс та графіка була сприятливі для всіх гравців.

1.1.2. Аналіз ігор жанру action-adventure з можливістю зміни кольорових схем

Проаналізовано, чи багато існує комп'ютерних ігор, що мають функцію адаптування кольорових схем для користувачів. Виявилось, що в 2011 році була випущена багатоплатформенна гра з назвою «Battlefield3» (рис. 1.1), що надавала можливість змінювати стиль із яскраво-зеленого й червоного на приглушені червоний і синій, які зручні для дальтоніків. Також замість червоних і зелених індикаторів для позначення команд-суперників використовуються альтернативні кольори: блакитний і помаранчевий. Таке рішення було застосовано в грі «Call of Duty: Black Ops» (1.2). Компанія «Treyarch» пішла ще далі, розробивши спеціальний тестер на колірну сліпоту (1.3).

Рис. 1.1. Гра «Battlefield3»

Рис. 1.2. Гра «Call of Duty: Black Ops»

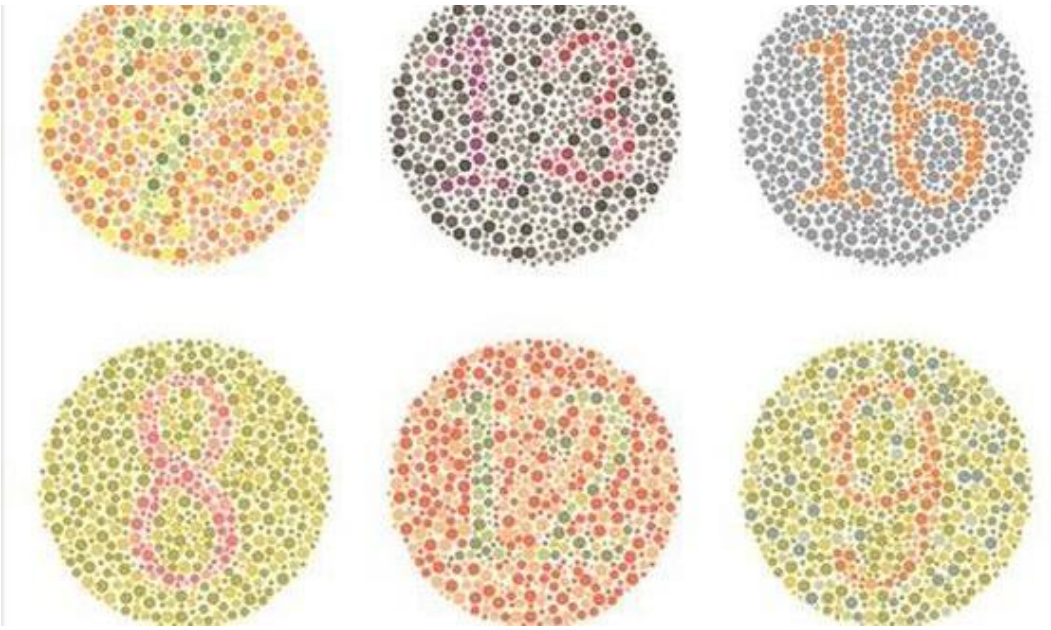

Рис. 1.3. Тест на колірну сліпоту

Пригодницький бойовик (англ. Action-adventure) - це жанр відеоігор, який поєднує характерні особливості пригодницьких відеоігор та ігор жанру бойовик. Від перших він отримав велику кількість персонажів, діалоги, інвентар, головоломки, акцент на дослідженні рівнів та складний сюжет, а від других динамічні сутички з ворогами, арсенал зброї, з пов'язаним менеджментом боєприпасів та систему окремих рівнів (мап).

Серед ігор-представників жанру action-adventure, що мають вищезазначену функцію, можна виділити такі проекти, як серія ігор «Red Dead Redemption», «Death Stranding» (рис. 1.4).

Рис. 1.4. Гра «Red Dead Redemption»

Колір відіграє важливу роль в сприйнятті сюжету та отриманні позитивних вражень від гри, але ці програми платні. Цікавими також є MOBA-ігри [14] «Heroes of the storm», «Dota 2» – ці проекти одні з небагатьох, що адаптують кольорову схему «під гравця» і є безкоштовними. Популярний «Minecraft» (рис. 1.5) також адаптований для людей з вадами зору, дозволяє змінювати рівень контрастності та колір в налаштуваннях гри.

Рис. 1.5. Гра «Minecraft»

Також був проаналізований сегмент ігор для смартфонів. Було знайдено декілька проектів, які враховують індивідуальні особливості гравців. Наприклад, «Moon Hunters» та «Hidden folks» (рис. 1.6). Але вони не є безкоштовними, тобто не зовсім доступні для дітей. Тобто дефіцит в сегменті мобільних програм досі наявний.

Рис. 1.6. Гра «Hidden folks»

1.1.3. Аналіз програм для зміни колірних схем

Розглянемо деякі найпоширеніші програми та технології, що враховують індивідуальні особливості людей з порушенням зору.

1. Режим контрастності у Windows 10 (рис. 1.7). Текст із низькою контрастністю може бути складно прочитати для людей зі слабким зором. Наприклад, є веб-сайти, на яких присутнє поєднання несумісних кольорів (наприклад, блакитні посилання на чорному фоні). Такий контент важко

сприймається навіть людьми з нормальним зором та може стати практично недоступним для людей із вадами зору. Дуже контрастні кольори можуть пришвидшити читання з комп'ютера та зробити його зручнішим. На жаль, функціонал цього режиму дуже обмежений, адже неможливо адаптувати графіку індивідуально, а деякі елементи взагалі спотворюються.

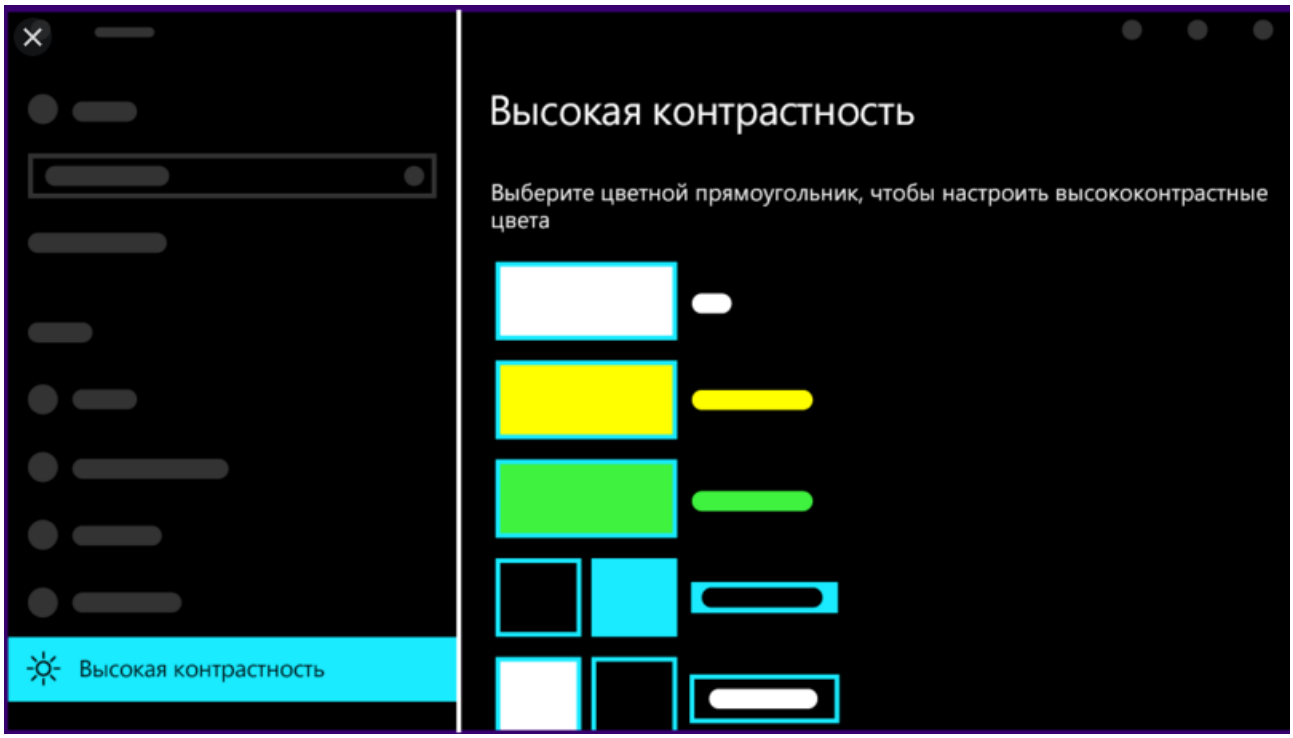

Рис. 1.7. Режим контрастності у Windows 10

2. Color Binoculars (рис. 1.8) - додаток для iOS, що використовує камеру телефону для попереднього перегляду в реальному часі і наступного налаштування зображення за допомогою спеціальних фільтрів. Має адаптованість до трьох різних типів дальтонізму. У ньому можна ввімкнути або вимкнути всі фільтри, щоб порівняти, як виглядає світ для людей, які не розрізняють кольори і для всіх інших. Програму неможливо зараз завантажити, а її підтримка здійснюється лише в деяких регіонах США.

Рис. 1.8. Додаток Color Binoculars

3. Color Blind Pal (рис. 1.9) – програма, що допомагає дальтонікам розрізняти кольори, які вони зазвичай не бачать. Також цікавою особливістю програми є режим «емпатії», який показує звичайним людям, як дальтоніки бачать світ. Зараз доступна на Android-пристроях. Прочитавши відгуки на play market, можна зробити висновок, що вона працює некоректно, що свідчить про необхідність створення нашої технології.

Рис. 1.9. Додаток Color Blind Pal

1.2. Призначення розробки та область застосування

Відомо, що колір має велике значення в найрізноманітніших сферах: у кіно, комп'ютерних іграх, дизайні сайтів, архітектурі, мистецтві тощо. Застосунки зі змінами колірних схем зможуть допомогти людям з особливими потребами сприймати контент більш комфортно. Технологія, яка зможе адаптувати колір у будь-якій грі під індивідуальні особливості гравця та його пристрою, є актуальною наразі.

Проаналізувавши ринок програмного забезпечення, було виявлено, що існують схожі програми як професійного рівня, так і більш прості. Але зазвичай вони надають низький функціонал або лише допомагають розробникам відчувати тонкощі дальтонізму при розробці додатків.

Призначенням роботи полягає у створенні технології зміни колірних схем у іграх, адаптованої для людей з особливими потребами. Додаток може бути запропонований розробникам програмного забезпечення як приклад програми, в якій створено комфортні умови гри для різних категорій користувачів.

Запропонована версія гри, яка на основі результатів тесту на дальтонізм враховує індивідуальні особливості кожного, допоможе охопити більшу аудиторію гравців.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка та програмна реалізація технології зміни колірних схем у додатках на прикладі гри в жанрі action-adventure» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Задачею кваліфікаційної роботи на тему «Розробка та програмна реалізація технології зміни колірних схем у додатках на прикладі гри в жанрі action-adventure» є дослідження та програмна реалізація способів створення комфортних умов гри для людей з колірною сліпотою, використовуючи технологію, що дозволяє змінити колірну схему на сприятливу із заданими параметрами у рушії Unreal Engine 4.

Мета кваліфікаційної роботи полягає в створенні технології зміни колірних схем у додатках на прикладі гри в жанрі action-adventure, адаптовану для мобільних пристроїв та призначену для людей з особливостями колірного сприйняття.

Для досягнення мети слід розв’язати такі задачі:

– проаналізувати сучасні тенденції розвитку програмного забезпечення для людей з дальтонізмом;

– проаналізувати особливості створення технології зміни кольору;

– порівняти наявні на ринку аналоги;

– висунути вимоги до колірних схем для людей з порушенням колірного сприйняття;

– спроектувати логіку гри;

– створити програмну реалізацію та здійснити її тестування;

– описати технологію зміни колірних схем.

В програмі необхідно створити технологію зміни колірних схем, застосувавши її на грі, яка буде адаптованою для людей з особливими потребами.

Не існує універсального рішення, яке потрібно використовувати в дизайні для дальтоніків, але є кілька принципів, які необхідно враховувати. Вимоги до дизайну програми-гри:

1. Не покладатися лише на колір для індикації стану.

2. Обмежити колірну палітру 2-3 кольорами.

3. Використовувати текстуру і візерунки, щоб показати контраст.

4. Використовувати контрастні кольори і відтінки, щоб виділити текстові посилання

5. Не використовувати небезпечні комбінації кольорів: зелений з червоним, зелений з блакитним, блакитний з сірим і т.д.

Для створення гри необхідно продумати компоненти гри та її логіку, а саме процес взаємодії гри з користувачем, умови виграшу, фіксації результатів і т. п.

Для її реалізації гри необхідно виконати наступні етапи проектування:

- візуалізація ігрового процесу;
- взаємодія з користувачем;
- логіка гри.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

В результаті розробки даного проекту повинна бути спроектована та створена гра у рушії Unreal Engine 4 на мові Java в середовищі середовищі Eclipse SDK, яка дозволяє змінити кольорову схему на більш контрастну із заданими параметрами, що є одним із способів завдання комфортних умов гри для людей з колірною сліпотою.

Гра повинна складатися з одного режиму. Гравець буде переміщатися по мапі, мати діалоги, інвентар та сутички з ворогами. Також повинно бути передбачено «Меню паузи» для того, щоб була можливість відпочити та змінити кольорову схему для людей з особливими потребами.

Для вирішення цих задач при розробці ігрового додатку повинні бути використані наступні рішення:

1. Розробка правил гри та стратегії поведінки.
2. Розробка моделей зображення графічних об’єктів.
3. Розробка логічної схеми додатку.
4. Розробка алгоритму роботи додатку.
5. Розробка графіки додатку з урахуванням вимог до гри для людей з вадами зору.
6. Розробка інтерфейсу користувача.
7. Розробка ігрового процесу.
8. Тестування та налагодження розробленої програми.

1.5.2. Вимоги до інформаційної безпеки

Інформаційна безпека прграмног забезпечення має надавати гарантію того, що досягаються такі цілі:

– конфіденційність критичної інформації;

– цілісність інформації та пов’язаних з нею процесів (створення, введення, обробка та відображення);

– актуальність (своєчасність відновлення) і доступність інформації (коли вона потрібна авторизованим користувачам);

– облік всіх процесів, пов’язаних з інформацією.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для підтримки коректної роботи комп’ютерної гри визначено рекомендовані вимоги до пристроїв, на яких він працюватиме. Така конфігурація обумовлена тим, що підтримує оптимальну роботу операційної системи, в результаті чого підвищується доступність додатку.

– процесор Intel Core i3-4160 3,6 ГГц, 2 ядра, 4 потоки;

– відеокарта GeForce GTX 550 Ti 1 ГБ, 4100 МГц;

– оперативна пам’ять Kingston DDR3 4 ГБ, 1333 МГц;

– материнська плата Gigabyte GA-H110-D3A s1151, DDR4;

– блок живлення LogicPower ATX-400W 400Вт;

– жорсткий диск Western Digital Blue 500 ГБ.

1.5.4. Вимоги до інформаційної та програмної сумісності

Даний проект має бути розроблений у рушії Unreal Engine 4 та за допомогою кросплатформової мови програмування на мові Java в середовищі середовищі Eclipse SDK, завдяки чому його можна буде використовувати під різними платформами (Android, Windows, IOS тощо).

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

В процесі роботи над кваліфікаційною роботою створено технологію зміни колірних схем у іграх, адаптовану для людей з особливими потребами.

Додаток може бути запропонований розробникам програмного забезпечення як приклад програми, в якій створено комфортні умови гри для різних категорій користувачів.

Програма призначена для реалізації способів створення комфортних умов гри для людей з колірною сліпотою використовуючи технологію, що дозволяє змінити колірну схему на сприятливу із заданими параметрами у рушії Unreal Engine 4, та на основі результатів тесту на дальтонізм враховує індивідуальні особливості кожного користувача.

2.2. Опис застосованих математичних методів

2.2.1. Загальна характеристика технології зміни контрастних схем

Існує керівництво доступності контенту для людей з особливими потребами [15], в якому пропонуються численні рекомендації, спрямовані на забезпечення більшого комфорту в роботі.

Наприклад, візуальне відображення тексту і текст на зображеннях повинні мати коефіцієнт контрастності не менше 7:1.

Коефіцієнт контрастності:

$$CR = (L1 + 0{,}05) / (L2 + 0{,}05), \quad (2.1)$$

де:

L1 – відносна яскравість найсвітлішого з кольорів;

L2 – відносна яскравість найбільш темного з кольорів.

Відносна яскравість будь-якої точки в колірному просторі, нормалізована щодо «0» для самого темного чорного і «1» для самого світлого білого.

У колірному просторі sRGB відносна яскравість кольору визначається як:

$$L = 0{,}2126 * R + 0{,}7152 * G + 0{,}0722 * B, \quad (2.2)$$

де R, G и B визначаються як:

- якщо R_{sRGB} <=0,03928, то R =R_{sRGB}/12,92, інакше

R = ((R_{sRGB}+0,055)/1,055) ^ 2,4;

- якщо G_{sRGB} <=0,03928, то G =G_{sRGB}/12,92, інакше

G = ((G_{sRGB}+0,055)/1,055) ^ 2,4;

- якщо B_{sRGB} <=0,03928, то B =B_{sRGB}/12,92, інакше

B = ((B_{sRGB}+0,055)/1,055) ^ 2,4;

і R_{sRGB}, G_{sRGB}, B_{sRGB} визначається як:

- $R_{sRGB} = R_{8bit}/255$;
- $G_{sRGB} = G_{8bit}/255$;
- $B_{sRGB} = B_{8bit}/255$.

