НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

"___" _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: «Система планування та розподілу ресурсів для створення засобів прийняття рішень»

Виконав (-ла):

студент (-ка) IV курсу, групи КВ-91
(шифр групи)

Сторожук Костянтин Валерійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент каф. СПСКС, Сергієнко П. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма

«Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» _____2023 р.

ЗАВДАННЯ
на дипломний проєкт студента
Сторожука Костянтина Валерійовича

1. Тема проєкту «Система планування та розподілу ресурсів для створення засобів прийняття рішень»,
керівник проєкту Сергієнко Павло Анатолійович,
затверджені наказом по університету від «31» травня 2023 р. № 2107-С

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: система планування та розподілу ресурсів для створення засобів прийняття рішень

4. Зміст пояснювальної записки
- Аналіз існуючих рішень та обґрунтування теми дипломного проєкту;
- Аналіз засобів реалізації;
- Розробка системи планування та розподілу ресурсів;
- Тестування системи.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

- Структурна схема взаємодії класів системи;
- Алгоритм розрахування наявних ресурсів та виконаних задач для певного кроку симуляції;
- Алгоритм відображення списку задач у системі;
- Алгоритм відображення взаємозв'язків між задачами в системі;
- Презентація за темою проєкту.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент каф. СПСКС		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	09.02.2023	
2.	Розроблення та узгодження технічного завдання	12.03.2023	
3.	Аналіз існуючих рішень	19.03.2023	
4.	Розроблення структури системи	18.04.2023	
5.	Програмна реалізація системи	14.05.2023	
6.	Підготовка матеріалів першого розділу дипломного проєкту	18.05.2023	
7.	Підготовка матеріалів другого розділу дипломного проєкту	21.05.2023	
8.	Підготовка матеріалів третього розділу дипломного проєкту	24.05.2023	
9.	Підготовка матеріалів четвертого розділу дипломного проєкту	27.05.2023	
10.	Підготовка графічної частини дипломного проєкту	30.05.2023	
11.	Оформлення документації дипломного проєкту	01.05.2023	
12.	Попередній огляд матеріалів диплому на кафедрі	02.06.2023	

Студент _____ Костянтин СТОРОЖУК
(підпис)

Керівник проєкту _____ Павло СЕРГІЄНКО
(підпис)

* Консультантом не може бути зазначено керівника дипломного проєкту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (50 с., 51 рис., 4 додатки).

Об’єкт розробки – створення системи планування та розподілу ресурсів для створення засобів прийняття рішень.

Система планування дозволяє: планувати задачі та дії, використовуючи різні ресурси, такі як час, гроші, матеріали тощо; переглядати наявні ресурси на кожному кроці та змінювати план дій відповідно до цього; створювати покрокові системи прийняття рішень, враховуючи наявні ресурси та обмеження, зберігати та завантажувати створенні системи для використання чи подальшого редагування.

В процесі розробки були використано багатоплатформовий інструмент для розроблення додатків Unity.

В ході розробки:

- проведено аналіз існуючих програм-планувальників та систем прийняття рішень;
- сформульовані вимоги до системи планування та розподілу ресурсів для створення засобів прийняття рішень;
- розроблено систему створення дерев взаємозв’язку задач, дій та ресурсів;
- розроблено користувацький інтерфейс програми, що дозволяє легко та швидко створювати покрокові системи прийняття рішень;
- розроблена система збереження та завантаження створених систем прийнять рішень;
- проведено тестування та валідацію програми.

Упровадження цієї програми дозволить ефективно планувати та приймати рішення, використовуючи наявні ресурси та обмеження та зберігати створенні системи.

Ключові слова:

СИСТЕМИ ПЛАНУВАННЯ, ПРИЙНЯТТЯ РІШЕНЬ, ОПТИМІЗАЦІЯ, РЕСУРСИ, UNITY.

ABSTRACT

The qualification work includes an explanatory note (50 p., 51 fig, 4 appendices).

The object of development is the creation of a planning and resource allocation system for the creation of decision-making tools.

Planning system allows you to: plan tasks and actions using different resources such as time, money, materials, etc.; review available resources at each step and modify the action plan accordingly; create step-by-step decision-making systems, taking into account available resources and constraints, save and load created systems for use or further editing.

The multi-platform application development tool Unity was used in the development process.

In the course of development:

- analysis of existing planning programs and decision-making systems was carried out;
- formulated requirements for the planning and resource allocation system for creating decision-making tools;
- the system for creating trees of interconnection of tasks, actions and resources was developed;
- the user interface of the program was developed, which allows you to easily and quickly create step-by-step decision-making systems;
- the system for saving and loading the created decision-making systems was developed;
- the program was tested and validated.

The implementation of this program will allow effective planning and decision-making, using available resources and constraints, and preserve the established system.

Keywords:

PLANNING SYSTEMS, DECISION MAKING, OPTIMIZATION, RESOURCES, UNITY.

	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.002 ТЗ	Система планування та	4		
			розподілу ресурсів для			
			створення засобів			
			прийняття рішень.			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ДП	Система планування та	2		
			розподілу ресурсів для			
			створення засобів			
			прийняття рішень.			
			Відомість дипломного			
			проекту			
	А4	ІАЛЦ.045490.004 ПЗ	Система планування та	50		
			розподілу ресурсів для			
			створення засобів			
			прийняття рішень.			
			Пояснювальна			
			записка			

					ІАЛЦ.045490.001 ОА
Змі	***Арк.***	***№ докум.***	***Підп.***	***Дата***	
Розроб.		Сторожук К. В.			Система планування та розподілу ресурсів для створення засобів прийняття рішень ***Опис альбому***
Перев.		Сергієнко П. А.			
Конс.					
Н.контр.		Клятченко Я. М.			
Зав.каф.		Романкевич В. О.			

Літ.	***Лист***	***Листів***
	1	2

НТУУ «КПІ ім. Ігоря Сікорського», ФПМ, КВ-91

	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.005 Д1	Взаємодія класів	1		
			системи.			
			Схема структурна			
	А4	ІАЛЦ.045490.006 Д2	Розрахування наявних	1		
			ресурсів та виконаних			
			задач для певного кроку			
			симуляції.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.007 Д3	Відображення списку	1		
			задач в системі.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.008 Д4	Відображення	1		
			взаємозв’язків між			
			задачами в системі.			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної			
			записки.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Система планування та розподілу ресурсів для створення засобів прийняття рішень».

Галузь застосування: організація створення планів та стратегій в умовах обмежених ресурсів.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення системи для створення планів та стратегій в умовах обмежених ресурсів що є гнучкою та зручною в використанні.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з операційними системами Windows 7 SP1+, 8, 10, 11
- можливість створення відображення дій, засобів та ресурсів;

- можливість створень планів та стратегій;
- можливість збереження створених систем;
- можливість редагування та відтворення збережених систем;

5.2. Вимоги до апаратного забезпечення

- Процесор: Архітектура X64 з підтримкою набору інструкцій SSE2;
- Оперативна пам’ять: 1 Гб;
- Простір диску: 1 Гб;

5.3. Вимоги до програмного забезпечення користувача

- Операційна система Windows 7 чи вище;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту
1.	Вивчення літератури за тематикою проєкту	09.02.2023
2.	Розроблення та узгодження технічного завдання	12.03.2023
3.	Аналіз існуючих рішень	19.03.2023
4.	Розроблення структури системи	18.04.2023
5.	Програмна реалізація системи	14.05.2023
6.	Підготовка матеріалів першого розділу дипломного проєкту	18.05.2023
7.	Підготовка матеріалів другого розділу дипломного проєкту	21.05.2023
8.	Підготовка матеріалів третього розділу дипломного проєкту	24.05.2023
9.	Підготовка матеріалів четвертого розділу дипломного проєкту	27.05.2023
10.	Підготовка графічної частини дипломного проєкту	30.05.2023
11.	Оформлення документації дипломного проєкту	01.05.2023
12.	Попередній огляд матеріалів диплому на кафедрі	02.06.2023

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ.045490.000	Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.045490.001 ОА	Опис альбому	2	
3	А4	ІАЛЦ.045490.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.045490.003 ДП	Відомість дипломного проєкту	1	
5	А4	ІАЛЦ.045490.004 ПЗ	Пояснювальна записка	50	
6	А4	ІАЛЦ.045490.005 Д1	Взаємодія класів системи. Схема структурна	1	
7	А4	ІАЛЦ.045490.006 Д2	Розрахування наявних ресурсів та виконаних задач для певного кроку симуляції. Схема алгоритму	1	
8	А4	ІАЛЦ.045490.007 Д3	Відображення списку задач в системі. Схема алгоритму	1	
9	А4	ІАЛЦ.045490.008 Д4	Відображення взаємозв’язків між задачами в системі. Схема алгоритму	1	

				ІАЛЦ.045490.003 ДП		
	ПІБ	Підп.	Дата			
Розробн.	Сторожук К.В.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Сергієнко П.А.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-91	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка
до дипломного проєкту

на тему: Система планування та розподілу ресурсів для створення засобів прийняття рішень

Київ – 2023 року

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045490.005 Д1. Взаємодія класів системи. Схема структурна
- ІАЛЦ.045490.006 Д2. Розрахування наявних ресурсів та виконаних задач для певного кроку симуляції. Схема алгоритму
- ІАЛЦ.045490.007 Д3. Відображення списку задач в системі. Схема алгоритму
- ІАЛЦ.045490.008 Д4. Відображення взаємозв’язків між задачами в системі. Схема алгоритму

Додаток 2. Фрагменти вихідного коду

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Компонента Unity (компонента) – функціональна частина об’єктів Unity.

Об’єкт Unity (об’єкт) – основний клас для всіх сутностей проєкту.

C# – мова програмування високого рівня.

Unity – рушій застосунку.

ВСТУП

У сучасному світі, де швидкість змін і конкуренція постійно зростають, ефективне прийняття рішень є критично важливим для досягнення успіху в бізнесі та управлінні проєктами. Процес прийняття рішень включає аналіз, оцінку альтернатив, визначення оптимального варіанту та планування дій для його реалізації.

Однак, прийняття рішень може бути складним завданням, особливо коли вимагається урахування багатьох факторів, обмежень та ресурсів. Система рішень що помилково не врахувала певні обмеження чи прорахувалась в отриманих та витрачених ресурсів призводить до негативних наслідків, аж до закриття проєктів та банкротства.

Системи що дають можливість враховувати різноманітні обмеження при симуляції плану дії можуть зменшити кількість помилок, підвищити швидкість розрахунків, надати можливість швидко побачити вплив на систему прийняття рішень зміну в обмеженнях та/чи ресурсах та адаптувати її до нових реалій.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1 Аналіз існуючих систем для планування задач та управління ресурсами

На теперішній час існує багато різних систем для планування задач та управління ресурсами. Розглянемо найпопулярніші з них.

ClickUp – позиціонує себе як хмарний інструмент для співпраці та керування проєктами, який підходить для підприємств будь-якого розміру та галузі [5]. Цей додаток дозволяє створювати проєкти які містять різноманітні задачі пов’язані з ним.

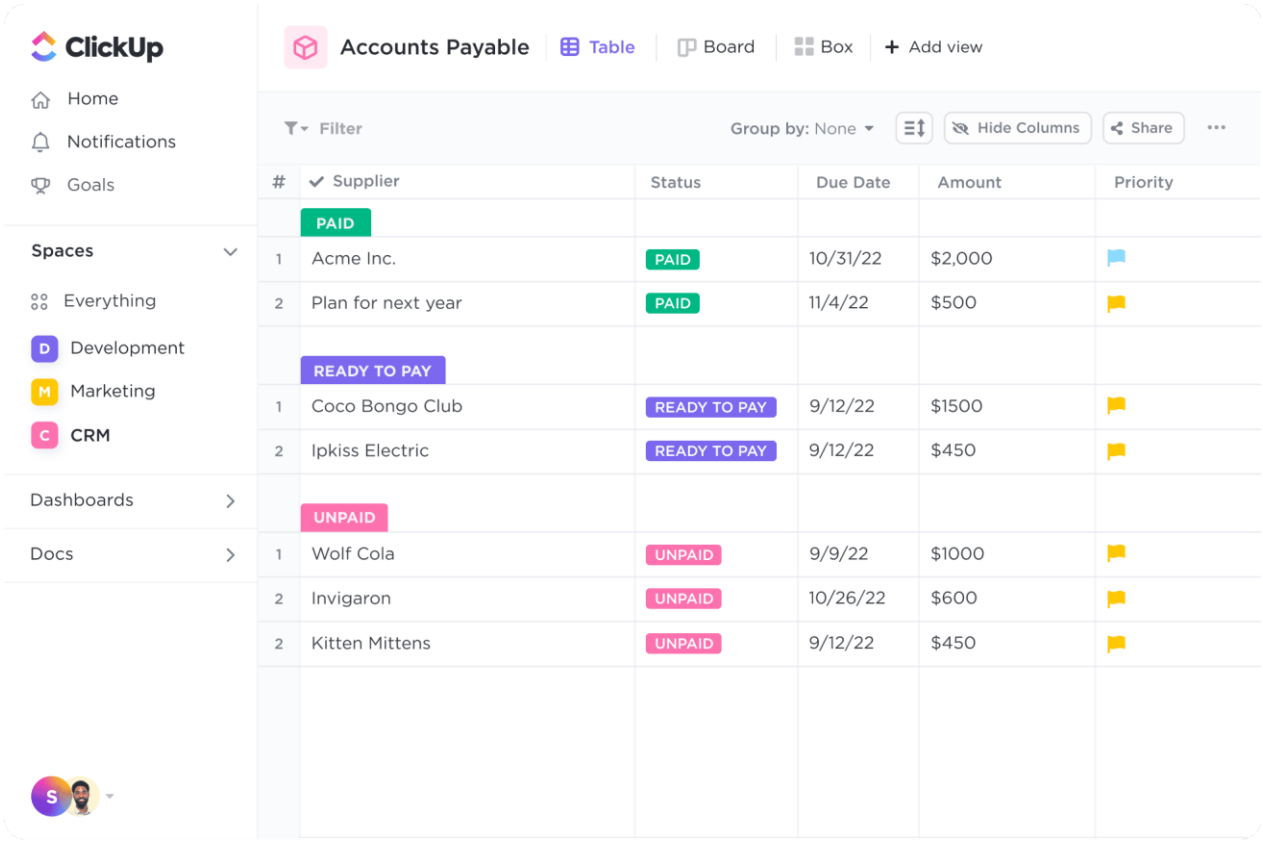

Рисунок 1 – Приклад проєкту в ClickUp

Кожній задачі може бути призначена людині та пріоритетність. Також може бути налаштовано автоматичне створення періодичних задач.

