НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

УДК 004.94

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: «*Петрі-об’єктне моделювання систем із використанням 3D-анімації*»

Виконав: студент *4* курсу, групи *ІС-52*
Мельничук Володимир Ігорович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *проф., д.т.н., доц., Стеценко І.В.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *ст.викл. Халус О. А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц. каф. ПЗКС ФПМ, к.т.н. Олещенко Любов Михайлівна* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) *6.050101*

«Комп’ютерні науки» («Інформаційні управляючі системи та технології»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)

“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Мельничуку Володимиру Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема проекту « *Петрі-об’єктне моделювання систем із використанням 3D-анімації* »

керівник проекту *Стеценко Інна Вячеславівна, д.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “*23*”*квітня* 2019 р. №1181-с

2. Термін подання студентом проекту “*03*”*червня* 2019 року

3. Вихідні дані до проекту

Технічне завдання

4. Зміст пояснювальної записки

1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд аналогів, постановка задачі

2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних

3. Математичне забезпечення: змістовна та математична постановки задачі, алгоритм імітації Петрі-об’єктної моделі та опис методу розв’язання

4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використань

3. Схема масивів інформації

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності

6. Схема структурна компонентів програмного забезпечення

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«15» лютого 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*16.02.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*18.02.2019*	
3.	*Постановка та формалізація задачі*	*19.02.2019*	
4.	*Розробка інформаційного забезпечення*	*20.02.2019*	
5.	*Алгоритмізація задачі*	*30.02.2019*	
6.	*Обґрунтування використовуваних технічних засобів*	*03.03.2019*	
7.	*Розробка програмного забезпечення*	*04.03.2019*	
8.	*Налагодження програми*	*10.05.2019*	
9.	*Виконання графічних документів*	*15.05.2019*	
10.	*Оформлення пояснювальної записки*	*28.05.2019*	
11.	*Подання ДП на попередній захист*	*30.05.2019*	
12.	*Подання ДП на основний захист*	*03.06.2019*	
13.	*Подання ДП рецензенту*	*05.06.2019*	

Студент _____ В.І. Мельничук
(підпис)

Керівник проекту _____ І.В. Стеценко
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
			Документи текстові		
1	А4	ДП ІС-5217.1181-с.ПЗ	Пояснювальна записка	74	
2	А4	ДП ІС-5217.1181-с.ТЗ	Технічне завдання	9	
3	А4	ДП ІС-5217.1181-с.ПМВ	Програма та методика випробувань	10	
			Матеріали графічні		
4	А3	ДП ІС-5217.1181-с.ССД	Схема структурна діяльності	1	
5	А3	ДП ІС-5217.1181-с.ССВ	Схема структурна варіантів використань	1	
6	А3	ДП ІС-5217.1181-с.СМІ	Схема масивів інформації	1	
7	А3	ДП ІС-5217.1181-с.ССК	Схема структурна класів програмного забезпечення	1	
8	А3	ДП ІС-5217.1181-с.ССП	Схема структурна послідовності	1	
9	А3	ДП ІС-5217.1181-с.ССК	Схема структурна компонентів програмного забезпечення	1	

Зм.	Арк.	ПІП	Підп.	Дата	ДП ІС-5217.1181-с.ВДП		
Розробн.		Мельничук В.І.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Стеценко І.В.					
Консульт.						КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52	
Н/контр.		Халус О.А.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проекту

на тему: Петрі-об’єктне моделювання систем із використанням 3D-анімації

Київ – 2019 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з шести розділів, містить 107 сторінок, 25 рисунків, 9 таблиць, 1 додаток, 23 джерела.

Дипломний проект присвячений розробці алгоритму імітаційного Петрі-об’єктного моделювання систем з використанням 3D-анімації.

У розділі загальних положень були розглянуті процеси діяльності, предметне середовище та наявні аналоги програмного продукту.

У розділі інформаційного забезпечення розглянуто вхідні та вихідні дані, описана структура масивів інформації.

У математичному розділі наведена змістовна постановка задачі, математична постановка задачі, алгоритм імітації Петрі-об’єктного моделювання, та опис методу імітації в режимі реального часу.

У розділі з програмного забезпечення наведена діаграма класів, їх взаємодія у вигляді послідовності та основні компоненти програми. Наведені вимоги до технічного забезпечення.

У технологічному розділі наведено керівництво користувача.

МЕРЕЖА ПЕТРІ, ПЕТРІ-ОБ'ЄКТНА МОДЕЛЬ, МОДЕЛЮВАННЯ СИСТЕМ, ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ, 3D-АНІМАЦІЯ, ДИСКРЕТНОПОДІЙНА СИСТЕМА

ABSTRACT

Structure and scope of work. The explanatory note of the graduation project consists of six sections, containing 107 pages, 25 pictures, 9 tables, 1 application, 23 sources.

This graduation project is dedicated to the development of Petri-object system simulation algorithm with use of 3D animation.

In the section of the general provisions, the processes of activity, the subject environment and the available analogues of the software product were considered.

In the section of the information support, the input and output data are considered, the database is described.

In mathematical section presents the mathematical formulation of the problem, the general formulation of the problem, the algorithm for solving problem of Petri-objects system simulation, and the method of solving this problem in real time.

In the software section, the structure of classes, their relations and interactions in sequence and the main components of the program are presented. The requirements for the technical support are specified.

The technology section provides user guides and software test methods.

PETRI NET, PETRI-OBJECT MODEL, SYSTEM SIMULATION, SIMULATION MODELING, 3D ANIMATION, DISCRETE-EVENT SYSTEM.

ЗМІСТ

ВСТУП

Моделювання – найефективніший спосіб аналізу комплексних систем не тільки масового обслуговування, але й іншого призначення – біологічних, інформаційних, технічних. Його використовують на етапі проектування та експлуатації систем. З розвитком обчислювальних можливостей сучасних комп'ютерів даний метод набирає все більшу популярність.

Розробка моделі системи – це нетривіальна задача, для виконання якої потрібно глибокі знання в галузі моделювання, а також креативне мислення, тому що не завжди всі процеси чи елементи реального життя можна відобразити на моделі.

Одним із найпотужніших математичних апаратів імітаційного моделювання вважають Петрі-об'єктне моделювання. На сьогоднішній день було розроблено багато Петрі-імітаторів, кожен зі своїми особливостями, які використовуються в різних сферах моделювання.

Хоча багато із сучасних програм для імітаційного моделювання дають розгорнутий звіт по результатам моделювання – для його розуміння потрібно бути спеціалістом в галузі моделювання систем. Але більшість людей не мають таких знань.

У зв'язку з цим використовуються анімовані 3D-об'єкти. Так набагато легше представити змодельовану систему людям, які не розуміються в галузі моделювання чи то замовникам розробки таких систем, чи то студентам, які тільки навчаються проводити імітаційне моделювання.

Саме тому призначенням цієї дипломної робити стало розробити такий програмний продукт, який би могли використовувати не тільки спеціалісти в галузі моделювання, але й люди, які тільки почали навчатися моделюванню.

Практичне значення одержаних результатів. Розроблено алгоритм Петрі-об'єктного моделювання систем, який працює з 3D-анімацією в режимі реального часу.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

З розвитком людської діяльності зростає складність систем, які ми розробляємо для полегшення якості нашого життя, все збільшується. Також збільшується кількість ресурсів, які ми витрачаєм на розробку таких систем. Тому потрібно думати не тільки про зручність використання різних систем людьми, але й ціну реалізації, експлуатації, ремонту і т.д. Щоб проаналізувати різні параметри майбутньої системи використовують моделювання.

Моделлю називається представлення об’єкта, системи чи поняття в деякій абстрактній формі, що є зручною для наукового дослідження [1].

Завдання моделювання полягає в знаходженні значень параметрів системи (вихідних даних) при відомих вхідних значеннях та відомій моделі.

Існує багато методів, за допомогою яких проводять моделювання. Але найчастіше виділяють: аналітичне моделювання, математичне моделювання, імітаційне моделювання.

Моделювання є аналітичним – якщо воно представлене певною залежністю вихідних змінних від вхідних та представляється набором відомих аналітичних функцій (функції, що розкладаються в ряд Тейлора).

Моделювання є математичним – якщо його неможливо представити у вигляді залежності вхідних від вихідних змінних, хоча його функціонування піддається опису аналітичними функціями. Розв’язок таких задач може бути знайдений в процесі виконання деякої кількості ітерацій, або за відомим алгоритмом, але не може бути представлений в аналітичній формі.

Моделювання є імітаційним – якщо модельовану систему не можна представити аналітичними функціями, але її можна подати певним алгоритмом імітації.

Імітація – це процес відтворення програмним чином функціонування системи протягом деякого часу. Для того, щоб отримати властивості модельованої системи виконують багато прогонів імітаційної моделі.

Сам процес моделювання можна описати такими етапами:

– визначення мети та задачі моделювання;
– формалізація моделі: складання концептуальної моделі системи, вибір теоретичної бази, визначення параметрів системи;
– створення моделі та оцінка її адекватності;
– дослідження моделі: планування та проведення експериментів, обробка результатів моделювання;
– оцінка точності результатів моделювання;
– формування висновку.

Імітаційне моделювання застосовується в областях:

– телекомунікаційні мережі;
– бізнес процеси;
– воєнні дії;
– логістика;
– транспортний рух;
– сервісні центри;
– екосистеми;
– ринкова економіка;
– динаміка популяції;
– управління проектами.

Адекватність моделі – збіг властивостей, функцій, параметрів, характеристик моделі і відповідних властивостей модельованого об'єкта. Адекватністю називається збіг моделі з модельованою системою у відношенні мети моделювання [2].

1.1.1 Опис процесу діяльності

Розглянемо процес діяльності системи. Для його відображення побудована схема структурна діяльності, що наведена в графічному матеріалі.

Спочатку для роботи з інструментарієм його потрібно імпортувати в ігровий рушій Unity3D. Після імпорту можна відкрити графічний редактор моделі, в якому можна створити та редагувати модель, а також зберегти її в файл.

Після того, як модель розроблена, програміст створює 3D сцену моделювання, розставляючи 3D-об’єкти, прив’язує до них 3D-анімацію, а також поведінку при наставанні подій, які буде генерувати модель.

Після цього можна запускати моделювання. Воно буде відбуватися в реальному часі, тому можливо в будь-який момент зупинити моделювання і отримати звіт за змодельований час.

1.1.2 Опис функціональної моделі

Актором системи є: Програміст моделі.

Дії або варіанти використання, що виконують в системі актори, наведені в таблиці 1.1, в якій описані актори, варіанти використання та їх описи дій.

Таблиця 1.1 – Типи залежностей між варіантами використання

Актор	*Варіант використа-ння*	*Опис дії варіанта використання*
Програмі ст моделі	Створення моделі	Програміст створює нову модель для її подальшої імітації
	Редагуванн я моделі	Програміст може редагувати створену раніше модель

Продовження таблиці 1.1

Актор	*Варіант використа-ння*	*Опис дії варіанта використання*
	Збереження	Програміст може зберегти модель у файл
	Прив’язка подій моделі до об’єктів 3D сцени	Програміст використовує наявний інструментарій для того, щоб прив’язати поведінку модельованих об’єктів до подій, які генерує модель
	Запуск моделювання	Програміст має запустити моделювання для отримання звіту
	Отримання звіту з моделю-вання	При завершенні моделювання програміст може отримати звіт по його результатам

Відповідно визначених варіантів використання побудована схема структурна варіантів використання, що наведена в графічному матеріалі

1.2 Огляд наявних аналогів

Імітаційне моделювання набуло великої популярності за останні п’ятдесят років. У ході пошуку таких систем було знайдено десяток програмних продуктів, які виконують імітаційне моделювання. Не всі вони використовують математичний апарат Петрі-об’єктних мереж, також не всі програми показують результати моделювання у вигляді 3D-анімації. На базі ігрового рушію Unity3D існують розробки для створення симуляцій, але вони напрямлені на вузькі спеціальності, та не використовують мережі Петрі як математичну модель.

Серед знайдених програмних засобів розглянуті наступні програмні продукти:

– GPSS;
– AnyLogic;
– PlantSim;
– Arena;
– Simio Simulation Software;
– ExtendSim;
– CPN Tools;
– SimViz.

GPSS – це мова програмування, яка призначена для імітаційного моделювання різних систем, особливо популярна для моделювання систем масового обслуговування. Коротко описуючи – це мова загального призначення, яка допомагає розв’язувати задачі по крокам на дискретних проміжках часу.

Системи в GPSS моделюються за допомогою блоків, по яким від одного блоку до іншого переміщаються транзакти, які позначають певний ресурс. Ця система є доволі зручною для, скажімо, моделювання виробничого процесу. Ця мова є не такою гнучкою, якщо порівнювати її з

багатьма іншими, сучаснішими мовами програмування, але завдяки своїй простоті вона досі залишається дуже популярною.

AnyLogic [3] – програмний засіб для імітаційного моделювання різного виду бізнес-процесів, який розроблений російським розробником – The AnyLogic Company. Цей інструментарій підтримує сучасніший графічний інтерфейс, порівнюючи його з GPSS, та використовує мову програмування Java для моделювання.

AnyLogic має також і графічний засіб моделювання, та дозволяє користувачу розширяти створені раніше моделі використовуючи мову програмування Java. Інтегрований компілятор Java в AnyLogic надає багато можливостей для створення моделей, та створення Java застосунків, які відкриватимуться у будь-якому сучасному браузері. Ці застосунки можна буде розмістити на веб-сторінках. Але для того, щоб ці застосунки можна було запускати без встановленого AnyLogic, потрібно буде придбати розширену версію програми.

Plant Simulation [5] - це програмне забезпечення, що розроблено розробником Siemens PLM Software для аналізу, візуалізації, моделювання та оптимізації різного роду виробничих систем та процесів, обігу матеріалів і логістичних операцій. Tecnomatix Plant Simulation використовують для оптимізації обігу матеріалів, утилізації ресурсів, а також роботи матеріально-технічного забезпечення для всіх структурних рівнів заводів від глобальних виробничих потужностей і аж до окремих конвеєрів.

У межах Plant Design und Optimization Solution група програмного забезпечення, до яких належить Plant Simulation , стоїть разом з такими продуктами як Digital Factory та Digital Manufacturing – які є частиною Product Lifecycle Management Software (PLM). Додаток дозволяє порівнювати складні виробничі альтернативи, в тому числі іманентну логіку процесів, за допомогою комп'ютерного моделювання. Plant Simulation використовується окремими планувальниками виробництва, а також міжнаціональними

підприємствами, в першу чергу для створення стратегічного макету плану, управління логікою і великими за розміром та складними виробничими інвестиціями. Це один з основних продуктів, які домінують на ринку цього простору.

Arena [4] – програма симуляції дискретно-подійних систем, розроблена Systems Modeling і придбана Rockwell Automation в 2000. Вона використовує SIMAN процесор і мову симуляції. Станом на 2016 вийшло уже 15 версій, які пропонують значні вдосконалення і оптимізації, анімацію, а також включають 64-бітні операції для процесів моделювання в Big Data.

У Arena, користувач створює модель експерименту, розміщуючи модулі (блоки різної форми), які представляють процеси або логіку. З'єднувальні лінії використовуються для об'єднання цих модулів разом і для визначення потоку об'єктів. Хоча модулі мають специфічні дії щодо об'єктів, потоку і синхронізації, точне представлення кожного модуля і об'єкта щодо об'єктів реального життя підлягає програмісту.

Статистичні дані, такі як час циклу та рівні WIP (робота в процесі), можуть бути записані і зроблені як звіти. Арена може бути інтегрована з технологіями Microsoft. Вона включає в себе Visual Basic для додатків, так що моделі можуть бути додатково автоматизовані, якщо потрібні конкретні алгоритми. Він також підтримує імпорт графічних схем Microsoft Visio, а також читання або надсилання даних до електронних таблиць Excel і баз даних Access. Також підтримуються елементи керування ActiveX.

Simio Simulation Software [6] забезпечує справжнє об'єктне середовище 3D-моделювання, яке дозволяє побудувати 3D-модель за один крок, з 2D-вигляду зверху вниз, перед тим, як миттєво перейти до 3D-вигляду системи. Потім 3D-об'єкти з бібліотеки об'єктів просто перетягуються і поміщаються у вигляд об'єкта моделі.

Усі продукти побудови моделей Simio безпосередньо інтегруються зі службою Google Warehouse, щоб забезпечити швидкий доступ до великої

бібліотеки вільно доступних 3D-символів, які можуть швидко і легко додати реалістичність моделям. Це призвело до успішного застосування програмного забезпечення Simio для моделювання в багатьох галузях промисловості, у тому числі для виробництва, у сфері охорони здоров'я, аерокосмічної та оборонної, гірничої промисловості, промислового будівництва.

Simio імітує системи та процеси з використанням своїх запатентованих інтелектуальних об'єктів (Патент № 8,156,468 C1: Система і спосіб створення інтелектуальних об'єктів моделювання з використанням описів графічного процесу). Вони можуть бути з попередньо побудованих бібліотек об'єктів або на замовлення, щоб представляти, наприклад, машини, роботи, літаки, клієнти, кораблі, лікарі тощо. Об'єкти об'єднують логіку і анімацію, відображаючи зміни в стані і виробляють необхідну модель.