Символ «^» означає операцію піднесення до степеня (формула взята з [15]).

Відомо, що майже всі сучасні системи використовують для відображення контенту кодування sRGB.

Відносна освітленість фонових пікселів безпосередньо біля об’єкту обчислюється також за формулою 2.2.

Взагалі, контролювати світловий контраст зручніше, якщо повністю відмовившись від кольору на початкових етапах проектування. Якщо ж відобразити графіку в відтінках сірого, то навіть користувачі з монохромією зможуть комфортно грати [15]. Відповідно, чим менше кольорів використовуємо, тим менше ймовірність отримати обмежений дизайн. Хоча ніколи не буде зайвим повністю знебарвити вже готовий інтерфейс заради підсумкової перевірки.

2.2.2. Використання HSL моделі

Виділяючи об'єкти, дальтоніки більше спираються на світловий контраст. Він так чи інакше присутній в будь-якому поєднанні кольорів, проте може бути різким або приглушеним [9]. Чим він нижчий, тим складніше людині розрізняти кольори. Тобто, постає потреба створити контрастну та приглушену кольорову схему для того, щоб інтерфейс та графіка була сприятливі для всіх гравців.

HSL (скорочено від англ. Hue, Saturation, Lightness) – колірна модель, в якій будь-який колір визначається трьома характеристиками: кольоровим тоном (англ. Hue), наприклад, синім, червоним, жовтим тощо; насиченістю (англ. Saturation), тобто частиною чистого кольору, без домішки чорної та білої фарб; «світлотою» (en; ru) (англ. Lightness), тобто близькістю до білого кольору.

Модель є нелінійним перетворенням моделі RGB. Колір, представлений в HSV, залежить від пристрою, на який він буде виведений, оскільки HSV – перетворення моделі RGB, яка теж залежить від пристрою. Модель HSV часто використовується в графічних програмах, оскільки зручна для роботи із зображеннями.

Візуальне подання колірної моделі HSV. Тривимірне простір HSV на двомірному екрані зазвичай представлено двома способами: у вигляді колірного кола і у вигляді колірного кільця.

Колірний круг – поперечний переріз циліндра, движок яскравості відповідає висоті циліндра. Візуалізація у вигляді колірного кола здобула широку популярність за першими версіями програм компанії Corel. На даний момент застосовується рідко, частіше використовують кільцеву модель, розроблену компанією Macromedia.

У колірному кільці колірний тон представлений у вигляді райдужного кільця. Насиченість і значення кольору (яскравість) вибираються за допомогою вписаного в це кільце трикутника. Його вертикальна вісь, як правило, регулює

насиченість, а горизонтальна дозволяє змінювати значення кольору. Для вибору кольору на кільці вказується тон, а потім необхідний колір з трикутника.

HSV і сприйняття кольору. Вважається, що модель HSV ближче до людського сприйняття кольорів, ніж RGB і CMYK. RGB і CMYK визначають колір як комбінацію основних кольорів, а HSV відображає інформацію про кольори більш ясною формі: "Що за колір? Наскільки він насичений? Наскільки він світлий чи темний ?"

Колірна модель HSL представляє колір схожим і навіть, можливо, більш зрозумілим чином, чим HSV.

Перевід RGB в HSB (HSV). У ході перетворення значення яскравостей по червоній, зеленій і синій складовій, які задані в діапазоні [0 .. 1], конвертуються в модель HSB (HSV). Отримують значення в наступних діапазонах:

– *H* – колірний тон (0-360°);
– *S* – насиченість (0-1);
– *B*(*V*) – яскравість (0-1).
– *Max* – функція визначення максимуму серед трьох складових *R*, *G*, і *B*.
– *Min* – функція визначення мінімуму серед трьох складових *R*, *G*, і *B*.

Алгоритм перекладу RGB в HSB (HSV) наведено на рис. 2.1:

$$H = \begin{cases} \text{не визначено, якщо } Max = Min; \\ 60^\circ \times \dfrac{G - B}{Max - Min} + 0^\circ, \text{якщо } Max = R \text{ і } G \geq B; \\ 60^\circ \times \dfrac{G - B}{Max - Min} + 360^\circ, \text{якщо } Max = R \text{ і } G < B; \\ 60^\circ \times \dfrac{B - R}{Max - Min} + 120^\circ, \text{якщо } Max = G; \\ 60^\circ \times \dfrac{R - G}{Max - Min} + 240^\circ, \text{якщо } Max = B. \end{cases}$$

$$S = \begin{cases} 0, \text{якщо } Max = 0; \\ 1 - \dfrac{Min}{Max}, \text{в інших випадках} \end{cases}$$

$$V = Max$$

Рис. 2.1. Алгоритм перекладу RGB в HSB (HSV)

2.3. Опис використаної архітектури та шаблонів проектування

Графічний ігровий контент несе в собі кілька функцій: семантичну (смислове, прояснює ігрову механіку) і декоративну (доповнює, урізноманітнення гру). Також, від якості графіки дуже залежить в тому числі і те, чи буде гравець «занурений» в атмосферу гри. Домогтися результату, який задовольняє всім цим вимогам, непросто, але тут накладаються ще й тимчасові рамки, адже створення ігрової графіки – це серйозна проектна задача. Потрібні ресурси: команда, бюджет, багато часу і вміння приймати грамотні рішення. Ігрова графіка має багато точок дотику з різними напрямками графічного дизайну. Очевидна близькість ігрової графіки і архітектури. Процес архітектурного проектування зазвичай ведеться в такому напрямку:

– вивчення функціональних завдань простору;
– художня організація;
– насичення виразними деталями.

Все це актуально і для ігор. Ріднить їх і принцип «Антропоморфності» – проектовані елементи повинні бути відповідні людині, ергономічні (дверні прорізи, сходи і т.п.). Слід відразу уточнити, що в іграх ергономічність повинна бути зручною для гравця, але не для його аватара. Так само як і архітектура, ігрова графіка повинна слідувати функції. Ігрову функцію задає гейм-дизайн, тому хороша ігрова графіка неможлива без гарного гейм-дизайн-документа.

Центральним моментом для графіки в ігрових проектах є поняття art direction. Зазвичай, під цим терміном мається на увазі наступне: графіка в грі повинна бути своєрідна і внутрішньо узгоджена, тобто у грі повинен бути присутнім стиль.

Щоб забезпечити якісний art direction, треба провести комплекс заходів, спрямованих на виготовлення багатого і цільного графічного контенту.

Створення ігрового контенту відбувається за звичайним ланцюжком: підготовка (pre-production), саме виробництво, складання і коригування.

Серед цих заходів однієї з найголовніших фаз є pre-production. саме по собі наявність pre-production – декларація того, що завдання своєрідності і узгодженості графіки ставиться і буде контролюватися. Кожен проект накладає свої обмеження: технологічні, ергономічні, тимчасові та інші. Основні питання, що виникають на цьому етапі – якими виразними засобами будуть передані загальна атмосфери гри, сеттинг, як будуть виглядати персонажі, основні візуальні елементи (логотипи і ключові локації, споруди). Дуже важливо зрозуміти, якими засобами буде реалізований стиль гри. Серед них орнаментальність текстур, найвища деталізація персонажів і інтерфейсу, стримана благородна колірна гамма. Або ж яскраві, сміливі колірні і світлові рішення, насиченість ефектами. Важливо, щоб такий образ або його передчуття був знайдений на самому ранньому етапі і ретельно документований. Підсумком pre-production повинен стати пакет концепт-документів (зазвичай званий art book) і приблизний план робіт на проекті, які обов'язково повинні бути доведені до всіх членів команди (включаючи програмістів). Саме по собі те, що цей образ буде знайдений і витриманий до кінця, незважаючи на всі перипетії, і є ознакою наявності art direction.

Наступною фазою є отримання списків контенту. На цьому етапі треба максимально докладно вивчити GDD, feature-листи, опитати гейм-дизайнера про все у ігровому контенті, при цьому записувати треба все, включаючи, наприклад, всі дерева, чагарники з усіма їх варіантами. Усе, зрозуміло, передбачити не вийде, але зазвичай в GDD на цьому етапі вже ясно описані ключові об'єкти, це вже 70% інформації, що можна вважати високим ступенем керованості проекту.

Стандартними напрямками виробництва ігрового контенту є локації, персонажі, інтерфейс, анімація, об'єкти (ті самі дерева, ліхтарики та інші дрібні деталі), освітлення і ефекти. Після аналізу цих списків може з'ясуватися, що десь доведеться робити конструктори, вводити уніфікацію – для здешевлення розробки (це треба окремо документувати). На базі цих рішень і списків контенту ми отримуємо план виробництва графіки для конкретного проекту.

Локації створюють ігровий простір, задають атмосферу гри, і від того, наскільки добре вони зроблені, багато в чому залежить чи відбудеться «Ефект занурення» гравця. Саме на локації витрачається максимум ресурсів, і цим треба перейматися в першу чергу.

Виробництво локацій починається з карт місій, зроблених гейм-дизайнером. Згідно з цими картками в тривимірних редакторах створюються заготовки локацій, визначається масштаб і положення об'єктів. Уже на цьому етапі необхідно починати ставити світло. Корисно відразу наповнити локацію приблизно тією ж кількістю об'єктів (будиночками, деревами та ін.), потім закріпити результати в ТЗ, з'ясувати, як експорт локації, який верхній стелю для будиночків, сегментів.

Виходячи з цих тестів, формується список технічних вимог до об'єктам для локацій – технічне завдання на різні типи об'ектов.

Обговорення заготовок локацій з гейм-дизайнером дасть інші ключові параметри: яка камера, який масштаб об'єктів і текстур. Одним з найважливіших складових ігровий локації є освітлення. На даному етапі можна відпрацьовувати колірні, світлові і композиційні рішення. Якісь загальні рецепти дати важко, кожен проект дуже індивідуальний, і саме ці настройки визначать те, як буде виглядати гра.

Постановка світла на локаціях потребують великого терпіння, хорошого інструментарію та потужні комплектуючі комп'ютера. Головне – зробити технологічні тести якомога раніше, щоб ця ємна за часом завдання йшла в правильному і перевіреному напрямку.

Потрібна велика кількість скетчів з різними варіантами освітлення на ранній стадії. Однак, є велика різниця між тим, як то або інше освітлення виглядає на скетчі і як воно виглядає в реальності. Як вплине освітлення на текстури, на сприйняття простору. Проводиться величезна кількість тестів, щоб переконатися, що вибране освітлення однаково ефектно виглядає на великих відкритих просторах і в невеликих вузьких коридорах.

2.4. Опис використаних технологій та мов програмування

2.4.1. Аналіз рушіїв для створення ігор

Гральний рушій (від англ. Game engine) – програмний рушій, центральна програмна частина будь-якої відеогри, за допомогою якої розроблюється гра та систематизуються всі її внутрішні структури. Гральний рушій відповідає за всю технічну складову відеогри [19].

Для вибору рушія для реалізації програми були розглянуті три найбільш популярні гральні рушії, за допомогою яких кожен охочий зможе створити свою власну гру:

1. Unity 3D. За допомогою Unity 3D можна створювати ігри різних жанрів. Кожен розробник має можливість легко імпортувати текстури, звуки та моделі. Даний рушій підтримує усі популярні формати зображень, а також надає можливість створювати скрипти за допомогою JavaScript, C# і Boo.

Unity – середовище для створення відеоігор, що має широкий спектр можливостей і зручний та дружній до користувача інтерфейс.

Unity – межплатформенне середовище розробки комп'ютерних ігор. Unity дозволяє створювати додатки, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші. Випуск Unity відбувся в 2005 році і з того часу йде постійний розвиток.

Основними перевагами Unity є наявність візуальної середовища розробки, міжплатформеній підтримки і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами і труднощі при підключенні зовнішніх бібліотек.

На Unity написані тисячі ігор, додатків та симуляцій, які охоплюють безліч платформ і жанрів. При цьому Unity використовується як великими розробниками, так незалежними студіями.

Можна відзначити його основні властивості:

– широкі можливості налаштування;

– доступний і зрозумілий інтерфейс;
– сценарії на C#, JavaScript і Boo;
– повна інтеграція ігрового рушія з середовищем розробки;
– підтримка перетягування об'єктів у редакторі;
– підтримка імпорту великої кількості форматів;
– вбудована підтримка мережі;
– підтримка фізики тканини (PhysX Cloth);
– можливість доповнення функціоналу;
– інструменти для спільної розробки;
– можливість використання систем контролю версій тощо.

2. Construct 2. Construct 2 дозволяє всім охочим створювати 2D-ігри будь-якої складності і будь-якого жанру, навіть не маючи навичок програмування. Ігри, зроблені за допомогою цього рушія, легко експортувати на всі основні платформи – PC, Mac, Linux, браузери з підтримкою HTML5, Android, iOS, Windows Phone, Blackberry 10, Amazon Appstore, Chrome Web Store, Facebook тощо. Підтримка iOS і Android здійснюється завдяки технологіям CocoonJS від Ludei, directCanvas від appMobi і Intel XDK, які використовують апаратне прискорення для збільшення продуктивності ігор в 5-10 разів. Інтерфейс програми інтуїтивно зрозумілий і простий в освоєнні, розробнику гри не потрібно знання з програмування і досвід у розробці програм. Логіка ігор в Construct 2 створюється за допомогою системи подій і пов'язаних з ними дій [13]. У минулому році при створенні творчої роботи ми же тестували його, але як показала практика, нам буде потрібне більш точне налаштування та індивідуальний підхід до особливостей зору та технічних засобів гравця.

3. Unreal engine 4. Програма має вбудовані редактори спрайтів, об’єктів, сценаріїв і кімнат. Написана мовою C ++, рушій дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X; консолей Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PSP, PS Vita, Wii, Dreamcast, GameCube і ін., а також на різних портативних пристроях, наприклад, пристроях Apple (iPad, iPhone), керованих

системою iOS і інших. За допомогою системи візуального створення скриптів Blueprints Visual Scripting можна створювати готові ігри, не написавши жодного рядка коду. У поєднанні зі зручним інтерфейсом це дозволяє швидко виготовляти робочі прототипи. Завдяки графічним можливостям рушія можливе створення більш точної технології зміни кольору для людей з колірною сліпотою.

Для спрощення портування рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic; раніше підтримувалися Glide API, S3 Metal, PowerVR SGL), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше підтримувалися A3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows, також планувалося для Linux і Mac), модулі для роботи з мережею й підтримка різних пристроїв вводу.

2.4.2. Обґрунтування вибору інструментів розробки

Серед рушіїв було обрано рушій Unreal Engine 4, тому що він має більший функціонал у використанні, є багатоплатформенним, має доступний інтерфейс, підтримує завантаження плагінів, використання коду, а також системи blueprints. Тому він цілком відповідає потребам, поставленим в постановці завдання.

Після встановлення обраного рушія одразу можна створювати новий проект Unreal Engine (рис. 2.1).

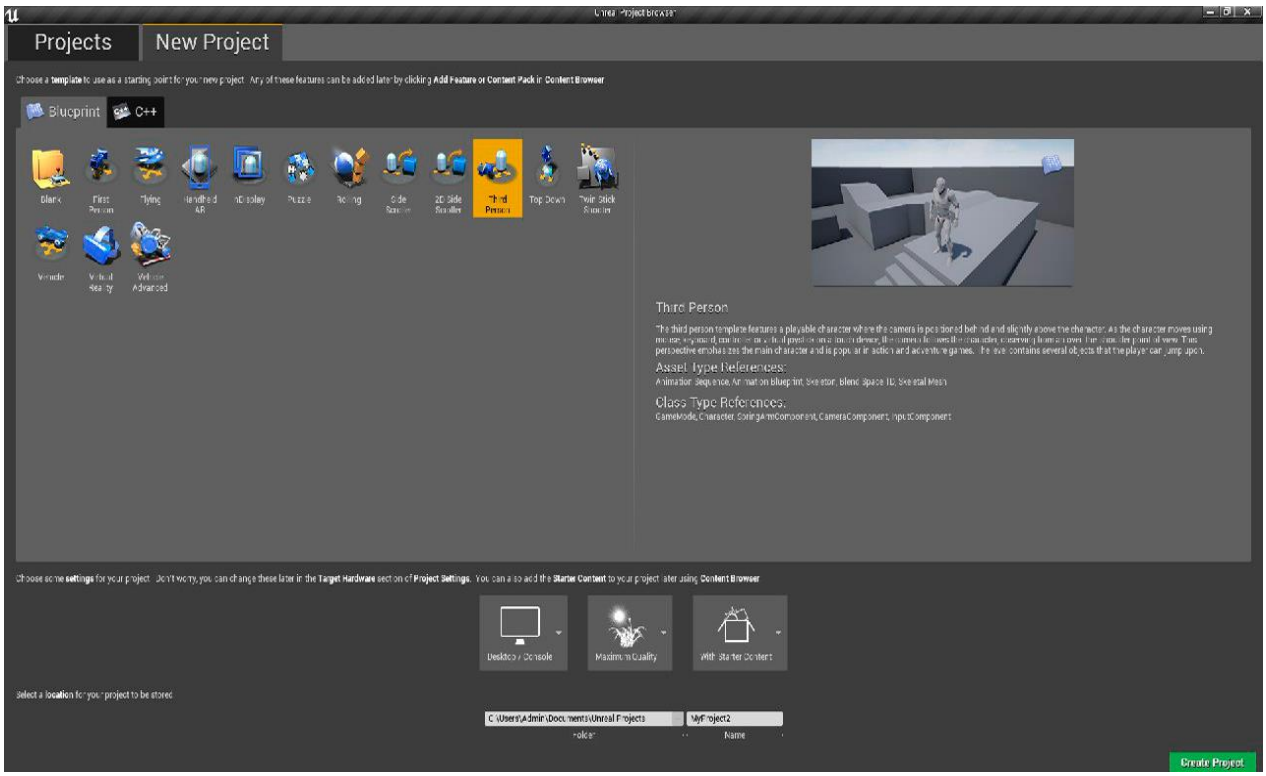

Рис. 2.1 Створення нового проекту

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Створення технології зміни колірної схеми

2.5.1.1. Розробка функцій кольорових схем

Відомо, що майже всі сучасні системи використовують для відображення контенту кодування sRGB.