Серед недоліків можна виділити відсутність концепції ресурсів відмінних від часу та автоматичного контролю можливості виконання задачі.

Trello – позиціонує себе як візуальний інструмент, який дає змогу керувати будь-яким типом проєкту, робочим процесом або відстеженням завдань [6]. Цей додаток дозволяє створювати дошки що містять списки з картками. Картки відображають задачі, а списки відображають етап на якому знаходиться задача.

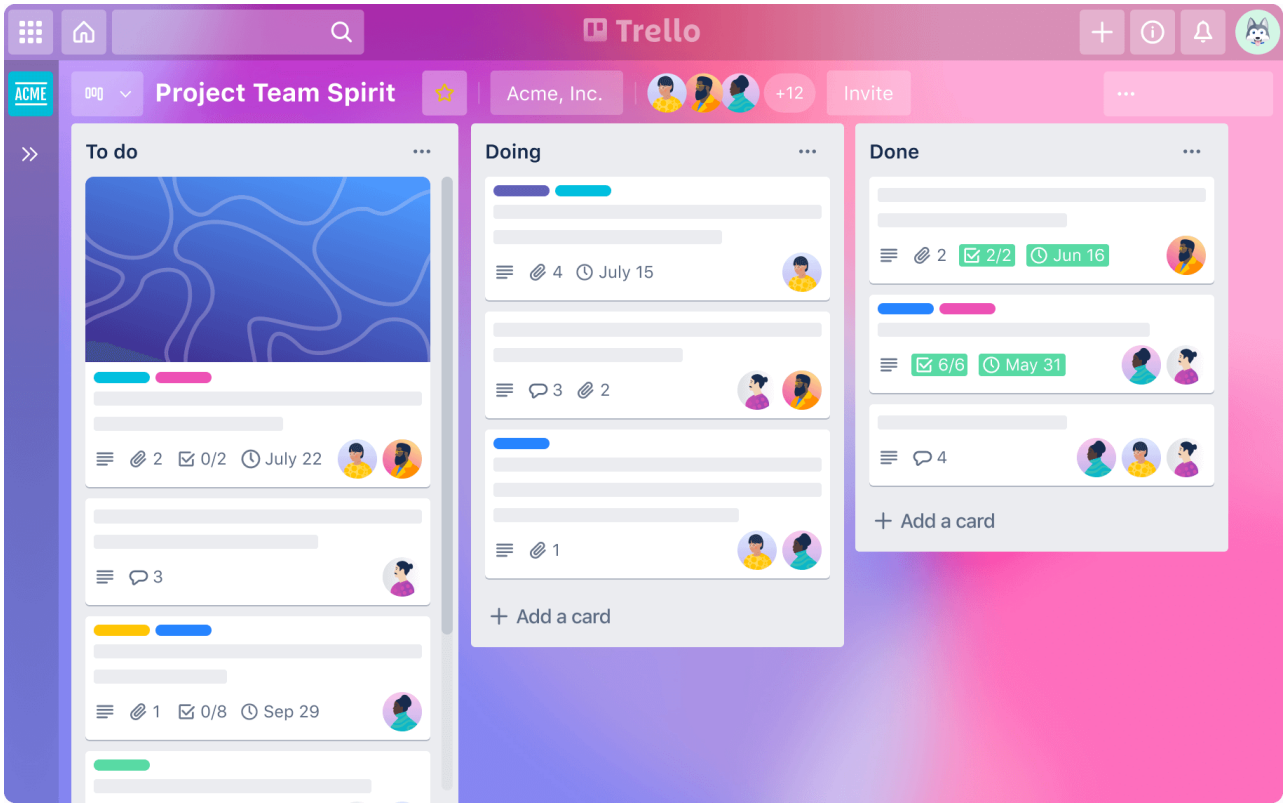

Рисунок 2 – Приклад дошки в Trello

Кожна картка може містити обговорення задачі у вигляді повідомлень між учасниками дошки, містити в собі прикріпленні файли.

Недоліками є, так само як в попередньому додатку, відсутність контролю ресурсами. Також система створення залежностей між картками не дає можливість заборонити виконання задачі для якої не виконані передумови.

Wrike – позиціонує себе як онлайн-сервіс для сумісної роботи і управління проєктами [7]. Цей сервіс дає можливість створювати задачі та розподіляти їх між людьми.

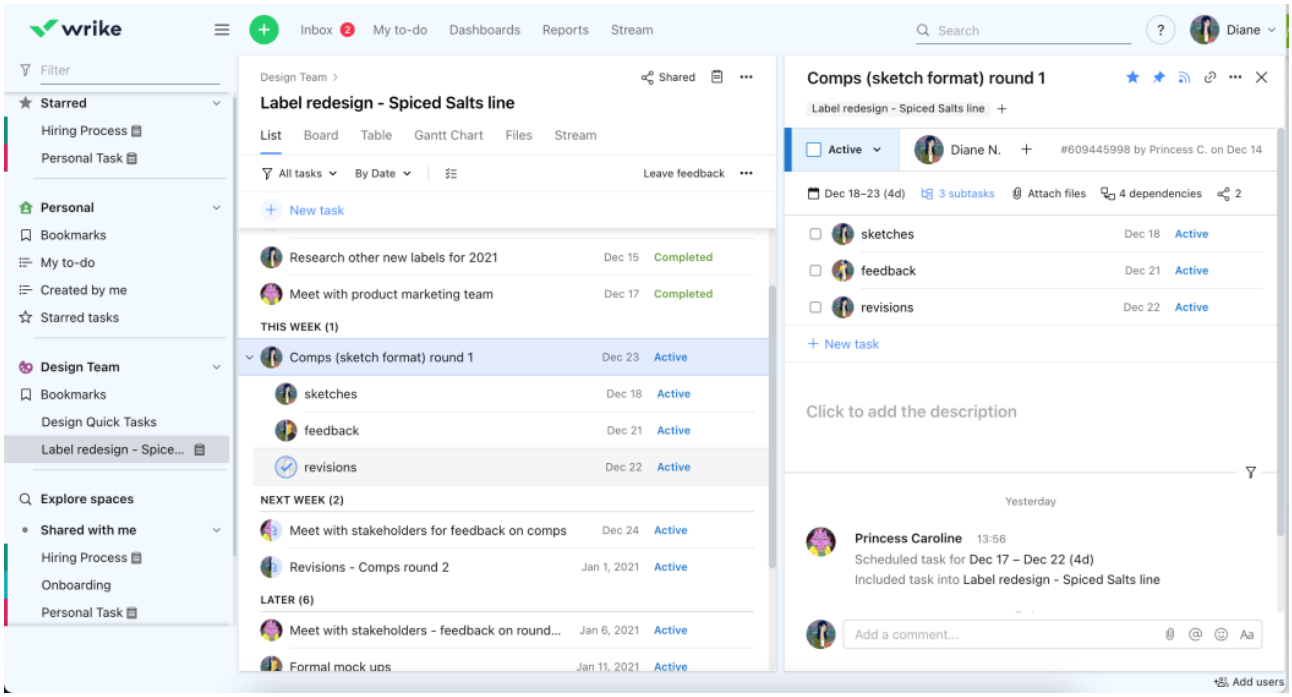

Рисунок 3 – Приклад проєкту в Wrike

Відзначною можливістю Wrike є можливість створення бюджетних планів пов’язаних з задачами проєкту. При створенні задачі є можливість встановити її вартість. Ця вартість автоматично вираховується з заданого бюджету, а при виході за межі бюджету видається відповідне попередження.

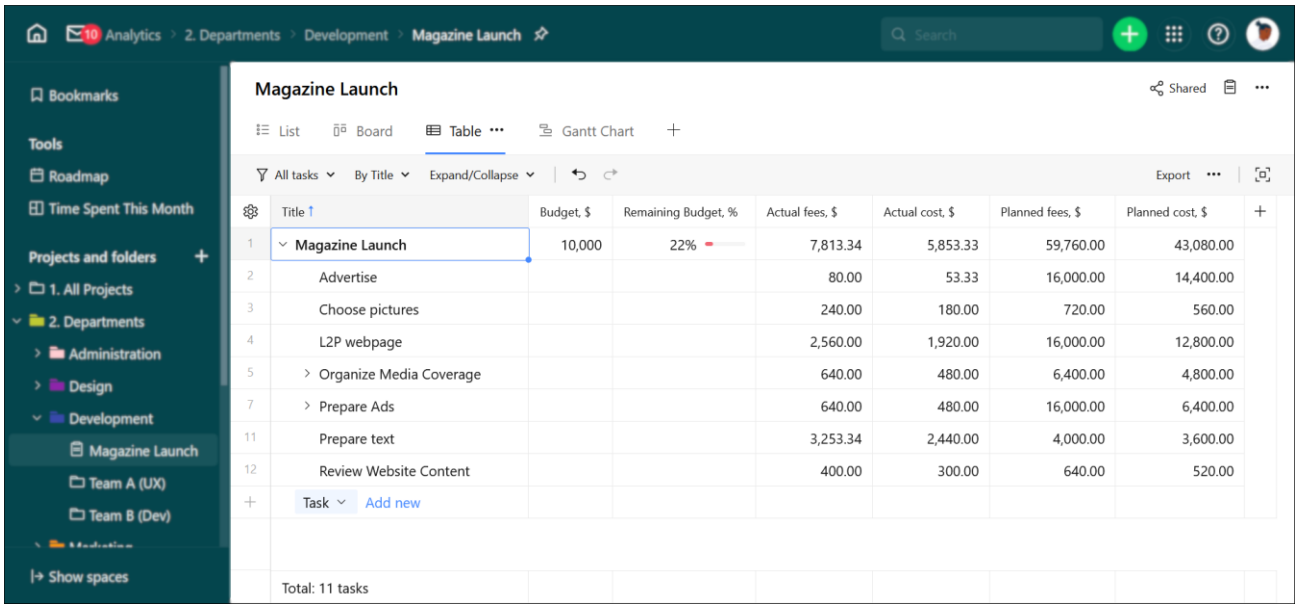

Рисунок 4 – Приклад таблиці бюджету в Wrike

Недоліками Wrike є відсутність можливості зручного врахування ресурсів відмінних від грошей та часу а також важкість первинного налаштування проєкту.

1.2 Обґрунтування теми дипломного проєкту

Виходячи з проведеного аналізу існуючих додатків та сервісів можна підкреслити, що існує багато різних засобів для планування дій та керування проєктами. Це можна пояснити наявністю в бізнесів та людей потреби в плануванні задач.

Однак ні один з знайдених інструментів не дає можливості створення планів виконання задач з урахуванням обмежень та ресурсів відмінних від часу та коштів. В той час як більшість існуючих рішень є зручними для відслідковування поточних задач та керування ними, їм бракує інструментів для створення довгострокових планів та систем рішень.

Було б доцільно створити систему що буде зручною на етапі розробки плану, буде враховувати обмеження в ресурсах та залежності між задачами, дозволить розглядати альтернативні шляхи та порівнювати їх. При необхідності, після обрання найкращого плану дій та симуляції його в додатку, план можна відтворити в одній з наведених вище програм для зручного розподілу праці в команді та його притриманню.

2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Опис особливостей рушія Unity

Unity є потужним інструментом для розробки відеоігор та застосунків. Його було обрано для створення системи планування дій та розподілу ресурсів через наявність якісної документації, зручність у використанні, великої кількості навчальних ресурсів по рушію та підтримку мови програмування C#.

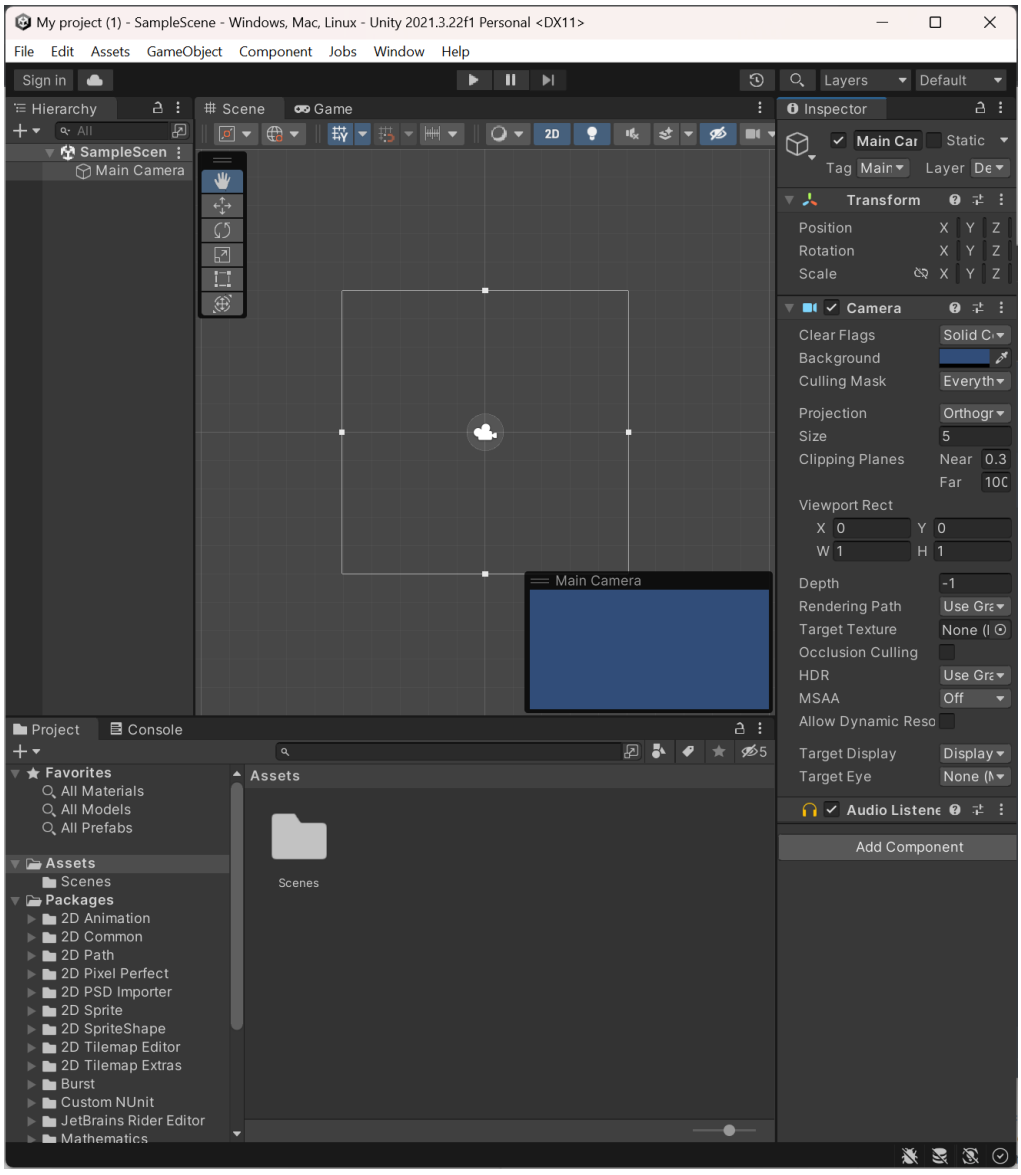

Рисунок 5 – Інтерфейс редактору Unity

Для функціонування будь-якого додатку на рушії Unity необхідні сцени. Сцена (Scene) зберігає в собі об’єкти додатку. В будь-який час виконання додатку що створений за допомогою Unity активною є рівно одна сцена.