ExtendSim [7] – сімейство програм, які були розроблені компанією Imagine That. Вони дозволяють проводити імітаційне моделювання динамічних систем. Системи будуються у вигляді блок-схем. Користувач може використовувати як і існуючими блоками, створеними розробниками, так і розширювати заготовлений функціонал власними блоками. Блок в собі містить код, властивості, анімацію та інтерфейс користувача. Модель імітації розробляється шляхом з’єднання блоків в мережу, та введення властивостей блоків через контекстні меню користувацького інтерфейсу. ExtendSim пропонує такі стандартні блоки моделювання як генератор, черга, набір ресурсів, діяльність та вихід. Сутності створюються в блоках генераторах та переміщаються по зв’язкам. Також, як і Arena, ExtendSim містить засоби анімації поведінки системи та статистичного аналізу даних.

CPN Tools [8] – інструментарій для редагування, симуляції та аналізу кольорових мереж Петрі. Інструмент має інкрементальну перевірку синтаксису та генерацію програмного коду, що відбувається під час створення мережі. Інструменти CPN складаються з двох основних

компонентів: графічного редактора та компонента симулятора. Графічний редактор написаний академічною мовою, бета-версією, а бекенд тренажера написаний у стандартному варіанті MC SML / NJ.

Програма підтримує анімацію потоку маркерів, експорт та імпорт моделей, аналіз результатів, властивостей там продуктивності моделі.

SimViz [9] – набір шаблонів, який дозволяє швидко розробити середовище симуляції для того, щоб розробники концентрувалися не на деталях реалізації алгоритмів імітації, а на масштабуванні і вдосконаленні програми у своїй предметній галузі. Інструментарій призначений для імітації процесів, які проходять у містах: транспорт, пішоходи, машини з сенсорами, які поводяться як справжні фізичні сенсори. Програмний засіб працює використовуючи алгоритми машинного навчання.

Хоча ці програми і пройшли перевірку часом, в багатьох із них є недоліків, а саме:

- власна скриптова мова програмування моделей. В межах сучасної IT індустрії, де кількість технологій якими повинен оперувати програмний інеженер велика як ніколи, мало кому захочеться вчити нову мову програмування. В моєму інструментарії буде використовуватись C# - одна із провідних мов програмування сучасності;
- низький графічний потенціал. Багато програм не розвивали графічні можливості і двигун для рендера з моменту їх створення в двохтисячних. Мій інструментарій буде інтегруватися в ігровий двигун Unity3D – графічні можливості якого вражають.

Також серед переваг вибору саме Unity3D як платформу розробки можна зазначити інтегрований фізичний двигун Physix, потужну рендер машину, систему анімації Mecanim, вбудовану систему пошуку шляхів переміщення NavMesh, а також наяність користувацьких розширень, які з легкістю можуть бути інтегровані в проект. Наявність системи префабів

(Prefab), об’єктів сцени, які серіалізуються як спеціальні файли, які можна перевикористовувати, а також за системою Prefab Workflow створювати так звані Prefab Variants створювати різні версії даного об’єкта.

Даний дипломний проект використовує технологію Петрі-об’єктного моделювання. Це потужний математичний апарат, точність і швидкість якого перевірена роками. Програми, які були наведені як приклад використовують інші математичні моделі власної розробки.

1.3 Постановка задачі

1.3.1 Призначення розробки

У звя’зку з тим що імітаційне моделювання з кожним роком все розвивається, а складність систем, які потрібно моделювати лише збільшується, тому даний інструментарій створюється для полегшення створення Петрі-об’єктних моделей і їх імітації.

1.3.2 Цілі та задачі розробки

Цілями розробки системи є:

- прискорення процесу моделювання систем;
- створення інструментарію, який можуть освоїти програмісти не знайомі з теоретичною базою систем імітації.

Для досягнення поставлених цілей необхідно реалізувати наступні задачі.

Задачами розробки є:

- створити редактор моделей, для швидкого моделювання;
- створити набір інструментів, які будуть необхідні для моделювання основних бізнес-процесів, та можливість їх розширювати за необхідності;

- збирання статистики за час моделювання системи;
- інтеграція математичного апарату Петрі-об’єктних мереж, як основу для імітаційного моделювання.

Із реалізацією задач було створено зручний інструментарій для створення імітаційного моделювання дискретних систем.

Висновок до розділу

У даному розділі здійснений детальний аналіз предметної області, визначені актори системи. Оглянуто та проаналізовано існуючі аналоги. Визначено цілі та задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідні дані представлені у таблиці 2.1.

Таблиця 2.1 – Вхідні дані

Данні	*Опис*
Модель системи	Петрі-об’єктна модель, яка зберігається у вигляді xml документа
3D моделі та анімації	3D моделі у форматі .fbx, або .obj

2.2 Вихідні дані

Вихідні дані представлені в таблиці 2.2.

Таблиця 2.2 – Вихідні дані

Дані	*Опис*
Звіт по моделюванню.	Звіт, який буде мати статистику за змодельований час

2.3 Структура масивів інформації

Для серіалізації мережі Петрі буде використано xml документ. Макет розмітки представлено нижче:

– модель мережі Петрі задається елементом PetriModel, який має в собі масиви позицій – Places, переходів – Transitions, вхідних зв’язків – InConnections, вихідних зв’язків – OutConnections, код має наступний вигляд:

```
<xs:element name="PetriModel">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Places"/>
    <xs:element ref="Transitions"/>
    <xs:element ref=="Objects"/>
    <xs:element ref="InConnections"/>
    <xs:element ref="OutConnections"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

– масив всіх позицій задається елементом Places, який має в собі елементи – Place, які мають атрибути:

1) id – що визначає унікальний ідентифікатор серед елементів;
2) name – унікальне ім’я, по якому можна буде звертатися до позиції;
3) tokens – кількість токенів, які наявні в даній позиції.

Код має наступний вигляд:

```
<xs:element name="Places">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Place"/>
   </xs:sequence>
   <xs:attribute name="type"  type="xs:NCName"/>
  </xs:complexType>
```

```
</xs:element>
<xs:element name="Place">
 <xs:complexType>
  <xs:attribute name="id"  type="xs:integer"/>
  <xs:attribute name="name"  type="xs:NCName"/>
  <xs:attribute name="tokens"  type="xs:integer"/>
 </xs:complexType>
</xs:element>
```

– масив всіх переходів задається елементом Transitions, який має в собі елементи – Transition, які мають атрибути:

1) id – що визначає унікальний ідентифікатор серед елементів;
2) name – унікальне ім’я, по якому можна буде звертатися до переходу;
3) time – час, за який відбувається перехід;
4) halfrange – відхилення від середнього часу за рівномірним розподілом;
5) probability – вірогідність наставання події;
6) priority – пріоритет вибору події при конфліктних переходах.

Код має наступний вигляд:

```
<xs:element name="Transitions">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Transition"/>
   </xs:sequence>
   <xs:attribute name="type"  type="xs:NCName"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="Transition">
  <xs:complexType>
```

```
<xs:attribute name="id"  type="xs:integer"/>
<xs:attribute name="name"  type="xs:NCName"/>
<xs:attribute name="time"  type="xs:decimal"/>
<xs:attribute name="halfrange"  type="xs:decimal"/>
<xs:attribute name="probability"  type="xs:decimal"/>
<xs:attribute name="priority"  type="xs: integer"/>
</xs:complexType>
</xs:element>
```

– масив всіх Петрі-об’єктів задається елементом Objects, який має в собі елементи – Object, які мають атрибути:

1) id – що визначає унікальний ідентифікатор серед елементів;
2) PetriGraph – шлях до файла, в якому зберігається інкапсульована мережа;
3) Label – назва Петрі-об’єкта;
4) InputPlaceLabel – вхідна спільна позиція;
5) OutputPlaceLabel – вихідни спільна позиція;

Код має наступний вигляд:

```
<xs:element name="Objects">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Object"/>
    </xs:sequence>
    <xs:attribute name="type"  type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="Object">
  <xs:complexType>
    <xs:attribute name="id"  type="xs:integer"/>
    <xs:attribute name="name"  type="xs:NCName"/>
```

```
<xs:attribute name="gpath"  type=" xs:NCName "/>
<xs:attribute name="inputp"  type=" xs:NCName "/>
<xs:attribute name=" outputp "  type=" xs:NCName "/>
</xs:complexType>
</xs:element>
```

– масив всіх вхідних зв’язків задається елементом InConnections, який має в собі елементи – InConnection, які мають атрибути:

1) id – що визначає унікальний ідентифікатор серед елементів.
2) weight – вага зв’язку, тобто кількість токенів, які заберуться в позиції при переході.
3) place – ім’я позиції, з якої виходить зв’язок.
4) transition – ім’я переходу, в який входить зв’язок.

Код має наступний вигляд:

```
<xs:element name="InConnections">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="InConnection"/>
   </xs:sequence>
   <xs:attribute name="type"  type="xs:NCName"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="InConnection">
  <xs:complexType>
   <xs:attribute name="id"  type="xs:integer"/>
   <xs:attribute name="place"  type="xs:NCName"/>
   <xs:attribute name="transition"  type="xs:NCName"/>
   <xs:attribute name="weight"  type="xs:integer"/>
  </xs:complexType>
 </xs:element>
```

– масив всіх вихідних зв’язків задається елементом OutConnections, який має в собі елементи – OutConnection, які мають атрибути:

1) id – що визначає унікальний ідентифікатор серед елементів.
2) weight – вага зв’язку, тобто кількість токенів, які добавляться в позицію при переході.
3) place – ім’я позиції, в яку входить зв’язок.
4) transition – ім’я переходу, з якого виходить зв’язок.

Код має наступний вигляд:

```
<xs:element name="OutConnections">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="OutConnection"/>
    </xs:sequence>
    <xs:attribute name="type"  type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="OutConnection">
  <xs:complexType>
    <xs:attribute name="id" type="xs:integer"/>
    <xs:attribute name="place" type="xs:NCName"/>
    <xs:attribute name="transition" type="xs:NCName"/>
    <xs:attribute name="weight"  type="xs:integer"/>
  </xs:complexType>
</xs:element>
```

Схема XML документу наведена в частині графічного матеріалу.

Для відображення вихідного звіту з моделювання було використано текстовий файл з такою структурою (рисунок 2.1):