Досить добре люди з більшістю видів дальтонізму сприймають використання блакитного та помаранчевого кольорів. Тобто гарним рішенням буде додати й таку колірну схему у даний проект. Також потрібно відмовитись від неможливих кольорів (відтінків, які схожі на обидва кольори, наприклад «червоно-зелений» або «жовто-блакитний»). Наприклад, червоний колір стимулює колбочки в сітківці, а зелений – пригнічує їх. Таким чином, червоний і зелений кольори перешкоджають сприйняттю один одного.

Взагалі, контролювати світловий контраст зручніше, якщо повністю відмовившись від кольору на початкових етапах проектування. Якщо ж

відобразити графіку в відтінках сірого, то навіть користувачі з монохромією зможуть комфортно грати [10]. Відповідно, що менше кольорів використовуємо, то менше ймовірність отримати обмежений дизайн. Хоча ми вважаємо, що інколи не буде зайвим повністю знебарвити вже готовий інтерфейс заради підсумкової перевірки.

Для кращого розуміння кольорової сліпоти при створенні технології, була використана програма Color Oracle. Вона є безкоштовнм симулятором сліпоти, працює з такими операційними системами, як Windows, Mac та Linux і відтворює в режимі реального часу те, що побачать люди із загальними порушеннями кольорового зору.

Для початку розробки функцій кольорових схем потрібно створити окремі колірні схеми для кожного з різновидів дальтонізму (рис.2.2), а саме:

– протанопії;
– дейтеранопії;
– тританопії.

Рис. 2.2 Відмінності кольорового сприйняття

Такі окремі колірні схеми для кожного з різновидів дальтонізму допоможуть враховуючи індивідуальні особливості гравця, підлаштувати кольорове відображення його монітору.

Створюючи функцію вибору кольорової схеми, були використані формули [2.1] та [2.2] та рекомендаціями проектування графіки для порушення колірного сприйняття.

Для вирішення задачі створення технології зміни колірної схеми необхідно розробити щонайменше п’яти кольорових схем: монохромної, звичайної та схеми для трьох різновидів дальтонізму(рис. 2.3).

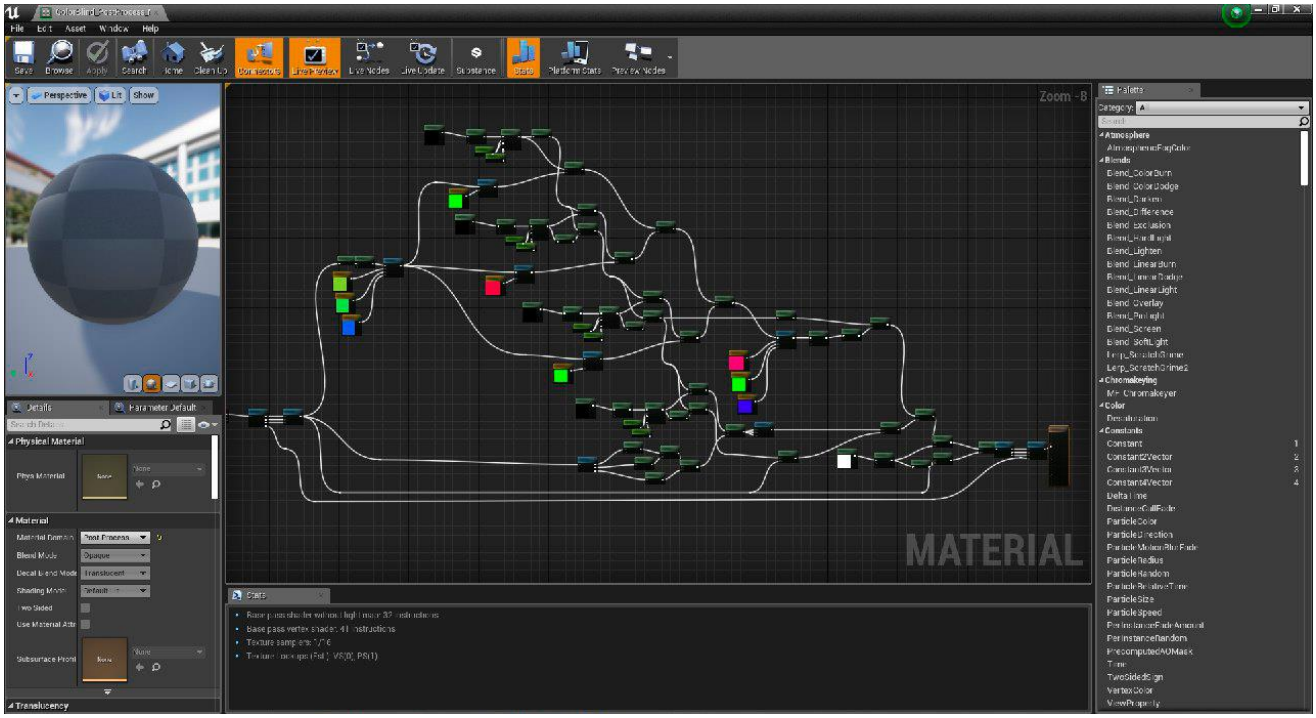

Рис. 2.3 Матеріал і функція технології змін кольору

Працюючи над монохромною схемою, були використали різні степені контрасту та освітлення (рис. 2.4). Для створення схеми різновидів порушень кольорового сприйняття (рис. 2.5-2.7) були використані різні налаштування функції та матеріалу, змінювались контраст, освітленість та колірний тон.

Рис. 2.4 Монохромна схема

Рис. 2.5 Схема для протанопії

Рис. 2.6 Схема для дейтеранопії

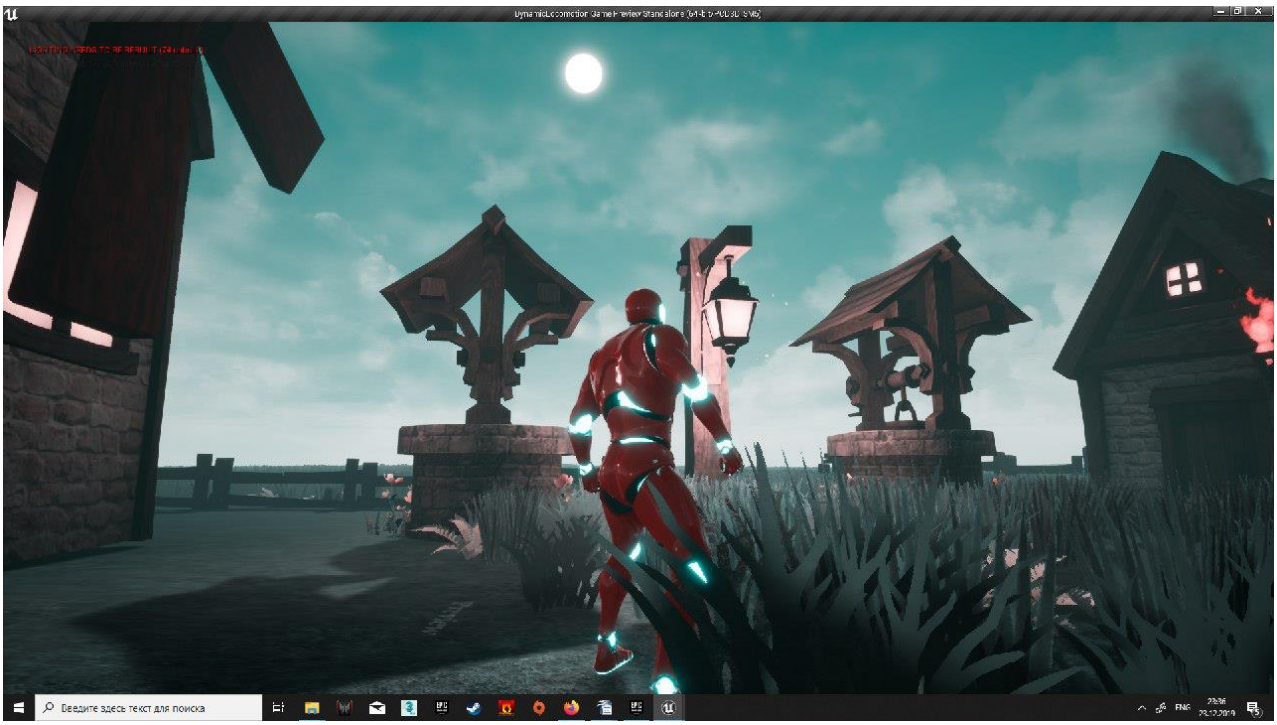

Рис. 2.7 Схема для тританопії

Вигляд стандартного графічного вікна можна побачити на рис. 2.8. В звичайну комфортну схему було додано холодних відтінків для більш комфортної гри в будь-який момент часу (вдень і вночі).

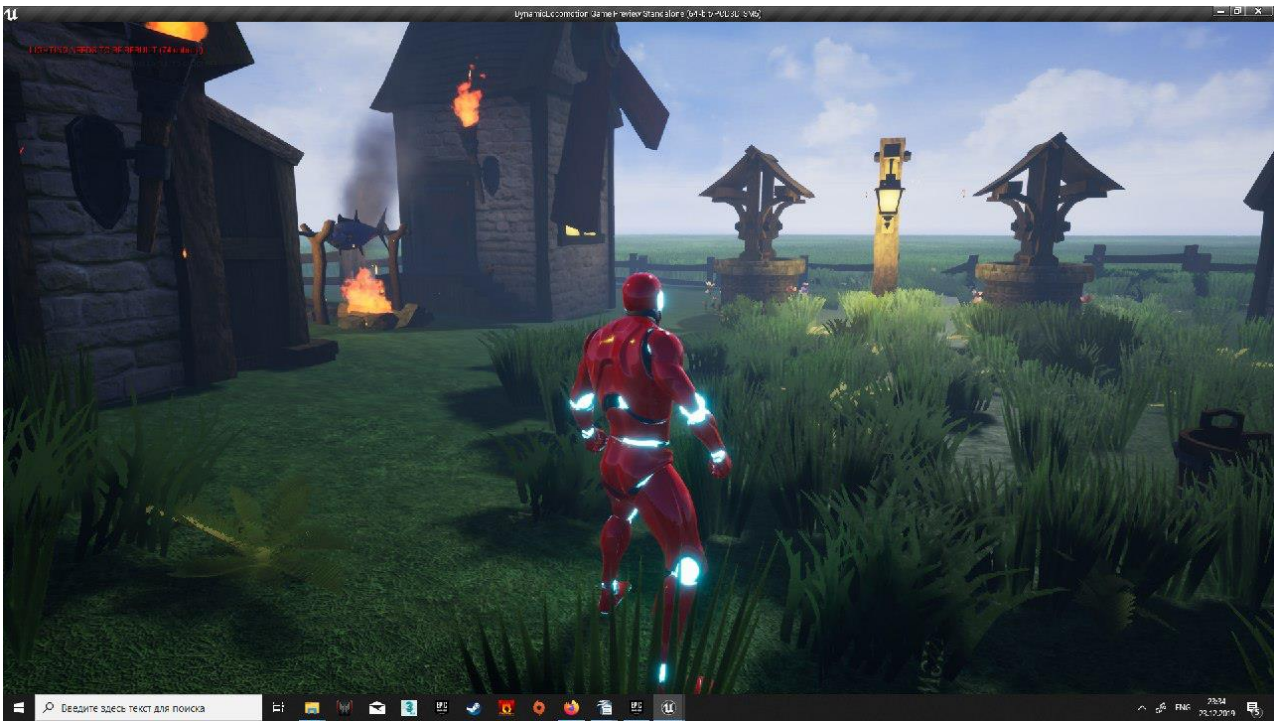

Рис. 2.8 Звичайна комфортна схема

Створюючи функцію вибору кольорової схеми, було використано також плагін AdjustHSL (зміна кольорів схеми HSL).

Працюючи над монохромною, ми використали різні ступені контрасту та освітлення. Для створення схеми з найбільшим контрастом були використані різні налаштування плагіну, змінювались контраст, освітленість та кольоровий тон.

Алгоритм для перетворення між кольором HSL в RGB має наступний вигляд:

```
function hslToRgb(h, s, l){
    var r, g, b;
    if(s == 0){
        r = g = b = l; // achromatic
    }else{
        var hue2rgb = function hue2rgb(p, q, t){
            if(t < 0) t += 1;
            if(t > 1) t -= 1;
```

```
            if(t < 1/6) return p + (q - p) * 6 * t;
            if(t < 1/2) return q;
            if(t < 2/3) return p + (q - p) * (2/3 - t) * 6;
            return p;
        }
        var q = l < 0.5 ? l * (1 + s) : l + s - l * s;
        var p = 2 * l - q;
        r = hue2rgb(p, q, h + 1/3);
        g = hue2rgb(p, q, h);
        b = hue2rgb(p, q, h - 1/3);
    }
    return [Math.round(r * 255), Math.round(g * 255), Math.round(b * 255)];
}
```

Алгоритм для зворотного перетворення між кольором RGB в HSL має наступний вигляд:

```
function rgbToHsl(r, g, b){
    r /= 255, g /= 255, b /= 255;
    var max = Math.max(r, g, b), min = Math.min(r, g, b);
    var h, s, l = (max + min) / 2;
    if(max == min){
        h = s = 0; // achromatic
    }else{
        var d = max - min;
        s = l > 0.5 ? d / (2 - max - min) : d / (max + min);
        switch(max){
            case r: h = (g - b) / d + (g < b ? 6 : 0); break;
            case g: h = (b - r) / d + 2; break;
            case b: h = (r - g) / d + 4; break;
        }
        h /= 6;
```

```
    }
    return [h, s, l];
}
```

2.5.1.2. Використання тесту дальтонізму для індивідуального налаштування схеми

Для того, щоб технологія мала адаптивний та індивідуальний характер було вирішено використати результати проходження гравцем тесту на дальтонізм перед початком гри (рис. 2.9) за методикою тестування з сайту enchroma.com [17]. Даний тест буде запропоновано пройти при першому запуску гри. В залежності від результату тесту буде обрана відповідна схема, яка за бажанням може бути змінена у головному меню. Таким чином,отримано повноцінну технологію адаптації колірних схем, яка буде враховувати індивідуальні особливості та налаштування передачі кольору монітором користувача.

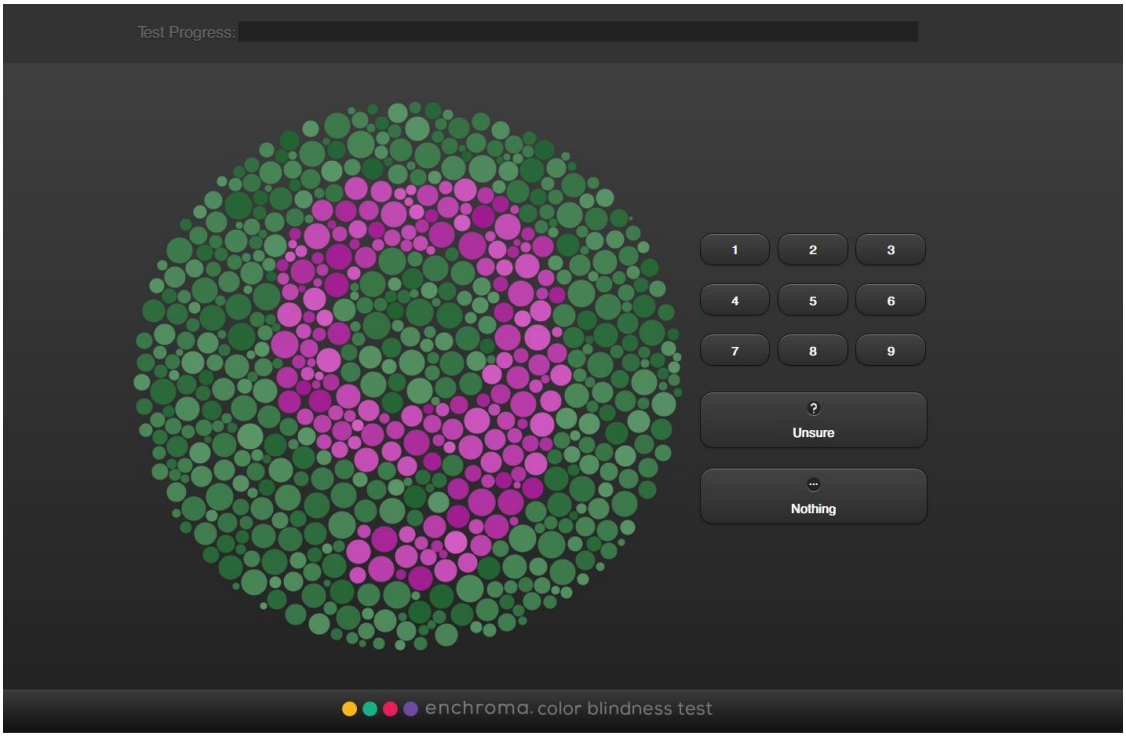

Рис. 2.9. Вигляд тесту на дальтонізм

2.5.2. Створення компонентів та логіки гри

2.5.2.1. Проектування місцевості у грі

Проектування компонентів гри починається з базового елемента, а саме локації гри та обрання текстури для неї. Для цього були використані текстури з сайту itch.io [16], додатково обравши потрібну категорію «Game assets» (рис. 2.10).