Об’єкти (GameObject) є фундаментальною частиною рушія. Вони слугують як контейнери для зберігання компонент. Вони є єдиним способом відображення інформації на екрані.

Компоненти це функціональна частина будь якого додатку Unity. Саме через них запускається будь який власний код. В рушії існує багато створених компонентів які можна додати до об’єктів. На рисунку 5 можна помітити компоненти об’єкту Main Camera в лівій частині екрану. Розглянемо їх.

- Transform – це компонента, що відповідає за розміщення об’єкту в просторі. Вона є наявною в всіх об’єктів.
- Camera – це компонента, що відповідає за відображення сцени на екрані користувача.
- Audio Listener – це компонента, що відповідає за відтворення звуку на аудіопристроях користувача.

Код на мові C# також може бути компонентою – для цього клас скрипту повинен унаслідуватись від класу MonoBehaviour. Такий клас може реалізувати функції що будуть викликані при певних умовах пов’язаних з компонентою чи відповідним їй об’єктом. Розглянемо декілька таких функцій що будуть важливими в проєкті.

- MonoBehaviour.Start() – це функція що виконується коли компонент вперше стає активним на сцені. Вона є корисною для ініціалізації стану компоненти,
- MonoBehaviour.Update() – це функція що виконується на кожному кадрі коли компонент є активним. Є корисним якщо стан об’єкту може змінитись на кожному кадрі додатку. Наприклад, якщо об’єкт повинен слідувати позиції вказівника.

Також існують різні інтерфейси від яких можна унаслідуватись що отримувати різні події, наприклад натискання вказівником на об’єкт, наведення миші на об’єкт тощо.

DraggableObject.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class DraggableObject : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler, IPointerClickHandler
{
    Vector2 startPositionMouse;
    Vector2 startPositionNode;
    Vector2 startPositionObj;

    int nodeNumber;

    public void OnBeginDrag(PointerEventData eventData) {
        MainController.fieldsController.SelectNode(nodeNumber);

        Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
        startPositionMouse = mousePos;
        startPositionNode = MainController.activeNodes[nodeNumber].pos;
        startPositionObj = ((RectTransform)this.transform).anchoredPosition;
    }

    public void OnDrag(PointerEventData eventData) {
        Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
        MainController.activeNodes[nodeNumber].pos = startPositionNode + mousePos - startPositionMouse;
        ((RectTransform)this.transform).anchoredPosition = startPositionObj + mousePos - startPositionMouse;
        MainController.ForceDisplayEdges();
    }

    public void OnEndDrag(PointerEventData eventData) {
        Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
        MainController.activeNodes[nodeNumber].pos = startPositionNode + mousePos - startPositionMouse;
        MainController.ForceDisplay();
    }

    void Start() {
        nodeNumber = this.GetComponent<NodeVisualized>().nodeNumber;
    }

    public void OnPointerClick(PointerEventData pointerEventData)
    {
        if (pointerEventData.button == PointerEventData.InputButton.Left) {
            MainController.fieldsController.SelectNode(nodeNumber);
        }
    }
}
```

Рисунок 6 – Реалізації компоненти що дає можливість переміщувати вузли по екрану

2.2 Опис інтерфейсу редактору Unity

Інтерфейс редактору Unity складається з декількох вікон які можуть бути розміщені на екрані будь яким чином в залежності від потреб розробника.

Вікно Project дає доступ до файлового провіднику. Він надає можливість переглядати використані файли, змінювати наявні скрипти, додавати посилання на файл компонентам об’єктів та інше.

Рисунок 7 – вікно Project редактору Unity

Вікно Console відображає повідомлення виведені протягом налагоджування проєкту, помилки компіляції та помилки що виникають під час виконання додатку.

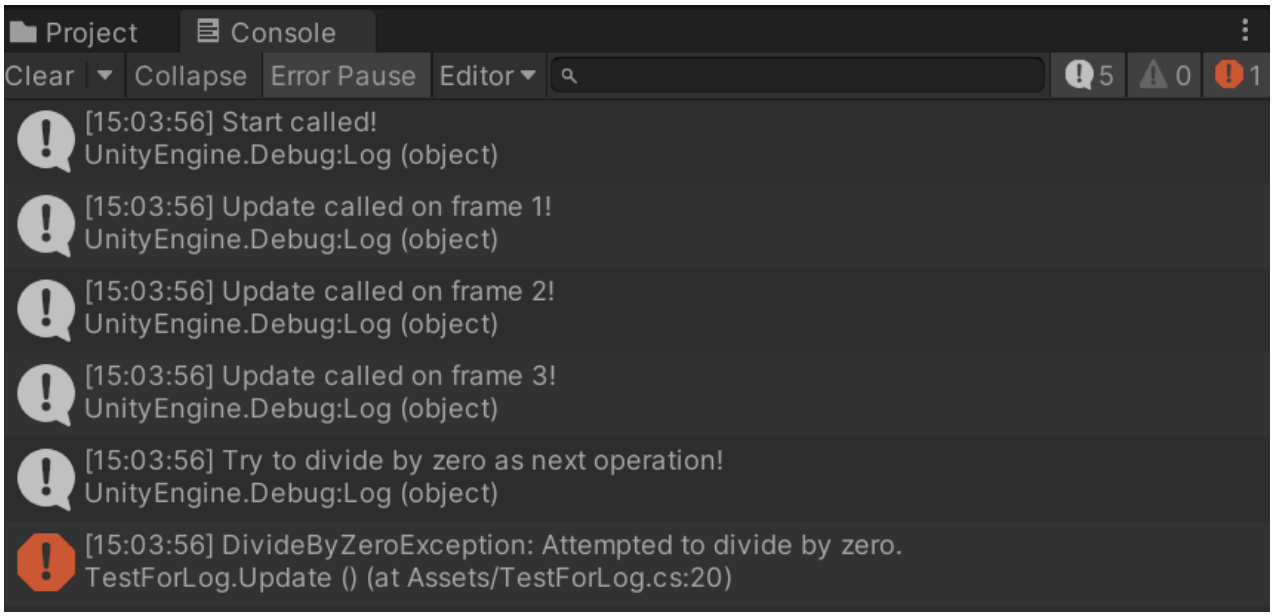

Рисунок 8 – вікно Console редактору Unity

Вікно Hierarchy дозволяє обрати поточну сцену, змінювати порядок об’єктів на сцені, створювати ієрархію об’єктів, виділяти елементи для відображення в вікні Inspector, видаляти та створювати сцени та об’єкти.

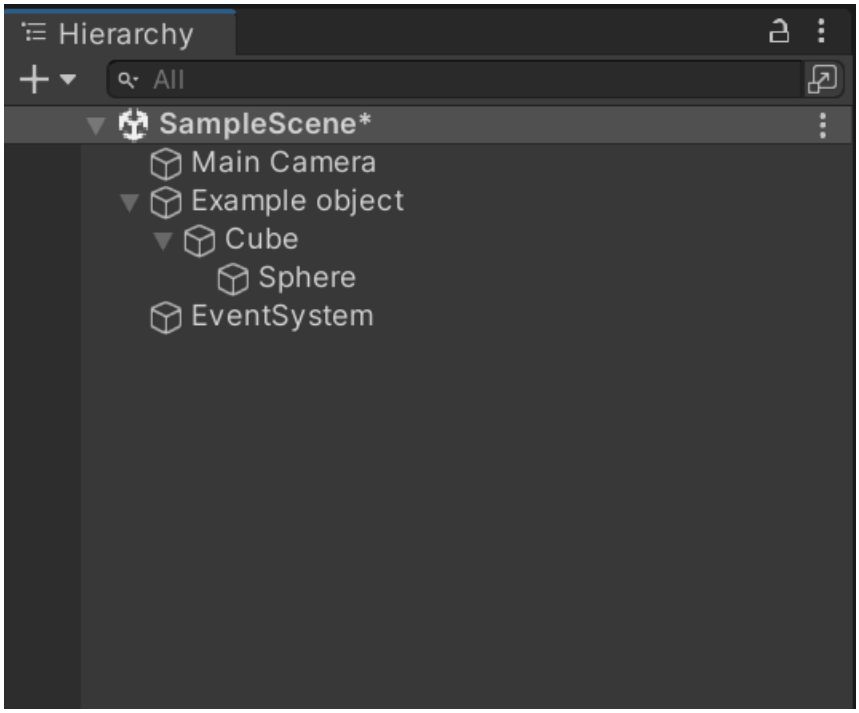

Рисунок 9 – вікно Hierarchy редактору Unity

Вікно Scene відображає обрану сцену у вигляді придатному для редагування. Вона не залежить від наявності камер, дозволяє рухати елементи та переглянути як виглядає будь яка частина сцени.

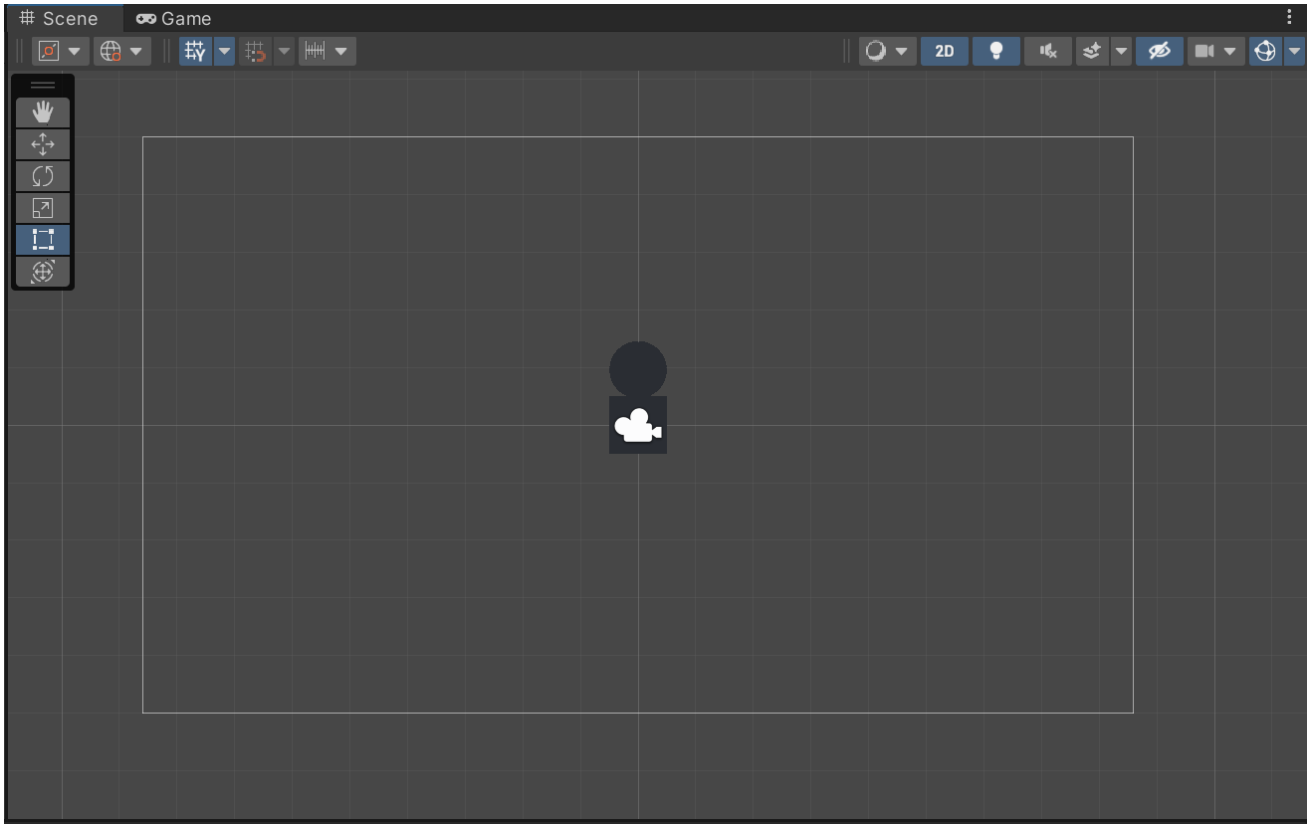

Рисунок 10 – вікно Scene редактору Unity

Вікно Game відображає обрану сцену поточну сцену так, як вона виглядає для користувача. Вона дозволяє змінювати розміри екрану та побачити як виглядає додаток на різних пристроях. Такою дозволяє взаємодіяти з грою під час запуску додатку в тестовому режимі.

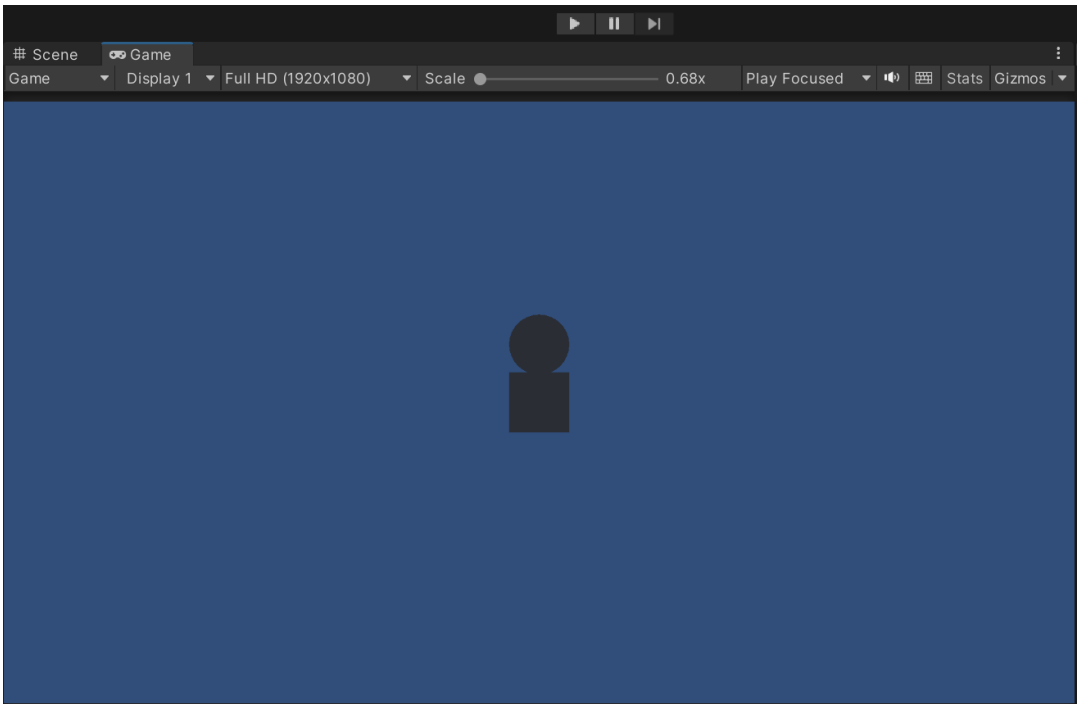

Рисунок 11 – вікно Game редактору Unity

Вікно Inspector дозволяє додавати та видаляти компоненти для обраного об’єкту. Також тут можна змінювати поля що помічені в коді як ті, що можуть бути редаговані з редактору.

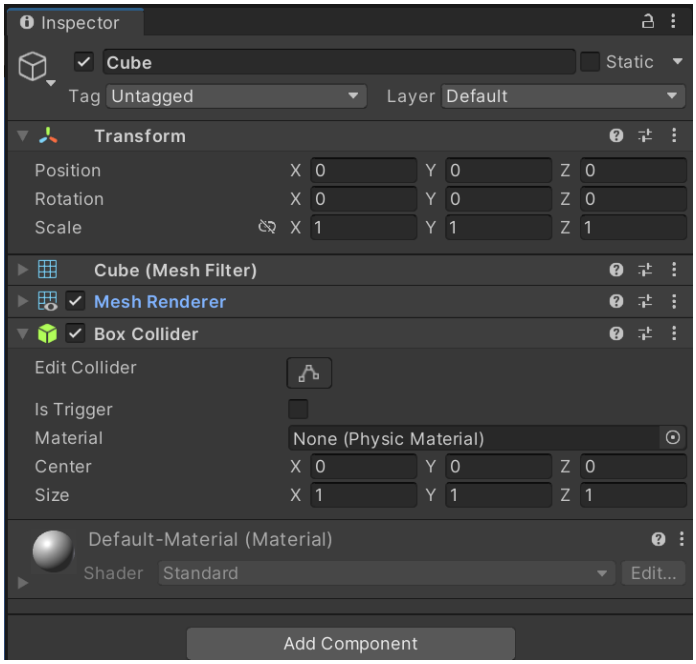

Рисунок 12 – вікно Inspector редактору Unity

2.3 Опис мови програмування C#

C# (C-Sharp) є потужною і сучасною мовою програмування, розробленою компанією Microsoft. Вона має широкий спектр застосувань і є основною мовою для розробки додатків в Unity. C# володіє численними функціями та можливостями, які роблять її привабливою для розробників.

Однією з ключових особливостей C# є його синтаксис, який схожий на синтаксис інших популярних мов, таких як C++ та Java. Це робить його легким для вивчення і розуміння розробниками, які вже знайомі з цими мовами. Крім того, C# має сучасну і елегантну структуру, яка сприяє покращенню продуктивності розробників і забезпечує зручну роботу з кодом.

C# підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє розробникам створювати класи, об'єкти, успадкування, поліморфізм та інші концепції ООП. Це сприяє структуруванню коду, меншу повторюваність і забезпеченню більшої модульності додатків.

3. РОЗРОБКА СИСТЕМИ ПЛАНУВАННЯ ТА РОЗПОДІЛУ РЕСУРСІВ

3.1 Відображення систем в пам’яті

При проєктуванні архітектури системи було відомо що системи створені в додатку буде необхідно зберігати на диску у вигляді файлів. В той же час відображення всієї системи на екрані пристрою є нетривіальною задачею, що потребує багато об’єктів, складної взаємодії між ними, та, як результат, робить збереження у файл складним.

Було вирішено знайти мінімальну можливість даних з яких можна відтворити будь яку створену систему у графічному вигляді. Виявилось, що достатньо всього трьох списків – списку задач у вигляді класу Node, списку ресурсів у вигляді класу Resource та списку активованих задач для кожного кроку симуляції у вигляді списків бітів.