```
==========================
ModelingTme: [Час моделювання]
==========================
| Place | M | AM | MM | SM |
==========================
……...
==========================
| Trans | U | UC | AT | ST |
==========================
……...
```

Рисунок 2.1 – Схема структури текстового файлу звіту

Пояснення до позначок:

– позиції:
  1) Place – назва позиції;
  2) M – поточне маркування;
  3) AM – середнє маркування;
  4) MM – максимальне маркування;
  5) SM – загальна кількість маркувань;
– переходи:
  1) Trans – назва переходу;
  2) U – кількість переходів;
  3) UC – коефіцієнт навантаженості переходу;
  4) AT – середній час переходу;
  5) ST – загальний час переходів.

Висновок до розділу

У даному розділі було розглянуто вхідні та вихідні дані, наведені в таблицях. Розглянута структура xml документа, в якому буде зберігатися модель, було описано всі її елементи та атрибути та наведено їх значення.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Дискретно-подійне моделювання [10] – різновид імітаційного моделювання. В дискретно-подійному моделюванні функціонування системи представляється як хронологічна дискретна послідовність подій. Подія відбувається у визначений момент часу та призводить до зміни стану системи. Між цими подіями система перебуває у незмінному стані.

Система яка проводить дискретно-подійне моделювання містить такі компоненти:

Стан системи – це набір параметрів, які визначають саму систему. Зміна стану відбувається в момент виникнення нової події.

Годинник – використовується для синхронізації всіх змін, які виникають в моделі.

Список подій – система підтримує список майбутніх подій, який називається списком очікуваних подій. Зазвичай використовується один список подій, оскільки при використанні декількох виникають складності в синхронізації. Список подій подається у вигляді пріоритетної черги, яка відсортована за часом виникнення події.

Генератори випадкових чисел – використовується для систем, події яких мають випадковий характер.

Статистика – дані, які збираються в модельованих системах, наприклад:

- середня доступність ресурсів;
- середня довжина черги;
- середній час очікування в черзі;
- коефіцієнт використання пристрою;
- кількість нульових входжень в чергу.

Умови завершення – проходження заданого часу, виконання заданої події, ручна зупинка, якщо модель виконується в режимі реального часу.

Існує багато різних методів імітаційного моделювання дискретно-подійних систем. Але для даної дипломної роботи був обраний алгоритм імітації саме на мережах Петрі.

Мережі Петрі – математичний апарат для моделювання динамічних дискретних систем. Вперше описані Карлом Петрі у 1962 році.

Мережі Петрі використовуються для моделювання систем, в яких складові елементи можуть виконуватися незалежно один від одного. Після проводення аналізу мережі Петрі можна отримати інформацію про стан системи в конкретний момент часу.

Зв'язок подій в системах асинхронного виду задається множиною відношень "умови-події". У мережах Петрі умовами є позиції, а подіями є переходи. Тому граф мережі Петрі є дводольним орієнтованим мультиграфом. Дуги можуть лише сполучати переходи і позиції в двох напрямках, але не можуть сполучати однотипні вершини. В мережах Петрі допускається кратність дуг, яка позначає велику кількість зв'язків.

В мережах Петрі кількісну характеристику позицію прийнято позначати числом маркерів.

Для того, що перехід спрацював потрібно щоб виконувалися такі правила – перехід спрацьовує, коли для всіх позицій, які в нього входять виконується умова $N_i \geq K_i$, де N_i – число маркерів в і-й вхідній позиції, K_i – вага дуги, що зв’язує і-ту позицію та перехід; після спрацювання – число маркерів в і-й вхідній позиції зменшується на K_i , а в j-й вихідній позиції збільшується на M_j , де M_j – вага дуги, що зв’язує перехід з j-ю позицією.

На рисунку 3.1 показаний приклад спрацювання переходу з маркуванням (2,2,3,1). Після того, як умови спрацьовування переходу виконаються маркування стає іншим: (1,0,1,4).

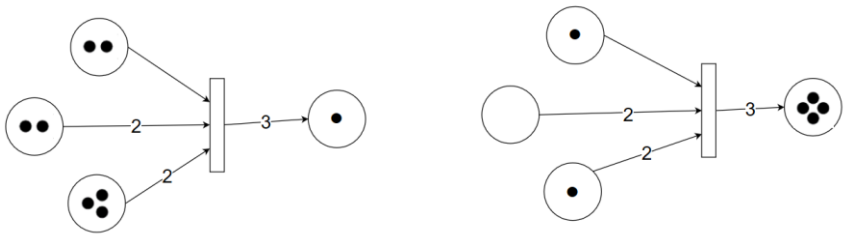


Рисунок 3.1 – Фрагмент мережі Петрі до а) і після б) спрацьовування переходу

Петрі-об’єктний підхід – це парадигма програмування моделей, суть якої полягає в представленні структури моделі як множину об’єктів. Функціонування самих об’єктів має бути представлене у вигляді стохастичної мережі Петрі. Цей підхід дозволить швидко та ефективно конструювати складні та комплексні дискретно-подійні системи. Таким чином всі об’єкти працюють за однаковими правилами, а об’єктно-орієнтований підхід дозволить клонувати об’єкти у будь-якій кількості.

Зв’язки між Петрі-об’єктами реалізуються за допомогою спільних позицій (позиція належить одночасно двом об’єктам)

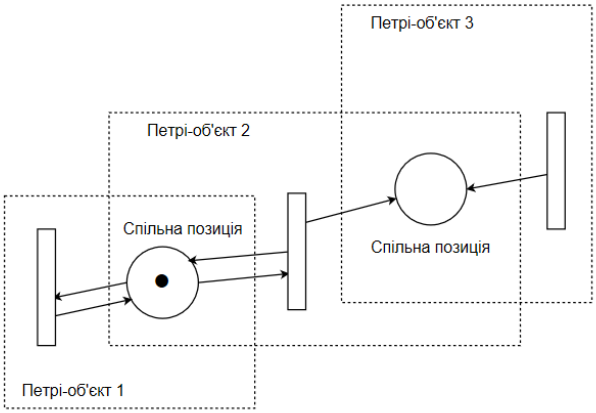


Рисунок 3.2 – Зв’язок двох Петрі-об’єктів за допомогою спільної позиції

3.2 Математична постановка задачі

Динаміка функціонування Петрі-об’єкта повністю задається його мережею Петрі. Тому математичний опис динаміки функціонування Петрі-об’єкта співпадає з математичним описом алгоритму імітації стохастичної мережі Петрі з багатоканальними і конфліктними переходами та з інформаційними зв’язками.

Стан тимчасової мережі Петрі в кожен момент часу описується станом її позицій *M(t)* і станом її переходів *E(t)*: *S(t) = (M(t), E(t))*.

Математичний опис функціонування Петрі об’єкта має вигляд [22]:

$$\begin{cases} t_n = \min_{T \in \mathbf{T}_N} \tau_T(t_{n-1}), t_n \geq t_{n-1}, \\ \mathbf{S}_N(t_1) = (D^-)^m\left(\mathbf{S}_N(t_0)\right), \\ \mathbf{S}_N(t_n) = (D^+)^m\left(D^+\left(\mathbf{S}_N(t_{n-1})\right)\right), \\ n = 2,3\ldots \end{cases} \quad (3.1)$$

де *N* – Петрі відповідного Петрі-об’єкта, $S_N(t_n)$ – стан Петрі-об’єкта, D^- – перетворення стану часової мережі Петрі, зв’язане з входом маркерів в переходи, D^+ – перетворення стану тимчасової мережі Петрі, зв’язане з входом маркерів в переходи.

Петрі-об’єктна модель описується стохастичною часовою мережею Петрі, яка є об’єднанням мереж Петрі-об’єктів, з яких вона складається – *Model = {O₁, O₂, … Oₙ}*.

Рівняння, що задають Петрі-об’єктну модель мають вигляд:

$$t_n = \min_N \tau_N, t_n \geq t_{n-1}, \quad (3.2)$$

$$\mathbf{S}(t_n) = \begin{pmatrix} (D^-)^m\left(D^+\left(\tilde{\mathbf{S}}_1(t_{n-1})\right)\right) \\ \cdots \\ (D^-)^m\left(D^+\left(\tilde{\mathbf{S}}_N(t_{n-1})\right)\right) \\ \cdots \\ (D^-)^m\left(D^+\left(\tilde{\mathbf{S}}_L(t_{n-1})\right)\right) \end{pmatrix}, \forall \tilde{\mathbf{S}}_N(t_n): \vee_{T \in \mathbf{T}_N} Z(T, t_n) = 0, \quad (3.3)$$

де *m* - кількість перетворень D^-, які здійснюються перед досягненням стану Петрі-об'єкта, при якому жоден із його переходів не запускається [24].

З алгоритмічної точки зору розбиття на Петрі-об’єкти дозволяє значно зменшити кількість елементарних операцій, які необхідно здійснити для перетворення мережі Петрі. Перетворення $D^+\left(\tilde{\mathbf{S}}_N(t_{n-1})\right)$ потребує фактичного проведення тільки коли $\tau_{\mathrm{N}} = t_n$, так як в іншому випадку значення предикатів $X(T, t_n) = 0 \quad \forall\, T \in \mathbf{T}_N$ і виконання перетворення $D^+\left(\tilde{\mathbf{S}}_N(t_{n-1})\right)$ не приведе до зміни стану Петрі-об’єкта. Перетворення $(D^-)^m$, що виконуються для кожного Петрі-об’єкта, потребують фактичного виконання m_J кратного входу маркерів в переходи до досягнення стану, при якому жоден з переходів Петрі-об’єкта не запускається. Так як $m_J < m$ для більшості Петрі-об’єктів і для ще більшої кількості Петрі-об’єктів $m_J = 0$, то фактична кількість входів маркерів в переходи окремих Петрі-об’єктів значно менше кількості m входів маркерів в переходи Петрі-моделі.

3.3 Алгоритм імітації Петрі-об’єктної моделі

Алгоритм імітації Петрі-об’єктної моделі, який би працював режимі реального часу:

Крок 1. Створити список Петрі-об’єктів, чергу майбутніх подій.

Крок 2. Записати початкове маркування.

Крок 3. Знайти час найближчої події:

Крок 4. Просунути час моделі на маленькій проміжок часу (дельту).

Крок 4.1. Якщо черга майбутніх подій не пуста і час моделі перевищує час настання цих подій – здійснити вхід до відповідних маркерів:

Крок 4.2. Якщо час моделі перевищує час настання найближчої події:

Крок 4.1.1. Вирішити конфлікт, якщо подій декілька.

Крок 4.1.2. Здійснити вихід з маркерів відповідного Петрі-об’єкта.

Крок 4.1.3. Записати подію входу маркерів в список майбутніх подій.

Крок 5. Відобразити результат моделювання.

3.4 Опис методів розв'язання

В комп'ютерних динамічних симуляціях в таких випадках використовують імітацію по дельтам (змінам часу). В комп'ютерній графіці часова дельта – це час, який пройшов між опрацюванням поточного кадру і попереднього. Її часто використовують в комп'ютерних симуляціях фізики, динаміки руху об'єктів, симуляції рідини, частинок і інших випадках. Тому в даному проекті використовується цей підхід в імітації Петрі-об'єктної моделі.

Для цього замість моделювання системи на всьому проміжку часу - вона моделюється по дискретних проміжках. Для цього потрібно знати одну подію наперед і кожного разу, коли ми моделюємо систему на наступну дельту часу, перевіряти, чи час моделювання не перевищив час наставання деякої події. Якщо так – то просуваємо системний час на момент наставання наступної події і знаходим час наставання наступної події. А якщо вийшло так, що реальний час перестрибнув через декілька системних подій, то потрібно їх змоделювати, і вже тоді знайти момент наставання наступної події.

Продемонструємо роботу алгоритму на схемі (рисунок 3.3). Чорні лінії – це дискретні проміжки часу. Стрілочки – це час наставання події деякого номеру. Незаповнена лінія – це час, коли було активовано певну подію.

В момент часу 0 – ми прораховуєм час наставання наступної події – це подія 1 і запам'ятовуєм цей час. Коли настав час t1, який перевищив час наставання події, то ми активуємо цю подію і прораховуєм наступну подію, що є подія номер 2. Таким чином ми просуваємся в реальному часі синхронізуючись при цьому з модельним часом.

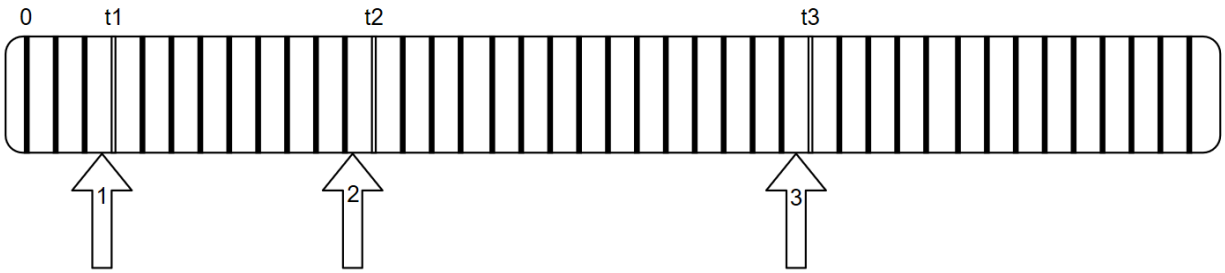


Рисунок 3.3 – Схема роботи алгоритму

Висновок до розділу

У даному розділі дипломного проекту було детально розібрано основні поняття імітаційного моделювання дискретно-подійних систем, основні положення, терміни та принцип роботи імітаційного Петрі-об’єктного моделювання. Було наведено математичний опис Петрі-об’єктної моделі, та наведені переваги використання саме такого підходу. Було наведено алгоритм імітації та його модифікація для симуляції систем в режимі реального часу.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Програмне забезпечення, що використовується при розробці:

- операційна система: **Windows 10** [11] – операційна система від компанії Microsoft для ПК, смартфонів, планшетів та ноутбуків;