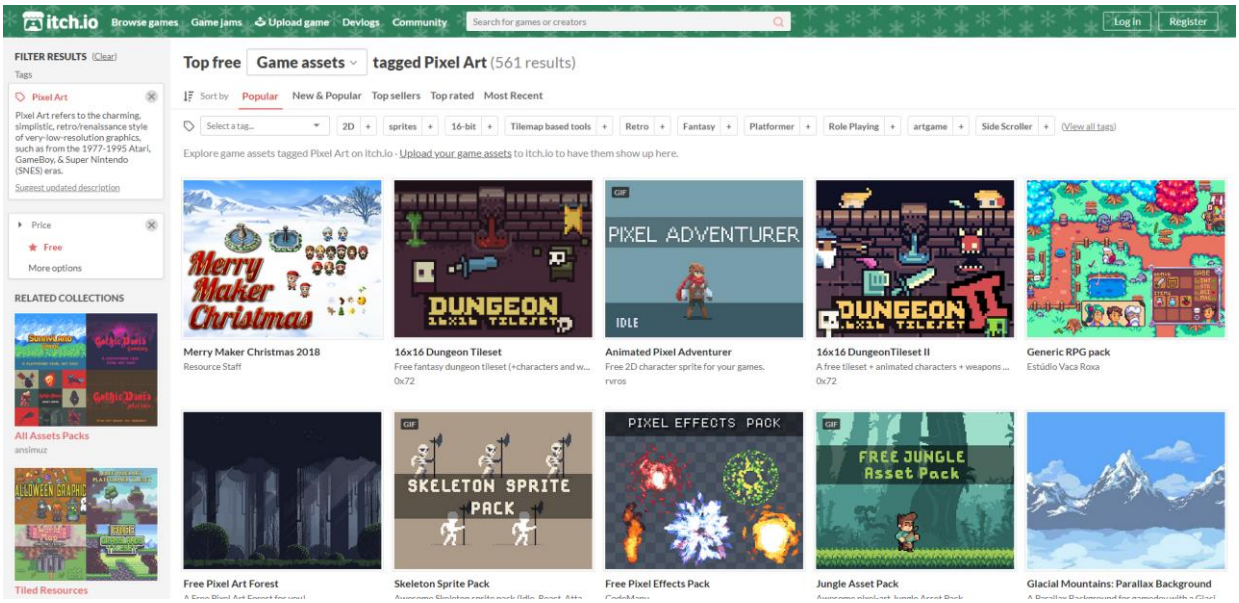

Рис. 2.10. Вибір текстури

Можна зазначити, що однією з переваг двигуна Unreal Engine 4 є безкоштовні бібліотеки текстур та щотижневі роздачі платних текстур високої якості (рис. 2.11).

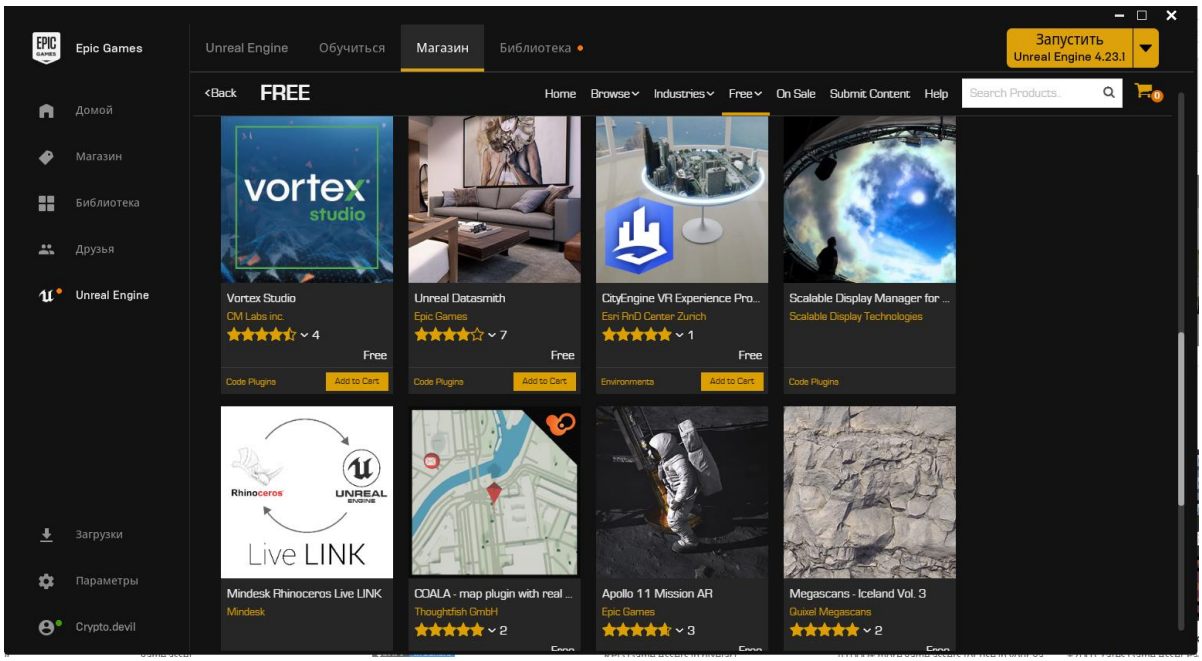

Рис. 2.11. Магазин текстур у Unreal Engine 4

Завантаживши відповідну текстуру, імпортуємо її у проект(Download ->Import -> обираємо проект, до якого імпортуємо ->Import). Після цього, надаємо назву нашій текстури.

Таким же чином знаходимо текстуру для будівель та прикрас (для перевірки роботи технології з більш складними об’єктами).

2.5.2.2. Створення механіки гри та головного меню

Для того, щоб гравець мав змогу у повній мірі оцінити комфортність схем та відчути їх роботу, було створено карту гри та її демо-рівень.

Головна камера «Main Camera» у грі прив’язана до користувача. Таким чином було отримано зображення повноцінного героя, гра з яким буде вестись з видом від третьої особи (рис. 2.12).

Встановлюючи світло було враховано його значимість для кожного об’єкту.

Рис. 2.12. Вид від третьої особи

У даній грі врахована можливість зберігати прогрес гравця при проходженні їм рівня. Алгоритм роботи технології та демо-гри зображено на рис. 2.13-2.14.

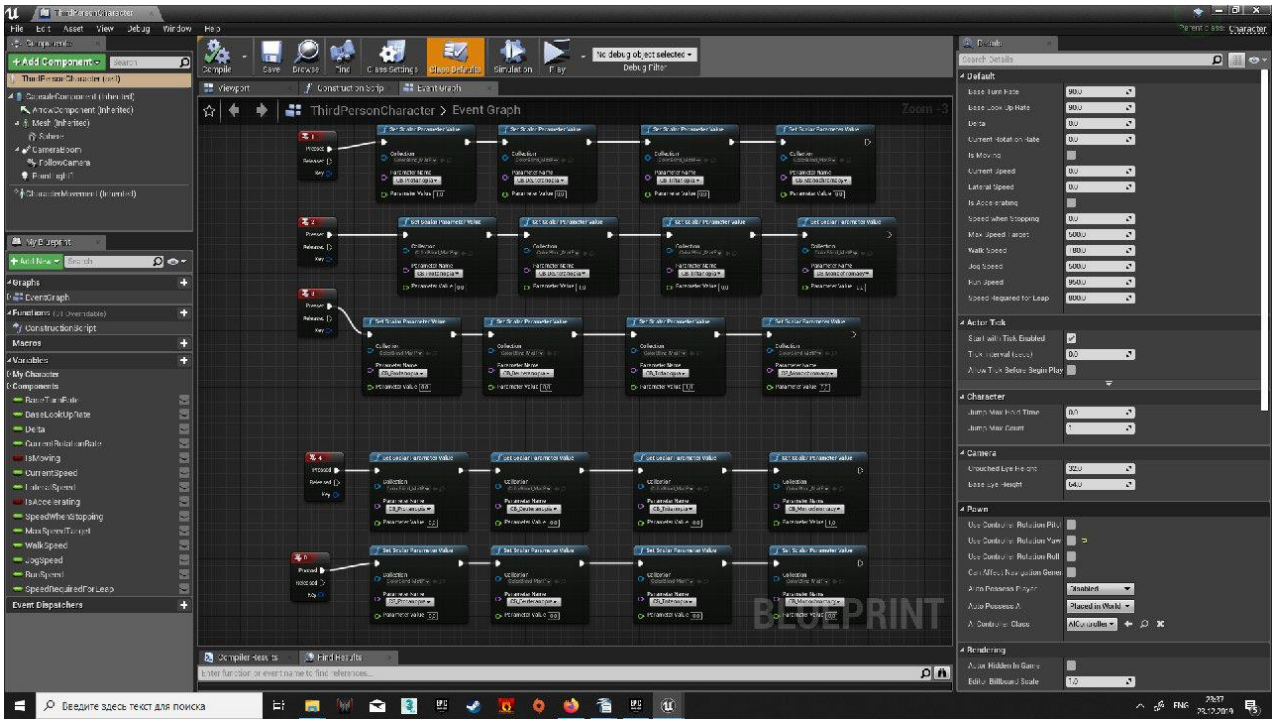

Рис. 2.13. Алгоритм роботи демо-гри та технології адаптації схем

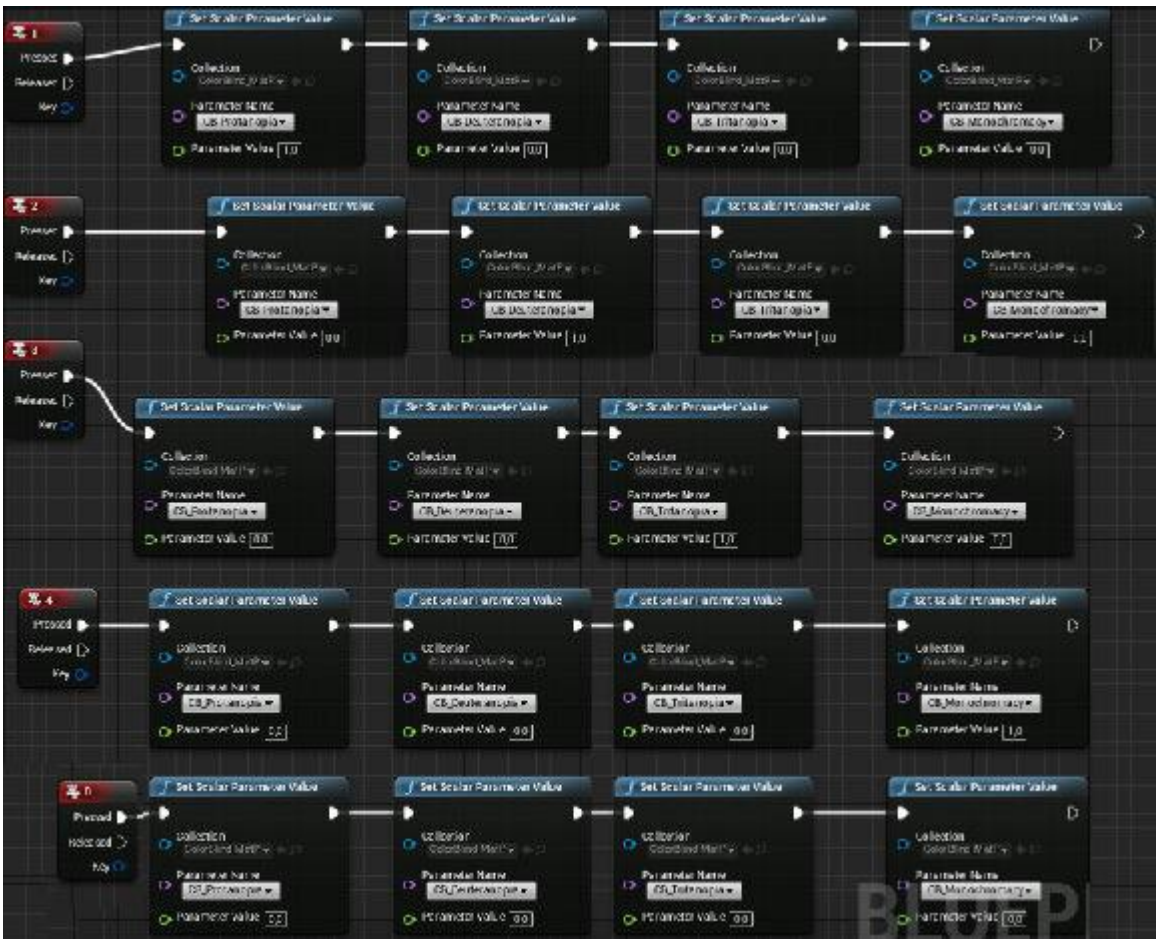

Рис. 2.14. Алгоритм роботи з програмою

Алгоритм роботи «Головного меню», яке надає можливість відпочити та змінити кольорову схему зображено на рис. 2.15.

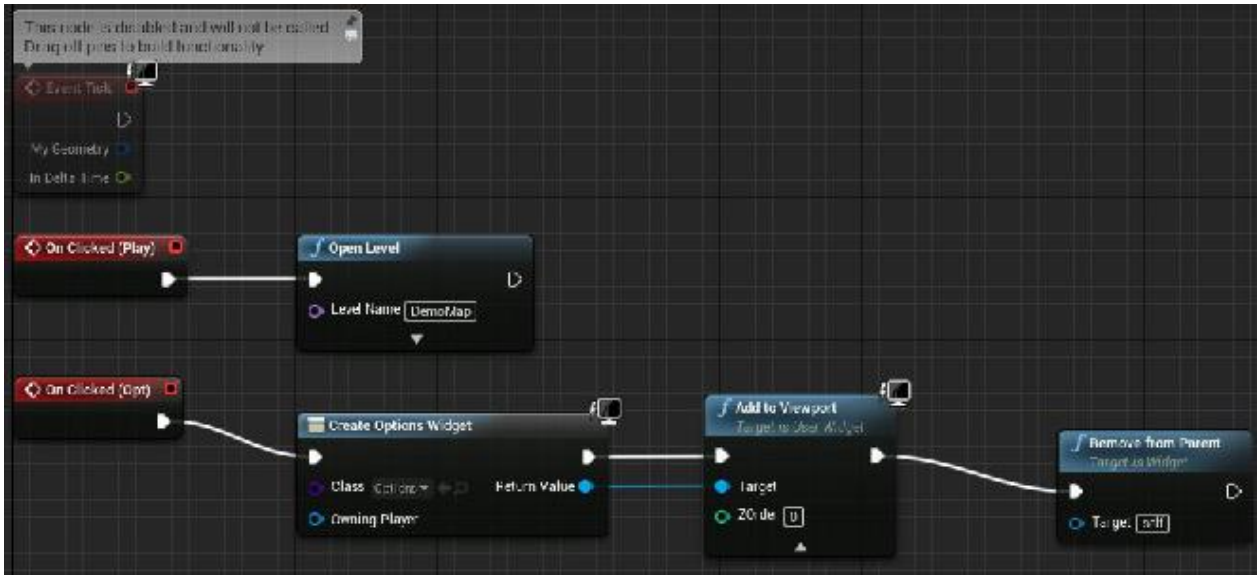

Рис. 2.15. Алгоритм керування меню

2.5.2.3. Алгоритм створення технології адаптації кольорових схем

На рис. 2.16-2.21 наведено алгоритм створення технології адаптації кольорових схем, який розроблено за допомогою функцій, реалізованих в рушії Unreal Engine 4 .