```
public class Node
{
    public List<Requirement> requirements;
    public string name;
    public Vector2 pos;
    public bool isRepeatable;

    public List<Node> edgesTo;

    public Node() {
        requirements = new List<Requirement>();
        edgesTo = new List<Node>();
        pos = Vector2.zero;
        name = "New node";
        isRepeatable = false;
    }
}
```

Рисунок 13 – клас Node

Клас Node представляє собою будь яку задачу що існує в поточній системі. Вона містить 5 полів що повністю відображають задачу. Розглянемо ці поля.

- requirements – це список витрат що необхідні для виконання задачі. Клас Requirement буде розглянутий далі.
- name – це заголовок задачі що відображається при її візуалізації.
- pos – це позиція задачі на екрані під час її візуалізації. Її типом є вбудована структура Vector2 що відображає двовимірний вектор.
- isRepeatable – це логічна змінна що відображає чи може задача бути виконана декілька разів.
- edgesTo – це список вказівників на інші задачі, що не можуть бути виконані раніше цієї задачі.

```
public class Requirement
{
    public Resource resource;
    public int price;
    public Requirement() {
        if (MainController.resources.Count > 0) {
            resource = MainController.resources[0];
        } else {
            resource = new Resource();
        }
        price = 10;
    }
}
```

Рисунок 14 – клас Requirement

Клас Requirement складається лише з двох полів. Розглянемо ці поля.

- resource – це ресурс який потрібно витратити для виконання задачі.
- price – це кількість ресурсу яку необхідно витратити для виконання задачі.

```
public class Resource
{
    public Sprite icon;
    public string icon_path;
    public string name;
    public int amount;
    public Resource() {
        name = "New resource";
        amount = 0;
        icon_path = "";
    }
}
```

Рисунок 15 – клас Resource

Клас Resource відображає ресурс що може бути призначений задачам. Цей клас складається з 4 полів. Розглянемо ці поля.

- icon – це завантажена текстура ресурсу
- icon_path – це шлях до файлу що зберігає текстури ресурсу
- name – це назва ресурсу
- amount – це початкова кількість ресурсу в системі

```
public class MainController : MonoBehaviour
{
    static public List<Resource> resources;
    static public List<Node> activeNodes;
    static public List<List<bool>> activatedNodes;
```

Рисунок 16 – мінімум для відображення системи

3.2 Збереження систем в файли

Оскільки ми можемо відтворити систему з трьох списків наведених вище – то для компактного збереження достатньо записати лише їх до відповідного файлу.

Для максимальної компактності запис буде відбуватись до бінарного файлу.

Щоб записати список – можемо спочатку записати кількість елементів в списку, потім записувати самі елементи. Таким чином при читанні ми зможемо точно знати де закінчується один список та починається наступний.

Іншою проблемою є наявність посилань в структурах які зберігаємо. Оскільки між записом та завантаженням може відбутись очищення оперативної пам’яті – ці посилання зламаються та не будуть відповідати реальним об’єктам. Щоб обійти цю проблему ми можемо покластись на те що порядок елементів в списках буде однаковий при записі та читанні. Тоді всі посилання в бінарному файлі можна замінити індексом відповідною структури в відповідному їй списку.

```
void WriteNodes(BinaryWriter writer) {
    var nodes = MainController.activeNodes;
    writer.Write(nodes.Count);
    foreach (var node in nodes) {
        writer.Write(node.name);
        writer.Write(node.pos.x);
        writer.Write(node.pos.y);
        writer.Write(node.isRepeatable);
        writer.Write(node.edgesTo.Count);
        foreach (var nodeTo in node.edgesTo) {
            writer.Write(MainController.GetNodeId(nodeTo));
        }
        writer.Write(node.requirements.Count);
        foreach (var req in node.requirements) {
            writer.Write(MainController.GetResourceId(req.resource));
            writer.Write(req.price);
        }
    }
}
```

Рисунок 17 – функція для запису списку задач в бінарний файл

Під час читання файлу – необхідно спочатку зчитати та створити всі структуру, і тільки після цього відтворювати посилання на структури. Інакше є імовірність зустріти елемент який ще не був створений та спробувати взяти на нього посилання, що призведе до помилки.

```
void ReadNodes(BinaryReader reader) {
    List<Node> nodes = new List<Node>();
    List<List<int>> edges = new List<List<int>>();
    List<List<Tuple<int,int>>> reqs = new List<List<Tuple<int,int>>>();
    int count = reader.ReadInt32();
    for (int i = 0; i < count; i++) {
        Node node = new Node();
        node.name = reader.ReadString();

        float x = reader.ReadSingle();
        float y = reader.ReadSingle();
        node.pos = new Vector2(x, y);

        node.isRepeatable = reader.ReadBoolean();

        int edgesCount = reader.ReadInt32();
        edges.Add(new List<int>());
        for (int j = 0; j < edgesCount; j++) {
            edges[i].Add(reader.ReadInt32());
        }

        int reqsCount = reader.ReadInt32();
        reqs.Add(new List<Tuple<int,int>>());
        for (int j = 0; j < reqsCount; j++) {
            reqs[i].Add(new Tuple<int,int>(reader.ReadInt32(), reader.ReadInt32()));
        }
        nodes.Add(node);
    }

    for (int i = 0; i < count; i++) {
        foreach (var edgeTo in edges[i]) {
            nodes[i].edgesTo.Add(nodes[edgeTo]);
        }
        foreach (var req in reqs[i]) {
            Requirement requirement = new Requirement();
            requirement.resource = MainController.resources[req.Item1];
            requirement.price = req.Item2;
            nodes[i].requirements.Add(requirement);
        }
    }

    MainController.activeNodes = nodes;
}
```

Рисунок 18 – функція для читання списку задач з бінарного файлу

Останньою проблемою є запис списку списків логічних значень. Стандартний BinaryReader.ReadBoolean та Write записують логічне значення цілим байтом. Щоб це обійти, можна спочатку записувати в змінну 8 біт, і тільки після цього виводити. Так само з читанням – читати одразу 8 біт і після цього перетворювати їх в 8 змінних.

```
void WriteActivatedNodes(BinaryWriter writer) {
    var activatedNodes = MainController.activatedNodes;
    writer.Write(activatedNodes.Count);
    int leftBits = 0;
    int leftover = 0;
    foreach (var layer in activatedNodes) {
        foreach (var b in layer) {
            leftover >>= 1;
            if (b) {
                leftover += 1<<7;
            }
            leftBits++;
            if (leftBits == 8) {
                writer.Write((byte)leftover);
                leftBits = 0;
                leftover = 0;
            }
        }
    }
    if (leftBits != 0) {
        leftover >>= 8 - leftBits;
        writer.Write((byte)leftover);
    }
}

void ReadActivatedNodes(BinaryReader reader, int nodesCount) {
    List<List<bool>> activatedNodes = new List<List<bool>>();
    int count = reader.ReadInt32();
    int leftBits = 0;
    int leftover = 0;
    for (int i = 0; i < count; i++) {
        activatedNodes.Add(new List<bool>());
        for (int j = 0; j < nodesCount; j++) {
            if (leftBits == 0) {
                leftover = reader.ReadByte();
                leftBits = 8;
            }
            activatedNodes[i].Add((leftover & 1) > 0);
            leftover >>= 1;
            leftBits--;
        }
    }
```

Рисунок 19 – функції запису та читання списку активованих задач з бінарного файлу

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
00000000	02	00	00	00	12	43	3a	5c	69	63	6f	6e	73	5c	6d	6f
00000010	6e	65	79	2e	70	6e	67	05	4d	6f	6e	65	79	1e	00	00
00000020	00	11	43	3a	5c	69	63	6f	6e	73	5c	68	61	74	73	2e
00000030	70	6e	67	04	48	61	74	73	00	00	00	00	03	00	00	00
00000040	08	42	75	79	20	68	61	74	73	00	00	00	00	00	00	00
00000050	00	01	01	00	00	00	01	00	00	00	02	00	00	00	00	00
00000060	00	00	0a	00	00	00	01	00	00	00	f6	ff	ff	ff	09	53
00000070	65	6c	6c	20	68	61	74	73	3e	bc	99	43	7a	90	21	43
00000080	01	00	00	00	00	02	00	00	00	01	00	00	00	0a	00	00
00000090	00	00	00	00	00	f1	ff	ff	ff	0a	42	75	69	6c	64	20
000000a0	73	68	6f	70	7c	7a	61	c3	7a	90	21	43	00	01	00	00
000000b0	00	00	00	00	00	02	00	00	00	00	00	00	00	0a	00	00
000000c0	00	00	00	00	00	0a	00	00	00	09	00	00	00	cc	b6	6d
000000d0	03															

Рисунок 19 – приклад бінарного файлу з системою

Equivalent ASCII characters

```
          C : \ i c o n s \ m o
n e y . p n g   M o n e y
    C : \ i c o n s \ h a t s .
p n g   H a t s
  B u y   h a t s