- ігровий рушій: **Unity3D** [12] – потужний сучасний рушій для симуляції графічних застосунків в реальному часі.
- середовище розробки: **Visual Studio** [13] – серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом.
- мова написання коду програми: **C#** [14] - об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET**;**
- мова розмітки, для серіалізації моделей мереж, **XML** [21] - стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет;
- **Enterprise Architect** [15] – це середовище для візуального проектування;
- **Draw.io** [16] – сайт для створення UML діаграм.

Серед альтернативних варіантів вибору засобів розробки можна відмітити Unreal Engine 4, Godot, CryEngine, написання власного рушію на графічній бібліотеці OpenGL / DirectX / Vulkan.

Написання власної рендер-машини було зразу відкинуто, адже для цього потрібно потратити дуже багато часу, і для того щоб спроектувати всі компоненти потрібна команда спеціалістів у сфері комп’ютерної графіки.

CryEngine [17] – це ігровий рушій створений німецьким розробником – Crytek, який вперше було застосовано у комп'ютерній ігрі Far Cry. Він був розроблений як техно-демонстрація GeForce 3 від Nvidia. Рушій підтримує скриптінг на C++, Lua та C#.

Го́до (Godot) [18] – open source-ний кросплатформовий 2D та 3D ігровий рушій під ліцензією MIT, що розробляється компанією Godot Engine Community. До релізу у вигляді відкритого програмного забезпечення рушій використовували всередині деяких компаній Латинської Америки. Середовище розробки працює на Windows, Unix, iOS та може збирати проекти на персональні комп'ютери, ігрові приставки, мобільні та веб платформи. Рушій підтримує скриптинг або повністю на C++, або застосовуючи власну скриптову мову програмування GDScript.

Unreal Engine [19] – ігровий рушій, розроблюваний і підтримуваний компанією Epic Games. Підтримує скриптінг на C++, власній скриптовій мові Unreal Script та візуальна мова програмування Blue Prints, рушій дозволяє створювати графічні застосування для багатьох платформ: на ПК, ігрових консолей та смартфонів.

Попри всі переваги вище описаних редакторів було вирішено інтегрувати інструментарій в Unity3D. Переваги цього рушія:

Першою перевагою я вважаю – сама мова C# та платформа .NET. Дана мова є високорівневою і дозволяє програмісту легко увійти в розробку ігор або інших симуляцій в реальному часі. А використання .NET є важливим моментом, тому що в ньому є багато елементів і прийомів, які вже реалізовані, і програмісту потрібно тільки скористатися ними.

Друга перевага – кросплатформенність, тобто один і той же код, написаний на рушії Unity, з мінімальними змінами може бути перенесений на різні платформи. Це великий плюс, який скорочує час розробки програми в декілька раз.

Третя перевага – це хороше Community. Це означає, що у різних функцій рушія є чіткий опис з прикладами на сайті розробника, звернутись до якого можна в будь-який момент. Якщо щось все ж лишилось незрозумілим, служба підтримки обов'язково відповість на ваше питання.

І четвертою перевагою є Asset Store, де знаходиться велика кількість різних плагінів і ресурсів для створення продукту. Зрозуміло, що деякі з них безплатні, а деякі платні, але вони всі зібрані в одному місці з зручним пошуком і можливістю завантажити, інтегрувати і отримати зразу робочий функціонал. Планується даний дипломний проект по завершенню розробки опублікувати безплатно саме на цьому маркеті.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Так як даний продукт являє собою інструментарій для створення 3D анімації Петрі-об'єктного моделювання, який інтегровано в рушій Unity3D, то потрібно мати встановлену останню версію рушію Unity 2018.3 і вище, а також інтегроване середовище розробки Visual Studio, Mono, або інше на вибір. Для нормальної роботи з Unity потрібно мати комп'ютер як мінімум такої конфігурації:

- процесор з набором інструкцій SSE2;
- графічна карта з підтримкою DirectX 9 (shader model 3.0);
- операційна система: Windows 7 SP1+, 8, 10, 64-bit; Mac OS X 10.9+.

4.3 Архітектура програмного забезпечення

Дана система складається з трьох модулів, які побудовані згідно принципів чистої архітектури. Першим модулем є редактор Петрі-об'єктної моделі. Він відповідає за створення, редагування та відображення на екрані мережі Петрі у вигляді графового редактора, основними компонентами якого є вузли та дуги. Основним класом цього модуля є *PetriModelEditor*, який

наслідується від абстрактного класу *EditorWindow*. Це дозволяє інтегрувати редактор у середовище ігрового рушія і він буде сприйматися як всі інші вікна.

Клас *PetriNode* – абстрактний клас, який відповідає за вузли на графі.Його реалізаціями є *ObjectNode* – Петрі-об'єкт, *PetriPlaceNode* – позиція, *PetriTransitionNode* – перехід.

Клас *PetriLink* – абстрактний клас, який відповідає за дуги на графі, його реалізаціями є *InLink* – дуга, що входить в перехід та *OutLink* – дуга, що виходить з нього.

Всі вузли та дуги містять відповідні їм параметри, які відповідають параметрам компонентів мережі Петрі, а також параметри відображення на екрані (позиція, колір) та параметри, які потрібні для серіалізації (*id*).

Даний модуль створює об'єкти класу *PetriNet*, який є основним класом другого модуля і відповідає за математичну модель.

PetriNet агрегує в собі класи *PetriObject, PetriPlace, PetriTransition, InArc, OutArc*. Їх призначення зрозуміле із назви і в основному вони використовуються для зберігання даних про відповідні їм компоненти мережі Петрі. Винятком є *PetriTransition*, якому крім того є два методи *TransitFrom()* та *TransitTo()*, значення яких буде описано в специфікації методів.

Третій модуль використовується для прив'язки 3D-анімації до подій моделі, які буде генерувати об'єкт класу *PetriNet.* Основним класом даного модуля є *ModelPlayback*, який агрегує в собі один об'єкт класу *PetriNet* та викликає його метод *Simulate(). ModelPlayback* наслідується від класу *MonoBehaviour*, який представляє основний інтерфейс взаємодії з «ігровим циклом» [23].

Також в цьому модулі знаходяться класи, які відповідають за 3D-акторів. Клас *PetriActor* використовується для задання поведінки актору. Актор вміє знаходити шлях до точки в трьох вимірному просторі, також вміє рухатись в довільному напрямку. Клас *PetriActorsPool* – структура даних у

вигляді черги, в яку приходять і виходять актори в порядку їх потрапляння у відповідний компонент.

Прив’язка подій моделі до поведінки 3D-акторів задається класам *PetriComponent, TransitionComponent, PetriQueue* та класам, що їх успадковують. Для цього переозначаються відповідні методи в цих класах. Сама прив’язка відбувається у класі *PetriModelSubsriber*, який зв’язує другий та третій модулі.

Щоб описати класи програми та їх зв’язки у графічному матеріалі наведена схема структурна класів.

Покажемо процес взаємодії класів, які відповідають за імітацію моделі та класів, які відповідають за поведінку акторів.

SceneView – це вікно перегляду 3D-сцени. Для того щоб на ньому відобразилися дії акторів об’єкт класу *PlaybackModel* викликає на кожному кадрі метод *Simulate()*, якщо подія на моделі наступила, тобто виконались умови переходу, то викликається метод *TransitionTo()*, який відповідає за передачу маркерів із вхідних позицій до вихідних. Дальше цей метод за допомогою зворотнього виклику виконує метод *HandleEvents()* класа *PetriObjectSubscriber*, який відповідає за прив’язку подій. Дальше він визиває методи всіх об’єктів, які на нього підписались. Для цього вони мають реалізувати метод *OnTransition(),* тобто вони наслідуються від класу *PetriTransitionComponent,* який має цей абстрактний метод. Подальші виклики залежать від реалізації самого метода класами наслідниками, але зазвичай використовується взаємодія із класом *PetriActor* і його методом *GoToPoint()*, яка наказує актору переміститись в тану точку в просторі. Після цих викликів обновляється стан 3D-сцени і результат виводиться на екран.

Для того, щоб зрозуміти цю взаємодію краще у графічному матеріалі наведена схема структурна послідовності.

Система складається з двох основних компонентів: компонента редактора (*Editor*) і компонента середовища виконання (*Runtime*).

Компоненти редактора призначені для відображення вікон ігрового рушію Unity та всіх розширень. *PetriModelEditorWindow* – це складова редактора, яка його розширює. Результатом взаємодії з редактором є створення об'єктів, які будуть існувати в середовищі виконання. В даній системі *PetriModelEditorWindow* створює об'єкти класів *PetriNet.* Це є основною взаємодією даних компонентів.

Середовище Unity працює так, що на етапі збирання програми компілятор видаляє всі зв'язки з компонентом редактора, тому в програмний продукт, який готується для випуску всі ті класи не попадають, а спроба викликати їх видасть помилку. Це зберігає велику кількість пам'яті і дозволяє розділити компоненти редактора і середовища виконання.

Схема структурна компонентів програмного забезпечення наведена в частині графічного матеріалу.

4.4 Специфікація функцій

У таблиці 4.1 показана специфікація методів:

Таблиця 4.1 – Опис методів класів

Назва класу	*Назва методу*	*Дія*
EditorWindow	void OnGui()	Метод, за допомогою якого можна створювати власні розширення редактора
PetriModelEditor	void Draw()	Відповідає за прорисовку всіх елементів
PetriModelEditor	void SaveModel()	Зберігає модель в xml файлі

Продовження Таблиці 4.1

Назва класу	*Назва методу*	*Дія*
PetriModelEditor	void LoadModel()	Загружає модель з xml файлу
PetriNode	void Draw()	Відповідає в прорисовці одної ланки в графовому редакторі
PetriNode	void HandleEvents(Event)	Відповідає за обробку подій такий як OnMouseClick, OnMouseDrag та інших
PetriLink	void Draw()	Прорисовує дугу
PetriLink	bool Connect(PetriNode, PetriNode)	Зв’язує дугою дві ланки, і повертає чи вдалось зв’язати. (Наприклад неможливо зв’язати Позицію і Позицію)
PetriModelSubscriber	void Subscribe(string, PetriEvent)	Клас, який зв’язує модель і презентацію. Підписує дії об’єктів сцени на події моделі
PetriModelSubscriber	void Unsubscribe(string, PetriEvent)	Відписує дії об’єктів сцени на події моделі
PetriModelSubscriber	void HandleEvents()	Викликає відповідні події у відповідних об’єктів

Продовження Таблиці 4.1

Назва класу	*Назва методу*	*Дія*
MonoBehaviour	void Awake()	Викликається зразу при запуску симуляції один раз
MonoBehaviour	void Start()	Викликається коли об’єкт симуляції стає активним один раз
MonoBehaviour	void Update()	Викликається коли об’єкт симуляції стає активним кожен фрейм
TransitionComponent	void TransitionAction()	Викликається як відповідь на подію в моделі
PetriQueue	void EnterQueue(PetriObjectBehabiour)	Викликається коли ресурс поступив в чергу
PetriQueue	void ExitQueue(PetriObjectBehabiour)	Викликається коли ресурс вийшов з черги

Продовження Таблиці 4.1

PetriEvent	void OnTransitionEnter(PetriEventArgs)	Метод, що викликається коли подія настала
PetriEvent	void OnTransitionExit(PetriEventArgs)	Метод, що викликається коли подія завершилася
PetriEvent	void OnTransitionStay(PetriEventArgs)	Метод, що викликається коли подія настала, але ще не завершилася
PetriNet	void Simulate(float)	Приймає дельту часу, на яку симулює модель
TextComponent	void SetText()	За допомогою цього методу можна вивести будь-який параметр мережі як текст
PetriActorsPool	PetriActor Dequeue()	Забирає актора із черги
PetriActorsPool	void Enqueue(PetriActor)	Забирає актора в чергу
PetriActor	void GoToPoint(Vector3)	Метод, що каже актору знайти шлях до заданої точки, та почати рухатись до неї

Висновок до розділу

В даному розділі було розглянуто програми, та мови програмування, за допомогою яких було реалізовано програмний продукт. Також показано три види діаграм із детальним поясненням до них. Розглянуто загальні вимоги для технічних засобів. Та показано опис дій методів.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для роботи потрібно встановити Unity 2018.3 або вище. Для того щоб почати роботу в редакторі натискаєм праву кнопку в вікні проекту потім Create – PetriNetGraph (рисунок 5.1). Після цього появиться пусте робоче полотно (рисунок 5.2), дане вікно можна переміщати в редакторі і прикріпляти до інших вікон [20].

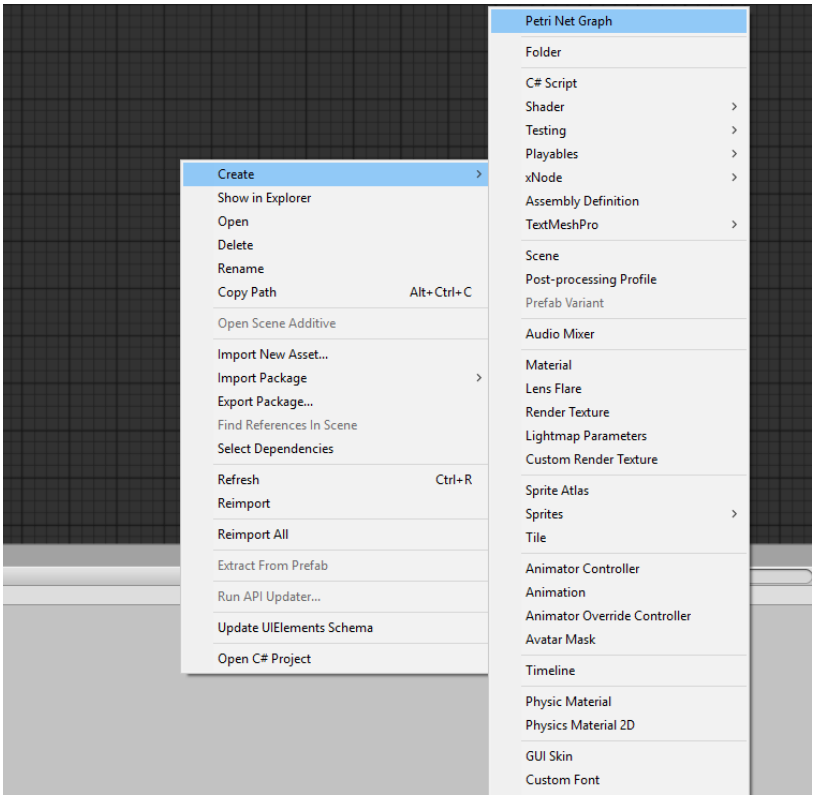


Рисунок 5.1 – Створення Петрі-моделі

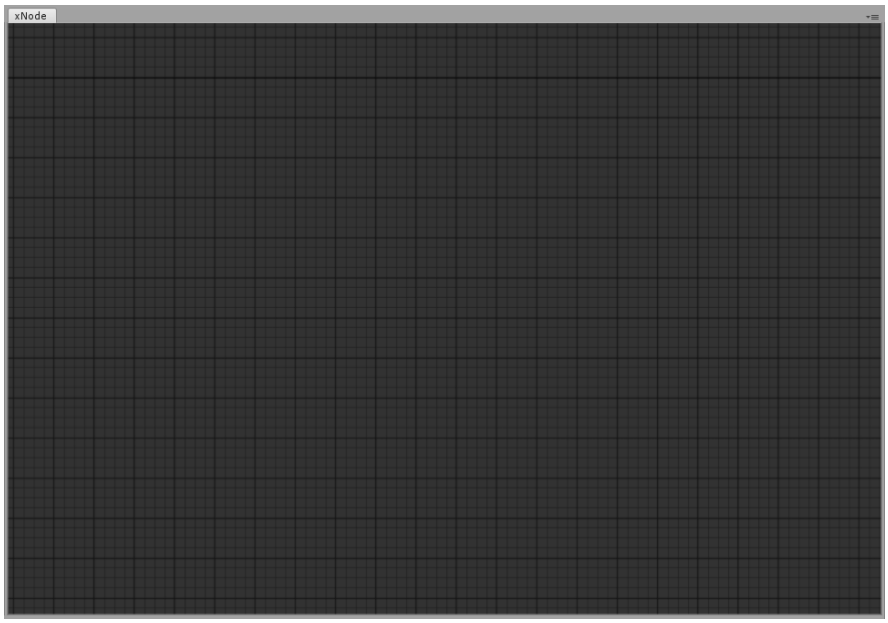


Рисунок 5.2 – Пусте полотно редактора

Для того, щоб добавити вузол потрібно натиснути праву кнопку і в контекстному меню вибрати потрібний вузол (рисунок 5.3). Всього існує три види вузлів: позиція, перехід та об’єкт (рисунок 5.4).

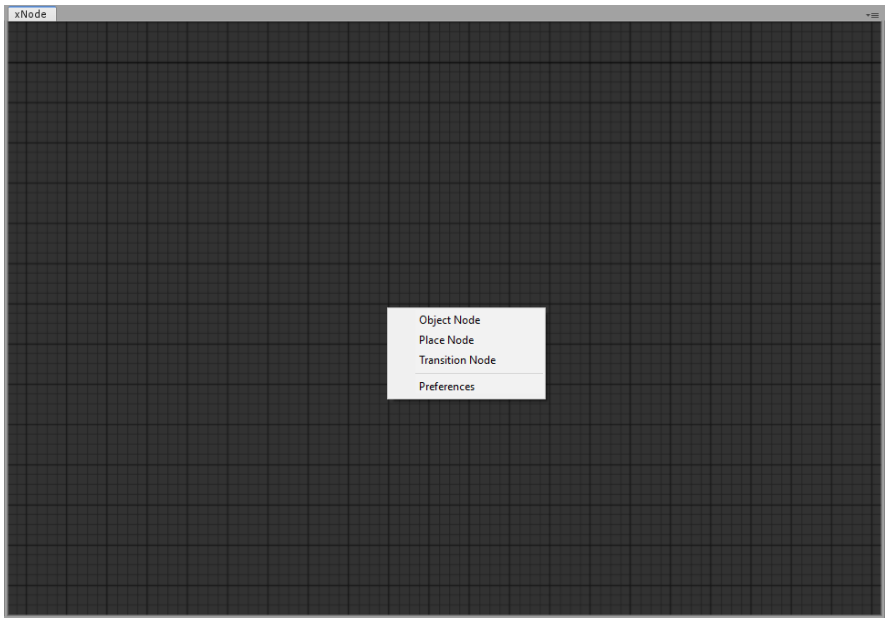


Рисунок 5.3 – Створення нового вузла

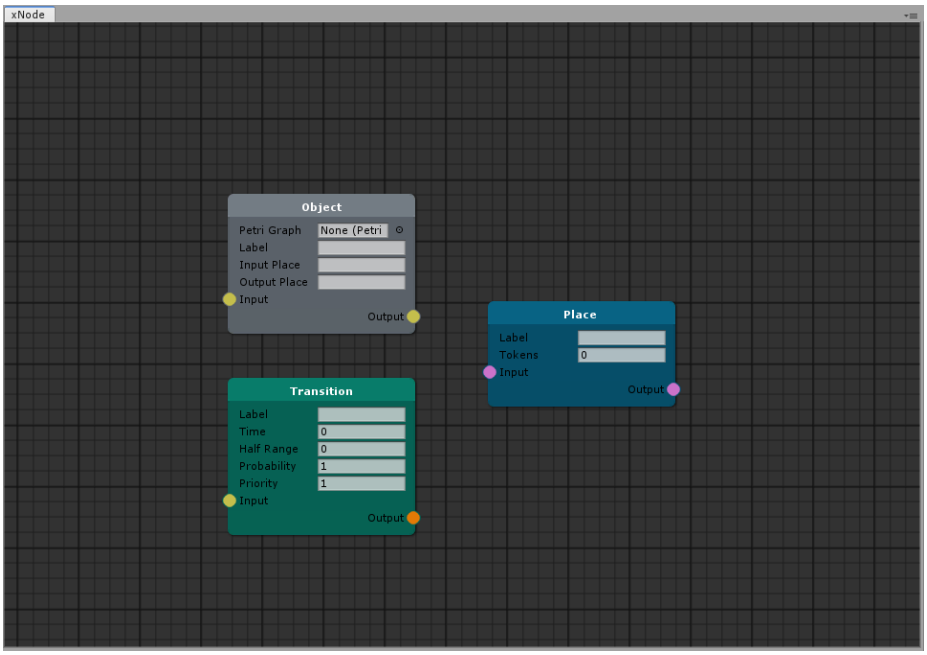


Рисунок 5.4 – Всі різні типи вузлів

Всі змінні параметри знаходяться всередині блока вузла. Для коректної роботи 3D-анімації бажано задати всім вузлам назву (поле Label). Воно буде використовуватись для прив’язки подій моделі до анімації на 3D-сцені.

Для того щоб створити зв’язок, потрібно перетягти мишкою з виходу вузла «Output» до входу іншого вузла «Input». Зауважу, що з’єднювати однотипні вузли неможливо (крім об’єктів, про які пізніше).

Вузлу «Place», що відповідає позиції в Петрі-об’єктній моделі можна задати параметри:

– Label – назва вузла;
– Tokens – початкове маркування.

Вузлу «Transition», що відповідає переходу в Петрі-об’єктній моделі можна задати параметри:

– Label – назва вузла;
– Time – час переходу;
– Half Range – відхилення від часу переходу (Time ± Half Range);
– Probability – ймовірність наставання події при конфліктному переході;

– Priority – пріоритет наставанння події при конфліктному переході (менше число – більший пріоритет).

Вузлу «Object», що відповідає об’єкту в Петрі-об’єктній моделі можна задати параметри:

– PetriGraph – мережа Петрі, яку описує даний об’єкт;
– Label – назва вузла;
– InputPlace – назва позиції, яка буде використовуватись як спільна позиція з’єднаного об’єкта до входу «Input»;
– OutputPlace – назва позиції, яка буде використовуватись як спільна позиція з’єднаного об’єкта з виходом «Output».

На рисунку 5.5 зображений приклад створеної мережі:

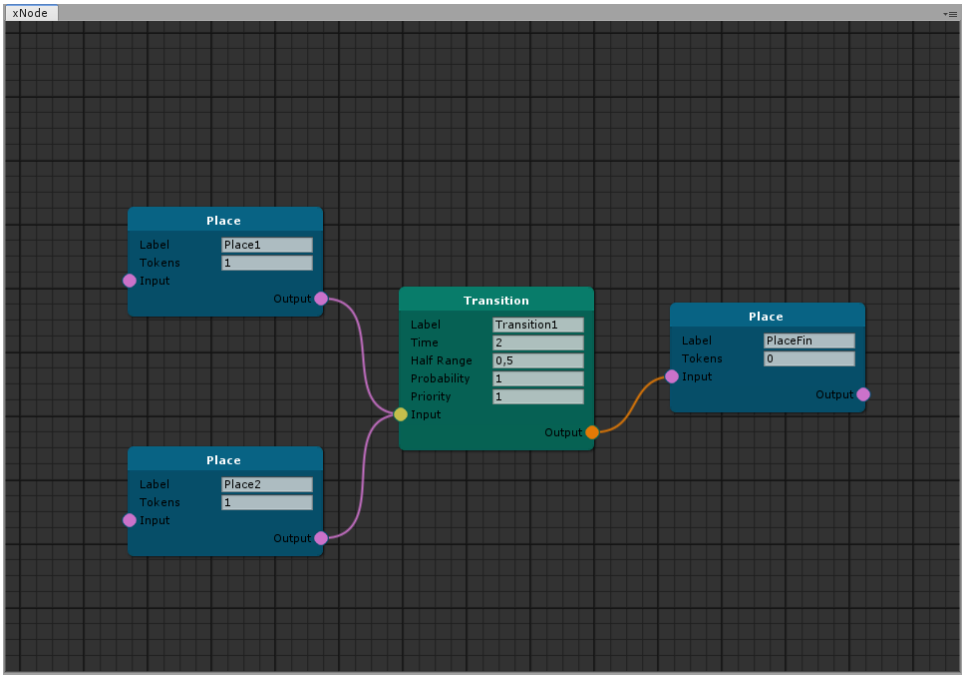


Рисунок 5.5 – Створений за допомогою редактора Петрі-граф

Створену вище мережу можна інкапсулювати в Петрі-об’єкт, що дозволить з легкістю тиражувати цілі блоки мереж як один об’єкт. На рисунку 5.6 зображено як будуть виглядати три таких послідовно з’єднаних об’єкта.

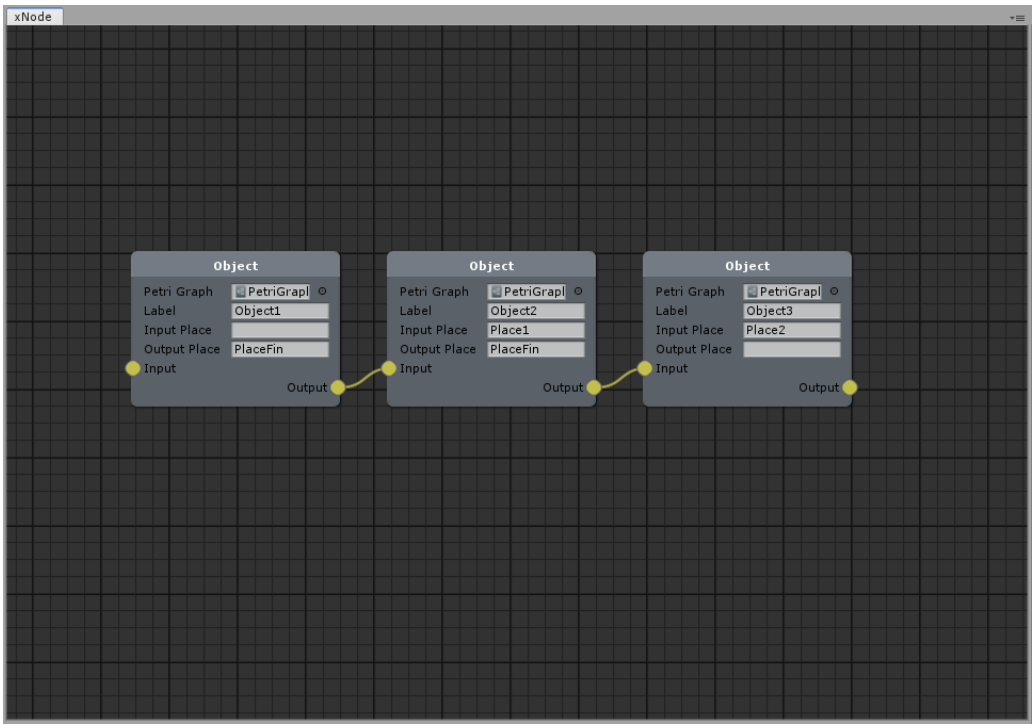


Рисунок 5.6 – Послідовно з'єднані Петрі-об'єкти

Якби цю мережу проектувати не Петрі-об'єктним способом, то вона мала б вигляд як на рисунку 5.7.

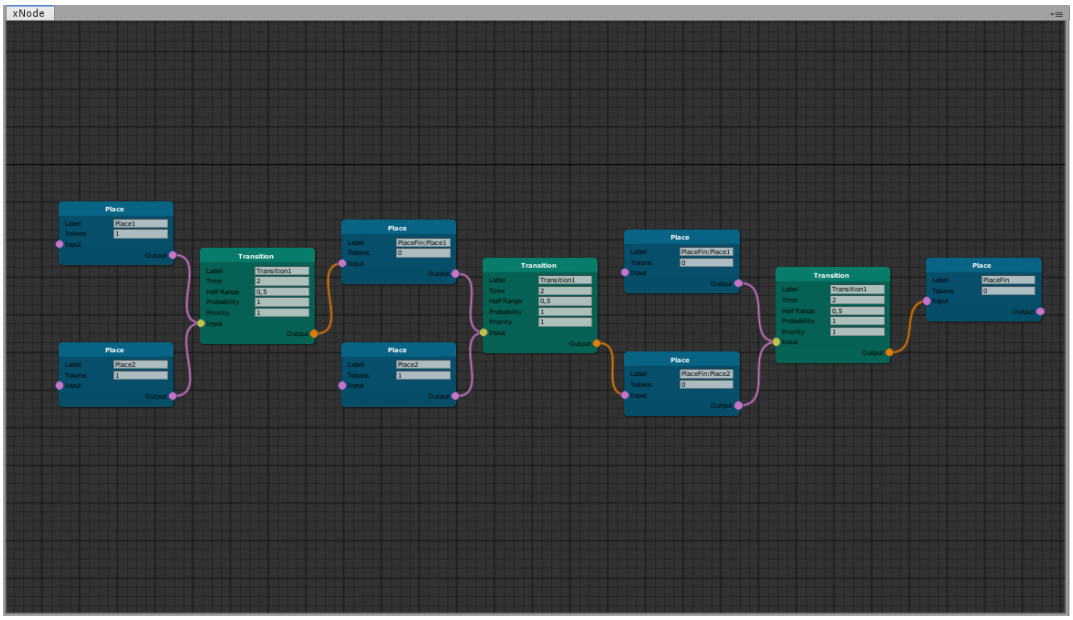


Рисунок 5.7 – Структура мережі, що не використовує Петрі-об'єкти

Для того, щоб додати ваги дуг, потрібно цю ж дугу протягнути ще раз. Якщо вага дуги буде більша одиниці, то вона стане відображатись на графі (рисунок 5.8).

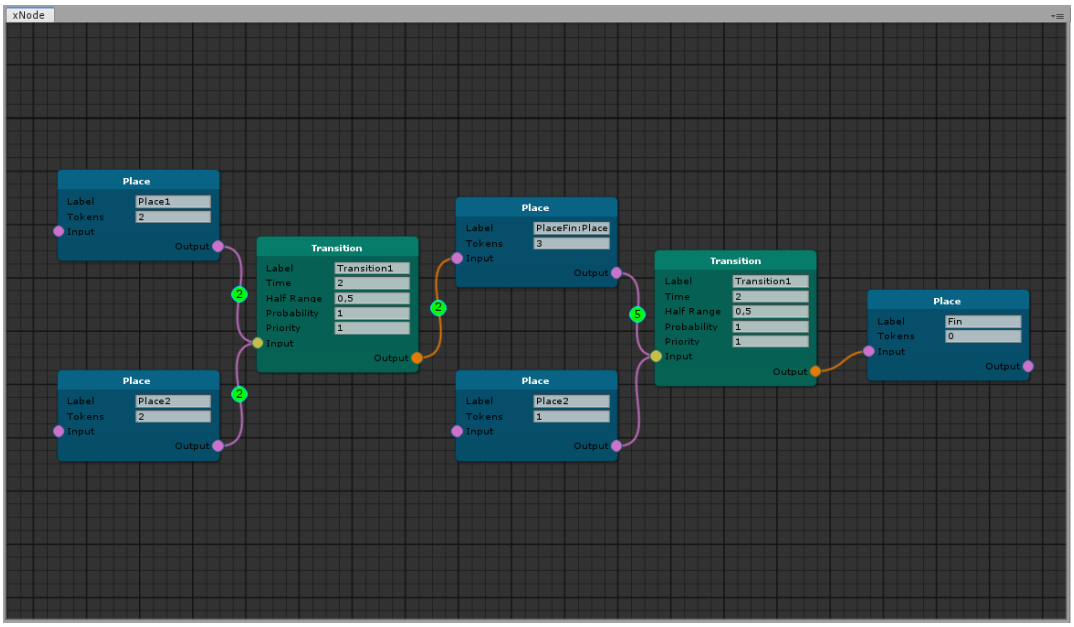


Рисунок 5.8 – Мережа зі зваженими дугами

Для роботи з полотном редактора використовуються звичні для кожного клавіші і комбінації:

– ЛКМ – для вибору, перетягування вузлів і створення зв'язків;
– ПКМ – для виклику контекстного меню;
– СКМ – затримуючи можна переміщатися по полотну, гортаючи кнопку вверх або вниз можна збільшути або зменшувати зображення;
– Delete – видаляє вузол;
– Ctrl + D – робить дублікат вибраного вузла.

Всі зміни при роботі з редактором зберігаються автоматично.

Для того, щоб додати модель на 3D-сцену потрібно створити пустий об’єкт і добавити йому в вікні інспектора компонент ModelPlayback. В поле Petrinet Graph потрібно додати створену за допомогою редактора мережу (рисунок 5.9). Якщо поставити галочку на параметрі Debug, то стан моделі буде виводитись в консоль.

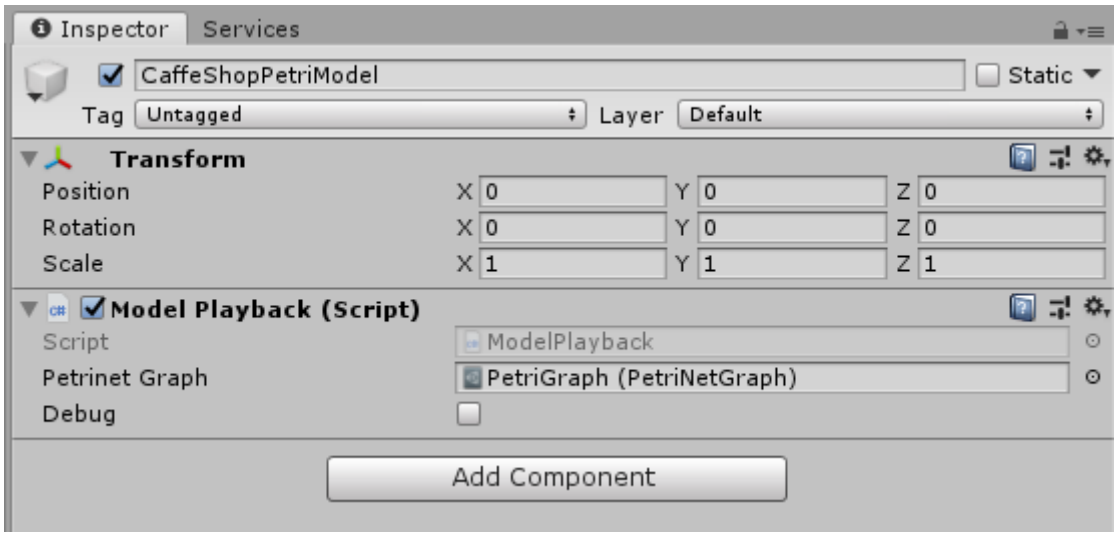


Рисунок 5.9 – Вікно інспектора об’єкта, який буде імітувати передану йому Петрі-об’єктну модель

В одній сцені може бути як завгодно багато моделей, які будуть імітуватися незалежно та паралельно.

Для зв’язку Петрі-об’єктної моделі і 3D-сцени використовуються спеціальні компоненти, які наслідуються від PetriComponent. Їм потрібно передати посилання на ModelPlayback, а також вказати назву Петрі позиції чи місця, якого стосується цей компонент.

Існують готові компоненти такі як: генератор, черга, компонент руху актора до конкретної точки, чи вільно в будь-якому напрямку, вивід статистики, вивід маркування як текст та інші. Цей функціонал можна розширювати створюючи нові класи, які наслідуються від PetriComponent – якщо потрібно просто взяти параметри моделі, PetriTransition – якщо потрібно виконати дію при виконанні події входу, або виходу маркувань в перехід на моделі Петрі, PetriQueue – якщо потрібно розширити функціонал черги, TextComponent – якщо потрібно вивести параметр моделі як текст.

Система використовує концепцію Петрі-акторів. Поведінка яких буде задаватися станом Петрі-об’єктної моделі. Створюються такі актори в компонентах-генераторах і добавляються в пул акторів.

Кожен об’єкт може мати вхідний та вихідний пули акторів. Вихідний пул одного компонента може вказатись вхідним для іншого. При виході актора з одного компонента він видаляється з його вихідного пула акторів і добавляється до вхідного пула акторів компонента, якому передається управління актором.

Сам актор – це 3D-модель, з власною машиною станів яка обробляє анімацію, і також модулем знаходження найближчого шляху до точки в просторі – NavMesh. Інтерфейс взаємодії з актором дозволяє посилати його в конкретну точку простору, або наказати рухатися в довільних напрямках. Швидкість, прискорення, кутову швидкість, ширину агента і інші параметри можна відредагувати в компоненті NavMeshAgent, який зображений на рисунку 5.10.

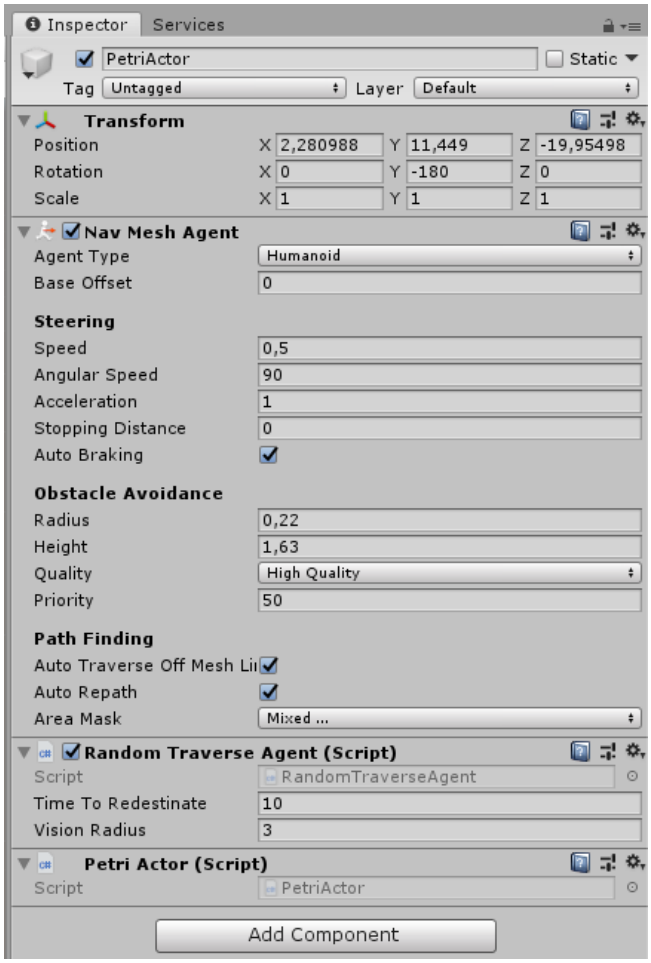


Рисунок 5.10 – Компоненти шаблонного Петрі-актора

Для прикладу створимо сцену (рисунок 5.11) в якій люди будуть приходити до магазину кави, ставати в чергу (обмежену 12 людьми), коли черга дійде до людини він затримається на деякий час біля прилавку (замовлення кави) і після цього піде в випадковому напрямку.

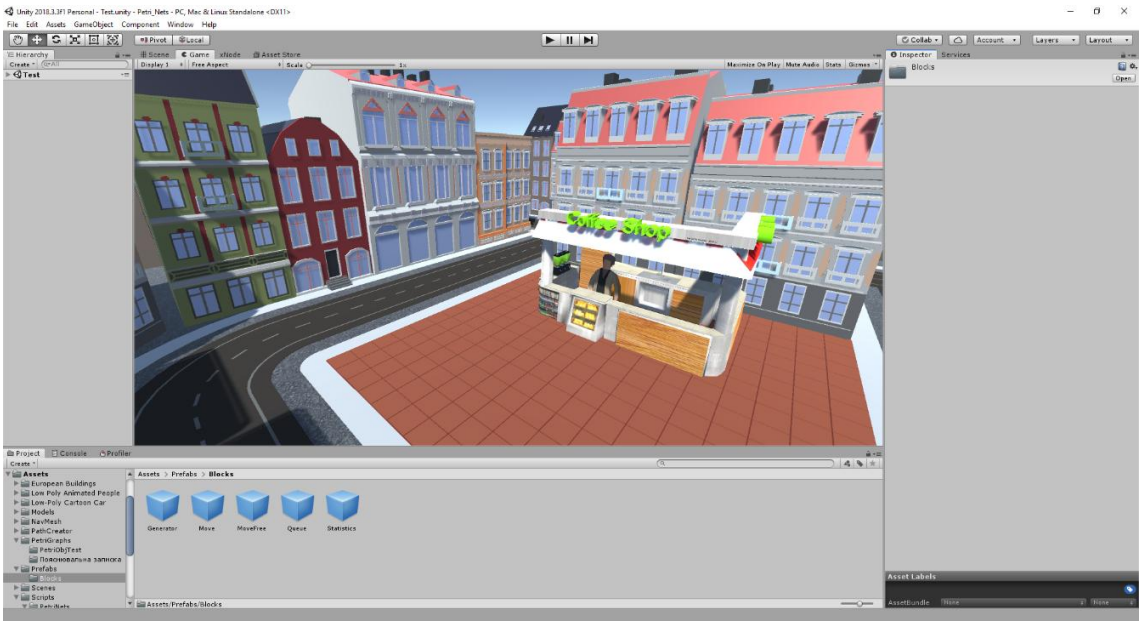


Рисунок 5.11 – 3D-сцена міста з магазином кави

На рисунку 5.12 зображена Петрі-модель такої системи:

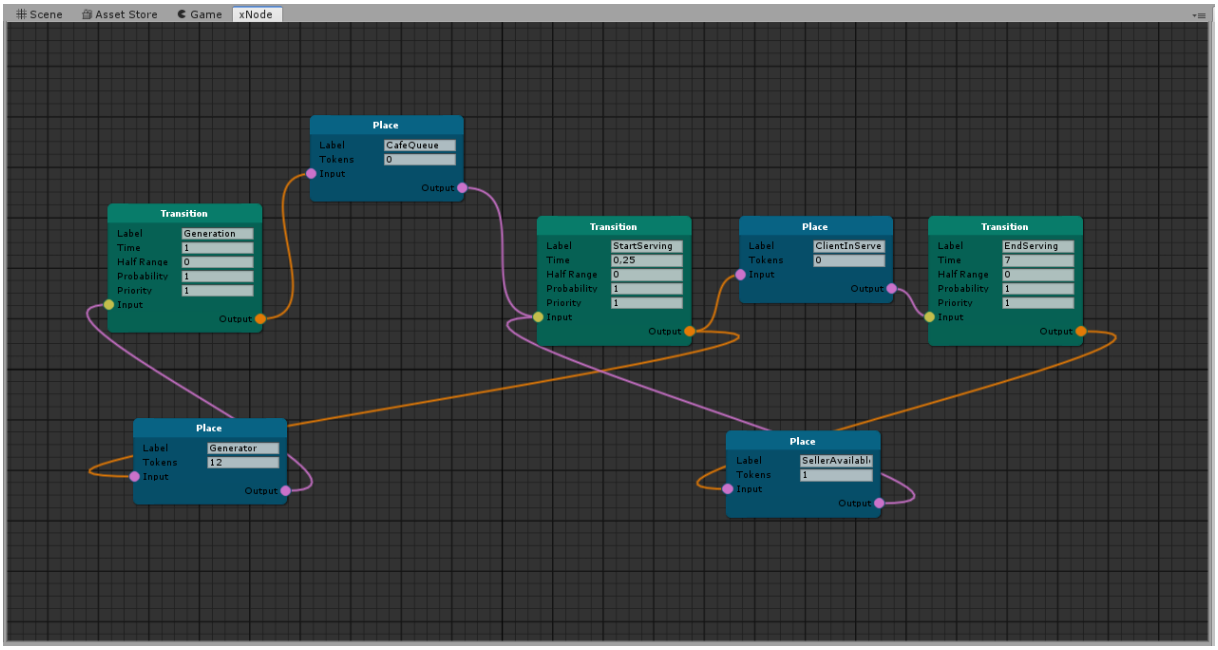


Рисунок 5.12 – Петрі-модель магазину кави

Добавим новий об'єкт ModelPlayback і передамо йому тільки що створену модель. Почнем моделювання з компонента генератора (рисунок 5.13). В цього компонента є такі параметри:

– Nets Reference – передаєм посилання на об’єкт ModelPlayback;
– Petri Label – назва переходу, який буде триггерити генерацію Петрі-акторів;
– Transition Event – перелічення з двох варіантів:
  1) Transition Enter – триггериться, коли почався перехід;
  2) Transition Exit – триггериться, коли перехід завершився;
– Input Actors Pool – вхідний пул акторів. Так як для генератора він не потрібен, то залишається пустим;
– Output Actors Pool – посилання на вихідний пул акторів, в якому будуть з’являтися згенеровані актори;
– Prefab – шаблон актора, який буде клонуватися;
– Instantiation Point – точка, з якої будуть появлятися актори;
– Parent Game Object – батьківський об’єкт для згенерованих акторів.

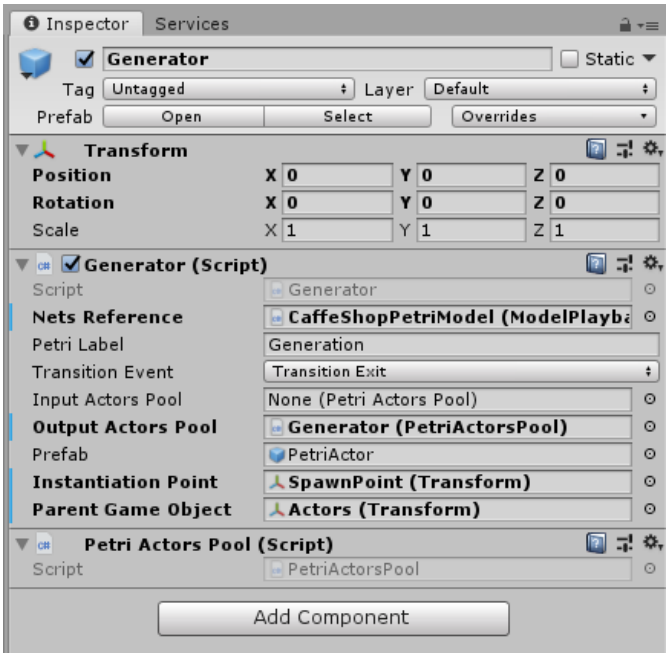


Рисунок 5.13 – Компонент Generator

Після цього добавим чергу, в яку будуть поступати актори (рисунок 5.14). За допомогою сплайна можна редагувати форму, яку буде приймати черга.

Рисунок 5.14 – Редагування форми черги

На рисунку 5.15 зображені параметри, які має компонент черги (похожі на описані раніше поля були опущені):

– Path – посилання на об’єкт, що визначає форму черги (в даному випадку це кубічний сплайн);
– Queue Enter Transition Label – назва переходу, який буде триггерити вхід актора до черги;
– Queue Exit Transition Label – назва переходу, який буде триггерити вихід актора з черги;
– Queue Size Petri Label – назва позицію, яка вказує на поточну довжину черги;
– Fraction Space Needed – число, яке позначає яку частину черги буде займати один актор.

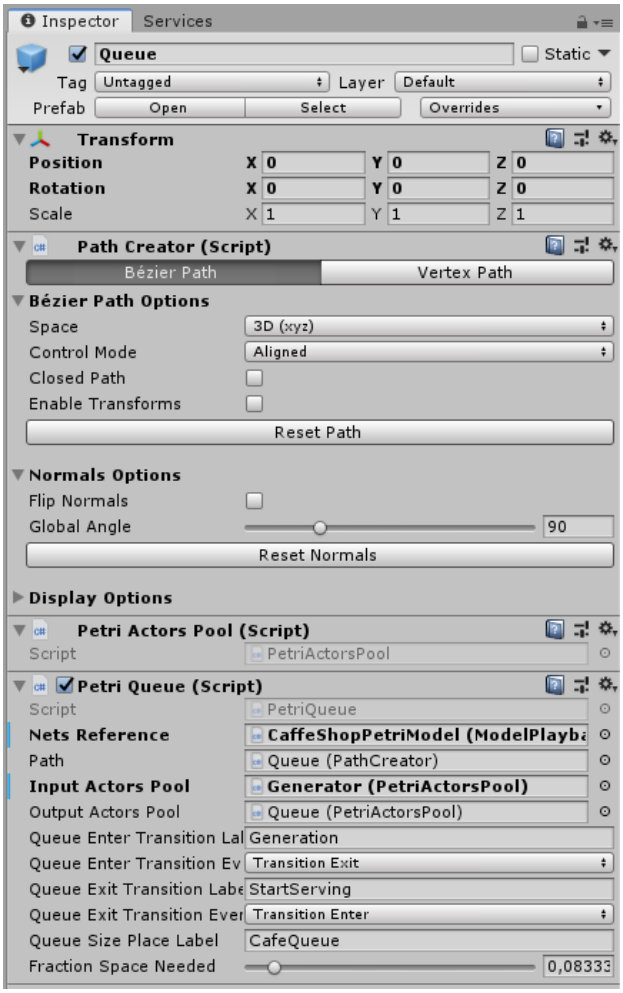


Рисунок 5.15 – Компонент Queue

Для того, щоб показати, що актор замовив каву використаєм компонент Move (рисунок 5.16). Особливість цього компонента в тому, що актор після того, як завершив шлях до вказаної точки – зупиняється. Так ми покажем як актор перемістився з голови черги до місця перед продавцем, де робиться замовлення.

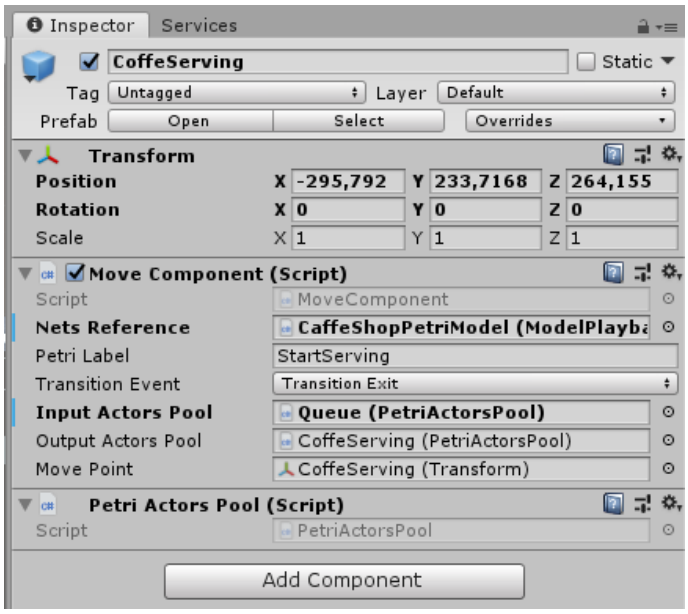


Рисунок 5.16 – Компонент Move

Для того щоб показати, що актор отримав своє замовлення і пішов своєю дорогою використаєм компонент MoveFree (рисунок 5.17), який похожий на компонент Move. Відрізняється лише тим, що переміщається не до конкретної точки, а наказує актору рухатись в довільних напрямках.

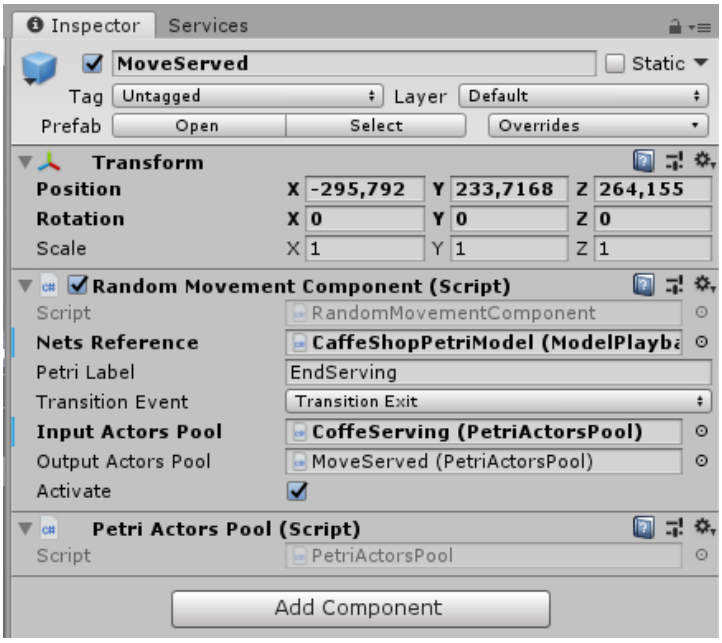


Рисунок 5.17 – Компонент MoveFree

Нажавши кнопку запуску на інтерфейсі Unity ми бачимо, через деякий час, що появляється перший клієнт (рисунок 5.18), а почекавши ще трохи ми бачимо, що черга до магазину уже максимально заповнена і більше людей не підходить (рисунок 5.19).

Рисунок 5.18 – Імітаційне моделювання (прийшов перший клієнт)

Рисунок 5.19 – Імітаційне моделювання (максимальна черга)

А тепер добавим виведення статистики. Для цього заготовлений спеціальний компонент – Statistica, який покаже всю статистику в реальному часі (рисунок 5.20). Статистика виводиться для всіх позицій і переходів.

Рисунок 5.20 – Імітаційне моделювання (вікно статистики)

Після завершення моделювання звіт з моделювання зберігається у вигляді текстового документу, який зображений на рисунку 5.21.

Modeling Example.txt — Блокнот

Файл Правка Формат Вид Справка

```
============================
ModelingTme: 134:32
============================
| Place | M | AM | MM | SM |
============================
| Generator | 0 | 0.04 | 1 | 57 |
| CafeQueue | 12 | 11.84 | 12 | 31 |
| ClientInService | 1 | 0.94 | 1 | 19 |
| SellerAvailable | 0 | 0.06 | 1 | 20 |
============================
| Trans | U | UC | AT | ST |
============================
| Generation | 56 | 0.42 | 1 | 56.28 |
| StartServing | 20 | 0.95 | 0.25 | 127.3 |
| EndServing | 18 | 0.96 | 7 | 128.94 |
```

Рисунок 5.21 – Звіт з моделювання

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності 3D анімаціям і подіям на сцені в реальному часі подіям згенерованими Петрі-об’єктною моделлю.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

За результатами тестів, була перевірена повна функціональність даної моделі. У наступних таблицях було наведено повний перелік випробувань функціональних можливостей, що вважаються основними.

Таблиця 5.1 – Редагування об’єктної моделі графічно

Мета тесту	**Перевірка функції «Редагування об’єктної моделі»**
Початковий стан моделі	Створена модель в якій є одна позиція і один перехід
Вхідні дані:	xml файл моделі
Схема проведення тесту:	Протягнути мишкою з вихідного порта позиції до вхідного порту переходу
Очікуваний результат	Появилась на екрані дуга яка веде з позиції до переходу
Стан моделі після проведення випробувань:	Над дугою появилась цифра «1», що означає стандартну вагу дуги

Таблиця 5.2 – Симуляція системи на незначний проміжок часу

Мета тесту	**Перевірка функції «Симуляції системи»**
Початковий стан моделі	Створена симуляція з усіма подіями, яка готова до запуску
Вхідні дані:	xml файл моделі та сцена з об’єктами
Схема проведення тесту:	Запуск симуляції на дуже маленький період часу, з який жодна із подій не мали настати.
Очікуваний результат	Запустяться тільки переходи з нульовим входженням.
Стан моделі після проведення випробувань:	Запустяться тільки переходи з нульовим входженням.

Таблиця 5.3 – Створення нової моделі та введення даних

Мета тесту	**Перевірка функції «Створення нової моделі та введення даних»**
Початковий стан моделі	Відкрите вікно редагування моделі
Вхідні дані:	-
Схема проведення тесту:	Натиснути кнопку «Створити модель», натиснути праву кнопку миші і в контекстному меню вибрати «Перехід». В полі «Час переходу» написати число
Очікуваний результат	На екрані намалюється перехід з введеним часом
Стан моделі після проведення випробувань:	На екрані намалюється перехід з введеним часом

Таблиця 5.4 – Завершення симуляції

Мета тесту	**Перевірка функції «Завершення симуляції»**
Початковий стан моделі	Запущена симуляція
Вхідні дані:	xml файл моделі та сцена з об’єктами
Схема проведення тесту:	Натиснути кнопку «Завершити симуляцію»
Очікуваний результат	Симуляція завершується без жодних помилок
Стан моделі після проведення випробувань:	На екран виводиться журнал симуляції

Таблиця 5.5 – Симуляція системи на значний проміжок часу

Мета тесту	**Перевірка функції «Запуск роботи алгоритму»**
Початковий стан моделі	Запущена симуляція
Вхідні дані:	xml файл моделі та сцена з об’єктами
Схема проведення тесту:	Просунути модель на великий проміжок часу, під час якого мали б відбутися сотні подій
Очікуваний результат	Модель успішно згенерує всі події і просунеться на час вказаний програмістом
Стан моделі після проведення випробувань:	Появиться вікно з індикатором загрузки і обробки даних

Висновок до розділу

В даному розділі було дано вичерпну користувацьку інструкцію користування програмним засобом, було перелічено повний список функціоналу, що був розроблений в рамках даної системи, за допомогою знімків екрану і їх пояснення.

Було наведено конкретний приклад моделі із детальним поясненням кожного кроку її розробки.

ЗАГАЛЬНІ ВИСНОВКИ

Даний дипломний проект розглядає роботу Петрі-об'єктного моделювання систем з використанням 3D-анімації.

У розділі загальних положень розглядається основний процес діяльності користувача та функціональна модель за допомогою схеми варіантів використання. З її допомогою було показано функціональні вимоги до системи. Крім того був наведений короткий опис предметного середовища. Було наведено відомості про дискретно-подійне моделювання, основні принципи роботи мереж Петрі та показано переваги Петрі-об'єктного підходу до моделювання систем. Також було описано мету та призначення розробки.

У розділі наявних аналогів були описані інші рішення, які вирішують потібну проблему та були наведені їх основні переваги. В кінці розділу було зазначено чим розроблений програмний продукт вигідно відрізняється від вказаних аналогів.

У розділі інформаційного забезпечення були описані дані, якими оперує система та структури даних для їх зберігання. Серед вхідних даних, з якими працює користувач є 3D-моделі, та граф мережі Петрі, який зберігається у форматі .xml. Після роботи з системою, а саме по закінченню моделювання користувач отримує звіт з моделювання у вигляді .txt документа. Структура вхідних та вихідних даних була наведена в цьому розділі.

У математичному розділі наведено теоретичні відомості про роботу Петрі-об'єктних мереж та наведено їх математичний опис. Було описано алгоритм імітації Петрі-об'єктних моделей.

У розділі опису методів розв'язання було наведено проблеми використання даного алгоритму імітації та були запропоновані методи, за допомогою яких Петрі-об'єктна імітація може працювати в режимі реального часу.

У розділі програмного та технічного забезпечення наведено опис використаних технологій, архітектури програмного забезпечення та побудована схема структурна класів для її відображення. Були представлені основні компоненти програмного забезпечення у вигляді схеми структурної компонентів. Основна взаємодія класів та об'єктів програмного забезпечення були відображені на схемі структурній послідовності. Представлені таблиці, що містять опис методів та їх параметрів, що представлені в схемі структурній класів.

У технологічному розділі наведено вичерпне керівництво користувача та екранні форми застосунку. Основні тестові сценарії та результат їх проходження описаний у розділі випробування програмного продукту.

Після проходження всіх етапів розробки та тестування можна зазначити, що програмний продукт відповідає всім функціональним вимогам, які до нього ставили у технічному завданні.

ПЕРЕЛІК ПОСИЛАНЬ

1. Стеценко І. В. Моделювання систем: навч. посіб. / І. В. Стеценко. – М-во освіти і науки України, Черк. держ. технол. ун-т. – Черкаси : ЧДТУ, 2010. – 399 с.
2. Імітаційне моделювання систем та процесів: Електронне навчальне видання. Конспект лекцій / В. Б. Неруш, В. В. Курдеча. – К.: НН ІТС НТУУ «КПІ», 2012. – 115 с.
3. AnyLogic: Simulation Modeling Software Tools & Solutions for Business [Електронний ресурс] – Режим доступу: https://www.anylogic.com/
4. Arena Simulation [Електронний ресурс] – Режим доступу: https://www.arenasimulation.com/
5. Plant Simulation and Throughput Optimization [Електронний ресурс] – Режим доступу: https://www.plm.automation.siemens.com/
6. Simulation, Production planning and Scheduling Software | Simio [Електронний ресурс] – Режим доступу: https://www.simio.com/index.php
7. Extend Sim Simulation Software [Електронний ресурс] – Режим доступу: https://extendsim.com/
8. CPN Tools: A tool for editing, analyzing, and simulating Colored Petri Nets [Електронний ресурс] – Режим доступу: http://cpntools.org/
9. Jump start your autonomous simulation development with Unity's SimViz Solution Template [Електронний ресурс] – Режим доступу: https://blogs.unity3d.com/2018/11/08/jump-start-your-autonomous-simulation-development-with-unitys-simviz-solution-template/
10. Імітаційне моделювання систем масового обслуговування: Практикум. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. – 307 с

11. Microsoft Windows [Електронний ресурс] – Режим доступу: https://www.microsoft.com/uk-ua/windows
12. Unity [Електронний ресурс] // Режим доступу: https://unity.com/
13. Microsoft Visual Studio [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
14. C Sharp [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp
15. Enterprise Architect [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Enterprise_Architect_(software)
16. Draw.io [Електронний ресурс] – Режим доступу: https://www.draw.io/
17. CryEngine [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/CryEngine_(серія_рушіїв)
18. Godot [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Godot
19. Unreal Engine [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Unreal_Engine
20. Unity Editor Window [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/editor-EditorWindows.html
21. XML [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/XML#cite_note-XmlOriginsGoals-1
22. Стеценко И. В. Теоретические основы Петри-объектного моделирования систем / И. В. Стеценко // Математичні машини і системи. – Киев, 2011. – № 4. – С. 136-148.
23. Game Loop (ігровий цикл) [Електронний ресурс] – Режим доступу: https://gameprogrammingpatterns.com/game-loop.html.

PetriNet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace PetriNetLight
{
    public class PetriNet
    {
        private PetriNetGraph graphAsset;

        private PetriPlace[] places;
        private PetriTransition[] transitions;

        private float SimulationTime { get; set; }
        public float Time { get; private set; }
        private float InitialTime { get; set; }
        private float previousTransitionTime = 0;

        private List<(float, PetriTransition)> lateExecutionQueue;

        public PetriNet()
        {
            places = new PetriPlace[0];
            transitions = new PetriTransition[0];
            lateExecutionQueue = new List<(float, PetriTransition)>();
        }

        public PetriNet(PetriNetGraph graphAsset) : this()
        {
            LoadFromGraphAsset(graphAsset);
        }

        public void SetPlaces(params PetriPlace[] places)
        {
            this.places = places;
        }
        public void SetTransitions(params PetriTransition[] transitions)
        {
            this.transitions = transitions;
        }
        public PetriPlace[] GetPlaces()
        {
            return places;
        }
        public PetriTransition[] GetTransitions()
        {
            return transitions;
        }
```

```
public void Simulate(float deltaTime)
{
   Time += deltaTime;

   if (lateExecutionQueue.Count > 0)
   {
      var readyFromQueue = from t in lateExecutionQueue
                           where Time >= t.Item1
                           select t;

      List<(float, PetriTransition)> removeList = new List<(float, PetriTransition)>();
      foreach (var t in readyFromQueue)
      {
         t.Item2.TransitTo(Time);
         removeList.Add(t);
      }

      foreach (var t in removeList)
      {
         lateExecutionQueue.Remove(t);
      }
   }

   while (true)
   {
      var rand = UnityEngine.Random.value;
      var readyToTransit = (from t in transitions
                            where !t.IsProccesing && t.TryTransition()
                            orderby t.TransitionPriority, t.TransitionProbability * UnityEngine.Random.value descending
                            select t).ToList();

      if (readyToTransit.Count() == 0)
         break;

      foreach (var t in readyToTransit)
      {
         if (t.IsProccesing || !t.TryTransition())
            continue;

         t.TransitFrom(Time);

         if (t.TransitionTime > 0)
            lateExecutionQueue.Add((t.TransitionTime + Time, t));
         else
         {
            t.TransitTo(Time);
         }
      }
   }
```

```
            // Save statistics
            foreach (var t in places)
                t.DoStatistic();
            foreach (var t in transitions)
                t.DoStatistic();
        }

        public void LoadFromGraphAsset(PetriNetGraph graphAsset)
        {
            this.graphAsset = graphAsset;

            List<(PetriPlace, string)> places = new List<(PetriPlace, string)>();
            List<(PetriTransition, string)> transitions = new List<(PetriTransition, string)>();
            List<ArcIn> arcsIn = new List<ArcIn>();
            List<ArcOut> arcsOut = new List<ArcOut>();
            List<(string, string, int)> arcsInIds = new List<(string, string, int)>();
            List<(string, string, int)> arcsOutIds = new List<(string, string, int)>();

            // Check Petri Objects
            #region PetriObjects
            foreach (ObjectNode pObj in graphAsset.nodes.Where((i) => i is ObjectNode))
            {
                foreach (var node in pObj.petriGraph.nodes)
                {
                    var np = node as PlaceNode;
                    var nt = node as TransitionNode;
                    if (np)
                    {
                        PetriPlace p = new PetriPlace() { Label = pObj.label + "." + np.label,
Tokens = np.tokens, RealTokens = np.tokens };
                        places.Add((p, pObj.id + "." + np.id));

                        foreach (var port in np.Outputs)
                        {
                            foreach (var con in port.GetConnections())
                            {
                                var connectedNt = con.node as TransitionNode;
                                arcsInIds.Add((pObj.id + "." + np.id, pObj.id + "." + connectedNt.id,
port.GetConnectionWeigth(port.GetConnectionIndex(con))));
                            }
                        }
                    }
                    if (nt)
                    {
                        PetriTransition t = new PetriTransition()
                        {
                            Label = pObj.label + "." + nt.label,
                            TransitionTime = nt.time,
                            TransitionProbability = nt.probability,
```

```
                        TransitionPriority = nt.priority
                    };
                    transitions.Add((t, pObj.id + "." + nt.id));

                    foreach (var port in nt.Outputs)
                    {
                        foreach (var con in port.GetConnections())
                        {
                            var connectedNp = con.node as PlaceNode;
                            arcsOutIds.Add((pObj.id + "." + nt.id, pObj.id + "." + connectedNp.id, port.GetConnectionWeigth(port.GetConnectionIndex(con))));
                        }
                    }
                }
            }
        }
        #endregion

        foreach (var node in graphAsset.nodes)
        {
            var np = node as PlaceNode;
            var nt = node as TransitionNode;
            var no = node as ObjectNode;
            if (no)
            {
                if (no.outputTransition != "")
                {
                    foreach (var port in no.Outputs)
                    {
                        foreach (var con in port.GetConnections())
                        {
                            var connectedNt = con.node as PlaceNode;
                            arcsOutIds.Add((transitions.Find((i) => i.Item1.Label == no.label + "." + no.outputTransition).Item2, connectedNt.id, port.GetConnectionWeigth(port.GetConnectionIndex(con))));
                        }
                    }
                }
            }
            if (np)
            {
                PetriPlace p = new PetriPlace() { Label = np.label, Tokens = np.tokens, RealTokens = np.tokens };
                places.Add((p, np.id));

                foreach (var port in np.Outputs)
                {
                    foreach (var con in port.GetConnections())
                    {
```

```
                        var connectedNt = con.node as TransitionNode;
                        var connectedToObj = con.node as ObjectNode;

                        if (connectedNt)
                            arcsInIds.Add((np.id,connectedNt.id,
port.GetConnectionWeigth(port.GetConnectionIndex(con))));

                        if (connectedToObj)
                        {
                            if (connectedToObj.inputTransition != "")
                            {
                                arcsInIds.Add((np.id, transitions.Find((i) => i.Item1.Label ==
connectedToObj.label + "." + connectedToObj.inputTransition).Item2,
port.GetConnectionWeigth(port.GetConnectionIndex(con))));
                            }
                        }
                    }
                }
            }

            if (nt)
            {
                PetriTransition t = new PetriTransition()
                {
                    Label = nt.label,
                    TransitionTime = nt.time,
                    TransitionProbability = nt.probability,
                    TransitionPriority = nt.priority
                };
                transitions.Add((t, nt.id));
                foreach (var port in nt.Outputs)
                {
                    foreach (var con in port.GetConnections())
                    {
                        var connectedNp = con.node as PlaceNode;
                        arcsOutIds.Add((nt.id, connectedNp.id,
port.GetConnectionWeigth(port.GetConnectionIndex(con))));
                    }
                }
            }
        }
        foreach (var ain in arcsInIds)
        {
            ArcIn arcIn = new ArcIn()
            {
                PlaceFrom = places.Find((t) => t.Item2 == ain.Item1).Item1,
                TransitionTo = transitions.Find((t) => t.Item2 == ain.Item2).Item1,
                Weight = ain.Item3
            };
```

```
                arcsIn.Add(arcIn);
            }

            foreach (var aout in arcsOutIds)
            {
                ArcOut arcOut = new ArcOut()
                {
                    PlaceTo = places.Find((t) => t.Item2 == aout.Item2).Item1,
                    TransitionFrom = transitions.Find((t) => t.Item2 == aout.Item1).Item1,
                    Weight = aout.Item3
                };

                arcsOut.Add(arcOut);
            }

            foreach (var t in transitions)
            {
                t.Item1.SetArcsIn(arcsIn.Where(i => i.TransitionTo == t.Item1).ToArray());
                t.Item1.SetArcsOut(arcsOut.Where(i => i.TransitionFrom == t.Item1).ToArray());
            }

            this.SetPlaces(places.Select(i => i.Item1).ToArray());
            this.SetTransitions(transitions.Select(i => i.Item1).ToArray());
        }

        public static PetriNet CreateTest()
        {
            PetriNet pNet = new PetriNet();

            PetriPlace p1 = new PetriPlace() { Tokens = 2, RealTokens = 2, Label = "Start" };
            PetriPlace p2 = new PetriPlace() { Tokens = 0, RealTokens = 0, Label = "CenterUp" };
            PetriPlace p3 = new PetriPlace() { Tokens = 0, RealTokens = 0, Label = "CenterDown" };
            PetriPlace p4 = new PetriPlace() { Tokens = 0, RealTokens = 0, Label = "Exit" };
            PetriTransition t1 = new PetriTransition() { TransitionTime = 3, Label = "T_CenterUp" };
            PetriTransition t2 = new PetriTransition() { TransitionTime = 1, Label = "T_CenterDown" };
            PetriTransition t3 = new PetriTransition() { TransitionTime = 1, Label = "T_Exit" };
            PetriTransition t0 = new PetriTransition() { TransitionTime = 1, Label = "T_UpperCenter" };
            ArcIn ai1 = new ArcIn() { Weight = 1, PlaceFrom = p1, TransitionTo = t1 };
            ArcIn ai2 = new ArcIn() { Weight = 1, PlaceFrom = p1, TransitionTo = t2 };
            ArcIn ai3 = new ArcIn() { Weight = 1, PlaceFrom = p2, TransitionTo = t3 };
            ArcIn ai4 = new ArcIn() { Weight = 1, PlaceFrom = p3, TransitionTo = t3 };
            ArcIn ai0 = new ArcIn() { Weight = 1, PlaceFrom = p1, TransitionTo = t0 };
```

```
            ArcOut ao1 = new ArcOut() { Weight = 1, TransitionFrom = t1, PlaceTo = p2 };
            ArcOut ao2 = new ArcOut() { Weight = 1, TransitionFrom = t2, PlaceTo = p3 };
            ArcOut ao3 = new ArcOut() { Weight = 3, TransitionFrom = t3, PlaceTo = p4 };
            ArcOut ao0 = new ArcOut() { Weight = 1, TransitionFrom = t0, PlaceTo = p2 };

            t1.SetArcsIn(ai1);
            t1.SetArcsOut(ao1);

            t2.SetArcsIn(ai2);
            t2.SetArcsOut(ao2);

            t3.SetArcsIn(ai3, ai4);
            t3.SetArcsOut(ao3);

            t0.SetArcsIn(ai0);
            t0.SetArcsOut(ao0);

            pNet.SetPlaces(p1, p2, p3, p4);
            pNet.SetTransitions(t1, t2, t3, t0);

            return pNet;
        }

        #region Events
        public enum TransitionEvent
        {
            TransitionEnter,
            TransitionExit
        }
        public void SubscribeEvent(string transitionLabel, Action action, TransitionEvent
transitionEvent)
        {
            foreach (var t in transitions)
            {
                if (t.Label == transitionLabel)
                {
                    switch (transitionEvent)
                    {
                        case TransitionEvent.TransitionEnter:
                            t.transitionEnter += action;
                            break;
                        case TransitionEvent.TransitionExit:
                            t.transitionExit += action;
                            break;
                        default:
                            break;
                    }
                    break;
                }
```

```
            }
        }
        #endregion

        public int GetTokens(string placeLabel, bool real = false)
        {
            foreach (var p in places)
            {
                if (p.Label == placeLabel)
                {
                    return real ? p.RealTokens : p.Tokens;
                }
            }

            throw new ArgumentException("Unexisting label: " + placeLabel);
        }

        public float GetAverageTokens(string placeLabel)
        {
            foreach (var p in places)
            {
                if (p.Label == placeLabel)
                {
                    return p.AverageTokens;
                }
            }

            throw new ArgumentException("Unexisting label: " + placeLabel);
        }

        public float GetUseCoefficient(string transitionLabel)
        {
            foreach (var t in transitions)
            {
                if (t.Label == transitionLabel)
                {
                    return t.UseCoefficient;
                }
            }

            throw new ArgumentException("Unexisting label: " + transitionLabel);
        }
    }
}
```

PetriTransition.cs

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace PetriNetLight
{
   public class PetriTransition
   {
      private ArcIn[] arcsIn;
      private ArcOut[] arcsOut;

      public string Label { get; set; }
      public float TransitionTime { get; set; }
      public float TransitionProbability { get; set; }
      public int TransitionPriority { get; set; }

      public bool IsProccesing { get; private set; }

      public Action transitionEnter;
      public Action transitionExit;

      // Statistics
      private float proccesingCumulate = 0;
      private int statsCheckCount = 1;
      public float UseCoefficient { get; private set; }



      public void SetArcsIn(params ArcIn[] arcsIn)
      {
         this.arcsIn = arcsIn;
      }

      public void SetArcsOut(params ArcOut[] arcsOut)
      {
         this.arcsOut = arcsOut;
      }
      public bool TryTransition()
      {
         if (arcsIn.Length == 0)
            return false;

         bool check = true;

         foreach (var aIn in arcsIn)
         {
            if (aIn.PlaceFrom.Tokens < aIn.Weight)
            {
```

```
                    check = false;
                    break;
                }
            }

            return check;
        }

        public void TransitFrom(float transitionTime)
        {
            foreach (var aIn in arcsIn)
            {
                aIn.PlaceFrom.SetTokens(aIn.PlaceFrom.Tokens - aIn.Weight, transitionTime);
            }

            IsProccesing = true;
            transitionEnter?.Invoke();
        }

        public void TransitTo(float transitionTime)
        {
            foreach (var aIn in arcsIn)
            {
                aIn.PlaceFrom.RealTokens -= aIn.Weight;
            }

            foreach (var aOut in arcsOut)
            {
                aOut.PlaceTo.SetTokens(aOut.PlaceTo.Tokens + aOut.Weight,
transitionTime);
            }

            foreach (var aOut in arcsOut)
            {
                aOut.PlaceTo.RealTokens += aOut.Weight;
            }

            IsProccesing = false;
            transitionExit?.Invoke();
        }

        public void DoStatistic()
        {
            proccesingCumulate += IsProccesing ? 1 : 0;
            statsCheckCount++;
            UseCoefficient = proccesingCumulate / statsCheckCount;
        }
    }
}
```

Додаток А

Тексти програмного коду

Бібліотека Петрі-об'єктного моделювання систем із використанням 3D анімації

(Найменування програми (документа))

DVD-R

(Вид носія даних)

10 арк, 42 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проекту	**В.о. завідувача кафедри**
_____ *І.В. Стеценко*	_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)	(підпис) (ініціали, прізвище)
“16” квітня 2019 р.	“17” квітня 2019 р.

Петрі-об’єктне моделювання систем із використанням 3D-анімації

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП ІС-5217.1181-с.ТЗ*

на 9 сторінках

Київ – 2019 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: *Петрі-об'єктне моделювання систем із використанням 3D-анімації*

1.2 Найменування організації-замовника та організації-учасника робіт

Генеральним замовником проекту являється кафедра Автоматизованих систем обробки інформації та управління НТУУ "КПІ". Представником замовника є Стеценко Інна Вячеславівна.

Розробником системи є студент групи ІС-52 факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського» Мельничук Володимир Ігорович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над Петрі-об’єктним моделюванням систем з використанням 3D-анімації – 5 лютого 2019 рік.

Плановий термін по закінченню роботи над Петрі-об’єктним моделюванням систем з використанням 3D-анімації – не пізніше 1 червня 2019 року.

2 ПРИЗНАЧЕННЯ І МЕТА СТВОРЕННЯ КОМПЛЕКСУ ЗАДАЧ

2.1 Призначення комплексу задач

Призначенням розробки є дослідження і аналіз процесу використання 3D-анімації для Петрі-об’єктного моделювання систем.

2.2 Цілі створення комплексу задач

Цілями розробки системи є:

- прискорення процесу моделювання систем;
- створення інструментарію, який можуть освоїти програмісти не знайомі з теоретичною базою систем імітації;

Для досягнення поставлених цілей необхідно реалізувати наступні задачі.

Задачами розробки є:

- створити редактор моделей, для швидкого моделювання.
- створити набір інструментів, які будуть необхідні для моделювання основних бізнес-процесів, там можливість їх розширювати за необхідності;
- збирання статистики за час моделювання системи;
- інтеграція математичного апарату мереж Петрі, як основу для імітаційного моделювання

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Працювати з системою можуть всі користувачі Unity3D. З її допомогою користувач зможе легко та швидко створити Петрі-об'єктну модель, та прив'язати її до 3D-сцени з анімаціями.

Об'єктом автоматизації є прив'язка 3D-анімацій до Петрі-об'єктних моделей.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Система має виконувати наступні функції:

1) Створення та редагування моделі у редакторі мереж.
2) Зв’язування подій на моделі з 3D-анімацією.
3) Отримання статистики за змодельований час;

4.2 Вимоги до надійності

Програма повинна зберігати працездатність і забезпечувати відновлення своїх функцій при виникненні наступних позаштатних ситуацій:

– при помилках в роботі апаратних засобів (крім носіїв даних і програм).

Продукт повинен поєднувати надійність і функціональність. У разі виникнення аварійних ситуацій необхідно сповіщати користувача та надавати інструкцію для подальших дій. Будь-які аварійні ситуації мають бути задокументовані у звіті, який при необхідності надсилається розробнику для визначення причини збою в роботі та усуненні помилок, які могли привести до нестабільної роботи програмного продукту.

4.3 Вимоги до складу і параметрів технічних засобів

Потрібно мати встановлену останню версію рушію Unity 2018.3, або вище, а також інтегроване середовище розробки Visual Studio, Mono, або інше на вибір. Для нормальної роботи з Unity потрібно мати комп’ютер як мінімум такої конфігурації:

- процесор з набором інструкцій SSE2;
- графічна карта з підтримкою DirectX 9 (shader model 3.0);
- операційна система: Windows 7 SP1+, 8, 10, 64-bit; Mac OS X 10.9+.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з Петрі-об’єктного моделювання систем із використанням 3D-анімації.

Таблиця 5.1 – Стадії та етапи розробки

Та№ п/п	**Назва етапу роботи**	**Термін виконання етапу**	**Результат виконання**
1.	Підготовка технічного завдання на розробку програмного продукту	07.02.2019	
2.	Розробка сценарію роботи	12.02.2019	
3.	Технічне проектування – функціональність, модулі, задачі, цілі тощо	20.02.2019	
4.	Узгодження з керівником інтерфейсу користувача	02.03.2019	
5.	Розробка інформаційного забезпечення	10.03.2019	
6.	Розробка програмного забезпечення	10.04.2019	
7.	Налагодження програми	20.04.2019	
8.	Тестування програми	16.05.2019	
9.	Здача готового програмного продукту замовнику	30.05.2019	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1 Види випробувань

Види, склад, об’єм і методи випробувань програмного продукту повинні бути викладені в програмі і методиці випробувань системи, що розробляється в складі робочої документації.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО

Керівник проекту

_____ *І.В. Стеценко*
(підпис) (ініціали, прізвище)

“13” травня 2019 р.

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)

“14” травня 2019 р.

Петрі-об'єктне моделювання систем із використанням 3D-анімації

ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

Шифр *ДП ІС-5217.1181-с.ПМВ*

на 10 сторінках

Київ – 2019 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАННЯ

1.1 Найменування програми

Повне найменування програми: Петрі-об’єктне моделювання із використанням 3D-анімації.

1.2 Область застосування

Функціонал призначений для створення та редагування Петрі-об’єктних моделей загального характеру та їх подальшої імітації з використанням 3D-анімації.

1.3 Умовне позначення програми

Умовне позначення: «PetriObjModel3D».

2 МЕТА ВИПРОБУВАНЬ

Метою випробувань є перевірка відповідності анімацій на 3D-сцені подіям, які імітуватиме Петрі-об’єктна модель.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Функціональні вимоги:

- створення та редагування Петрі-об’єктних моделей у редакторі мереж;
- прив’язка об’єктів 3D-сцени до Петрі-об’єктів;
- проведення імітаційного моделювання на 3D-сцені;
- отримання звіту з моделювання;
- використання компонентів для задання поведінки 3D-акторів

3.1.1 Вимоги до складу виконуваних функцій

Функціонал повинен забезпечувати можливість виконання перерахованих нижче функцій:

- створення та редагування Петрі-об’єктних мереж;
- використання базових компонентів для задання поведінки акторів та можливість їх розширення;
- запуск моделювання та отримання звіту;

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація має складатися з керівництва користувача та вихідних текстів програмного коду.

5 СКЛАД І ПОРЯДОК ВИПРОБУВАНЬ

Етапи випробувань:

- ознайомчий;
- виконавчий.

На ознайомчому етапі проводиться:

- перевірка комплектності програмної документації;
- перевірка комплектності складу технічних і програмних засобів.

Під час виконавчого етапу проводиться:

- перевірка відповідності технічних характеристик системи;
- перевірка ступеню виконання вимог функціонального призначення системи.

Функції, що підлягають перевірці:

- створення та редагування мереж;
- проведення імітаційного моделювання;
- прив’язка поведінки акторів до Петрі-об’єктів;
- отримання звіту з моделювання.

6 МЕТОДИ ВИПРОБУВАНЬ

У процесі тестування була перевірена основна функціональність засобу для моніторингу та прогнозування фінансових показників. У таблицях 6.1 – 6.5 наведено перелік випробувань основних функціональних можливостей.

Таблиця 6.1 – Редагування об’єктної моделі графічно

Мета тесту	**Перевірка функції «Редагування об’єктної моделі»**
Початковий стан моделі	Створена модель в якій є одна позиція і один перехід
Вхідні дані:	xml файл моделі
Схема проведення тесту:	Протягнути мишкою з вихідного порта позиції до вхідного порту переходу
Очікуваний результат	Появилась на екрані дуга яка веде з позиції до переходу
Стан моделі після проведення випробувань:	Над дугою появилась цифра «1», що означає стандартну вагу дуги

Таблиця 6.2 – Симуляція системи на незначний проміжок часу

Мета тесту	**Перевірка функції «Імітації системи»**
Початковий стан моделі	Створена симуляція з усіма подіями, яка готова до запуску
Вхідні дані:	xml файл моделі та сцена з об’єктами
Схема проведення тесту:	Запуск симуляції на дуже маленький

	період часу, з який жодна із подій не мали настати.

Продовження таблиці 6.2

Очікуваний результат	Запустяться тільки переходи з нульовим входженням
Стан моделі після проведення випробувань:	Запустяться тільки переходи з нульовим входженням

Таблиця 6.3 – Створення нової моделі та введення даних

Мета тесту	**Перевірка функції «Створення нової моделі та введення даних»**
Початковий стан моделі	Відкрите вікно редагування моделі
Вхідні дані:	-
Схема проведення тесту:	Натиснути кнопку «Створити модель», натиснути праву кнопку миші і в контекстному меню вибрати «Перехід». В полі «Час переходу» написати число
Очікуваний результат	На екрані намалюється перехід з введеним часом
Стан моделі після проведення випробувань:	На екрані намалюється перехід з введеним часом

Таблиця 6.4 – Завершення симуляції

Мета тесту	**Перевірка функції «Завершення симуляції»**
Початковий стан моделі	Запущена симуляція
Вхідні дані:	xml файл моделі та сцена з

	об’єктами
Схема проведення тесту:	Натиснути кнопку «Завершити симуляцію»

Продовження таблиці 6.4

Очікуваний результат	Симуляція завершується без жодних помилок
Стан моделі після проведення випробувань:	На екран виводиться журнал симуляції

Таблиця 6.5 – Симуляція системи на значний проміжок часу

Мета тесту	**Перевірка функції «Запуск роботи алгоритму»**
Початковий стан моделі	Запущена симуляція
Вхідні дані:	xml файл моделі та сцена з об’єктами
Схема проведення тесту:	Просунути модель на великий проміжок часу, під час якого мали б відбутися сотні подій
Очікуваний результат	Модель успішно згенерує всі події і просунеться на час вказаний програмістом
Стан моделі після проведення випробувань:	Появиться вікно з індикатором загрузки і обробки даних

Графічний матеріал
до дипломного проекту

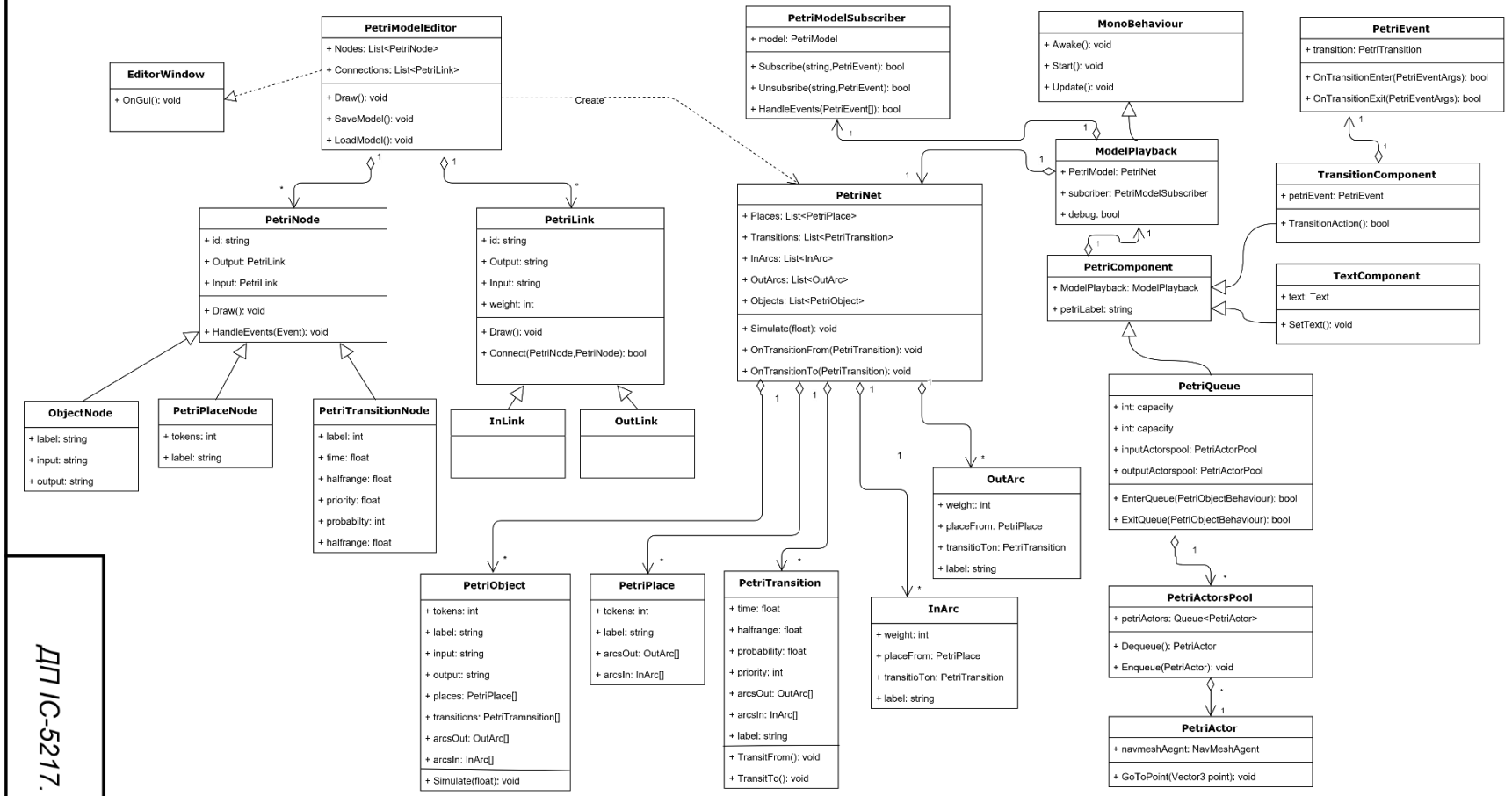
на тему: Петрі-об’єктне моделювання систем із використанням 3D-анімації

Київ – 2019 року

ДП ІС-5217.1181-с.ССВ

					ДП ІС-5217.1181-с.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера / Маса / Масштаб
Розробив		Мельничук В.І.				
Перевірив		Стеценко І.В.				Аркуш 1 / Аркушів 1
Т. кон.					Петрі-об'єктне моделювання систем із використанням 3D-анімації	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52
Н. кон.		Халус О.А.				
Затвердив		Стеценко І.В.				

					ДП ІС-5217.1181-с.ССД	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Мельничук В.І.			Схема структурна діяльності	Літера / Маса / Масштаб
						Аркуш 1 / Аркушів 1
Перевірив		Стеценко І.В.				
Т. кон.					Петрі-об'єктне моделювання систем із використанням 3D-анімації	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52
Н. кон.		Халус О.А.				
Затвердив		Стеценко І.В.				



ДП ІС-5217.1181-с.ССК

ДП ІС-5217.1181-с.ССК

Схема структурна класів програмного забезпечення

Петрі-об'єктне моделювання систем із використанням 3D-анімації

КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52

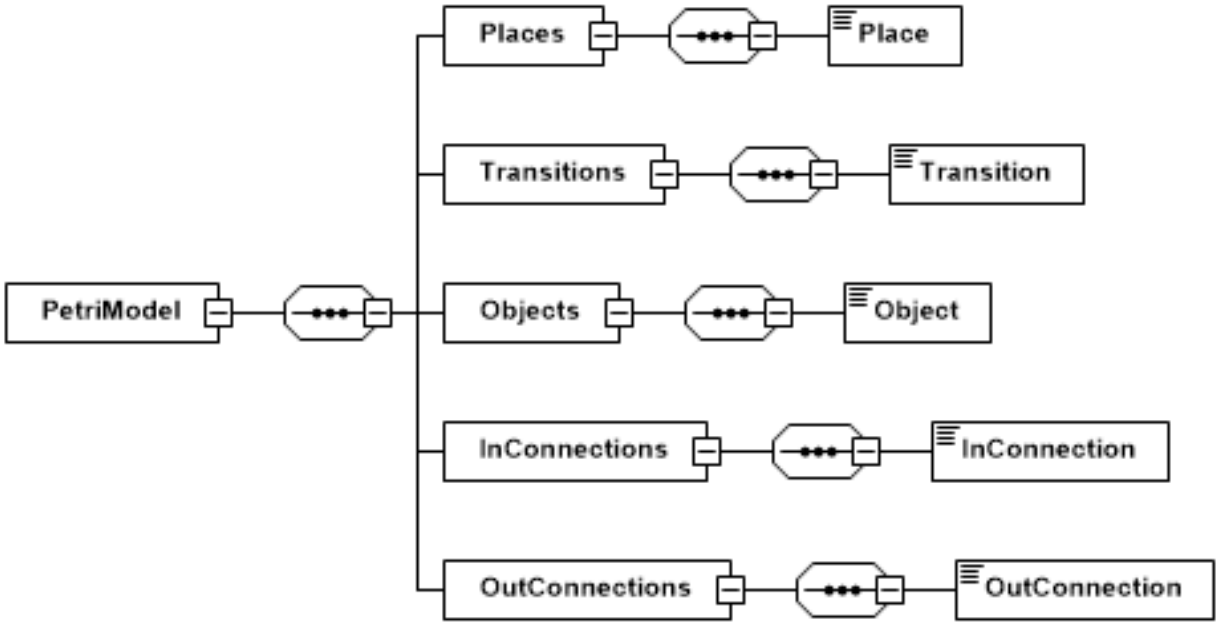
Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

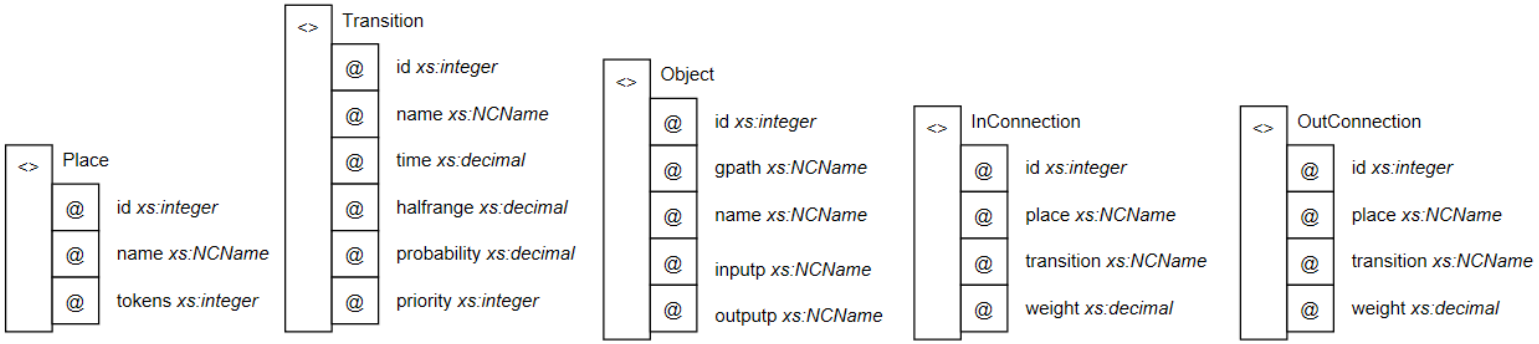
Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Мельничук В.І.		
Перевірив		Стеценко І.В.		
Т. кон.				
Н. кон.		Халус О.А.		
Затвердив		Стеценко І.В.		

					ДП ІС-5217.1181-с.ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Мельничук В.І.					
						Аркуш 1	Аркушів 1
Перевірив		Стеценко І.В.					
Т. кон.					Петрі-об'єктне моделювання систем із використанням 3D-анімації	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52	
Н. кон.		Халус О.А.					
Затвердив		Стеценко І.В.					

ДП ІС-5217.1181-с.ССП

					ДП ІС-5217.1181-с.ССП	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності	Літера / Маса / Масштаб
Розробив		Мельничук В.І.				
Перевірив		Стеценко І.В.				Аркуш 1 / Аркушів 1
Т. кон.					Петрі-об'єктне моделювання систем із використанням 3D-анімації	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52
Н. кон.		Халус О.А.				
Затвердив		Стеценко І.В.				





ДП ІС-5217.1181-с.СМІ

ДП ІС-5217.1181-с.СМІ

Схема масивів інформації

Петрі-об'єктне моделювання систем із використанням 3D-анімації

КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52

Літера	Маса	Масштаб

Аркуш 1 Аркушів 1

Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Мельничук В.І.		
Перевірив		Стеценко І.В.		
Т. кон.				
Н. кон.		Халус О.А.		
Затвердив		Стеценко І.В.		