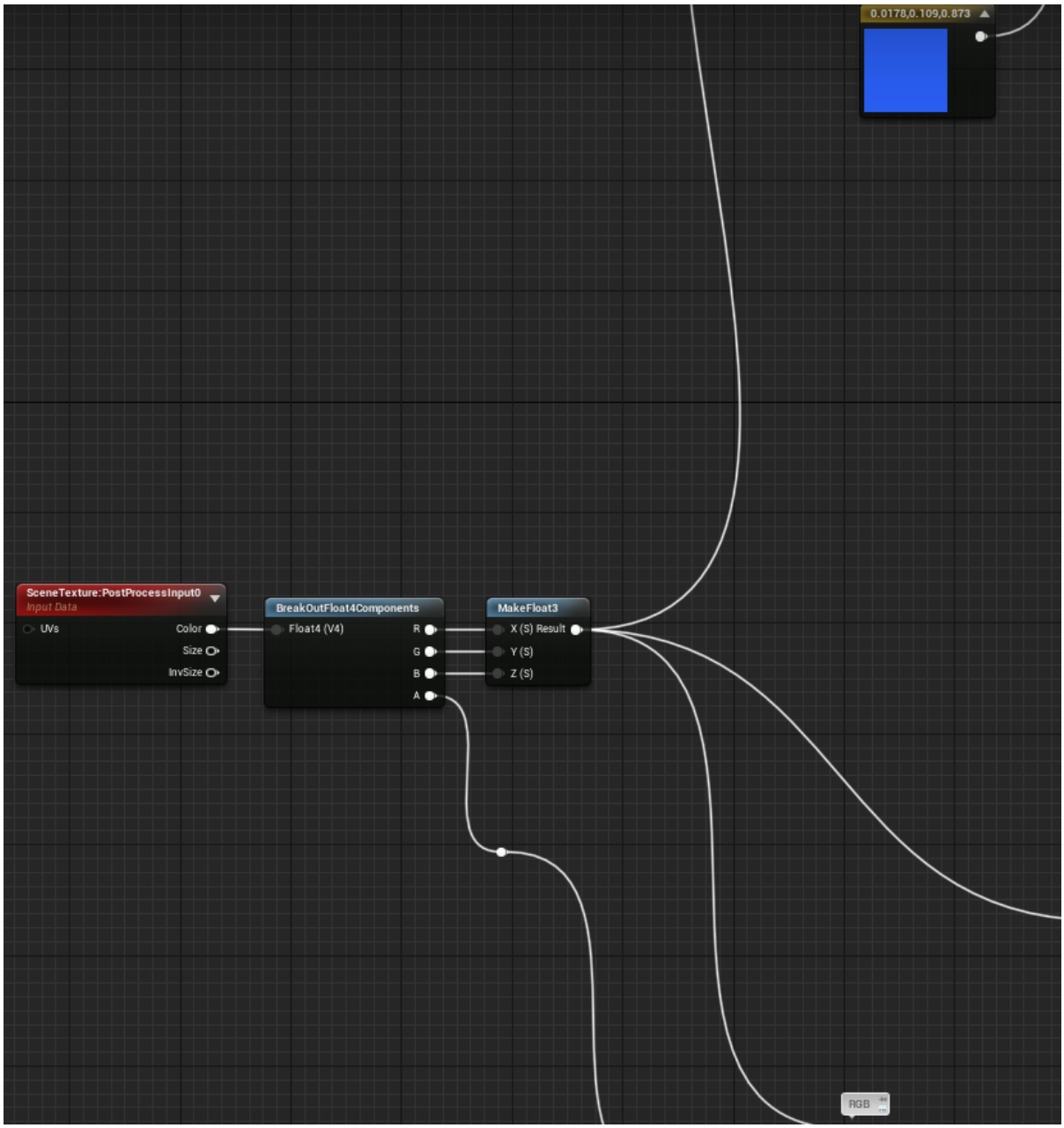

Рис. 2.16. Схема алгоритму технології адаптації кольорових схем. Частина 1

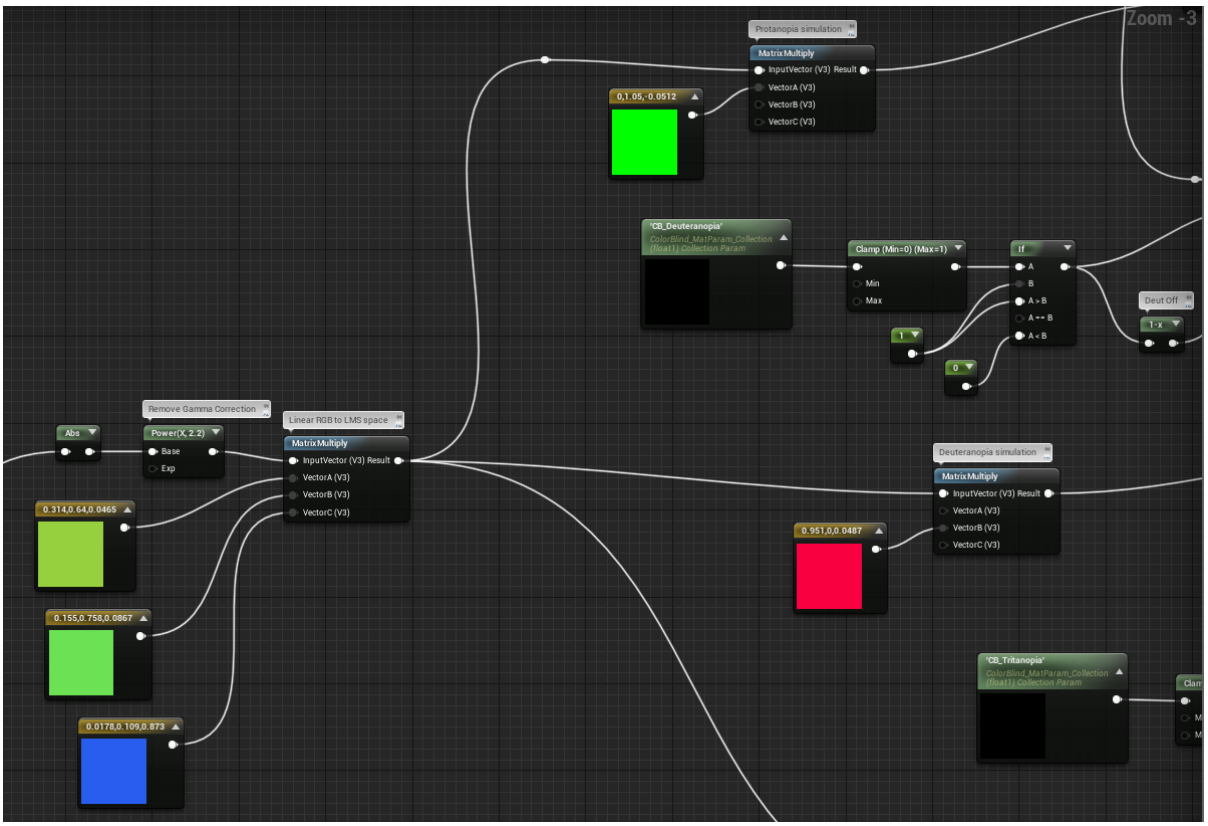

Рис. 2.17. Схема алгоритму технології адаптації кольорових схем. Частина 2

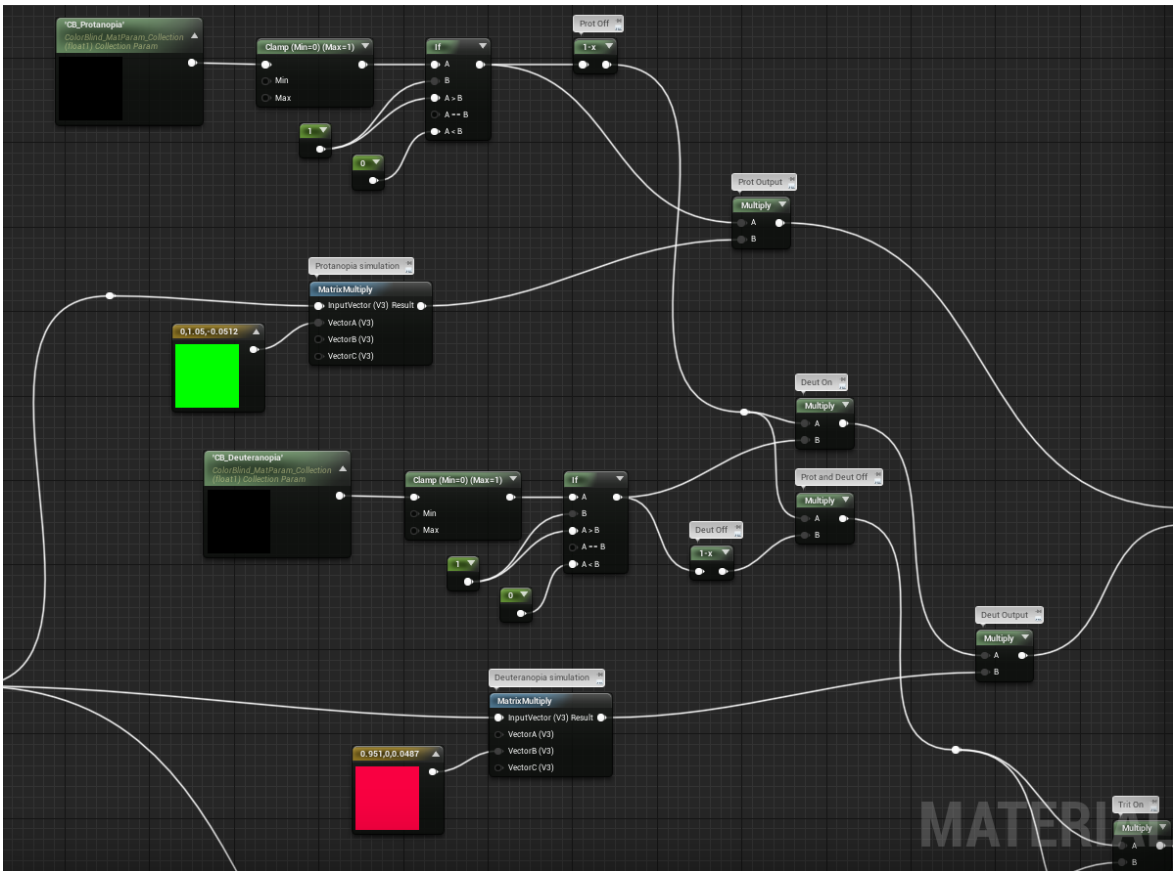

Рис. 2.18. Схема алгоритму технології адаптації кольорових схем. Частина 3

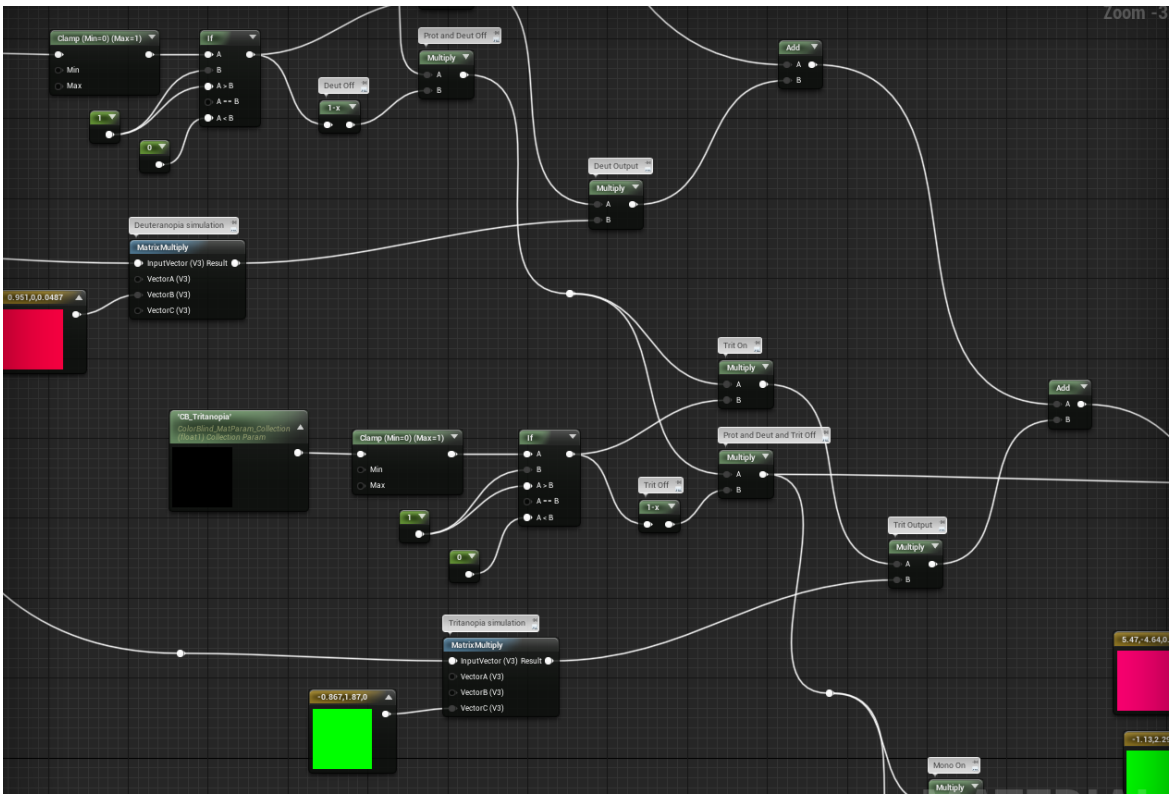

Рис. 2.19. Схема алгоритму технології адаптації кольорових схем. Частина 4

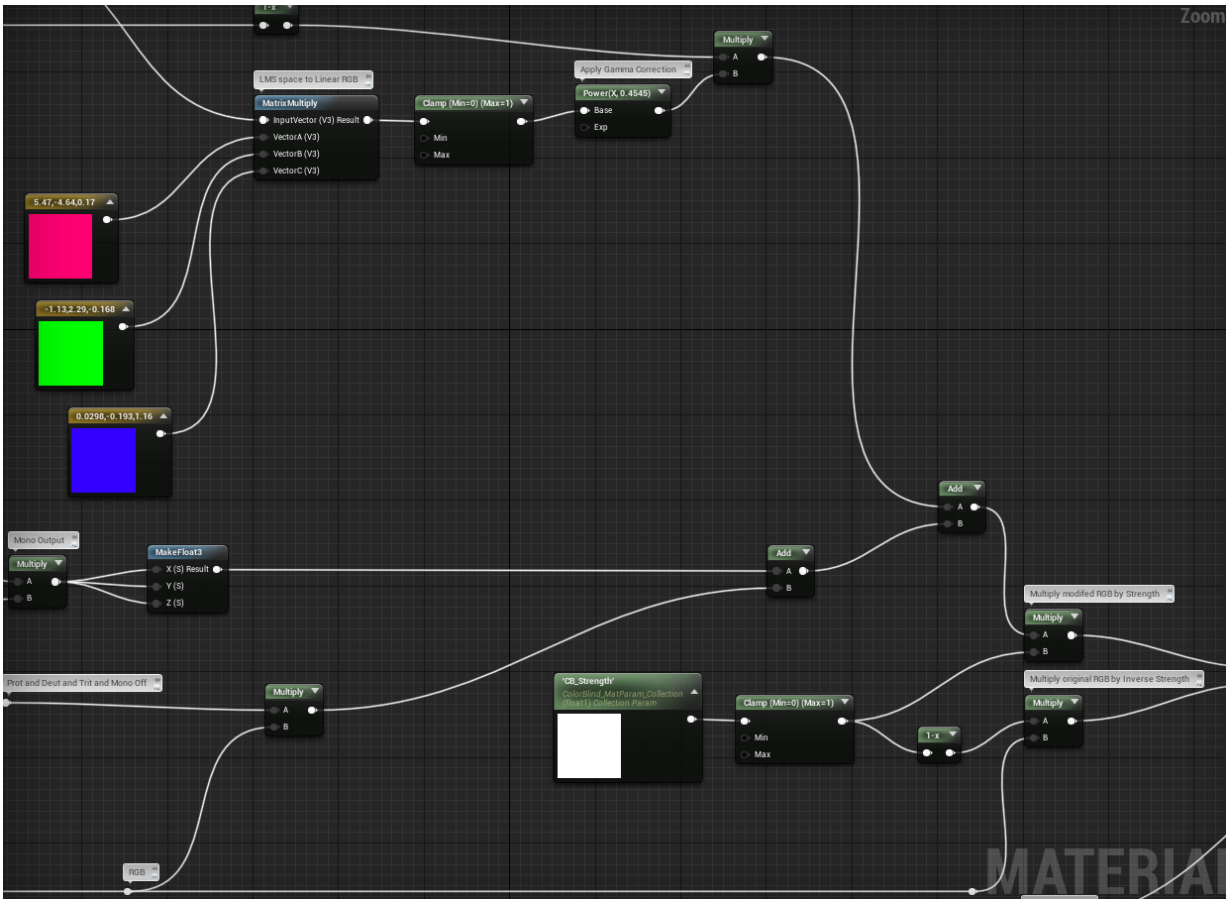

Рис. 2.20. Схема алгоритму технології адаптації кольорових схем. Частина 5

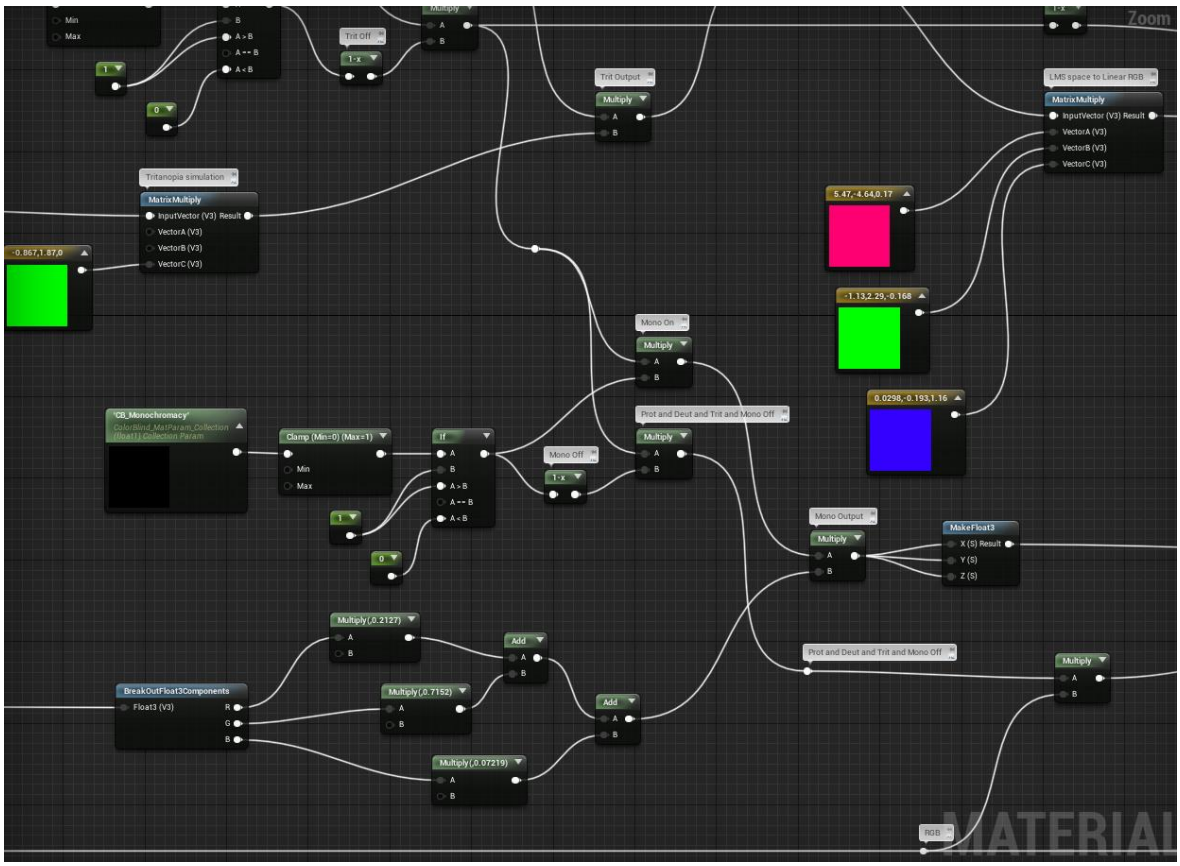

Рис. 2.21. Схема алгоритму технології адаптації кольорових схем. Частина 6

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними для роботи додатку є інформація, яка вводиться та відноситься до групи вхідного потоку:

– початкові дані персонажу;

– команди для керування ходом ігрового процесу, введені за допомогою пристроїв вводу – миші або сенсору.