                              S
e l l   h a t s >     C z   ! C

                    B u i l d
s h o p | z a   z   ! C


                              m
```

Рисунок 20 – приклад бінарного файлу з системою з еквівалентними символами ASCII

3.3 Візуалізація графу задач

Для візуалізації графу задач створено компоненту Visualizer. Ця компонента містить дві основні функції що стосуються графу.

Display(List<Node>) – функція що приймає на вхід список задач та відображає його на екрані. Для цього вона створює копію об’єкту-заготовки для кожної задачі що була надана, переміщує її на позицію вказану в класі Node. Далі породжує нащадків від об’єкту поточної задачі для кожного елементу с списку requirements шляхом клонування об’єктів-заготовок для вартості.

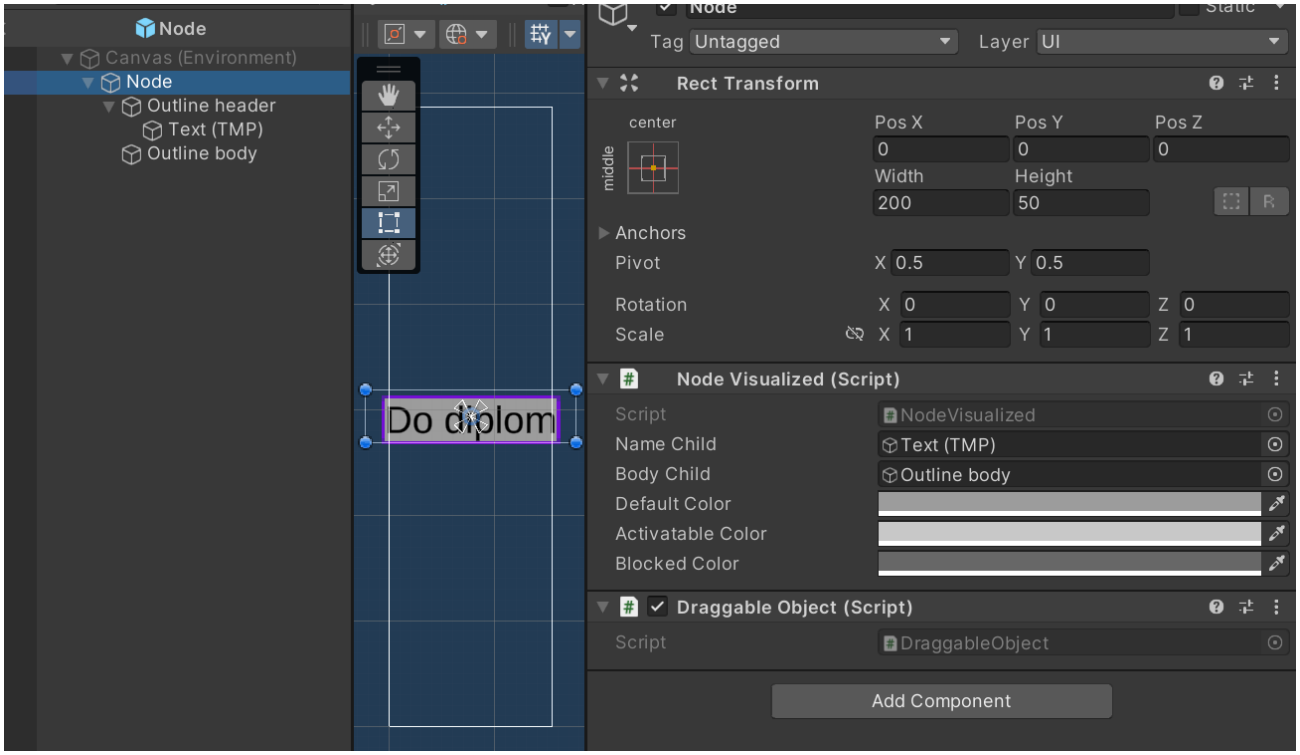

Рисунок 21 – об’єкт-заготовка для задачі

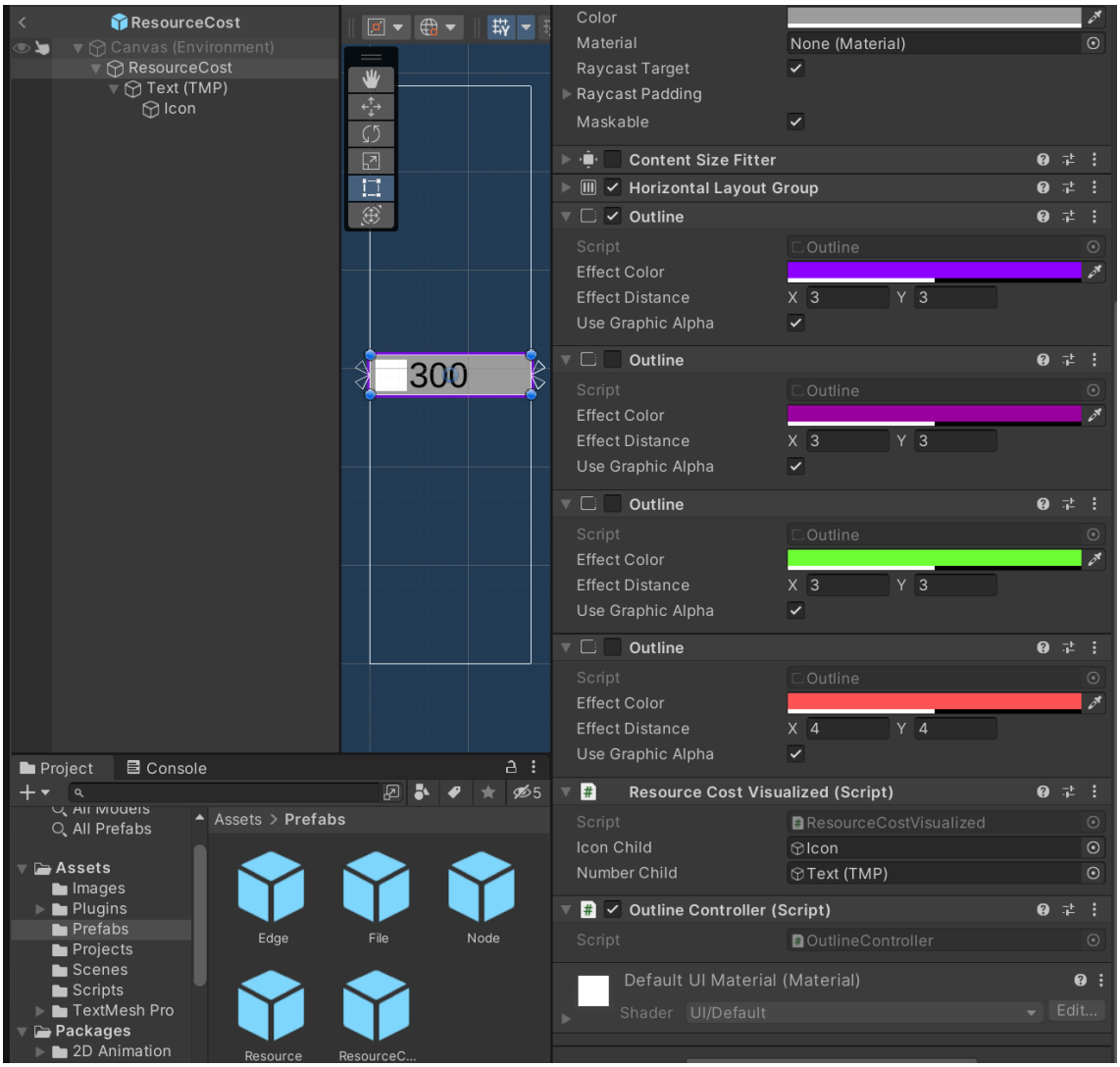

Рисунок 22 – об’єкт-заготовка для вартості задачі

В кінці виконання функція Display також викликає функцію DisplayNodes(List<Node>).

DisplayNodes(List<Node>) – функція що приймає на вхід список задач та відображає на екрані взаємозв’язки між цими задачами. Для цього вона створює об’єкт-заготовку орієнтованого ребра. Точками початку та кінця дуги є не просто точки об’єктів для яких вона створюється. Для правильного розташування також повинні бути враховані ширини об’єктів а також сторона з якої дуга виходить (зліва чи справа). Після цього створений об’єкт дуги розташовується в точці початку, нормалізує свою довжину та множить на

дистанцію між точками початку та кінця. Останнім кроком є встановлення напряму об’єкту на точку кінця дуги.

```
public Tuple<Vector2, Vector2> GetEdgesPosFromNode(GameObject nodeFromObj, GameObject nodeToObj) {
    Vector2 arrowFromPos = ((RectTransform)nodeFromObj.transform).anchoredPosition;
    Vector2 arrowToPos = ((RectTransform)nodeToObj.transform).anchoredPosition;

    var headerParentTransformFrom = nodeFromObj.GetComponent<NodeVisualized>().nameChild.transform;
    var headerParentTransformTo = nodeToObj.GetComponent<NodeVisualized>().nameChild.transform;
    if (arrowFromPos.x < arrowToPos.x) {
        arrowFromPos += new Vector2(((RectTransform)headerParentTransformFrom).sizeDelta.x / 2, 0);
        arrowToPos -= new Vector2(((RectTransform)headerParentTransformTo).sizeDelta.x / 2, 0);
    } else {
        arrowFromPos -= new Vector2(((RectTransform)headerParentTransformFrom).sizeDelta.x / 2, 0);
        arrowToPos += new Vector2(((RectTransform)headerParentTransformTo).sizeDelta.x / 2, 0);
    }

    return Tuple.Create(arrowFromPos,arrowToPos);
}
```

Рисунок 23 – функція що повертає точки виходу та входу дуг між об’єктами

Ці дві функції реалізують більшу частину відображення графу задач. Існує ще декілька скриптів що контролюють вигляд об’єктів до яких вони відносяться в залежності від поточного стану програми.

OutlineController – компонент що керує контуром об’єкту задачі на екрані.

NodeVisualized – компонент що керує кольором заднього фону об’єкту задачі.

Рисунок 24 – візуалізований граф

3.4 Створення інтерактивного поля

Ідея інтерактивного поля схожа з популярними реалізаціями електронних мап, де користувач може самостійно обирати поточний масштаб за допомогою колеса миші та пересувати екран на цікавлячу його територію за допомогою перетягування карти мишею.

Інтерактивне поле є дуже зручним елементом візуалізації. Воно дозволяє рухатись по полю що містить задачі за допомогою миші, віддаляти та приближати його. Це дозволяє користувачу створювати більші за масштабом системи, оскільки він не є обмежений розмірами екрану.

Для його реалізації можна використати властивості ієрархії об’єктів Unity. В цьому рушії якщо призначити один об’єкт нащадком іншого, то позиція об’єкту нащадку буде розрахована відносно батьківського об’єкту. Тобто разом з батьківським об’єктом будуть рухатись всі його нащадки.

Інтерактивне поле реалізується скриптом DraggableField. За допомогою інтерфейсів IBeginDragHandler, IDragHandler та IEndDragHandler ми можемо реєструвати коли користувач починає тягнути за поле, тягне за поле та закінчує тягнути. Все що залишається – це змінювати позицію батьківського об’єкту всіх вершин та зв’язків під час виклику цих функцій таким чином, щоб при перетягування миші її позиція відносно графу залишалась незмінною.

```
public void OnBeginDrag(PointerEventData eventData) {
    Vector2 mousePos = GetMousePositionInCanvas(Input.mousePosition);
    startPositionMouse = mousePos;
    startPositionObj = ((RectTransform)graph.transform).anchoredPosition;
}

public void OnDrag(PointerEventData eventData) {
    Vector2 mousePos = GetMousePositionInCanvas(Input.mousePosition);
    ((RectTransform)graph.transform).anchoredPosition = startPositionObj + (mousePos - startPositionMouse);
}

public void OnEndDrag(PointerEventData eventData) {
    Vector2 mousePos = GetMousePositionInCanvas(Input.mousePosition);
    ((RectTransform)graph.transform).anchoredPosition = startPositionObj + (mousePos - startPositionMouse);
    MainController.ForceDisplay();
}
```

Рисунок 25 – реалізація функцій перетягування інтерактивного поля

Для зміни масштабу можна використати схожу техніку. Всі об’єкти-нащадки будуть змінювати свій розмір якщо ми будемо масштабувати батьківський об’єкт.

За допомоги функції OnGUI() можна реєструвати крутіння колеса миші. Отже все що залишається – змінювати масштаб батьківського об’єкту в сторону, в яку було прокручене колесо.

```
void OnGUI()
{
    graph.transform.localScale *= (1 + Input.mouseScrollDelta.y * scaleRate);
}
```

Рисунок 26 – реалізація функцій масштабування інтерактивного поля

Для покращення інтерактивності поля можна надати можливість користувачу переміщувати задачі мишею. Це можна зробити додавши схожу компоненту на кожний об’єкт задач (тобто до об’єкту-заготовки). Відмінністю буде те, що рухатись буде не весь граф, а тільки певна задача, а також те що потрібно буде оновлювати позицію задачі в основному списку задач.

```
public void OnBeginDrag(PointerEventData eventData) {
    MainController.fieldsController.SelectNode(nodeNumber);

    Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
    startPositionMouse = mousePos;
    startPositionNode = MainController.activeNodes[nodeNumber].pos;
    startPositionObj = ((RectTransform)this.transform).anchoredPosition;
}

public void OnDrag(PointerEventData eventData) {
    Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
    MainController.activeNodes[nodeNumber].pos = startPositionNode + mousePos - startPositionMouse;
    ((RectTransform)this.transform).anchoredPosition = startPositionObj + mousePos - startPositionMouse;
    MainController.ForceDisplayEdges();
}

public void OnEndDrag(PointerEventData eventData) {
    Vector2 mousePos = MainController.edgeCreator.GetMousePositionInCanvas(Input.mousePosition);
    MainController.activeNodes[nodeNumber].pos = startPositionNode + mousePos - startPositionMouse;
    MainController.ForceDisplay();
}
```

Рисунок 27 – реалізація функцій масштабування інтерактивного поля

3.5 Симуляція створених систем

Режим симуляції представляє собою особливий режим роботи системи. Під час симуляції заборонено змінювати структуру проєкту – додавати чи знищувати задачі та створювати між ними залежності. Натомість з'являється можливість керування списками активованих задач та покроково симулювати систему.

Для керування програмою в даному режимі була розроблена компонента SimulationController. В період симуляції вона завжди підтримує поточний крок симуляції. При зміні кроку вона може відобразити необхідні зміни за допомогою функції DisplayStep.

```
public void DisplayStep() {
    if (!isSimulating) {
        return;
    }
    if (currentStep == MainController.activatedNodes.Count) {
        List<bool> newLayer;
        if (currentStep == 0) {
            newLayer = new List<bool>();
            for (int i = 0; i < MainController.activeNodes.Count; i++) {
                newLayer.Add(false);
            }
        } else {
            newLayer = new List<bool>(MainController.activatedNodes[currentStep - 1]);
            for (int i = 0; i < MainController.activeNodes.Count; i++) {
                if (!MainController.activeNodes[i].isRepeatable) {
                    newLayer[i] = false;
                }
            }
        }
        MainController.activatedNodes.Add(newLayer);
    }

    var resourcesCalculated = CalculateResourcesValues();
    resourcesValues = resourcesCalculated.Item1;
    completed = resourcesCalculated.Item2;

    UpdateResourceCounts();
    MainController.ForceDisplay();
}
```

Рисунок 28 – реалізація функції відображення кроку симуляції

Якщо ми переходимо до нового кроку симуляції – ми копіюємо попередній список активованих задач, при умові що вони можуть бути виконані повторно. Інакше ми беремо список відповідного кроку.

Для розрахунку кількості поточних ресурсів дана компонента має функцію CalculateResourcesValues. Вона проходить по всіх активованих задачах та вираховує їх вартість з кількості доступних ресурсів. В процесі вона також формує список задач які були виконані до поточного кроку симуляції.