В результаті обробки цих даних, здійснюється виведення певної інформації або реакція додатку, яка відноситься до групи вихідного потоку, що забезпечує безпосередньо результат роботи на певному етапі функціонування додатку.

Вихідними даними є:

– зображення персонажів гри;

– інформація про поточний стан гри;

– підлаштування колірної схеми гри в залежності від результату проходження тесту на дальтонізм..

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Для розробки даного веб-додатку використовувався ПК з наступними характеристиками:

- тип процесора: процесор з частотою 2.2 ГГц;
- ОЗУ об'ємом 4 Гб;
- 30 Мб доступного простору на жорсткому диску;
- жорсткий диск з частотою обертання 5400 об / хв.;
- дозвіл екрану 1024x768;
- клавіатура, маніпулятор “миша”.
- вихід до мережі Інтернет.

2.7.2. Використані програмні засоби

Дана гра була розроблена на рушії Unreal Engine 4 за допомогою мови програмування Java в середовищі розробки Eclipse SDK Version: Neon (4.6)

2.7.3. Виклик та завантаження програми

Відкомпільована гра запускається виконуванням файлу Daltonizm.exe

2.7.4. Опис інтерфейсу користувача

При першому запуску гри користувачу буде запропоновано пройти тест на колірне сприйняття (рис. 2.22). В залежності від результату тесту буде обрана

відповідна схема, яка за бажанням може бути змінена у головному меню (рис. 2.23 – 2.24). Таким чином,отримано повноцінну технологію адаптації колірних схем, яка буде враховувати індивідуальні особливості та налаштування передачі кольору монітором користувача.

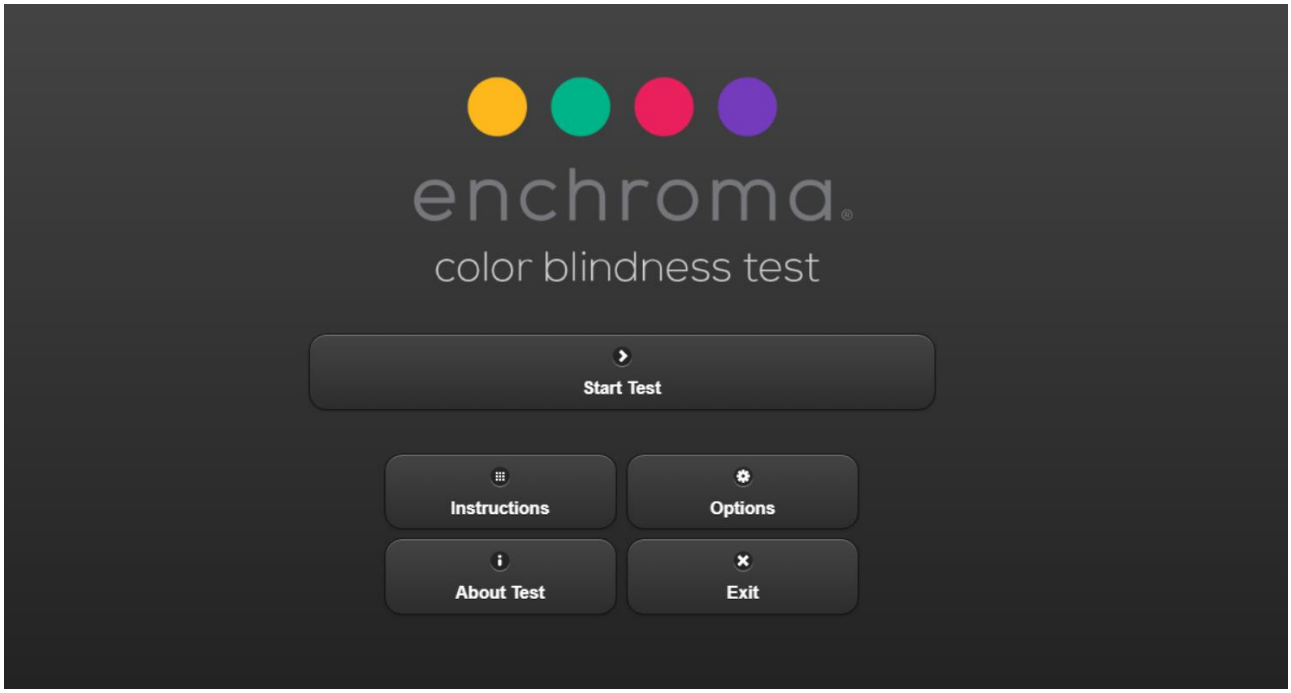

Рис. 2.22. Меню проходження тесту

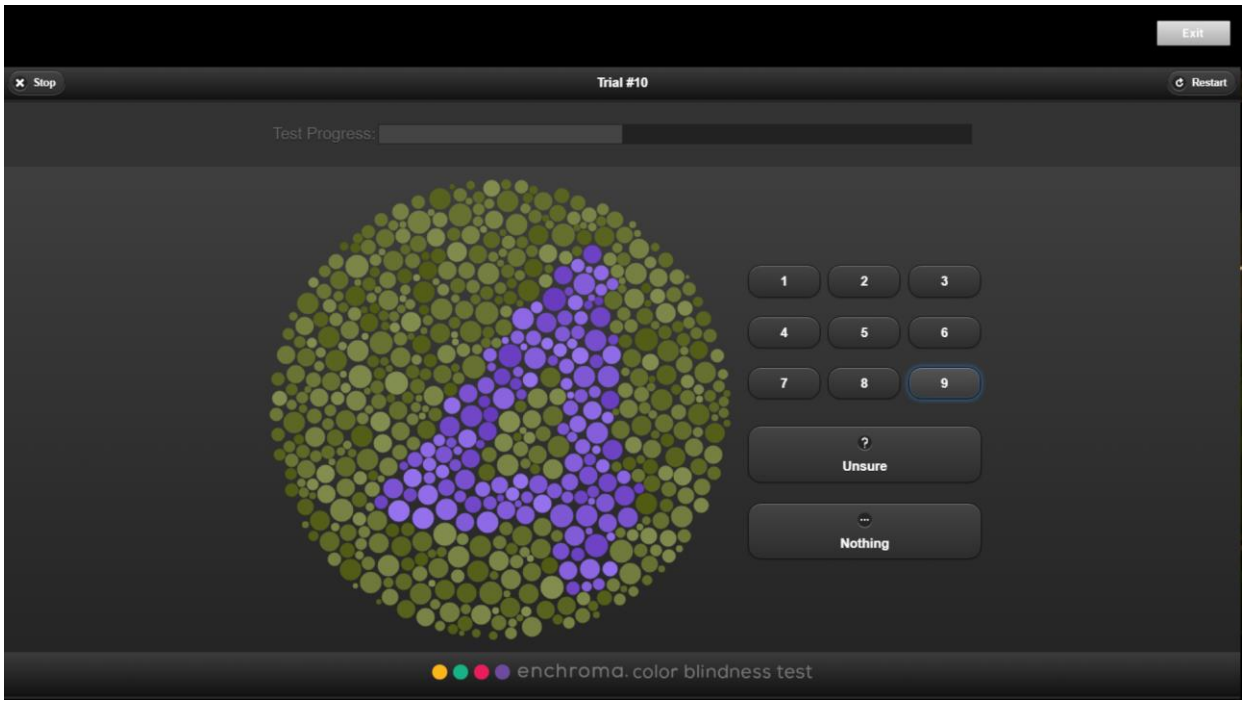

Рис. 2.23. Зображення проходження тесту на колірне сприйняття

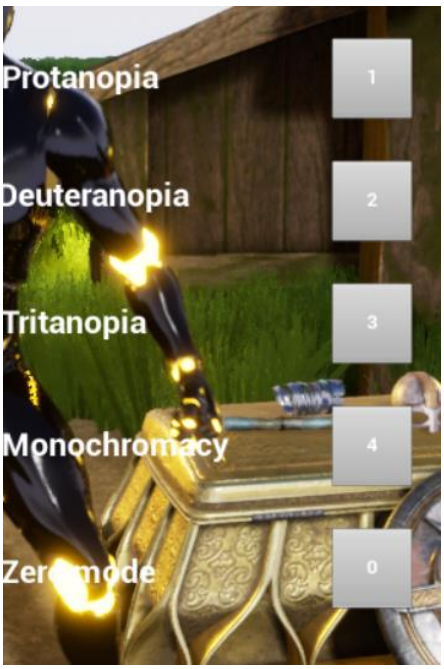

Рис. 2.24. Результат проходження тесту та налаштування під нього системи

Головне меню гри наведено на рис. 2.25.

Рис. 2.25. Вигляд головного меню

Головне меню відкриває доступ до налаштувань (рис. 2.26), початку гри та виходу з неї.

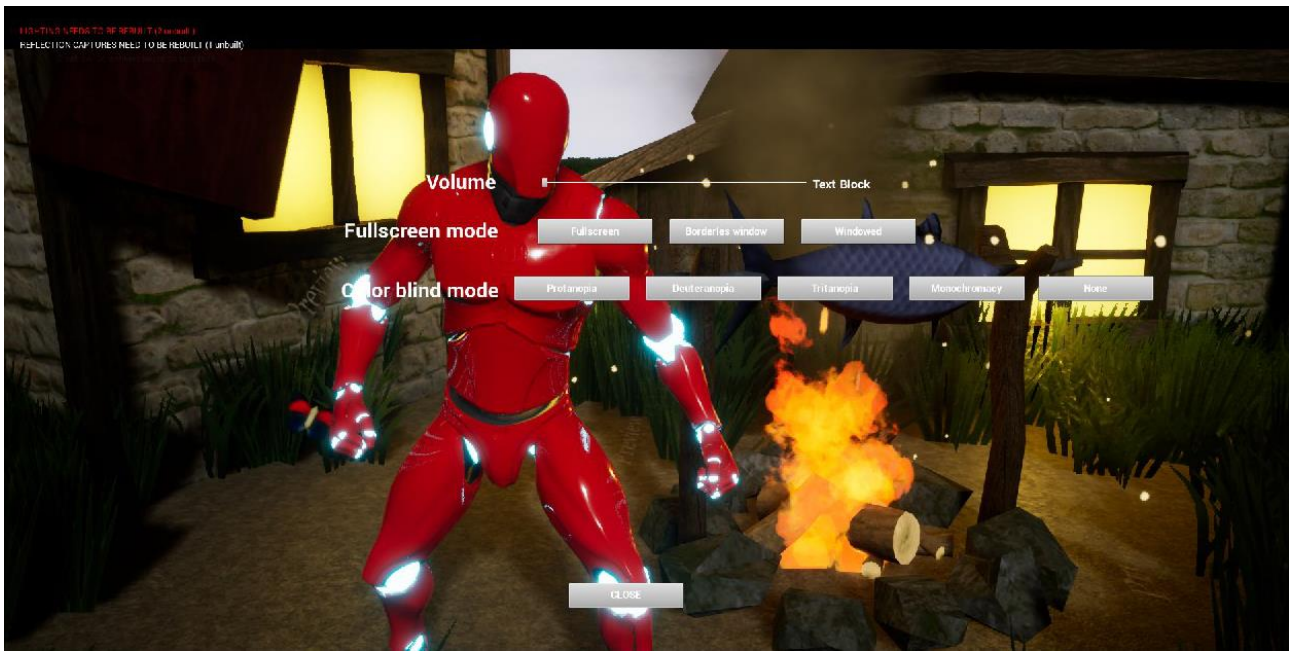

Рис. 2.26. Меню налаштувань системи

Гра складається з одного режиму. Гравець буде переміщатися по мапі, мати діалоги, інвентар та сутички з ворогами.

Зображення ігрового процесу, виконаного з використанням різних кольорових схем, наведено на рис. 2.27-2.32.

Рис. 2.26. Зображення ігрового процесу

Рис. 2.27.Зображення ігрового процесу з додаванням світла

Рис. 2.28. Зображення процесу гри

Рис. 2.29. Зображення ігрового процесу в монохромному режиму

Рис. 2.30. Зображення ігрового процесу для протанопії

Рис. 2.31. Зображення ігрового процесу для дейтеранопії

Рис. 2.32. Зображення ігрового процесу для тританопії

Також передбачено «Меню паузи» для того, щоб була можливість відпочити та змінити кольорову схему для людей з особливими потребами.

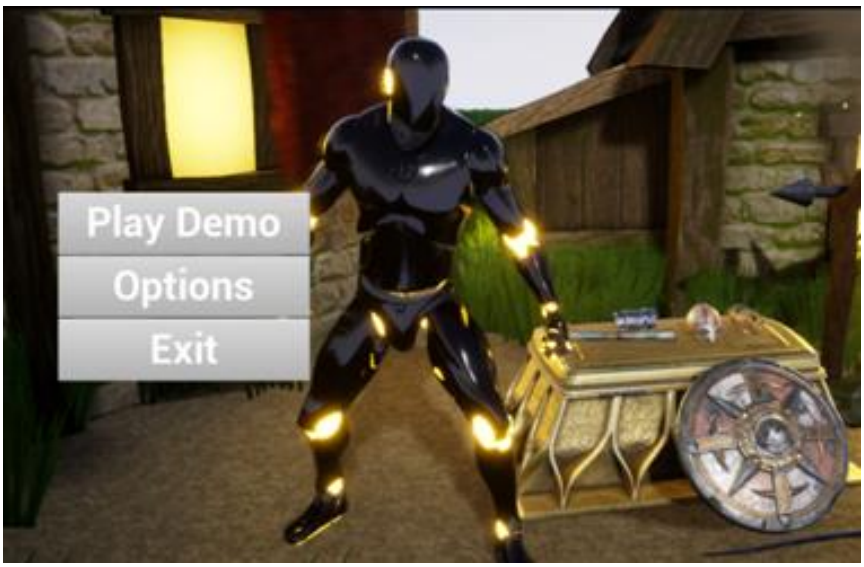

Рис. 2.33. Меню паузи гри

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів – 1500;
- коефіцієнт складності програми – 2;
- коефіцієнт корекції програми в ході її розробки – 0,08;
- годинна заробітна плата програміста, грн./год – 100.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де to – витрати праці на підготовку й опис поставленої задачі (50),

ти – витрати праці на дослідження алгоритму рішення задачі,

та – витрати праці на розробку блок-схеми алгоритму,

тп – витрати праці на програмування по готовій блок-схемі,

тотл – витрати праці на налагодження програми на ЕОМ,

тд – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \quad (3.2)$$

де q - передбачуване число операторів,

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

$$Q = 1500 \cdot 2 \cdot (1 + 0{,}08) = 3240 \text{ людино-годин.} \quad (3.3)$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75...85)K}, \text{ людино-годин,} \quad (3.4)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; B=1.2 … 1.5,

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{3240 \cdot 1{,}3}{80 \cdot 1{,}2} = 44, \text{ людино-годин.} \quad (3.5)$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \quad (3.6)$$

$$t_a = \frac{3240}{22 \cdot 1{,}2} = 123 \text{ людино-годин.} \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \tag{3.8}$$

$$t_n = \frac{3240}{23 \cdot 1{,}2} = 117 \text{ людино-годин.} \tag{3.9}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4...5)K} \quad \text{людино-годин.} \tag{3.10}$$

$$t_{отл} = \frac{3240}{5 \cdot 1{,}2} = 540 \text{ людино-годин.} \tag{3.11}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \tag{3.12}$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K}, \text{ людино-годин.} \tag{3.13}$$

$$t_{др} = \frac{3240}{18 \cdot 1{,}2} = 150 \text{ людино-годин,} \tag{3.14}$$

де tдо - трудомісткість редагування, печатки й оформлення документації.

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин.} \tag{3.15}$$

$$t_{до} = 150 + 73 = 223 \text{ людино-годин.} \tag{3.16}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 44 + 123 + 117 + 540 + 223 = 1097 \text{ людино-годин.} \tag{3.17}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.18)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.19)$$

де t - загальна трудомісткість, людино-годин,

Спр - середня годинна заробітна плата програміста, грн/година.

$$Ç_{ÇÏ} = 1097 \cdot 100 = 109700 \text{ ãðí.} \quad (3.20)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \times C_{М}, \text{ грн,} \quad (3.21)$$

де tотл - трудомісткість налагодження програми на ЕОМ, год.,

Смч - вартість машино-години ЕОМ, грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 540 \times 5 = 2700 \text{ грн.}, \quad (3.22)$$

$$Êïï = 2700 + 109700 = 102400 \text{ ãðí.} \quad (3.23)$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.,} \tag{3.24}$$

де Bk - число виконавців,

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1097}{1 \cdot 176} = 6.2 \text{ міс.} \tag{3.25}$$

Визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений розрахунок вартості роботи по створенню програми (102400 грн) та визначено час на створення (6,2 міс).

ВИСНОВКИ

Аналіз сучасного стану ринку програмного забезпечення показав, що є дефіцит як серед програм адаптації кольору для людей з вадами зору, так і серед ігор, які дозволяють користувачам з дальтонізмом обирати комфортний саме для них інтерфейс гри.

Головним результатом роботи є завершена технологія змін кольорових схем у іграх, що дозволяє людям з вадами зору змінювати їх для власної зручності.

Додаток може бути запропонований розробникам програмного забезпечення як приклад програми, в якій створено комфортні умови гри для різних категорій користувачів.

Запропонована демо-версія гри, яка на основі результатів тесту на дальтонізм враховує індивідуальні особливості кожного, допоможе всім зацікавленим весело та з користю організувати своє дозвілля.

В майбутньому створена технологія може використовуватись у різних галузях життя людини, а не тільки при розробці та адаптуванні комп’ютерних ігор.

В економічному розділі визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (102400 грн) та розраховано час на його створення (6,2 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

2. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2021.

3. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

4. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

5. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

6. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

7. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 121 «Програмна інженерія» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

8. Сайт Як проектувати для людей з колірною сліпотою URL: https://medium.com/@churakova_nastya/как-проектировать-для-людей-сцветовой-слепотой-5bbf9790748e дата звернення: 15.03.2021.

9. Сайт Дальтонізм у інтерфейсах: як створити доступний URL: https://vc.ru/design/44115-daltonizm-v-interfeysah-kak-sozdavat-dostupnyy-dizayn дата звернення: 15.03.2021.

10. Сайт Krita URL: https://krita.org/en/ дата звернення: 15.03.2021.

11. Сайт MOBA URL: https://uk.wikipedia. org/wiki/ MOBA дата звернення: 15.03.2021.

12. Сайт Color Contrast Analyzer URL: http://tools.cactusflower.org/analyzer/ дата звернення: 15.03.2021.

13. Сайт Керівництво по забезпеченню доступності веб-контенту URL: https://www.w3.org/Translations/WCAG20-ru/#content-structure-separation дата звернення: 15.03.2021.

14. Сайт з інді-іграми онлайн безкоштовно URL: https://itch.io/ дата звернення: 15.03.2021.

15. Сайт з методикою тесту на дальтонізм URL: https://enchroma.com/pages/color-blindness-test?format2=number#test дата звернення: 15.03.2021.

16. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К.: Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

17. Симонович С.В., Євсєєв Г.А. Цікавий комп’ютер: Книга для дітей, вчителів, батьків – М.: АСТ-ПРЕСС КНИГА - 2001. -368 с.

ДОДАТОК А

КОД ПРОГРАМИ

logik.jc

```
import types from './Types';
import Render from './Render';

import ExportCSHARP from './export/csharp';

const version = "1.0.0";
const defaults = [];

export default ({store}) => {
    const openfile = document.createElement('input');
    openfile.type = 'file';
    openfile.accept = '.sfep';
    openfile.style = 'display:none';
    document.body.appendChild(openfile);
    let canvas = null;
    let render = null;
    const getMousePos = () => {
        if(window.SF.mouse.length != 0){
            let rect = canvas.getBoundingClientRect();
            return [
                window.SF.mouse[0] - rect.left,
                window.SF.mouse[1] - rect.top
            ];
        }
        else return null;
    };
    const getScaledMousePos = () => {
        let mousePos = getMousePos();
        if(mousePos != null){
            return [
```

```
            mousePos[0] / window.SF.scale + window.SF.camera[0] / window.SF.scale,
            mousePos[1] / window.SF.scale + window.SF.camera[1] / window.SF.scale
        ];
    }
    else return null;
}
const getClickedObjectId = (x, y) => {
    for (const [key, value] of Object.entries(window.SF.episode).reverse()) {
        if(x >= value.position.x && x <= value.position.x + types[value.type].width && y >= value.position.y && y <= value.position.y + types[value.type].height) return key;
    }
    return null;
}
const getClickedPoint = (x, y) => {
    for (const [key, value] of Object.entries(window.SF.episode).reverse()) {
        const array = types[value.type].points;
        for (let index = 0; index < array.length; index++) {
            const point = array[index];
            if(x >= value.position.x + (point.position.x - 0.5) && x <= value.position.x + (point.position.x + 0.5) && y >= value.position.y + (point.position.y - 0.5) && y <= value.position.y + (point.position.y + 0.5)) return { module: {id: key, type: value.type}, point: {index, data: point}};
        }
    }
    return null;
}
const isvalid = (obj) => {
    try{
        if(typeof obj == "object"){
            for (const [key, value] of Object.entries(obj)) {
                if(typeof key != "string") return false;
                if(value.type == undefined) return false;
                if(types[value.type] == undefined) return false;
                if(value.position == undefined) return false;
                if(value.position.x == undefined) return false;
```

```
            if(value.position.y == undefined) return false;
            if(value.points == undefined || typeof value.points != 'object') return false;
        }
        return true;
    }
    else return false;
  }
  catch{
    return false;
  }
};
const updateEpisode = () => {
  localStorage.setItem("episode", JSON.stringify(window.SF.episode));
};
let saved = localStorage.getItem("episode");
const moveToEnd = (id) => {
  let newep = {};
  for (const [key, value] of Object.entries(window.SF.episode)) {
    if(key != id) newep[key] = value;
  }
  for (const [key, value] of Object.entries(window.SF.episode)) {
    if(key == id) newep[key] = value;
  }
  window.SF.episode = newep;
  updateEpisode();
}
const clearPoint = (id, index) => {
  delete window.SF.episode[id].points[index];
  for (const [key, value] of Object.entries(window.SF.episode)) {
    for (const [key2, value2] of Object.entries(value.points)) {
      if(value2.id == id && value2.index == index)
        delete window.SF.episode[key].points[key2];
    }
  }
}
```

```
    const select = (id) => {
        if(id != undefined && window.SF.episode[id] != undefined){
            window.SF.selectedId = id;
            moveToEnd(window.SF.selectedId);
            store.commit('selectId', window.SF.selectedId);

            /*let html = "<p><b>Selected:</b> " + types[episode[selectedId].type].menu.name +
"<br><b>Description:</b><br>" + types[episode[selectedId].type].menu.description + "</p>";
            if(!types[episode[selectedId].type].default) html += '<btn class="remove-btn"
onclick="nav.objects.remove(\" + id + '\')">Remove</btn>';
            $("controls").html(html);*/
        }
        else{
            window.SF.selectedId = "";
            store.commit('selectId', window.SF.selectedId);
            //$("controls").html("<b>Nothing is selected<b/>");
        }
    }
    select();
    const updateCanvas = (c, cf) => {
        if(!render) render = Render({cframe: cf, canvas: c, types, getMousePos, getScaledMousePos,
getClickedObjectId, getClickedPoint, updateEpisode, clearPoint, select});
        canvas = c;
        canvas.addEventListener("mousemove", (evt) => {
            window.SF.mouse = [evt.clientX, evt.clientY];
        });
        canvas.addEventListener("mouseleave", () => {
            window.SF.mouse = [];
        });
        render.updateCanvas(c, cf);
    };
    const makeNewEpisode = () => {
        window.SF.episode = {};
        defaults.forEach(el => {
            window.SF.episode[el.type] = el;
```

```
        });
        updateEpisode();
    }
    const makeId = (length) => {
        var result           = '';
        var characters       =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
        var charactersLength = characters.length;
        for ( var i = 0; i < length; i++ ) {
          result += characters.charAt(Math.floor(Math.random() * charactersLength));
        }
        return result;
    }
    const makeUniqueId = () => {
        let result = "";
        while(true){
            result = makeId(8);
            let isUnique = true;
            for (const [key] of Object.entries(window.SF.episode)) {
                if(key == result) isUnique = false;
            }
            if(isUnique == true) break;
        }
        return result;
    }
    const addObj = (type) => {
        if(types[type] != undefined){
            window.SF.episode[makeUniqueId()] = {type: type, position: {x:
Math.round((window.SF.camera[0] + canvas.width / 2) / window.SF.scale), y:
Math.round((window.SF.camera[1] + canvas.height / 2) / window.SF.scale)}, points: {}}
        }
    }
    const removeObj = (id) => {
        if(window.SF.episode[id] != undefined){
            for (const [key, value] of Object.entries(window.SF.episode)) {
```

```
            for (const [key2, value2] of Object.entries(value.points)) {
                if(value2.id == id)
                    delete window.SF.episode[key].points[key2];
            }
        }
        select();
        delete window.SF.episode[id];
    }
}
window.SF.removeObj = removeObj;
window.SF.updateEpisode = updateEpisode;

store.state.navbar.episode.push({
    click: () => {makeNewEpisode()},
    lang: "newep",
    class: {"fa-file": true}
});
store.state.navbar.episode.push({
    click: () => {openfile.click()},
    lang: "open",
    class: {"fa-folder-open": true}
});
openfile.addEventListener('change', () => {
    var reader = new FileReader();
    reader.readAsText(openfile.files[0], "UTF-8");
    reader.onload = (e) => {
        try{
            if(isvalid(JSON.parse(e.currentTarget.result).content)) window.SF.episode =
JSON.parse(e.currentTarget.result).content;
            updateEpisode();
        }
        catch{}
    };
});
store.state.navbar.episode.push({
```

```
        click: () => {
            let blobData = new Blob([JSON.stringify({meta: {version}, content: window.SF.episode})],
{type: "text/plain"});
            let url = window.URL.createObjectURL(blobData);
            let a = document.createElement("a");
            a.style = "display: none";
            document.body.appendChild(a);
            a.href = url;
            a.download = "episode.sfep";
            a.click();
            window.URL.revokeObjectURL(url);
            a.remove();
        },
        lang: "saveas",
        class: {"fa-save": true}
    });
    store.state.navbar.episode.push({
        click: () => {
            try{
                let namespace = prompt("Please enter namespace", "StoryFineExported");
                if(!namespace) return;
                let classname = prompt("Please enter class name", "Exported");
                if(!classname) return;
                let exported = ExportCSHARP(window.SF.episode, namespace, classname);
                let blobData = new Blob([exported], {type: "text/plain"});
                let url = window.URL.createObjectURL(blobData);
                let a = document.createElement("a");
                a.style = "display: none";
                document.body.appendChild(a);
                a.href = url;
                a.download = classname + ".cs";
                a.click();
                window.URL.revokeObjectURL(url);
                a.remove();
            }
```

```
            catch (e) {
                alert('Error during export process: \n' + e);
            }
        },
        lang: "export",
        class: {"fa-file-export": true}
    });
    for (const [key, value] of Object.entries(types)) {
        if(value.default) defaults.push({type: key, position: {x: value.default[0], y: value.default[1]},
points: {}})
        else{
            let c = {};
            if(value.icon){
                value.icon.split(' ').forEach(element => {
                    c[element] = true;
                });
            }
            value.icon
            store.state.navbar.objects.push({
                click: () => {addObj(key)},
                lang: key,
                class: c
            });
        }
    }
    store.state.navbar.functions.push({
        click: () => {
            if(localStorage.getItem("debug") == undefined || localStorage.getItem("debug") == "false"){
                window.SF.debug.enabled = true;
                localStorage.setItem("debug", "true");
            }
            else{
                window.SF.debug.enabled = false;
                localStorage.setItem("debug", "false");
            }
```

```
        },
        lang: "debug",
        class: {"fa-bug": true}
    });
    store.state.navbar.help.push({
        click: () => {
            let win = window.open('https://telegra.ph/StoryFine-Editor---Basic-guide-02-09', '_blank');
            win.focus();
        },
        lang: "basic-guide",
        class: {"fa-external-link-alt": true}
    });
    try{
        if(saved != undefined)
            if(isvalid(JSON.parse(saved))) window.SF.episode = JSON.parse(saved);
            else makeNewEpisode();
        else makeNewEpisode();
    }
    catch{makeNewEpisode();}
    if(localStorage.getItem("debug") == "true") window.SF.debug.enabled = true;
    return {updateCanvas};
            }
```

Main. Jc

```
import Vue from 'vue';
import Vuex from 'vuex';
import Store from './store/Store';
import Langs from './langs/index';
import App from './components/App';
import '@fortawesome/fontawesome-free/js/fontawesome';
import '@fortawesome/fontawesome-free/js/solid';
import '@fortawesome/fontawesome-free/js/regular';
import '@fortawesome/fontawesome-free/js/brands';
import 'normalize.css'
import './Style'
```

```
import Logic from './Logic';

String.prototype.lang = Langs.lang;
String.prototype.inject = Langs.inject;

String.prototype.replaceAll = function(search, replace){
   return this.split(search).join(replace);
}

const LANGS = Langs.list;
Vue.use(Vuex);
const store = new Vuex.Store(Store(LANGS, LANGS[0]));

window.SF = {
   debug: {
      enabled: false,
      data: []
   },
   selectedId: '',
   episode: {},
   camera: [0, 0],
   mouse: [0, 0],
   scale: 20
}
const {updateCanvas} = Logic({store});
window.SF.updateCanvas = updateCanvas;

new Vue({
   store,
   render: h => h(App)
         }).$mount('#app');

         Index.jc
import LANG_EN from './en';
```

```
const langs = {
    ru: LANG_EN
}

const fallbacks = [
    'en'
]

const replaceAll = function(text, search, replace){
    return text.split(search).join(replace);
}

export default {
    lang: function(lang) {
        let translated = "[UNTRANSLATED]";
        let value = undefined;
        try{
            value = eval(`langs['${lang}'][` + replaceAll(this, '.', "'][") + "']");
        }
        catch{}
        if(value === undefined){
            for (let index = 0; index < fallbacks.length; index++) {
                const element = fallbacks[index];
                try{
                    value = eval(`langs['${element}'][` + replaceAll(this, '.', "'][") + "']");
                }
                catch{}
                if(value === undefined) continue;
                else break;
            }
        }
        if(value !== undefined) translated = value;
        return translated;
    },
```

```
    inject: function(data) {
        let text = this;
        for (const [key, value] of Object.entries(data)) {
            text.replace(`{${key}}`, value);
        }
    },
    list: Object.keys(langs).sort()
            }


            Store.jc
import Navbar from './Navbar';

const store = {
    state: {
        selectedId: '',
        lang: {
            list: [],
            selected: ''
        },
        navbar: {
            episode: [],
            objects: [],
            functions: [],
            help: []
        }
    },
    mutations: {
        selectLanguage: function(state, lang) {
            state.lang.selected = lang;
        },
        selectId: function(state, id) {
            state.selectedId = id;
        }
    }
};
```

```
const modules = [
   Navbar
];
modules.forEach(module => {
   for (const [key1] of Object.entries(module)) {
      if(!store[key1]) store[key1] = {};
      for (const [key2] of Object.entries(module[key1])) {
         store[key1][key2] = module[key1][key2];
      }
   }
});
export default (LANGS, SELECTED) => {
   store.state.lang.list = LANGS;
   store.state.lang.selected = SELECTED;
   return store;
         }
         Type.jc
const getImage = (src) => {
   let image = new Image();
   image.src = src;
   return image;
};

export default {
   "entry": {
      default: [0, 0],
      width: 6,
      height: 6,
      texture: getImage("data:image/ "),
      points:[
         {
            type: "main-out",
            position: {
               x: 6,
               y: 3
```

```
            }
        }
    ]
},
"checkpoint":{
    icon: "fas fa-flag-checkered",
    width: 8,
    height: 4,
    texture: getImage("data:image/png; =="),
    points:[
        {
            type: "main-out",
            position: {
                x: 8,
                y: 2
            }
        },
        {
            type: "main-in",
            position: {
                x: 0,
                y: 2
            }
        },
        {
            type: "logical-out",
            position: {
                x: 8,
                y: 0
            }
        }
    ]
},
"condition":{
    icon: "fas fa-code-branch",
```

```
    width: 8,
    height: 6,
    texture: getImage("data:image/ "),
    points:[
        {
            type: "logical-in",
            position: {
                x: 0,
                y: 1
            }
        },
        {
            type: "main-in",
            position: {
                x: 0,
                y: 3
            }
        },
        {
            type: "main-out",
            position: {
                x: 8,
                y: 2
            }
        },
        {
            type: "main-out",
            position: {
                x: 8,
                y: 4
            }
        }
    ]
},
"choice":{
```

```
    icon: "fas fa-vote-yea",
    width: 8,
    height: 4,
    texture: getImage("data:image/png; "),
    points:[
        {
            type: "main-in",
            position: {
                x: 0,
                y: 2
            }
        },
        {
            type: "main-out",
            position: {
                x: 8,
                y: 1
            }
        },
        {
            type: "main-out",
            position: {
                x: 8,
                y: 3
            }
        }
    ]
},
"searcher":{
    icon: "fas fa-search",
    width: 6,
    height: 4,
    texture: getImage("data:image/png; "),
    points:[
        {
```

```
            type: "logical-out",
            position: {
                x: 6,
                y: 2
            }
        }
    ]
}
        };
```