```
Tuple<List<int>, List<bool>> CalculateResourcesValues() {
    List<int> resourcesValues = new List<int>();
    List<bool> completed = new List<bool>();
    foreach (var resource in MainController.resources) {
        resourcesValues.Add(resource.amount);
    }
    foreach (var node in MainController.activeNodes) {
        completed.Add(false);
    }

    for (int i = 0; i < currentStep; i++) {
        for (int nodeId = 0; nodeId < MainController.activeNodes.Count; nodeId++) {
            if (MainController.activatedNodes[i][nodeId] == false) {
                continue;
            }
            foreach (var req in MainController.activeNodes[nodeId].requirements) {
                if (MainController.GetResourceId(req.resource) != -1) {
                    resourcesValues[MainController.GetResourceId(req.resource)] -= req.price;
                }
            }
            completed[nodeId] = true;
        }
    }

    return new Tuple<List<int>, List<bool>>(resourcesValues, completed);
}
```

Рисунок 29 – реалізація функції обрахунку ресурсів

3.6 Взаємодія користувача з системою

Для коректної роботи додатку необхідно надати користувачу можливість вносити зміни до системи за допомогою графічних елементів. Це забезпечується кнопками, полями для вводу та розкривними списками.

Ця взаємодія забезпечується переважно двома створеними компонентами.

EdgeCreator – це компонента що забезпечує створення взаємозв’язків між об’єктами. При натисканні кнопки що створена за допомогою редактора Unity викликається функція StartCreating цієї компоненти.

Ця функція починає роботу компоненти та дозволяє їй відслідковувати події наведення миші на об’єкти задач. Вона відслідковує номер задачі на якій було почато створення взаємозв’язку та номер задачі на яку в поточний момент наведений вказівник.

Після початку створення взаємозв’язку починає працювати функція Update що викликається на кожному кадрі програми. Вона створює направлене ребро від задачі на якій було почато створення та до вказівника миші чи задачі на яку вказує вказівник миші, якщо вона є.

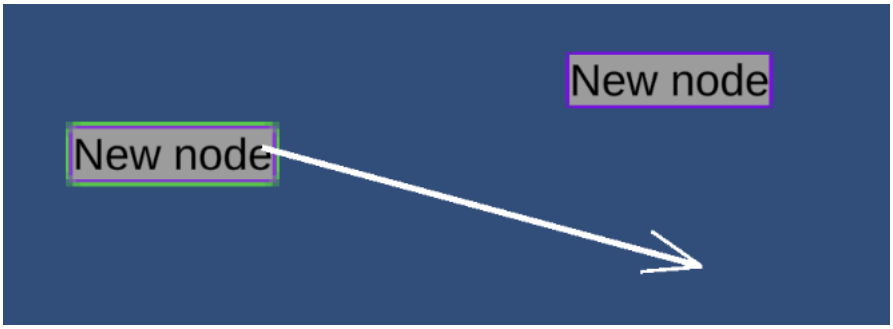

Рисунок 30 – вигляд тимчасової дуги, коли вказівник не вказує на задачу

Якщо миша вказує на задачу – то дугу треба відображати не в точці вказівника, а зліва чи справа від заголовку самої задачі. Для цього можна перевикористовувати функцію компоненти Visualizer – GetEdgesPosFromNode що була розглянута вище.

Рисунок 31 – вигляд тимчасової дуги, коли вказівник вказує на задачу

Для підтвердження створення дуги користувач повинен натиснути лівою кнопкою миші на існуючий ресурс. Це закінчить процес створення та додасть взаємозв’язок між задачами.

```
void Update() {
    if (arrowObj != null) {
        Destroy(arrowObj);
    }
    if (!isCreating) {
        return;
    }

    Vector2 arrowToPos;
    Vector2 arrowFromPos;

    if (node2Id == -1) {
        arrowToPos = GetMousePositionInCanvas(Input.mousePosition) - graphTransform.anchoredPosition / graphTransform.localScale;
        var arrowToPosDir = new Vector2(arrowToPos.x, arrowToPos.y);
        arrowFromPos = visualizer.GetEdgesPosFromNode(visualizer.GetNodeObj(MainController.activeNodes[node1Id]), arrowToPos);
    } else {
        var arrowPos = visualizer.GetEdgesPosFromNode(
            visualizer.GetNodeObj(MainController.activeNodes[node1Id]), visualizer.GetNodeObj(MainController.activeNodes[node2Id]));
        arrowFromPos = arrowPos.Item1;
        arrowToPos = arrowPos.Item2;
    }
        arrowObj = Instantiate(edgePrefab, Vector3.zero, Quaternion.identity, edgesParent.transform);
        var arrow_line_transform = (RectTransform)(arrowObj.GetComponent<EdgeVisualized>().arrowLine.transform);

        int offset = 0;
        int sizeDecrease = 5;
        if (node2Id != -1) {
            offset = 5;
            sizeDecrease = 0;
        }
        arrow_line_transform.sizeDelta = new Vector2(Vector2.Distance(arrowFromPos, arrowToPos) - offset, arrow_line_transform.rect.height);

        Vector2 direction = arrowToPos - arrowFromPos;
        direction.Normalize();
        ((RectTransform)arrowObj.transform).anchoredPosition = arrowToPos - direction * sizeDecrease;
        arrowObj.transform.right = arrowToPos - arrowFromPos;
}
```

Рисунок 32 – реалізація функції відображення дуги що створюється

NodeDataFieldsController – це компонента що забезпечує відображення та можливість зміни значень полів задач, а також дає можливість обирати задачі.

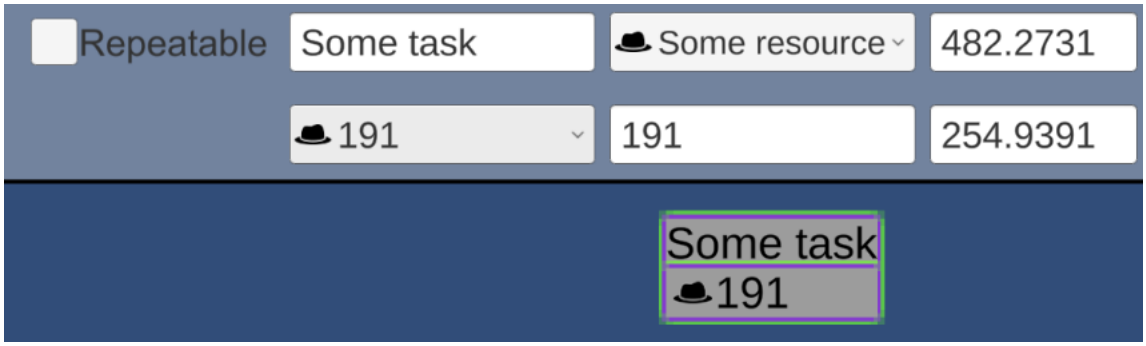

Рисунок 33 – поля задач що доступні користувачу

Після обрання задачі значення її змінних записуються в поля наведені на рисунку 33. Для цього NodeDataFieldsController має 7 функцій.

UpdateXField – оновлює значення поля координати x.

UpdateYField – оновлює значення поля координати y.

UpdateRequirementField – оновлює значення випадного списку поля потреб. Для цього з поля видаляються всі значення, а потом додаються всі потреби обраної задачі. Також вона додає пункт для створення нової потреби в кінці списку поля. Блокує поля ресурсу та ціни потреби якщо після оновлення потреба не обрана.

UpdateResourceField – оновлює значення випадного списку поля ресурсів поточної потреби. Для цього з поля видаляються всі значення, а потім додає всі існуючі ресурси.

UpdatePriceField – оновлює кількість ресурсів поточної потреби.

UpdateNameField – оновлює значення поля на назву поточної задачі.

UpdateRepeatable – оновлює значення флагу повторюваності задачі.

Також існує функція UpdateAllObj що оновлює всі поля за допомогою виклику наведених вище функцій.

```
public void UpdateRequirementField() {
    if (currentNode == -1) {
        return;
    }
    requirementField.ClearOptions();
    List<TMP_Dropdown.OptionData> reqs = new List<TMP_Dropdown.OptionData>();
    reqs.Add(new TMP_Dropdown.OptionData("    "));

    foreach(var req in MainController.activeNodes[currentNode].requirements) {
        reqs.Add(new TMP_Dropdown.OptionData(req.price.ToString(), req.resource.icon));
    }

    reqs.Add(new TMP_Dropdown.OptionData("New requirement", newButtonImage));

    requirementField.AddOptions(reqs);

    if (currentRequirement != -1) {
        requirementField.SetValueWithoutNotify(currentRequirement + 1);
    } else {
        currentRequirement = requirementField.value - 1;
    }

    resourceField.interactable = (currentRequirement != -1);
    priceField.interactable = (currentRequirement != -1);

}
```

Рисунок 34 – функція оновлення випадного списку поля потреб

При зміні значень в полях – ці значення потрібно передати в головний контролер та оновити там відповідні списки. Для цього NodeDataFieldsController реалізує 7 функцій: OverwriteXField, OverwriteYField, OverwriteRequirementField, OverwriteResourceField, OverwritePriceField, OverwriteNameField та OverwriteRepeatableField.

Ці функції є схожими на 7 функцій описаних вище, та вони записують значення в іншу сторону – з значення поля в значення списку.

```
public void OverwritePriceField() {
    if (currentNode == -1) {
        return;
    }
    if (priceField.interactable == false) {
        return;
    }

    MainController.activeNodes[currentNode].requirements[currentRequirement].price = ParseInt(priceField.text);
    MainController.ForceDisplay();
    MainController.simulationController.DisplayStep();
}
```

Рисунок 35 – функція перезапису ціни потреби

Для числових значень є окрема умова що строка «-» відповідає нулю та не перезаписується. Це реалізовано за допомогою функцій ParseInt та ParseFloat та зроблено для того щоб можна було зручно вводити від'ємні числа. Інакше потрібно було спочатку вводити число а вже після цього ставити мінус на початку.

```
float ParseFloat(string s) {
    if (string.IsNullOrEmpty(s)) {
        return 0.0f;
    }
    if (s == "-") {
        return -0.0f;
    }
    return float.Parse(s);
}

int ParseInt(string s) {
    if (string.IsNullOrEmpty(s)) {
        return 0;
    }
    if (s == "-") {
        return -0;
    }
    return int.Parse(s);
}
```

Рисунок 36 – функції перетворення строк в числові типи

Для взаємодії з користувачем також існує декілька невеликих компонент що слідкують за взаємодію курсора миші з різними елементами системи.

Наприклад, компонента NodeVisualized що відслідковує натиск курсору на задачу та передає її відповідному контролеру. Для лівої кнопки миші – передає компоненті що створює ребра подію обрання задачі. Для правої кнопки миші передає головному контролеру що потрібно видалити задачу чи активізувати її для наступного кроку симуляції, в залежності від того чи режим симуляції є ввімкнутим.

```
public void OnPointerClick (PointerEventData pointerEventData)
{
    if (pointerEventData.button == PointerEventData.InputButton.Left) {
        MainController.edgeCreator.nodePointerClick(nodeNumber);
    }

    if (pointerEventData.button == PointerEventData.InputButton.Right) {
        if (!MainController.simulationController.isSimulating) {
            MainController.RemoveNode(nodeNumber);
        } else {
            if (MainController.simulationController.IsActivatable(nodeNumber)) {
                MainController.simulationController.ActivateNode(nodeNumber);
            }
        }
    }
}
```

Рисунок 37 – функція натискання курсором миші на задачу

4. ТЕСТУВАННЯ СИСТЕМИ

4.1 Інтерфейс та структура додатку

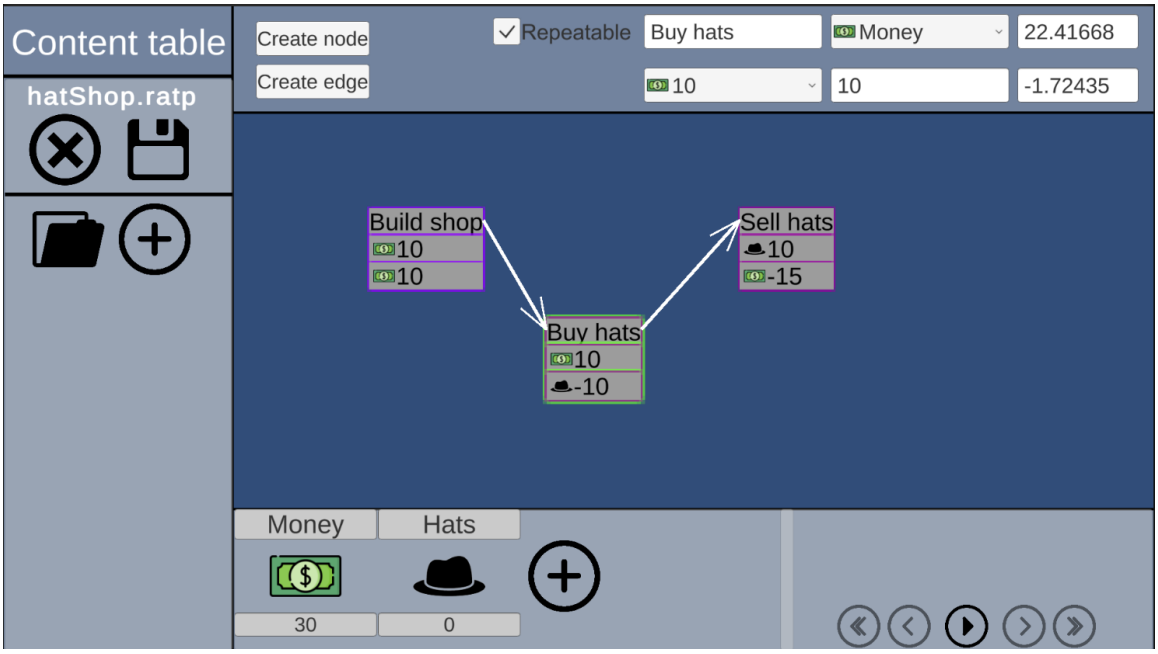

Рисунок 38 – інтерфейс додатку в звичайному режимі

Інтерфейс додатку поділено на 5 областей.

Ліва частина дозволяє керувати файлами з системами. Там розташовані кнопки для створення нових файлів та відкриття вже існуючих файлів. Також, для вже відкритих файлів, там знаходяться кнопки повторного відкриття файлу (якщо файл не відкритий) чи збереження (якщо файл вже відкритий) а також кнопка видалення файлу з історії. Файл що є активним в поточний момент робить свою назву жирною.

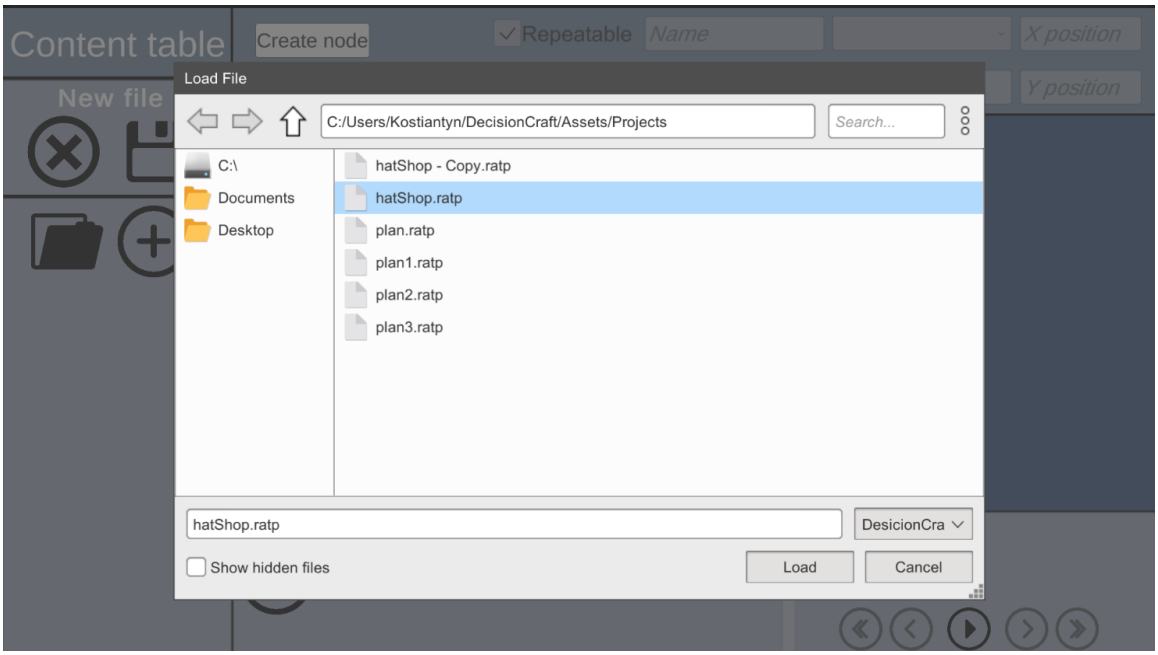

Рисунок 39 – інтерфейс додатку під час обрання файлу для відкриття

Під час обрання файлу для завантаження користувачу відкривається файловий провідник. Під час того як провідник відкритий – всі інші дії в системі заблоковані. Це зробленого для того, щоб в той час поки користувач користується інспектором він випадково не змінював систему.

Верхня частина додатку дозволяє керувати структурою графу та значеннями полів задач. Там розташовані кнопки для створення взаємозв’язків між задачами а також поля самих задач.

Рисунок 41 – інтерфейс керування структурою графу та задачами

Всі поля що там розташовані пов’язані з поточно обраною задачею. Обрати задачу можна натиснувши на неї лівої кнопкою миші. Для зручності, обрана задача відображається з зеленим контуром.

Загалом там розташовано 7 полів:

Багаторазова – поле що відображає та дає можливість змінити змінну isRepeatable для кожної задачі. Задачі що мають цей флаг встановленим мають спеціальний колір контуру та можуть бути виконані безліч раз на етапі симуляції.

Рисунок 40 – одноразова задача (зліва) та багаторазова (справа)

Назва – поле що дає можливість змінити назву задачі. Назва задачі відображається в заголовку кожної задачі що знаходиться в графі.

Вартість – поле що керує вартістю задачі. Воно являє собою спадний список вартості ресурсу. Цей спадний список також містить кнопку для створення додаткової вартості для обраного ресурсу.

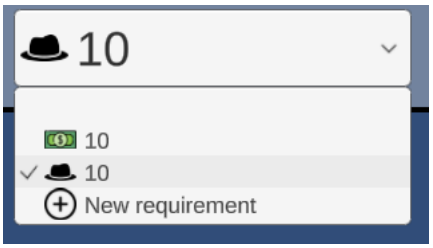

Рисунок 42 – спадний список вартості

Ресурс поточної вартості – поле що дає можливість змінити ресурс, який необхідно витратити для виконання задачі. Воно є активним тільки тоді, коли в спадному списку вартості обрано поле. Воно являє собою спадний список що містить у собі всі створені ресурси.

Рисунок 43 – спадний список ресурсів

Кількість ресурсу – поле що дає можливість змінити кількість ресурсі які необхідно витратити для виконання задачі. Воно є активним тільки тоді, коли в спадному списку вартості обрано поле. Воно являє собою поле для вводу цілих чисел.

Координата X – місце розташування ресурсу на полі по осі X. Також може бути змінена за допомогою перетягування задачі мишею.

Координата Y – місце розташування ресурсу на полі по осі Y. Також може бути змінена за допомогою перетягування задачі мишею.

Кнопка для створення задач створює пусту задачу в центрі графу.

Кнопка для створення взаємозв’язків починає процес створення зв’язку. Після її натискання з’являється тимчасовий зв’язок від обраної задачі до позиції курсору користувача. Якщо в цей час натиснути лівою кнопкою миші на будь-яку іншу задачу – між нею та поточною задачею буде створено взаємозв’язок. Якщо натомість повторно натиснути на кнопку створення взаємозв’язків – процес створення закінчиться без створення зв’язку.

Центральна частина додатку відображає граф та дозволяє керувати ним за допомогою інтерактивного поля та інтерактивних задач.

Користувач має змогу змістити поле в будь яку сторону за допомогою перетягування мишею. Також є можливість змінювати положення задач шляхом перетягування їх мишею.

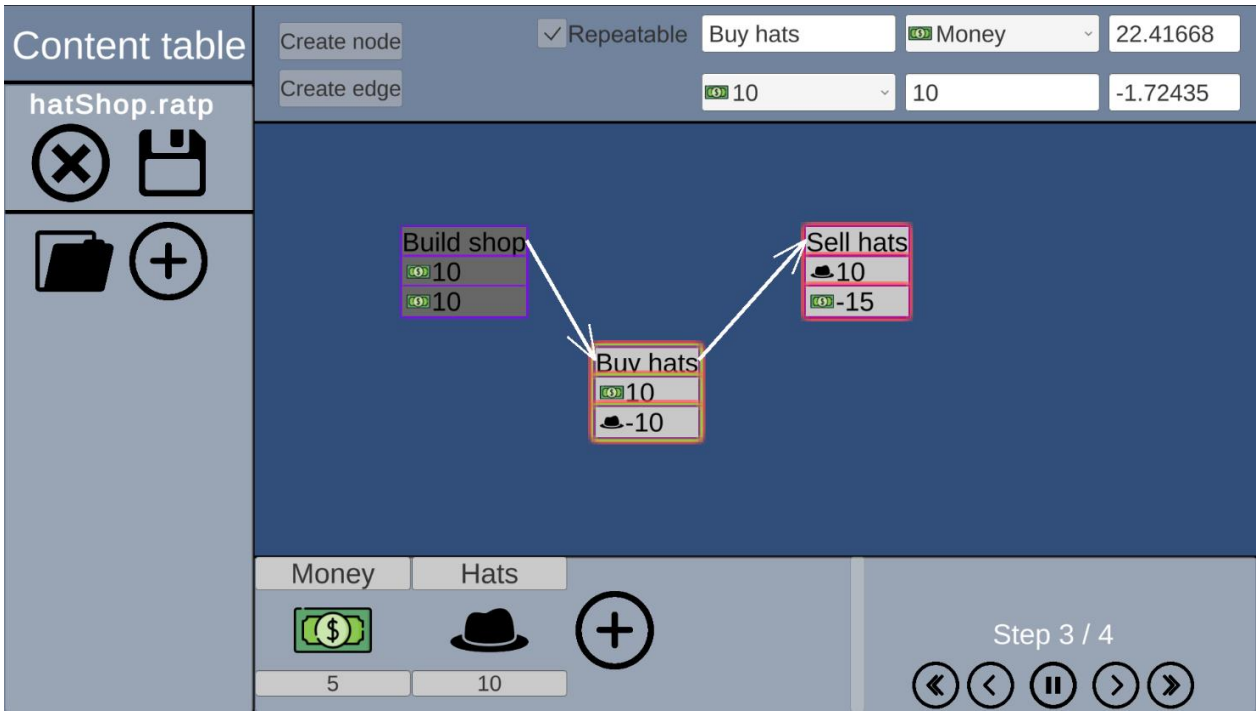

Рисунок 44 – інтерфейс додатку в режимі симуляції

Нижня частина додатку дозволяє керувати ресурсами, їх іконками, значеннями та назвами. Там розташована кнопка для створення нових ресурсів.

Кнопка створення ресурсу створює пустий ресурс, з білим прямокутником в якості іконки.

Поля імені та кількості ресурсів можуть бути змінені шляхом введення в них значень.

Іконка ресурсу може бути змінена шляхом натискання лівою клавішою миші на неї.

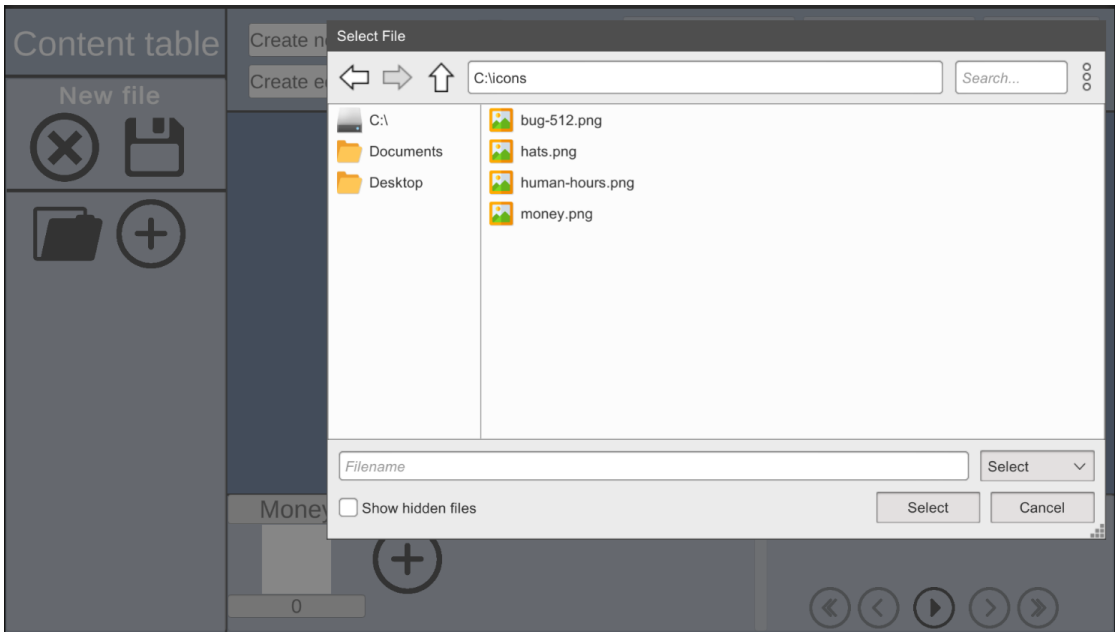

Рисунок 45 – інтерфейс додатку під час зміни картинки ресурсу

При її натисканні решта елементів блокуються і відображається активне вікно для обрання файлів. При обранні файлу формату png іконка ресурсу змінюється усюди де вона відображається.

Нижня права частина додатку дозволяє керувати симуляцією системи. Там розташовано 5 кнопок та текст що відображає номер поточного кроку. Розглянемо ці кнопки.

Перший крок – це кнопка що переводить симуляцію на її перший крок. На цьому кроці не було виконано ще ні одної задачі, отже кількість ресурсів співпадає з їх початковою кількістю.

Попередній крок – це кнопка що переводить симуляцію на попередній крок. При цьому значення активованих задач на поточному кроці також зберігається, що дає можливість повернутись до нього пізніше.

Запуск/Пауза симуляції – це кнопка що починає симуляцію якщо вона ще не ввімкнена чи виключає її якщо вона ввімкнена.

Наступний крок – це кнопка що переводить симуляцію на наступник крок. Якщо наступний крок ще не був створений – генерує новий крок.

Останній крок – це кнопка що переводить симуляції в останній крок який був створений.

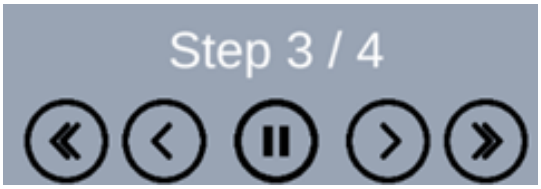

Рисунок 46 – кнопки керування симуляцією

Взаємозв’язки між задачами, самі задачі а також ресурси можуть бути видалені з системи за допомогою натискання правої кнопки миші по ним в звичайному режимі роботи.

При видалені ресурсу – всі задачі що коштують цей ресурс оновлюються щоб більше не включати цей ресурс.

При видалені задачі – всі зв’язки з та до цієї задачі видаляються.

В Режимі симуляції права кнопка миші використовується для активізації задачі на поточному кроці.

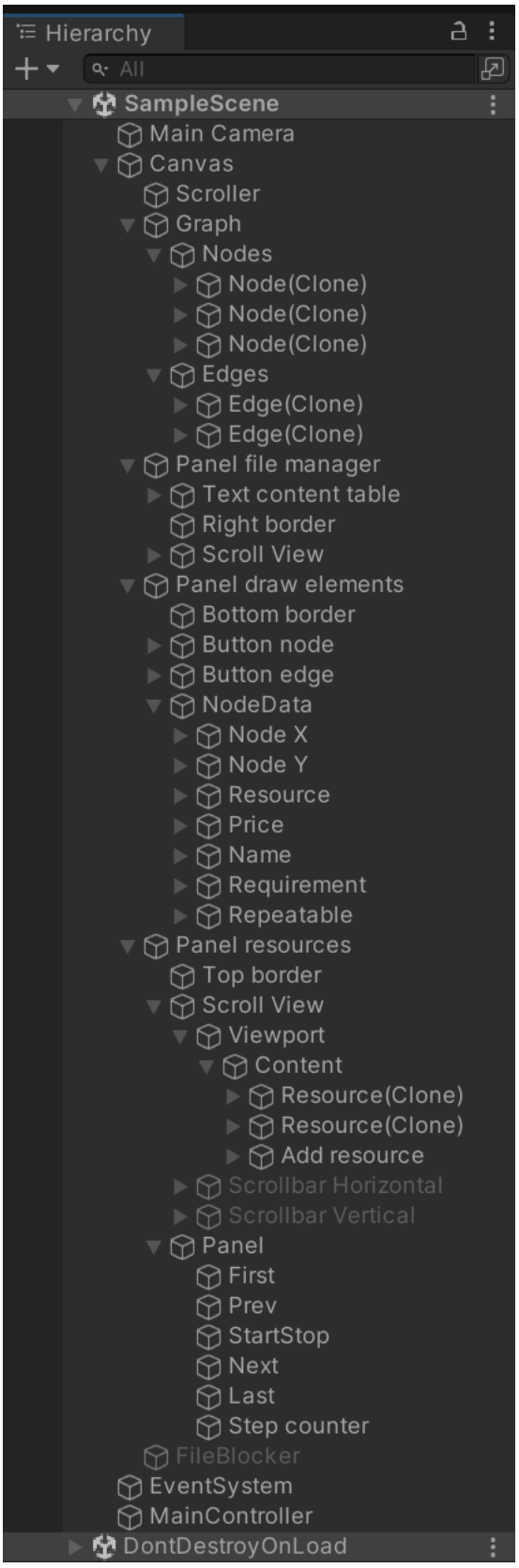

Рисунок 47 – ієрархія основної сцени проєкту

4.2 Система планування та керування ресурсами стартапу

Розглянемо Систему планування та керування ресурсами стартапу створену за допомогою розробленого додатку.

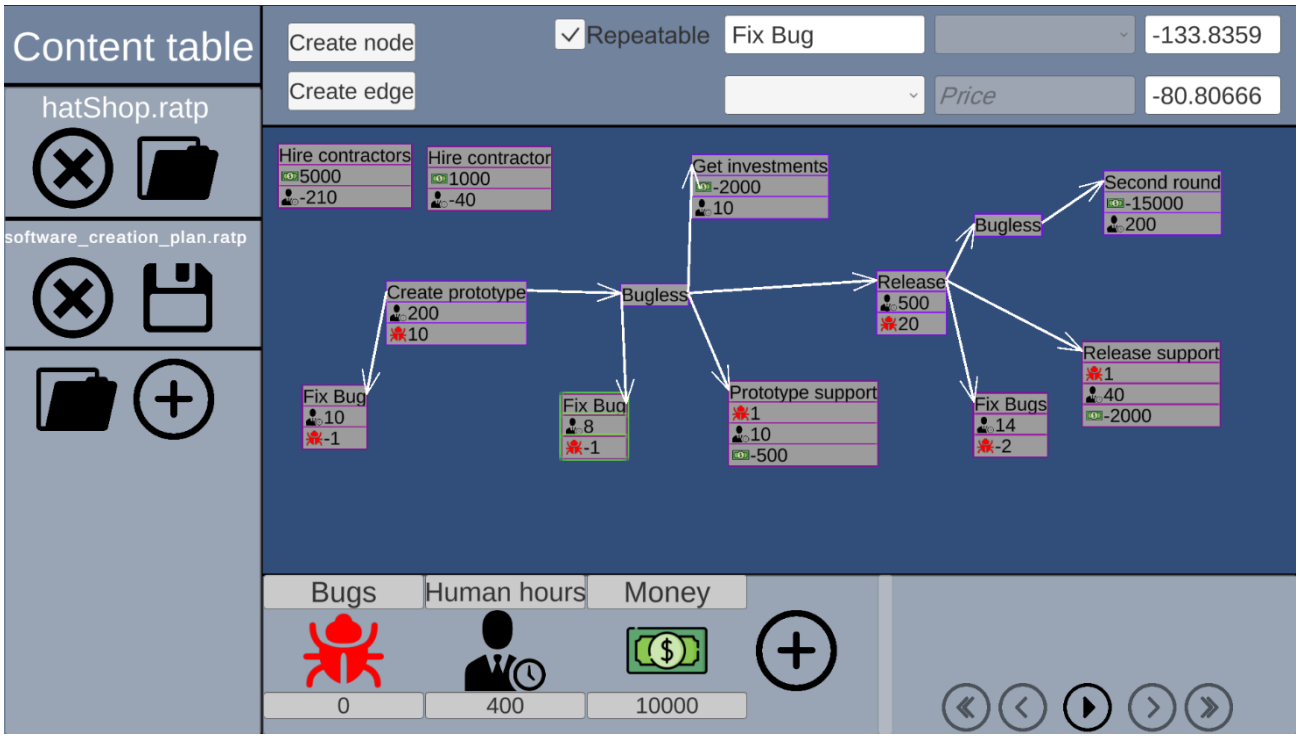

Рисунок 48 – початковий стан системи стартапу

Система планування та керування ресурсами стартапу

Дана система має 3 види ресурсів.

- Кількість помилок в програмі
- Доступні людино-години
- Гроші

Нехай метою є досягнення другого раунду інвестицій.

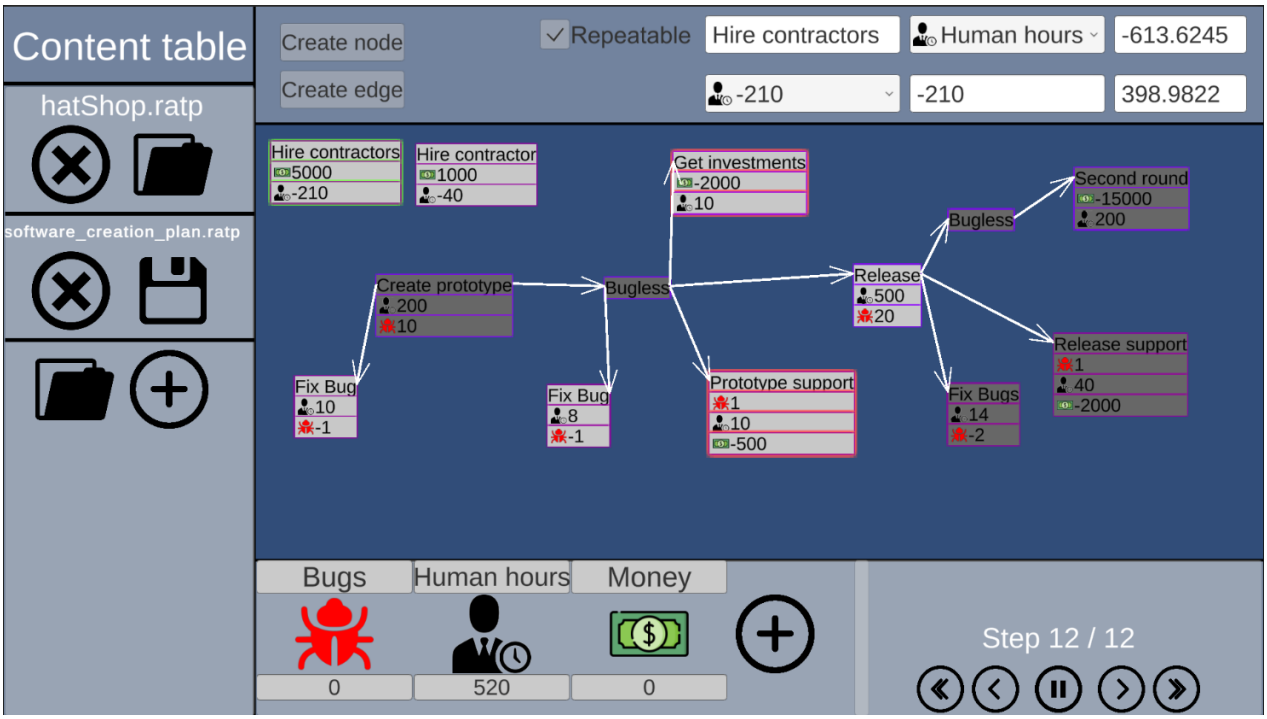

Рисунок 49 – проміжний етап симуляції системи стартапу

За 12 кроків було створено прототип, приведено кількість помилок до нуля та заплановано отримати інвестиції. Також декілька разів виконувався набір людей.

В процесі також відкрилась можливість виконати задачу запуску програми, що в свою чергу відкриє додаткові можливості.

В результаті кількість грошей та помилок була рівна нулю, в той час як компанія мала в наявності 520 людино-годин.

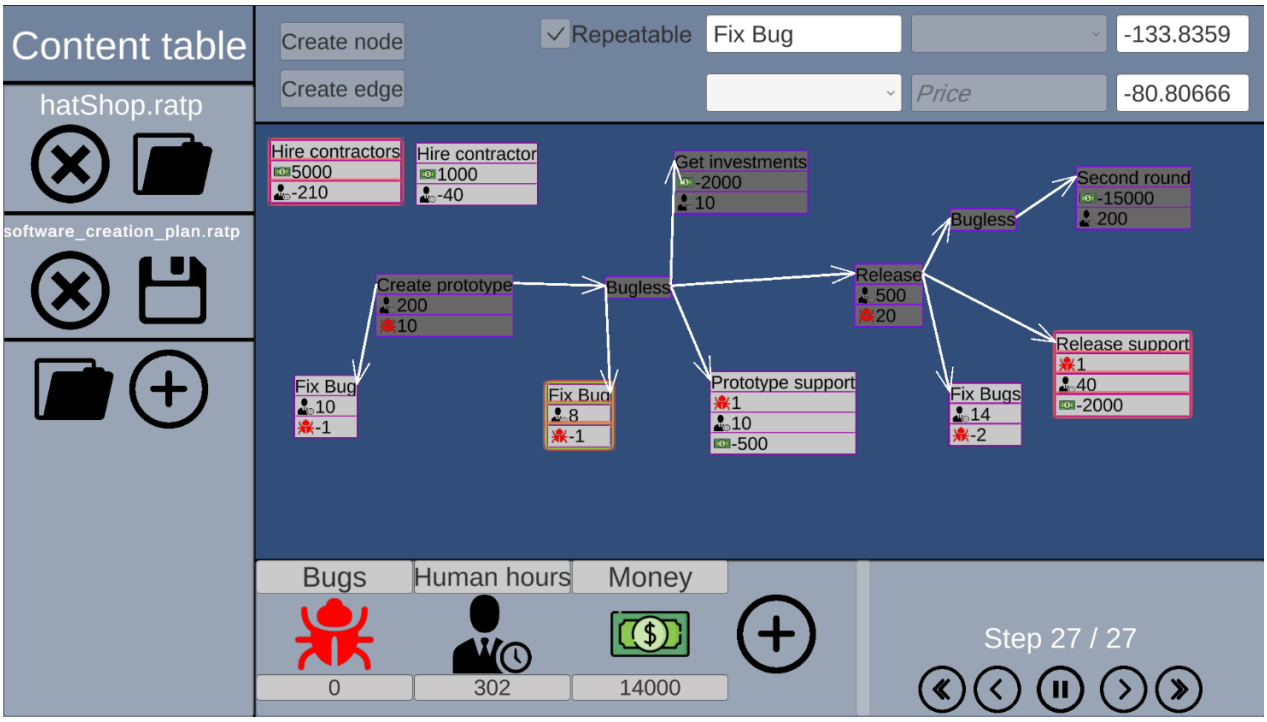

Рисунок 50 – останній крок симуляції системи стартапу

На кроці 27 було успішно виконано задачу другого раунду інвестицій.

Під час симуляції система правильно відображала кількість наявних ресурсів, не давала виконати задачі передумови яких не були виконані.

Покрокове керування давало можливість побачити результат при певному наборі активованих задач та, за необхідністю, повернутись назад та змінити набір на інший.

Це дало можливість розробити систему що виконує поставлену задачу та впевнитись в її вірності.

За допомогою панелі керування файлами ця система була збережена в файл software_creation_plan.ratp. При перезапуску програми та повторному відкриттю цього файлу система була відтворена в тому ж стані, в якому вона була на момент збереження.

ВИСНОВОК

Система планування та розподілу ресурсів для створення засобів для прийняття рішень, що є результатом дипломного проєкту, є зручною в використанні.

Аналіз існуючих рішень, що був проведений в першому розділі дипломного проєкту, виявив що наразі не існувало такої системи, що могла б дозволити планувати та симулювати системи що містять користувацький набір ресурсів, тому розробка була доцільною.

Створена система дозволяє виконувати такі функції:

- Створювати в графі задачі що мають свою вартість в довільних ресурсах
- Створювати взаємозв’язки між задачами виду: В може бути виконано лише після виконання А
- Виконувати покрокову симуляцію створених систем з автоматичним обрахуванням поточної кількості ресурсів та забезпеченням притриманню створеним взаємозв’язкам
- Зручно керувати положення графу. Це виражається можливостями рухати як весь граф, так і окремі задачі
- Зберіганням створених систем в бінарні файли малих розмірів
- Завантаженням створених систем з бінарних файлів.

При реалізації системи було розроблено алгоритми відображення та зберігання систем планування.

Для зберігання системи достатньо трьох списків – списку задач у вигляді класу Node, списку ресурсів у вигляді класу Resource та списку активованих задач для кожного кроку симуляції у вигляді списків бітів.

На основі цих списків можна відобразити довільну систему у вигляді зручному для користувача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Годболд Е. Mastering UI Development with Unity. Packt, 2018. 468 с.
2. Документація Unity. *Unity*: URL: https://docs.unity.com/ (дата звернення 21.05.2023).
3. Документація C#. *Microsoft*: URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення 21.05.2023).
4. 10 Best Project Management Software Of 2023. *Forbes*: URL: https://www.forbes.com/advisor/business/software/best-project-management-software/ (дата звернення 21.05.2023).
5. *ClickUp*: URL: https://clickup.com/ (дата звернення 21.05.2023).
6. *Trello*: URL: https://trello.com/uk (дата звернення 21.05.2023).
7. *Wrike*: URL: https://www.wrike.com/ (дата звернення 21.05.2023).