Render.jc

```
export default ({cframe, canvas, types, getMousePos, getScaledMousePos, getClickedObjectId,
getClickedPoint, updateEpisode, select, clearPoint}) => {
    let ctx = canvas.getContext('2d');
    const updateCanvas = (c, cf) => {
        cframe = cf;
        canvas = c;
        ctx = c.getContext('2d');
    }
    const updateSize = ()=>{
        canvas.height = cframe.offsetHeight;
        canvas.width = cframe.offsetWidth;
    };
    window.onresize = updateSize;
    var lastFrameTime = performance.now();
    var fps = 0;
    updateSize();
    const loop = ()=>{
        fps = Math.round(1000/(performance.now() - lastFrameTime));
        lastFrameTime = performance.now();
        draw();
        window.requestAnimationFrame(loop);
    };
    window.requestAnimationFrame(loop);
    const updateDebug = (data)=>{
```

```
        let d = ["Debug info:"];
        for (const [key, value] of Object.entries(data)) {
            d.push(`${key}: ${value}`);
        }
        window.SF.debug.data = d;
    };
    window.SF.camera = [3 * window.SF.scale - canvas.width / 2, 3 * window.SF.scale -
canvas.height / 2];
    const isInView = (x, y) => {
        let left = window.SF.camera[0] / window.SF.scale;
        let right = left + (canvas.width / window.SF.scale);
        let top = window.SF.camera[1] / window.SF.scale;
        let bottom = top + (canvas.height / window.SF.scale);
        return (x >= left && x <= right && y >= top && y <= bottom);
    };
    const toCanvasCoords = (x, y) => {
        return [
            x * window.SF.scale - window.SF.camera[0],
            y * window.SF.scale - window.SF.camera[1]
        ];
    };
    var movingCamera = [];
    var movingObject = [];
    var movingPoint = [];
    const getPair = (pointType) => {
        if(pointType == 'main-out') return 'main-in';
        if(pointType == 'main-in') return 'main-out';
        if(pointType == 'logical-out') return 'logical-in';
        if(pointType == 'logical-in') return 'logical-out';
    }
    const handleMovePointEnd = (mp) => {
        if(mp.length != 0){
            let click = getScaledMousePos();
            let point = getClickedPoint(click[0], click[1]);
            if(point){
```

```
            if(mp[0].module.id != point.module.id){
                if(point){
                    if(getPair(mp[0].point.data.type) == point.point.data.type){
                        clearPoint(mp[0].module.id, mp[0].point.index);
                        clearPoint(point.module.id, point.point.index);
                        let elfrom = window.SF.episode[mp[0].module.id];
                        let elto = window.SF.episode[point.module.id];
                        if(mp[0].point.data.type.endsWith('-out')){
                            elfrom.points[mp[0].point.index] = {id: point.module.id, index:
point.point.index}
                        }
                        else{
                            elto.points[point.point.index] = {id: mp[0].module.id, index:
mp[0].point.index}
                        }
                        updateEpisode();
                    }
                }
            }
        }
    };
    canvas.addEventListener("mousedown", () => {
        let click = getScaledMousePos();
        let clickPoint = getClickedPoint(click[0], click[1]);
        if(!clickPoint){
            let clickObj = getClickedObjectId(click[0], click[1]);
            if(click != null && clickObj != null){
                select(clickObj);
                if(!types[window.SF.episode[clickObj].type].default){
                    let coords = getMousePos();
                    let coordsOnCanvas = toCanvasCoords(window.SF.episode[clickObj].position.x,
window.SF.episode[clickObj].position.y);
                    if(coords != null) movingObject = [clickObj, window.SF.camera[0] +
coordsOnCanvas[0] - coords[0], window.SF.camera[1] + coordsOnCanvas[1] - coords[1]];
```

```
            }
        }
        else{
            select();
            let coords = getMousePos();
            if(coords != null) movingCamera = [window.SF.camera[0] + coords[0],
window.SF.camera[1] + coords[1]];
        }
    }
    else{
        let coords = getMousePos();
        if(coords){
            let coordsOnCanvas =
toCanvasCoords(window.SF.episode[clickPoint.module.id].position.x +
types[clickPoint.module.type].points[clickPoint.point.index].position.x,
window.SF.episode[clickPoint.module.id].position.y +
types[clickPoint.module.type].points[clickPoint.point.index].position.y);
            movingPoint = [clickPoint, coordsOnCanvas[0], coordsOnCanvas[1], coords[0],
coords[1]];
            movingPoint[5] = ((movingPoint[1] + (coords[0] - movingPoint[3]) +
window.SF.camera[0]) / window.SF.scale);
            movingPoint[6] = ((movingPoint[2] + (coords[1] - movingPoint[4]) +
window.SF.camera[1]) / window.SF.scale);
        }
    }
  })
  canvas.addEventListener("mousemove", (evt) => {
    if(movingCamera.length != 0){
        let rect = canvas.getBoundingClientRect();
        window.SF.camera[0] = movingCamera[0] - evt.clientX - rect.left;
        window.SF.camera[1] = movingCamera[1] - evt.clientY + rect.top;
    }
    else if(movingObject.length != 0){
        let rect = canvas.getBoundingClientRect();
```

```
            window.SF.episode[movingObject[0]].position.x = Math.round((movingObject[1] +
evt.clientX - rect.left) / window.SF.scale);
            window.SF.episode[movingObject[0]].position.y = Math.round((movingObject[2] +
evt.clientY - rect.top) / window.SF.scale);
            updateEpisode();
        }
        else if(movingPoint.length != 0){
            let coords = getMousePos();
            movingPoint[5] = ((movingPoint[1] + (coords[0] - movingPoint[3]) +
window.SF.camera[0]) / window.SF.scale);
            movingPoint[6] = ((movingPoint[2] + (coords[1] - movingPoint[4]) +
window.SF.camera[1]) / window.SF.scale);
        }
    });
    canvas.addEventListener("mouseup", () => {
        movingCamera = [];
        movingObject = [];
        handleMovePointEnd(movingPoint);
        movingPoint = [];
    })
    canvas.addEventListener("mouseleave", () => {
        movingCamera = [];
        movingObject = [];
        handleMovePointEnd(movingPoint);
        movingPoint = [];
    })
    canvas.addEventListener("wheel", (evt) => {
        let mousePos = getMousePos();
        if(movingCamera.length == 0 && movingObject.length == 0 && mousePos != null){
            if(evt.deltaY > 0)
                if(window.SF.scale > 5){
                    window.SF.scale -= 5;
                    window.SF.camera[0] -= (((window.SF.camera[0] + mousePos[0]) * (window.SF.scale
+ 5)) - ((window.SF.camera[0] + mousePos[0]) * window.SF.scale)) / (window.SF.scale + 5);
```

```
                window.SF.camera[1] -= (((window.SF.camera[1] + mousePos[1]) * (window.SF.scale
+ 5)) - ((window.SF.camera[1] + mousePos[1]) * window.SF.scale)) / (window.SF.scale + 5);
            }
        if(evt.deltaY < 0)
            if(window.SF.scale < 40){
                window.SF.scale += 5;
                window.SF.camera[0] += (((window.SF.camera[0] + mousePos[0]) *
window.SF.scale) - ((window.SF.camera[0] + mousePos[0]) * (window.SF.scale - 5))) /
(window.SF.scale - 5);
                window.SF.camera[1] += (((window.SF.camera[1] + mousePos[1]) *
window.SF.scale) - ((window.SF.camera[1] + mousePos[1]) * (window.SF.scale - 5))) /
(window.SF.scale - 5);
            }
    }
  })
  setInterval(()=>{
    let obj = {};
    obj["FPS"] = fps;
    obj["Camera"] = "[" + Math.round(window.SF.camera[0] * 100) / 100 + ", " +
Math.round(window.SF.camera[1] * 100) / 100 + "]";
    obj["Scale"] = window.SF.scale;
    let mousePos = getMousePos();
    if(mousePos != null) obj["Mouse"] = "[" + Math.round(mousePos[0] * 100) / 100 + ", " +
Math.round(mousePos[1]) / 100 + "]";
    let scaledMousePos = getScaledMousePos();
    if(scaledMousePos != null) obj["Scaled mouse"] = "[" + Math.round(scaledMousePos[0] *
100) / 100 + ", " + Math.round(scaledMousePos[1] * 100) / 100 + "]";
    if(window.SF.selectedId != "") {
        obj["Selected id"] = window.SF.selectedId;
        obj["Selected coords"] = "[" + window.SF.episode[window.SF.selectedId].position.x + ", "
+ window.SF.episode[window.SF.selectedId].position.y + "]";
    }
    if(movingPoint.length != 0) {
        obj["Point"] = JSON.stringify(movingPoint[0]);
        obj["Point coords"] = JSON.stringify(movingPoint.slice(1));
```

```
        }
        updateDebug(obj);
    }, 50);
    const drawGrid = ()=>{
        ctx.fillStyle = '#eeeeee';
        let startX = - ((window.SF.camera[0] / window.SF.scale) % 1);
        let startY = - ((window.SF.camera[1] / window.SF.scale) % 1);
        for (let position = startX * window.SF.scale; position < canvas.width; position +=
window.SF.scale) {
            ctx.fillRect(position, 0, 1, canvas.height);
        }
        for (let position = startY * window.SF.scale; position < canvas.height; position +=
window.SF.scale) {
            ctx.fillRect(0, position, canvas.width, 1);
        }
    };
    const drawPoint = (c, type) => {
        if(type == "main-out"){
            ctx.fillStyle = 'rgb(255, 255, 255)';
            ctx.beginPath();
            ctx.arc(c[0], c[1], window.SF.scale / 2 + 1, 0, 2 * Math.PI);
            ctx.fill();
            ctx.fillStyle = 'rgb(100, 100, 255)';
            ctx.beginPath();
            ctx.arc(c[0], c[1], window.SF.scale / 2, 0, 2 * Math.PI);
            ctx.fill();
            ctx.fillStyle = 'rgb(50, 50, 150)';
            ctx.beginPath();
            ctx.moveTo(c[0] + window.SF.scale / 4 + window.SF.scale / 10, c[1]);
            ctx.lineTo(c[0] - window.SF.scale / 4, c[1] - window.SF.scale / 4);
            ctx.lineTo(c[0] - window.SF.scale / 4, c[1] + window.SF.scale / 4);
            ctx.fill();
        }
        if(type == "main-in"){
            ctx.fillStyle = 'rgb(255, 255, 255)';
```

```
    ctx.beginPath();
    ctx.arc(c[0], c[1], window.SF.scale / 2 + 1, 0, 2 * Math.PI);
    ctx.fill();
    ctx.fillStyle = 'rgb(100, 100, 255)';
    ctx.beginPath();
    ctx.arc(c[0], c[1], window.SF.scale / 2, 0, 2 * Math.PI);
    ctx.fill();
    ctx.fillStyle = 'rgb(50, 50, 150)';
    ctx.beginPath();
    ctx.moveTo(c[0] + window.SF.scale / 4 + window.SF.scale / 10, c[1]);
    ctx.lineTo(c[0] - window.SF.scale / 4, c[1] - window.SF.scale / 4);
    ctx.lineTo(c[0] - window.SF.scale / 8, c[1]);
    ctx.lineTo(c[0] - window.SF.scale / 4, c[1] + window.SF.scale / 4);
    ctx.fill();
}
if(type == "logical-out"){
    ctx.fillStyle = 'rgb(255, 255, 255)';
    ctx.beginPath();
    ctx.arc(c[0], c[1], window.SF.scale / 2 + 1, 0, 2 * Math.PI);
    ctx.fill();
    ctx.fillStyle = 'rgb(175, 175, 175)';
    ctx.beginPath();
    ctx.arc(c[0], c[1], window.SF.scale / 2, 0, 2 * Math.PI);
    ctx.fill();
    ctx.fillStyle = 'rgb(100, 100, 100)';
    ctx.beginPath();
    ctx.lineTo(c[0] + window.SF.scale / 10, c[1] + window.SF.scale / 4);
    ctx.lineTo(c[0] + window.SF.scale / 4, c[1]);
    ctx.lineTo(c[0] + window.SF.scale / 10, c[1] - window.SF.scale / 4);
    ctx.lineTo(c[0] - window.SF.scale / 4, c[1] - window.SF.scale / 4);
    ctx.lineTo(c[0] - window.SF.scale / 4, c[1] + window.SF.scale / 4);
    ctx.fill();
}
if(type == "logical-in"){
    ctx.fillStyle = 'rgb(255, 255, 255)';
```

```
            ctx.beginPath();
            ctx.arc(c[0], c[1], window.SF.scale / 2 + 1, 0, 2 * Math.PI);
            ctx.fill();
            ctx.fillStyle = 'rgb(175, 175, 175)';
            ctx.beginPath();
            ctx.arc(c[0], c[1], window.SF.scale / 2, 0, 2 * Math.PI);
            ctx.fill();
            ctx.fillStyle = 'rgb(100, 100, 100)';
            ctx.beginPath();
            ctx.lineTo(c[0] + window.SF.scale / 4, c[1] + window.SF.scale / 4);
            ctx.lineTo(c[0] + window.SF.scale / 4, c[1] - window.SF.scale / 4);
            ctx.lineTo(c[0] - window.SF.scale / 4, c[1] - window.SF.scale / 4);
            ctx.lineTo(c[0] - window.SF.scale / 8, c[1]);
            ctx.lineTo(c[0] - window.SF.scale / 4, c[1] + window.SF.scale / 4);
            ctx.fill();
        }
    }
    const drawBlocks = ()=>{
        for (const [key, value] of Object.entries(window.SF.episode)) {
            types[value.type].points.forEach((element, index) => {
                let c = toCanvasCoords(value.position.x + element.position.x, value.position.y +
element.position.y);
                let pointData = value.points[index];
                if(pointData){
                    let coords = toCanvasCoords(window.SF.episode[pointData.id].position.x +
types[window.SF.episode[pointData.id].type].points[pointData.index].position.x,
window.SF.episode[pointData.id].position.y +
types[window.SF.episode[pointData.id].type].points[pointData.index].position.y);
                    ctx.beginPath();
                    ctx.moveTo(c[0], c[1]);
                    ctx.lineTo(coords[0], coords[1]);
                    ctx.strokeStyle = "rgb(0, 0, 0)";
                    ctx.lineWidth = 1;
                    ctx.stroke();
                }
```

```
        });
        if(isInView(value.position.x, value.position.y) || isInView(value.position.x +
types[value.type].width, value.position.y + types[value.type].height)){
            let a = toCanvasCoords(value.position.x, value.position.y);
            let b = toCanvasCoords(value.position.x + types[value.type].width, value.position.y +
types[value.type].height);
            ctx.drawImage(types[value.type].texture, a[0], a[1], b[0] - a[0], b[1] - a[1]);
            if(window.SF.selectedId == key){
               ctx.strokeStyle = "rgb(0, 0, 255)";
               ctx.lineWidth = 2;
               ctx.strokeRect(a[0] - 1, a[1] - 1, b[0] - a[0] + 2, b[1] - a[1] + 2);
            }
            types[value.type].points.forEach((element, index) => {
               let c = toCanvasCoords(value.position.x + element.position.x, value.position.y +
element.position.y);
               let pointData = value.points[index];
               if(pointData){
                  let coords = toCanvasCoords(window.SF.episode[pointData.id].position.x +
types[window.SF.episode[pointData.id].type].points[pointData.index].position.x,
window.SF.episode[pointData.id].position.y +
types[window.SF.episode[pointData.id].type].points[pointData.index].position.y);
                  ctx.beginPath();
                  ctx.moveTo(c[0], c[1]);
                  ctx.lineTo(coords[0], coords[1]);
                  ctx.strokeStyle = "rgb(0, 0, 0)";
                  ctx.lineWidth = 1;
                  ctx.stroke();
               }
               if(movingPoint.length == 0 || element.type == getPair(movingPoint[0].point.data.type))
                  drawPoint(c, element.type);
            });
            let data = [];
            data.push(value.id || key);
            if(value.descr) data.push(value.descr);
            let shift = 0;
```

```
            data.forEach(line => {
                ctx.fillStyle = 'rgb(0, 0, 0)';
                ctx.font = `${Math.round(16 * window.SF.scale / 20)}px Calibri`;
                ctx.fillText(line, a[0], a[1] - Math.round(6 * window.SF.scale / 20) - shift);
                shift += Math.round(16 * window.SF.scale / 20);
            });
        }
    }
    if(movingPoint.length != 0){
        let c2 = toCanvasCoords(movingPoint[5], movingPoint[6]);
        drawPoint([movingPoint[1], movingPoint[2]], movingPoint[0].point.data.type);
        ctx.beginPath();
        ctx.moveTo(movingPoint[1], movingPoint[2]);
        ctx.lineTo(c2[0], c2[1]);
        ctx.strokeStyle = "rgb(0, 0, 0)";
        ctx.lineWidth = 1;
        ctx.stroke();
    }
};
const drawDebug = () => {
    if(window.SF.debug.enabled){
        /*ctx.rect(10, 10, 100, 100);
        ctx.fill();*/

        ctx.fillStyle = 'rgb(0, 0, 0)';
        ctx.font = "16px Consolas";
        let shift = 0;
        window.SF.debug.data.forEach(line => {
            ctx.fillText(line, 5, 20 + shift);
            shift += 18;
        });
    }
}
//const coords
const draw = ()=>{
```

```
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawGrid();
    drawBlocks();
    drawDebug();
    /*ctx.fillStyle = '#000';
    ctx.fillText(fps, 10, 10)*/
};
return {updateCanvas};
        }
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи