

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»  
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту  
«Моделювання систем. Курсова робота»

Тема: Імітаційна модель системи аеропорту у формалізмі мереж Петрі

**Керівник:**  
ст. вик. Дифучин А. Ю.  
  
«Допущено до захисту»

«\_\_\_\_» \_\_\_\_\_ 2025 р.

Захищено з оцінкою

Члени комісії:

---

---

**Виконавець:**  
Панасюк О. Ю.  
студент групи ІП - 22  
залікова книжка № ІП-2220

«26» листопада 2025 р.

Інна СТЕЦЕНКО

Антон ДИФУЧИН

Київ – 2025

## ЗАВДАННЯ

Виконати дослідження для такої системи:

Потік літаків, що потребують посадки в аеропорті, пуассонівський з інтенсивністю 10 літаків за годину. В аеропорту є дві посадкові смуги. Літак, зробивши посадку на смугу, звільняє її через 35 хвилин. Якщо літак потребує посадки, коли усі смуги зайняті, він очікує на посадку. Якщо очікування на посадку триває більше ніж  $70 \pm 10$  хвилин, то виникає необхідність дозаправки літака, на що аеропорту доведеться витратити  $1000 \pm 200$  грошових одиниць. Після 140 хвилин очікування літак відправляється на посадку в інший аеропорт. За кожний літак, що здійснив посадку без очікування, аеропорт має прибуток 2000 гривень, а за кожний, що здійснив посадку з очікуванням -  $1500 \pm 100$  гривень. Витрати на будівництво додаткової посадкової смуги складають 3000000 гривень. В аеропорту не може бути побудовано більше ніж 10 додаткових смуг. Метою моделювання є визначення кількості посадкових смуг, при якій інтервал часу їх окупності буде мінімальним, якщо час окупності додаткових посадкових смуг складає:  $T = S / p$ , де  $S$  – витрати на побудову смуг,  $p$  - прибуток за один рік.

Для вищезазначеної системи:

- 1) розробити опис концептуальної моделі системи;
- 2) виконати формалізацію опису об'єкта моделювання в термінах визначеного у завданні формалізму;
- 3) розробити алгоритм імітації моделі дискретно-подійної системи у відповідності до побудованого формального опису;
- 4) для доведення коректності побудованого алгоритму виконати верифікацію алгоритму імітації;
- 5) визначити статистичні оцінки заданих характеристик моделі, що є метою моделювання;
- 6) провести експериментальне дослідження моделі;
- 7) інтерпретувати результати моделювання та сформулювати пропозиції щодо поліпшення функціонування системи;

- 8) зробити висновки щодо складності розробки моделі та алгоритму імітації на основі використаного формалізму, отриманих результатів моделювання та їх корисності.

## АНОТАЦІЯ

Панасюк Олександр Юрійович. Курсова робота на тему «Імітаційна модель системи аеропорту у формалізмі мереж Петрі». Текстова частина курсової роботи складається із вступу, 5 розділів, висновків, списку використаних джерел з 5 найменувань та 2 додатків, які викладено на 45 сторінках. В тексті роботи оформлено 8 рисунків, 3 таблиць.

У роботі поставлено завдання розробити та дослідити імітаційну модель системи аеропорту з метою визначення оптимальної кількості додаткових посадкових смуг, яка забезпечить мінімальний час окупності капіталовкладень

Імітаційна модель розроблена у формалізмі мереж Петрі з розширеними механізмами, що включають типи маркерів для фіксації фінансових показників та варійовану кратність дуги для моделювання складної логіки взаємодії ресурсів.

Проведено дослідження впливу кількості посадкових смуг на час окупності інвестицій, для чого було застосовано метод Фібоначчі на дискретній множині значень.

Результати моделювання свідчать, що мінімальний час окупності досягається при загальній кількості 3 посадкових смуги, що означає економічну доцільність будівництва однієї додаткової смуги

Зроблено висновки, що імітаційне моделювання системи є успішним, оскільки поставлена оптимізаційна задача вирішена.

Ключові слова: імітаційна модель, алгоритм імітації, мережа Петрі, час окупності, оптимізація, метод Фібоначчі, C#.

**ЗМІСТ**

Вступ.....	6
1 Концептуальна модель системи.....	7
1.1 Текстовий опис моделі системи.....	7
1.2 Опис параметрів, вхідних та вихідних змінних.....	7
1.3 Схематичне представлення процесу.....	8
2 Формалізована модель системи.....	11
3 Алгоритмізація та програмна реалізація імітаційної моделі системи.....	15
3.1 Вибір мови програмування та бібліотек.....	15
3.2 Розробка алгоритму імітації.....	15
3.3 Програмна реалізація.....	19
3.4 Верифікація роботи алгоритму імітації.....	21
4 Експериментальне дослідження моделі.....	23
5 Інтерпретація результатів моделювання.....	26
Висновки.....	27
Список використаних джерел.....	28
Додаток А. Лістинг коду алгоритму імітації.....	29
Додаток Б. Лістинг коду для проведення експериментів.....	44

## ВСТУП

Моделювання складних систем є одним із найважливіших напрямів сучасної прикладної інформатики, оскільки дозволяє досліджувати поведінку об'єктів і процесів без необхідності втручання в реальні системи. Особливе значення такі методи мають у сфері авіаційного транспорту, де ефективність функціонування аеропортів безпосередньо впливає на економічні показники, безпеку польотів та комфорт пасажирів. Одним із ключових завдань управління роботою аеропорту є оптимізація процесу посадки літаків, адже саме від правильного розподілу ресурсів, зокрема кількості посадкових смуг, залежить швидкість обслуговування повітряних суден, рівень затримок, а також загальні прибутки або втрати підприємства.

Зростання інтенсивності авіаперевезень вимагає від аеропортів підвищення ефективності використання своїх потужностей. Неефективне планування кількості посадкових смуг може привести як до простоїв та фінансових збитків, так і до перевитрат на будівництво зайвої інфраструктури. Для оптимізації таких комплексних систем доцільно застосувати мережі Петрі, який дозволяє доволі точно описати динаміку системи та взаємодію між потрібними її елементами.

У даній роботі розглядається процес функціонування аеропорту за допомогою моделювання у формалізмі мереж Петрі. Основна увага приділяється визначенню такої кількості посадкових смуг, за якої досягається найвища ефективність роботи аеропорту та мінімальний період окупності витрат на їх будівництво. Результати дослідження можуть бути використані для вдосконалення системи управління аеропортами, підвищення економічної ефективності їх діяльності та оптимізації використання ресурсів у транспортній галузі.

## 1 КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ

### 1.1 Текстовий опис моделі системи

Потік літаків, що потребують посадки в аеропорті, пуассонівський з інтенсивністю 10 літаків за годину. В аеропорту є дві посадкові смуги. Літак, зробивши посадку на смугу, звільняє її через 35 хвилин. Якщо літак потребує посадки, коли усі смуги зайняті, він очікує на посадку. Якщо очікування на посадку триває більше ніж  $70 \pm 10$  хвилин, то виникає необхідність дозаправки літака, на що аеропорту доведеться витратити  $1000 \pm 200$  грошових одиниць. Після 140 хвилин очікування літак відправляється на посадку в інший аеропорт. За кожний літак, що здійснив посадку без очікування, аеропорт має прибуток 2000 грошових одиниць, а за кожний, що здійснив посадку з очікуванням -  $1500 \pm 100$  грошових одиниць. Витрати на будівництво додаткової посадкової смуги складають 3000000 грошових одиниць. В аеропорту не може бути побудовано більше ніж 10 додаткових смуг.

Метою моделювання є визначення кількості посадкових смуг, при якій інтервал часу їх окупності буде мінімальним.

### 1.2 Опис параметрів, вхідних та вихідних змінних

На основі складеного текстового опису системи виділимо параметри системи та запишемо їх числові значення.

Таблиця 1.1 – Параметри моделі системи

Параметр	Чисрова характеристика
Інтенсивність надходження літаків	10 літаків в годину
Час перебування літака на смузі	35 хв
Максимальний час очікування без дозаправки літака	$70 \pm 10$ хв
Максимальний час очікування звільнення смуги	140 хв
Вартість дозаправки	$1000 \pm 200$ грошових одиниць
Прибуток за посадку літака без очікування	2000 грошових одиниць

### Продовження таблиці 1.1

Параметр	Числова характеристика
Прибуток за посадку літака з очікуванням	$1500 \pm 100$ грошових одиниць
Витрати на будівництво додаткової посадкової смуги	3000000 грошових одиниць
Максимальна кількість смуг	10 смуг

Далі наведемо вхідні та вихідні змінні, що будуть задовольняти мету моделювання системи.

Кількість додаткових посадкових смуг – вхідна зміна, що відповідає кількості додатково побудованих смуг в аеропорті, на які можуть здійснювати посадку літаки. Змінна має діапазон варіювання від 1 до 10.

Витрати на дозаправку – вихідна змінна, що означає кількість грошових одиниць витрачених на дозаправку літаків.

Дохід аеропорту – вихідна змінна, що описує кількість грошових одиниць, які отримав аеропорт за посадки літаків

Прибуток аеропорту – вихідна змінна, що відображає різницю між отриманим доходом від посадок літаків та витратами на дозаправку.

Час окупності побудованих смуг – вихідна змінна, що відповідає інтервалу часу, який потрібен, щоб отримати прибуток рівний витратам на будівництво додаткових смуг. Для спостереження даної змінної потрібно знати витрати на будівництво смуг та прибуток аеропорту за один рік. Обчислити змінну можна за наступною формулою.

$$T = \frac{S}{p}, p > 0$$

де  $S$  – витрати на побудову смуг,  $p$  - прибуток за один рік. У випадку якщо прибуток від'ємний або 0, будем вважати що  $T = \infty$

### 1.3 Схематичне представлення процесу

Для опису процесу посадки літака у аеропорті схематично зобразимо послідовність подій у вигляді діаграми BPMN<sup>[2]</sup> (рис 1.1).

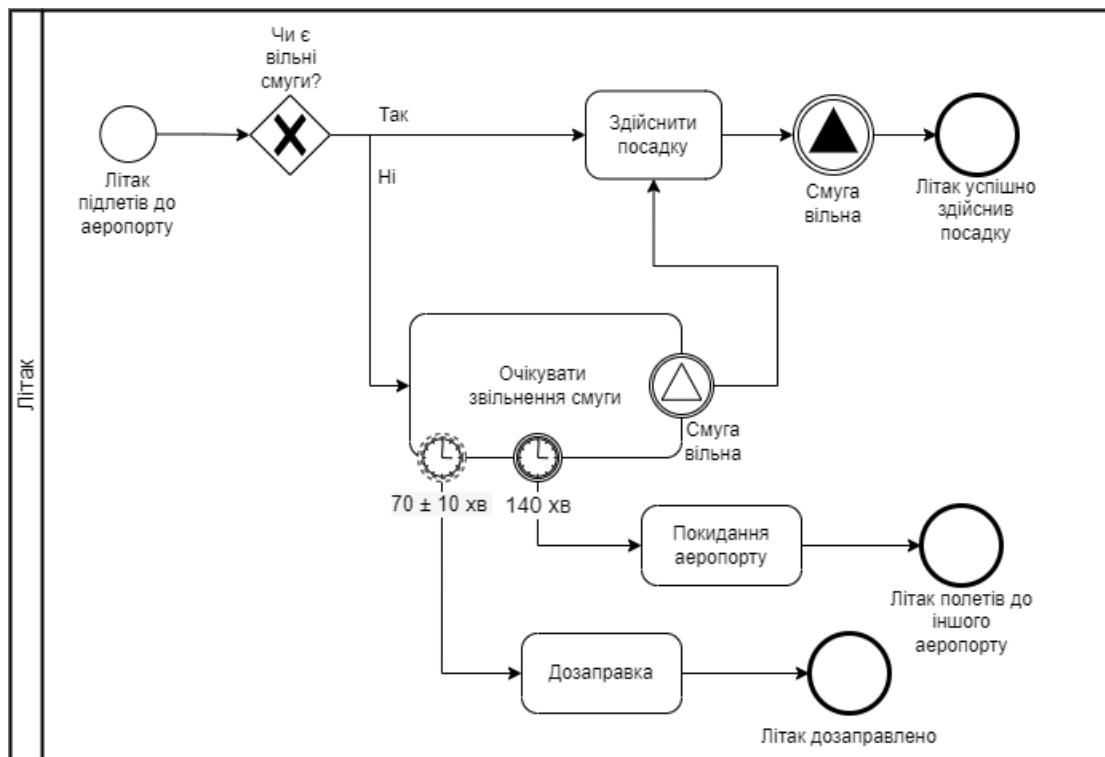


Рисунок 1.1 – BPMN діаграма процесу посадки літака.

Для кращого розуміння діаграми бізнес процесу опишемо два сценарії даної системи.

#### Сценарій 1. Успішна посадка без очікування

- 1) Літак піdlітає до аеропорту;
- 2) Система перевіряє, чи є вільна посадкова смуга;
- 3) Якщо хоча б одна смуга вільна, літак одразу отримує дозвіл на посадку;
- 4) Літак здійснює посадку;
- 5) Після завершення посадки система сигналізує про звільнення смуги;
- 6) Процес завершується подією “Літак успішно здійснив посадку”.

#### Сценарій 2. Посадка після очікування

- 1) Літак піdlітає до аеропорту, але всі посадкові смуги зайняті;
- 2) Літак переходить у стан очікування звільнення смуги;

- 3) Якщо очікування триває більше  $70 \pm 10$  хвилин, виникає необхідність у дозаправці літака, що виконується паралельно з подальшим очікуванням;
- 4) Якщо протягом 140 хвилин смуга так і не звільняється, літак залишає аеропорт і вирушає до іншого;
- 5) Якщо смуга звільняється раніше, літак переходить до етапу посадки;
- 6) Після завершення посадки система сигналізує про звільнення смуги;
- 7) Процес завершується подією “Літак успішно здійснив посадку”.

Таким чином, модель відображає два основні сценарії функціонування аеропорту: оперативну посадку без затримок та посадку після очікування, що може супроводжуватись додатковими витратами або навіть втратами рейсу. Це дозволяє проаналізувати вплив пропускної здатності аеропорту на фінансові результати та визначити оптимальну кількість посадкових смуг для досягнення найменшого часу окупності.

## 2 ФОРМАЛІЗОВАНА МОДЕЛЬ СИСТЕМИ

На основі концептуальної моделі складемо модель у формалізмі мереж Петрі. Для цього спочатку кожної події системи поставимо у відповідність перехід мережі Петрі. У системі аеропорту маємо наступні події:

- “Надходження літака до аеропорту”;
- “Посадка”;
- “Звільнення смуги”;
- “Спрацювання таймера дозаправки”;
- “Скидання таймера дозаправки”;
- “Дозаправка”;
- “Спрацювання таймера залишення аеропорту”;
- “Скидання таймера залишення аеропорту”;
- “Залишення аеропорту”.

Також виділимо два типи маркерів, що відповідають літакам. Це необхідно для коректного обрахування прибутку. Мережа має наступні типи маркерів.

- Без очікування;
- З очікуванням.

Далі складемо мережу Петрі для опису всіх взаємодій у системі. На рисунку 2.1 наведена розроблена мережа. З метою спрощення графічного представлення частину назв подій і переходів не буде наведено.

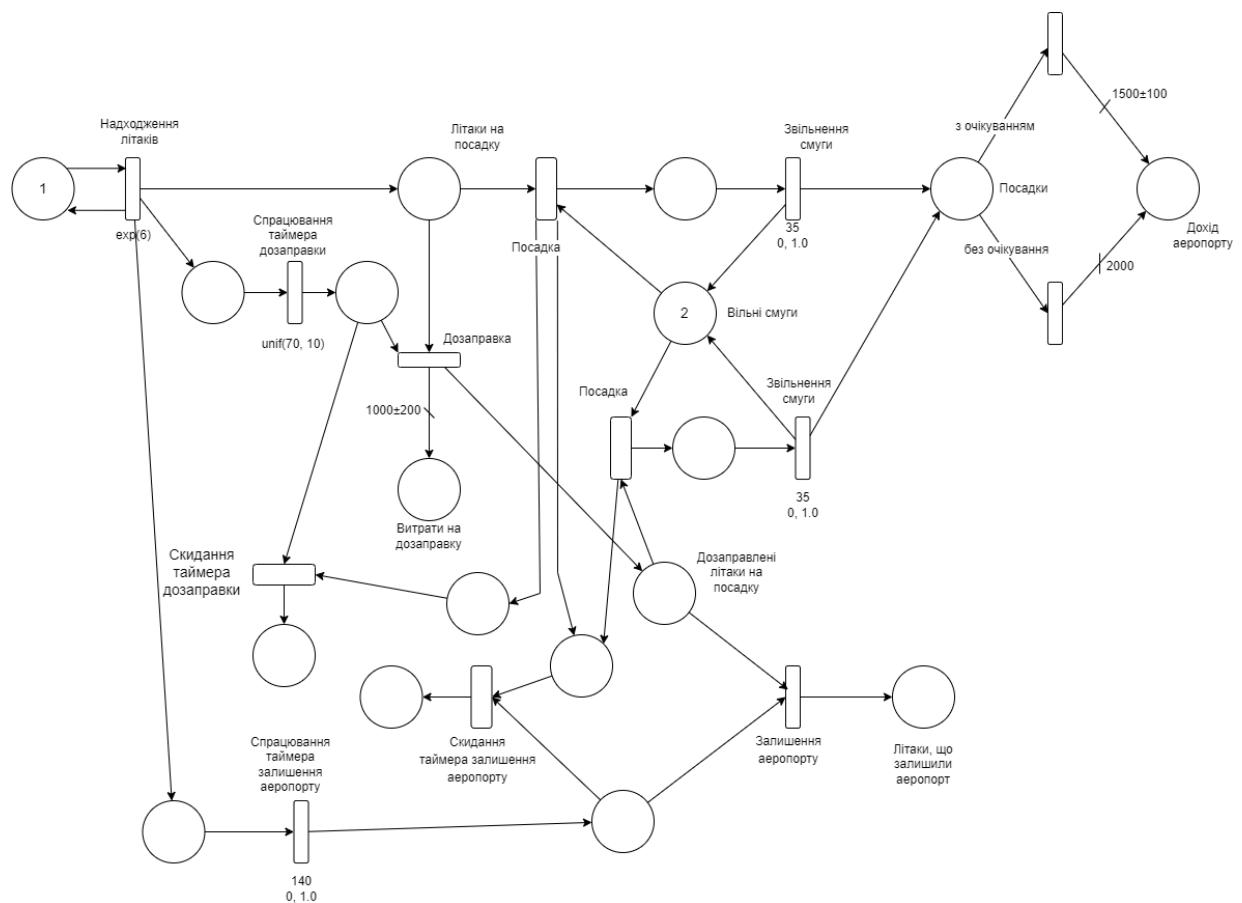


Рисунок 2.1 – мережа Петрі, що формально представляє процес посадки літаків в аеропорту

Для кожного переходу визначені параметри: часова затримка, значення пріоритету та значення ймовірності запуску (таблиця 2.1)

Таблиця 2.1 – Параметри переходів мережі Петрі

Назва переходу	Часова затримка	Значення пріоритету	Ймовірність запуску
Надходження літаків	$\text{exp}(6)$	0	1.0
Посадка	0	1	1.0
Звільнення смуги	35	0	1.0
Спрацювання таймера дозаправки	$\text{unif}(70, 10)$	0	1.0
Дозаправка	0	0	1.0

Продовження таблиці 2.1

Скидання таймера дозаправки	0	1	1.0
Спрацювання таймера залишення аеропорту	140	0	1.0
Залишення аеропорту	0	0	1.0
Скидання таймера залишення аеропорту	0	1	1.0

Робота системи розпочинається із спрацювання переходу “Надходження літаків”, який додає маркер літака у позицію “Літаки на посадку” та додає маркери в позиції для старту таймерів. Якщо є маркери у позиції “Вільні смуги”, то літак заходить на посадку, займаючи одну вільну смугу. При цьому після спрацювання переходу “Посадка” надходять маркери для скидання таймерів. Після посадки з часовою затримкою 35 хвилин виконується перехід “Звільнення смуги”, який повертає маркер у позицію “Вільні смуги” та додає маркер у позицію “Посадки”. Якщо маркер таймера дозаправки надійшов швидше, ніж встигла звільнитись смуга, то виконується перехід “Дозаправка”, який нараховує від 800 до 1200 маркерів у позицію “Витрати на дозаправку” та повертає маркер літака до позиції “Дозаправлені літаки на посадку”. З неї вони також можуть зайти на посадку. Якщо після дозаправки смуга не звільнилась та прийшов маркер таймера залишення аеропорту, то виконується перехід “Залишення аеропорту”.

Розмежування черги літаків на дозаправлених та недозаправлених, реалізовано для того щоб, правильно скидати таймери. Оскільки у випадку, коли дозаправлений літак здійснює посадку, то потрібно скинути тільки таймер для очікування залишення аеропорту. У випадку коли літак не мав дозаправки та здійснив посадку потрібно скинути обидва таймери.

Також для правильного обрахунку прибутку аеропорту в мережі реалізоване розгалуження літаків за їх типом. Таким чином з позиції “Посадки” маркери переходят у позицію “Дохід аеропорту” через два переходи. Перший спрацьовує для літаків, що приземлились без очікування, такий переход нараховує 2000 маркерів. Інший переход спрацьовує тільки для літаків, що приземлились з очікуванням, після спрацювання він нараховує від 1400 до 1600 маркерів у вихідну позицію.

Визначення типу маркера відбувається при переході у позицію “Літаки на посадку”. Коли маркер заходить у позицію, у випадку якщо в позиції “Вільні смуги” є хоча б один маркер та черги літаків порожні, тоді маркер що заходить на позицію буде мати тип “Без очікування” у інакшому випадку тип “З очікуванням”.

Вихідні змінні для доходу та витрат на дозаправку будуть відповідати кількості маркерів в однайменних позиціях. Вихідна змінна “Прибуток аеропорту” обчислюється як різниця кількості маркерів в позиції “Дохід аеропорту” та позиції “Витрати на дозаправку”. Час окупності смуг будемо розраховувати наступним чином:

$$T = \frac{3000000 \cdot N}{\max(p, 0)}$$

, де  $N$  – кількість побудованих смуг,  $p$  – річний прибуток аеропорту. Формально застосування функції  $\max$  потрібно, щоб правильно обраховувати час при від'ємних прибутках. В такому разі час окупності буде наблизатись до нескінченності.

### 3 АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ

#### 3.1 Вибір мови програмування та бібліотек

Для реалізації алгоритму імітації у формалізмі мереж Петрі було обрано мову програмування **C#**, оскільки вона поєднує високу продуктивність із зручністю розробки та добре підходить для моделювання складних систем. Мова має розвинену екосистему та надійні інструменти, включно з механізмами роботи з потоками та асинхронними операціями, що є важливим при відтворенні паралельної поведінки мереж Петрі. Крім того, об'єктно-орієнтований підхід C# природно узгоджується зі структурою мереж Петрі, де позиції, переходи та маркування можна подати у вигляді окремих сущностей із чіткими взаємозв'язками<sup>[5]</sup>.

Для генерації випадкових величин, зокрема часових затримок, було використано бібліотеку `MathNet.Numerics`. Вона надає широкий набір готових реалізацій статистичних розподілів та генераторів випадкових чисел, що дозволяє уникнути необхідності власної реалізації цих механізмів. Бібліотека вирізняється надійністю, точною реалізацією чисельних методів та простотою інтеграції в C#-проекти, що робить її оптимальним вибором для побудови стохастичних компонентів у моделі<sup>[4]</sup>.

#### 3.2 Розробка алгоритму імітації

На основі створеної формалізованої моделі створимо алгоритм імітації, що буде точно моделювати задану систему за правилами формалізму мереж Петрі. За основу візьмемо алгоритм стохастичної мережі Петрі з багатоканальними переходами та розв'язуванням конфліктів між переходами. Щоб точно реалізувати імітацію моделі, розширимо алгоритм наступним функціоналом:

- Типи маркерів, визначення типу маркера
- Спрацювання переходу тільки для маркера певного типу
- Варійована кратність дуги

Алгоритм імітації симулює розвиток подій як послідовність запусків переходів. Час настання найближчої події визначається за мінімальним значенням серед усіх моментів виходу маркерів із переходів.

Під час просування моделюального часу до цього найближчого моменту відбувається зміна стану мережі Петрі: здійснюється вихід маркерів із переходу, для якого поточний момент часу збігається з одним із його моментів виходу, а також вхід маркерів у ті переходи, для яких виконується умова їх запуску<sup>[1]</sup>. Алгоритм для просування часу та запуску подій системи можна представити у вигляді псевдокоду наступним чином:

**procedure** Run(tModel):

    t  $\leftarrow$  0

**while** t < tModel:

        tNext  $\leftarrow$  minimum NearestTMoment among all transitions

**if** tNext  $\neq \infty$ :

            transition  $\leftarrow$  the transition whose NearestTMoment = tNext

            perform token exit event for transition

            t  $\leftarrow$  tNext

**else**:

            stop simulation

**endif**

    transitionsForEntrance  $\leftarrow$  all transitions that can be triggered

    randomly shuffle transitionsForEntrance

    prioritizedTransitions  $\leftarrow$  transitionsForEntrance sorted by Priority descending

**while** prioritizedTransitions is not empty:

        transition  $\leftarrow$  first element of prioritizedTransitions

        remove transition from prioritizedTransitions

        perform token entrance event for transition

```

prioritizedTransitions ← all transitions in prioritizedTransitions
    that can be triggered
    sorted by Priority descending
endwhile

```

**endwhile**

Далі розробимо алгоритми для входу маркерів до переходу та їх виходу.

Для спрощення розробки переходи були розділені на два типи:

- **Звичайні** – можуть мати декілька вхідних та вихідних позицій. Кратність дуг завжди рівна одиниці.
- **Переходи для підрахунку вихідних змінних** – завжди мають одну вхідну та одну вихідну позицію, умова спрацювання переходу може включати тип маркера, кратність вихідної дуги може бути або сталою числом або випадковою цілою змінною, не мають часових затримок.

Далі буде наведений псевдокод, що описує вхід маркерів для звичайного переходу:

**procedure** EntryTokens(tCurrent):

**while** CanTrigger:

    tokenType ← None

**for each** place **in** inputPlaces:

        token ← get token from place

**if** place = targetTypePlace **and** tokenType = None:

            tokenType ← token.Type

**endif**

**endfor**

    t ← tCurrent

```

if Generator ≠ null:
    delay ← generate time delay
    t ← t + delay
endif

add (t, tokenType) to tMoments
endwhile

```

Даний алгоритм також має механізм збереження типа маркера, це важливо для переходів, до яких входить маркер, що представляє літак. Для таких переходів є важливим, щоб при переміщенні маркера з одної позиції на іншу тип маркера зберігся. В даному алгоритму targetTokenTypePlace представляє позицію яка буде визначати тип вихідного маркера.

Алгоритм для виходу маркерів із звичайного переходу можна записати у вигляді псевдокоду наступним чином:

```

procedure ExitTokens(t):
    (tMoment, tokenType) ← the element in tMoments where element.time = t
    remove (tMoment, tokenType) from tMoments

```

```

token ← new Token
token.Type ← tokenType

```

```

for each place in outputPlaces:
    add token to place
endfor

```

Далі будуть також наведені алгоритми виходу та входу маркерів для переходу, що обраховує вихідні змінні:

```

procedure EntryTokens(tCurrent):
    while CanTrigger:

```

```

if tokenType ≠ None:
    get token with specified type from input place
else:
    get token from input place
endif

    add tCurrent to tMoments
endwhile

procedure ExitTokens(t):
    remove t from tMoments

    if generator ≠ null:
        tokenCount ← integer part of generated random number
    else:
        tokenCount ← outputMultiplicity
    endif

    for i ← 1 to tokenCount:
        add token to output place
    endfor

```

### 3.3 Програмна реалізація

Оскільки модель розробляється згідно формалізму мереж Петрі, то елементи моделі можна представити у коді у вигляді окремих класів. Діаграма класів системи зображена на рисунку 3.1.

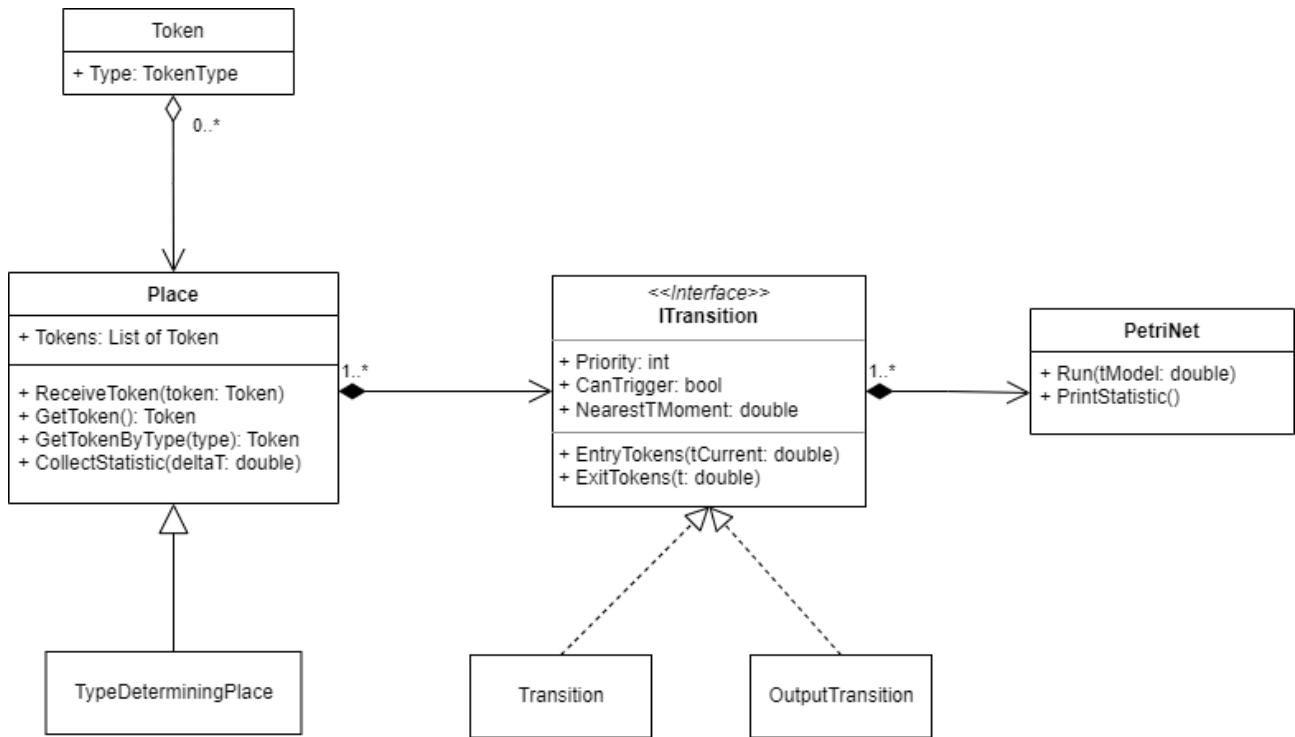


Рисунок 3.1 – Діаграма класів системи

В розробленій системі будуть наявні наступні сутності:

- PetriNet – клас, що відповідає за мережу Петрі;
- ITransition – інтерфейс, що описує як взаємодіяти з переходом мережі Петрі;
- Transition – клас, що реалізує інтерфейс ITransition. Відтворює поведінку звичайного переходу мережі Петрі;
- OutputTransition – клас, що реалізує інтерфейс ITransition. Функціонує як перехід для обрахунку вихідних змінних;
- Place – клас, що відповідає з позицією мережі Петрі;
- TypeDeterminingPlace – клас, що наслідує Place. Також поводиться як позиція, але додатково ще визначає тип маркера, залежно від стану моделі;
- Token – клас, що відповідає маркеру мережі Петрі.

На основі розроблених алгоритмів та моделі класів системи було реалізоване програмне забезпечення для імітації мережі Петрі, що моделює роботу запропонованої системи. Програмний код застосунку наведений у додатку А а також у репозиторії GitHub<sup>[3]</sup>.

Виконаємо тестовий запуск програми. Фрагмент протоколу подій буде наведено на рисунку 3.2, результати моделювання на рисунку 3.3.

```
----- T Next - 795,2213250968904-----
Plane generator: 0
Refueling timer start: 0
Refueling timer reset: 0
Refueling timer end: 0
Reseted refueling timers: 25
Leave timer start: 0
Leave timer end: 0
Leave timer reset: 0
Reseted leave timers: 46
Refuelings: 0
Refueling expenses: 102238
Leaves: 72
Free runways: 0
Plane queue (refueled): 11
Plane queue: 9
Landing 1: 0
Landing 2: 0
Landed planes: 0
Income for landings with waiting: 62846
Income for landings without waiting: 4000
Transition 8 was triggered
-----
```

Рисунок 3.2 – Фрагмент протоколу імітації

На рисунку зображений момент часу моделювання, кількість маркерів у кожній позиції та перехід, з якого відбувся вихід маркерів.

```
Refuelings expenses: 124717
Leaves: 93
Income With waiting: 77364
Income Without waiting: 4000
Profit: -43353
```

Рисунок 3.3 – Результати тестового запуску імітації

### 3.4 Верифікація роботи алгоритму імітації

Проведемо детальну верифікацію роботи алгоритму, щоб впевнитись в правильній роботі програми. Результати верифікації наведені в таблиці 3.1

Таблиця 3.1 - Результати верифікації алгоритму імітації.

Конфігурація	Літаки, що залишили аеропорт	Витрати на дозаправку	Прибуток за посадки з очікування	Прибуток за посадки без очікування
Початкова	80	116058	81370	4000
Кількість смуг - 4	42	94495	162157	8000
Інтенсивність надходження - 20 літаків в годину	245	289232	81000	4000
Час до залишення аеропорту - 280 хв	41	97588	80798	4000
Час звільнення смуги - 17 хв	40	102124	170707	4000

Спостерігаємо, що збільшення смуг у два рази відчутно збільшило дохід аеропорту, адже пропускна здатність аеропорту зросла. Також бачимо що дохід за посадки без очікування стала 80000, що означає що перші 4 літака приземлились без очікування. На конфігурації з збільшеним надходженням літаків, спостерігаємо, що прибуток залишився таким самим, але витрати на дозаправку та кількість літаків, що залишили аеропорт, суттєво збільшилися. Така поведінка є очікуваною, адже аеропорт не може обробити більшу кількість літаків і тому більше літаків йдуть на дозаправку а потім покидають аеропорт.

## 4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ

Перед початком експериментального дослідження важливим етапом є визначення довжини перехідного періоду та кількість повторних замірів вихідних змінних. Для точного обрахунку параметрів дослідження спочатку обчислимо статистичні характеристики відгуку (рис 4.1).

**N=3: Mean payback time = 0.023241019618056042,  
StdDev = 0.0013152255040191963**

Рисунок 4.1 - Статистичні характеристики відгуку

Далі виконаємо розрахунок оптимальної кількості повторних замірів за рівнянням Чебишева. Візьмемо довірчу ймовірність - 0,95 та точність рівній одному стандартному відхиленню.

$$p = \frac{\sigma^2}{\varepsilon^2(1-\beta)} = \frac{1}{0,05} = 20$$

Для дослідження перехідного періоду, була проведена серія експериментів з значеннями: 2 години, 12 годин та 1 день. Як з'ясувалось при тривалості моделювання в 1 день різниця між відгуком моделі була менша за задану точність. Тому тривалість імітації має бути, щонайменше 1 день. Результати експериментів наведені на рисунку 4.2

**2 hours: Max diff: 0.00734  
12 hours: Max diff: 0.00237  
1 day: Max diff: 0.00035**

Рисунок 4.2 - Результат визначення перехідного періоду

Максимальна різниця відгуку  $0.00035 < 0.00131$ , отже значення в 1 день це оптимальна тривалість імітації.

Згідно до мети моделювання задачею є оптимізація моделі. Для досягнення мети моделювання доцільніше буде використати повний перебір, можливих варіантів, оскільки їх кількість не велика. Щоб зменшити кількість

експериментів будемо відкидати одразу гірші варіанти. Для оптимізації відгуку моделі застосуємо метод Фібоначчі для однієї змінної.

Суть метода полягає в тому, що інтервал, у межах якого може міститися мінімум, поступово звужується, причому схема звуження визначається співвідношеннями сусідніх чисел Фібоначі.

На початку маємо відрізок  $[a, b]$  і заздалегідь обираємо найменше число  $F_N$  з ряду Фібоначчі, що є не меншим за довжину інтервалу. Використовуючи його, всередині початкового інтервалу розміщують дві точки.

$$x_1 = a + \frac{F_{N-2}}{F_N} (b - a), \quad x_2 = a + \frac{F_{N-1}}{F_N} (b - a),$$

у яких обчислюють значення функції. Залежно від того, у якій точці значення функції менше, одна з частин інтервалу відкидається. Якщо  $f(x_1) > f(x_2)$ , то мінімум може бути лише в правій частині, і новим інтервалом стає  $[x_1, b]$ . Якщо ж  $f(x_1) \leq f(x_2)$ , залишають ліву частину  $[a, x_2]$ .

Послідовність таких кроків повторюється з використанням менших чисел Фібоначі, і на кожному етапі одна з двох точок збігається з точкою попередньої ітерації, що мінімізує кількість нових обчислень значень функції. Після завершення всіх кроків залишається дуже вузький інтервал дискретних значень аргументу, серед яких мінімум визначається однозначно.

Відповідно до обраного підходу був розроблений код для проведення експерименту. Повний програмний код наведений в додатку Б. Результати експериментального дослідження наведено на рисунку 4.3.

```
N=6: Mean payback time = 0.07007190280418396
N=8: Mean payback time = 0.10405416182022831
N=4: Mean payback time = 0.03930893916490192
N=5: Mean payback time = 0.05512607572969493
N=3: Mean payback time = 0.023039576136694967
N=4: Mean payback time = 0.038708302255962924
Best number of runways: 3
```

Рисунок 4.3 - Результати експериментального дослідження

З рисунку 4.3 можемо побачити, що алгоритм за 3 ітерації та виконав всього 6 замірів відгуку та знайшов оптимальне значення параметра.

Найменший час окупності додаткових смуг досягається при 3 смугах. Отже найоптимальніше буде побудувати одну додаткову смугу.

## 5 ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ

У процесі розробки та дослідження імітаційної моделі системи аеропорту у формалізмі мережі Петрі було акумульовано низку важливих висновків. Спочатку було підтверджено коректність моделі шляхом верифікації. Аналіз різних конфігурацій показав, що збільшення кількості смуг з двох до чотирьох суттєво збільшує дохід та зменшує кількість літаків, що залишили аеропорт. Водночас, збільшення інтенсивності надходження літаків при незмінній інфраструктурі призводить до очікуваного погіршення: зростають витрати на дозаправку та кількість втрачених рейсів, оскільки аеропорт не справляється з потоком. Ключовою метою моделювання була оптимізація кількості посадкових смуг для мінімізації часу окупності інвестицій. Для досягнення цієї мети було застосовано метод Фібоначчі для повного перебору можливих варіантів. Експериментальне дослідження чітко встановило, що мінімальний середній час окупності досягається при 3 смугах. Подальше збільшення кількості смуг не дає пропорційного зростання річного прибутку, що збільшує час окупності.

На основі отриманих результатів, для поліпшення функціонування системи надаються пропозиції: по-перше, варто зосередитися на зменшенні часу звільнення смуги (з 35 до, наприклад, 17 хвилин), оскільки це рішення без додаткових капітальних витрат значно підвищує прибуток і знижує втрати літаків. По-друге, слід проаналізувати політику дозаправки, оскільки це є значною статтею витрат, які можуть бути оптимізовані.

Таким чином, модель не лише визначила оптимальну кількість смуг, а й підкреслила важливість оперативної ефективності як критичного фактора прибутковості аеропорту. На основі цих висновків можна рекомендувати, як найбільш доцільну зміну, будівництво однієї додаткової смуги та оптимізацію процесів наземного обслуговування.

## ВИСНОВКИ

У даній курсовій роботі було успішно розроблено та досліджено імітаційну модель системи аеропорту у формалізмі мереж Петрі, призначенну для вирішення оптимізаційної задачі: визначення оптимальної кількості додаткових посадкових смуг, що забезпечить мінімальний час окупності інвестицій. Застосування імітаційного методу моделювання, зокрема формалізму мереж Петрі, виявило свою перевагу у здатності точно відтворювати динаміку системи та паралельну взаємодію стохастичних процесів. Перевагою розробленого алгоритму є його ефективна робота з конфліктами між переходами та точне врахування фінансових показників за допомогою маркерів різних типів. Через складність коректного скидання таймерів, не набуло детального відтворення чітке розмежування черг на «дозаправлені» та «недозаправлені» з єдиною динамічною чергою. Загалом, імітаційне моделювання системи визнається успішним, оскільки воно дозволило сформулювати ключовий висновок: мінімальний час окупності досягається при загальній кількості 3 посадкових смуги, що означає необхідність побудови однієї додаткової смуги. Для подальшого дослідження системи рекомендується:

- Проаналізувати вплив зменшення часу звільнення смуги;
- Дослідити чутливість моделі до варіювання часу дозаправки;

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стеценко І. В. Моделювання систем : Навчальний посібник. Київ. 407 с.
2. BPMN 2.0 symbols - A complete guide with examples. *Camunda*. URL: <https://camunda.com/bpmn/reference/> (date of access: 14.10.2025).
3. GitHub - sashapanasiuk5/SystemModeling\_CW. *GitHub*. URL: [https://github.com/sashapanasiuk5/SystemModeling\\_CW](https://github.com/sashapanasiuk5/SystemModeling_CW) (date of access: 22.11.2025).
4. Math.NET numerics. *Math.NET Numerics*. URL: <https://numerics.mathdotnet.com/> (date of access: 31.10.2025).
5. .NET documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/uk-ua/dotnet/> (date of access: 22.11.2025).

## ДОДАТОК А. ЛІСТИНГ КОДУ АЛГОРИТМУ ІМІТАЦІЇ

```

namespace CourseWork;

public class Place
{
    private readonly List<Token> _tokens = new();
    private double _meanTokenCount = 0;

    public string Name { get; }
    public int TokenCount => _tokens.Count;
    public IReadOnlyCollection<Token> Tokens =>
        _tokens.AsReadOnly();

    public Place(string name)
    {
        Name = name;
    }

    public Place(string name, int initialTokenCount)
    {
        Name = name;
        for (int i = 0; i < initialTokenCount; i++)
        {
            _tokens.Add(new Token());
        }
    }

    public virtual void ReceiveTokens(List<Token> tokens)
    {
        _tokens.AddRange(tokens);
    }

    public virtual void ReceiveToken(Token token)

```

```
        _tokens.Add(token);
    }

    public virtual Token GetToken()
    {
        var token = _tokens.FirstOrDefault();
        _tokens.RemoveAt(0);
        return token;
    }

    public virtual Token GetTokenByType(TokenType tokenType)
    {
        var token = _tokens.FirstOrDefault(t => t.Type == tokenType);
        _tokens.Remove(token);
        return token;
    }

    public virtual List<Token> GetTokens(int count)
    {
        var tokens = _tokens.GetRange(0, count);
        _tokens.RemoveRange(0, count);
        return tokens;
    }

    public void CollectStatistic(double deltaT)
    {
        _meanTokenCount += deltaT * TokenCount;
    }

    public double GetMeanTokenCount(double totalTime)
    {
        return _meanTokenCount / totalTime;
    }
}
```

```
}

}

namespace CourseWork;

public class TypeDeterminingPlace: Place
{
    private Place _freeRunways;

    private Place _queueRefueled;

    public TypeDeterminingPlace(string name, Place freeRunways, Place queueRefueled) : base(name)
    {
        _freeRunways = freeRunways;
        _queueRefueled = queueRefueled;
    }

    public TypeDeterminingPlace(string name, Place freeRunways, Place queueRefueled, int initialTokenCount) : base(name, initialTokenCount)
    {
        _freeRunways = freeRunways;
        _queueRefueled = queueRefueled;
    }

    public override void ReceiveToken(Token token)
    {
        token.Type = GetTokenType();
        base.ReceiveToken(token);
    }

    public override void ReceiveTokens(List<Token> tokens)
    {
        foreach (var token in tokens)
        {
```

```

        ReceiveToken(token);
    }

}

private TokenType GetTokenType()
{
    if (_freeRunways.TokenCount > 0 && TokenCount == 0 &&
_queueRefueled.TokenCount == 0)
    {
        return TokenType.LandingWithoutWaiting;
    }
    return TokenType.LandingWithWaiting;
}

public class Transition: ITransition
{
    private List<Place> _inputPlaces;
    private List<Place> _outputPlaces;
    public IGenerator Generator { get; set; }

    private List<(double,TokenType)> _tMoments = new();

    private Place? _targetTokenTypePlace;

    public int Id { get; }
    public int Priority { get; }

    public double NearestTMoment
    {
        get
        {
            if(_tMoments.Count == 0)

```

```

        return Double.MaxValue;
    return _tMoments.Min(x => x.Item1);
}
}

public bool CanTrigger => _inputPlaces.All(p =>
p.TokenCount > 0);

public Transition(int id, List<Place> inputPlaces,
List<Place> outputPlaces, int priority = 0, Place?
targetTokenTypePlace = null)
{
    _inputPlaces = inputPlaces;
    _outputPlaces = outputPlaces;
    Priority = priority;
    _targetTokenTypePlace = targetTokenTypePlace;
    Id = id;
}

public void EntryTokens(double tCurrent)
{
    while (CanTrigger)
    {
        var tokenType = TokenType.None;
        foreach (var place in _inputPlaces)
        {
            var token = place.GetToken();
            if (place == _targetTokenTypePlace && tokenType
== TokenType.None)
                tokenType = token.Type;
        }

        var t = tCurrent;
        if (Generator != null)
        {

```

```

        t += Generator.Generate();
    }
    _tMoments.Add((t, tokenType));
}
}

public void ExitTokens(double t)
{
    var (tMoment, tokenType) = _tMoments.Find(x => x.Item1
== t);
    _tMoments.Remove((tMoment, tokenType));

    var token = new Token() { Type = tokenType };
    foreach (var place in _outputPlaces)
    {
        place.ReceiveToken(token);
    }
}

public class OutputTransition: ITransition
{
    public int Priority { get; }

    public int Id { get; }

    public bool CanTrigger
    {
        get
        {
            if (_tokenType != TokenType.None)
            {
                return _inputPlace.Tokens.Any(t => t.Type ==
_tokenType);
            }
        }
    }
}
```

```
        return _inputPlace.TokenCount > 0;
    }

}

public double NearestTMoment
{
    get
    {
        if(_tMoments.Count == 0)
            return Double.MaxValue;
        return _tMoments.Min();
    }
}

private List<double> _tMoments = new();

private readonly Place _inputPlace;
private readonly Place _outputPlace;

private readonly IContinuousDistribution? _generator;

private readonly int _outputMultiplicity;

private readonly TokenType _tokenType;

public OutputTransition(int id, Place inputPlace, Place outputPlace, int upperValue, int lowerValue, TokenType tokenType = TokenType.None, int priority = 0)
{
    _inputPlace = inputPlace;
    _outputPlace = outputPlace;
    _generator = new ContinuousUniform(lowerValue,
upperValue);
    _tokenType = tokenType;
}
```

```

Priority = priority;
Id = id;
}

public OutputTransition(int id, Place inputPlace, Place
outputPlace, int outputMultiplicity, TokenType tokenType =
TokenType.None, int priority = 0)
{
    _inputPlace = inputPlace;
    _outputPlace = outputPlace;
    _outputMultiplicity = outputMultiplicity;
    _tokenType = tokenType;
    Priority = priority;
    Id = id;
}

public void EntryTokens(double tCurrent)
{
    while (CanTrigger)
    {
        if (_tokenType != TokenType.None)
        {
            _inputPlace.GetTokenByType(_tokenType);
        }
        else
        {
            _inputPlace.GetToken();
        }
        _tMoments.Add(tCurrent);
    }
}

public void ExitTokens(double t)
{
    _tMoments.Remove(t);
}

```

```
        int tokenCount;
        if (_generator != null)
        {
            tokenCount = (int)_generator.Sample();
        }
        else
        {
            tokenCount = _outputMultiplicity;
        }

        for (int i = 0; i < tokenCount; i++)
        {
            _outputPlace.ReceiveToken(new Token());
        }
    }

}

public class PetriNet
{
    private readonly List<ITransition> _transitions;

    private readonly List<Place> _places;

    private readonly bool _printInfo;

    public PetriNet(List<ITransition> transitions, List<Place>
places, bool printInfo = true)
    {
        _transitions = transitions;
        _places = places;
        _printInfo = printInfo;
    }

    public void Run(double tModel)
```

```

    {

        double t = 0.0;
        while (t < tModel)
        {
            var tNext = _transitions.Min(x =>
x.NearestTMoment);

            if (tNext != Double.MaxValue)
            {
                var transition = _transitions.First(x =>
x.NearestTMoment == tNext);

                transition.ExitTokens(tNext);
                if (_printInfo)
                {
                    Console.WriteLine("----- T Next -
"+ tNext + "-----");
                    foreach (var place in _places)
                    {
                        place.CollectStatistic(tNext - t);
                        Console.WriteLine(place.Name + ": " +
place.TokenCount);
                    }
                    Console.WriteLine("Transition " +
transition.Id + " was triggered");
                }
                Console.WriteLine("-----\n");
            }
            t = tNext;
        }

        var transitionsForEntrance = _transitions.Where(x => x.CanTrigger).ToArray();

        Random.Shared.Shuffle(transitionsForEntrance);
    }
}

```

```

var prioritizedTransitions =
transitionsForEntrance.OrderByDescending(x =>
x.Priority).ToList();

while (prioritizedTransitions.Count > 0)
{
    var transition = prioritizedTransitions[0];
    prioritizedTransitions.RemoveAt(0);

    transition.EntryTokens(t);
    prioritizedTransitions =
prioritizedTransitions.Where(x =>
x.CanTrigger).OrderByDescending(x => x.Priority).ToList();
}

if (_printInfo)
{
    Console.WriteLine("----- Statistics -----");
    foreach (var place in _places)
    {
        var meanTokenCount =
place.GetMeanTokenCount(tModel);
        Console.WriteLine("Mean token count for place " +
place.Name + ": " + meanTokenCount);
    }
}

Console.WriteLine("-----\n");
}

}
}

var p1 = new Place("Plane generator", 1);

var refuelingTimerStart = new Place("Refueling timer start");

```

```

var refuelingTimerReset = new Place("Refueling timer reset");
var refuelingTimerEnd = new Place("Refueling timer end");
var refuelingTimerReseted = new Place("Reseted refueling
timers");

var leaveTimerStart = new Place("Leave timer start");
var leaveTimerEnd = new Place("Leave timer end");
var leaveTimerReset = new Place("Leave timer reset");
var leaveTimerReseted = new Place("Reseted leave timers");

var refuelings = new Place("Refuelings");
var refuelingExpenses = new Place("Refueling expenses");
var leaves = new Place("Leaves");

var freeRunways = new Place("Free runways", 8);

var planeQueue_refueled = new Place("Plane queue (refueled)");

var planeQueue = new TypeDeterminingPlace("Plane queue",
freeRunways, planeQueue_refueled);

var landing1Place = new Place("Landing 1");
var landing2Place = new Place("Landing 2");

var landedPlanes = new Place("Landed planes");

var income1 = new Place("Income for landings with waiting");
var income2 = new Place("Income for landings without
waiting");

var generator = new Transition(1, [p1], [p1, planeQueue,
leaveTimerStart, refuelingTimerStart])
{
    Generator = new DistributionAdapter(new Exponential(1.0 /
6.0))
}

```

```

    } ;

    var refuelTimer = new Transition(2, [refuelingTimerStart],
[refuelingTimerEnd])
{
    Generator = new DistributionAdapter(new
ContinuousUniform(60, 80))
};

var refuelTimerReseter = new Transition(3, [refuelingTimerEnd,
refuelingTimerReset], [refuelingTimerReseted], 1);

var refueling = new Transition(4, [planeQueue,
refuelingTimerEnd], [refuelings, planeQueue_refueled], 0,
planeQueue);

var refuelingExpenseCounter = new OutputTransition(5,
refuelings, refuelingExpenses, 1200, 800);

var leaveTimer = new Transition(6, [leaveTimerStart],
[leaveTimerEnd])
{
    Generator = new ConstantGenerator(140.0)
};

var leaveTimerReseter = new Transition(7, [leaveTimerEnd,
leaveTimerReset], [leaveTimerReseted], 1);

var leaving = new Transition(8, [planeQueue_refueled,
leaveTimerEnd], [leaves]);

var landing1 = new Transition(9, [planeQueue, freeRunways],
[refuelingTimerReset, leaveTimerReset, landing1Place], 1,
planeQueue);

```

```
var landing2 = new Transition(10, [planeQueue_refueled,
freeRunways], [leaveTimerReset, landing2Place], 1,
planeQueue_refueled);

var countIncomeForLandingWithWaiting = new
OutputTransition(11, landedPlanes, income1, 1600, 1400,
TokenType.LandingWithWaiting);

var countIncomeForLandingWithoutWaiting = new
OutputTransition(12, landedPlanes, income2, 2000,
TokenType.LandingWithoutWaiting);

var releaseRunway1 = new Transition(13, [landing1Place],
[freeRunways, landedPlanes], 0, landing1Place)
{
    Generator = new ConstantGenerator(17)
};

var releaseRunway2 = new Transition(14, [landing2Place],
[freeRunways, landedPlanes], 0, landing2Place)
{
    Generator = new ConstantGenerator(17)
};

var model = new PetriNet([
    generator,
    refuelTimer,
    refuelTimerReseter,
    refueling,
    leaveTimer,
    leaveTimerReseter,
    leaving,
    landing1,
    landing2,
    releaseRunway1,
    releaseRunway2,
```

```

    refuelingExpenseCounter,
    countIncomeForLandingWithoutWaiting,
    countIncomeForLandingWithWaiting
], [
    p1,
    refuelingTimerStart,
    refuelingTimerReset,
    refuelingTimerEnd,
    refuelingTimerReseted,
    leaveTimerStart,
    leaveTimerEnd,
    leaveTimerReset,
    leaveTimerReseted,
    refuelings,
    refuelingExpenses,
    leaves,
    freeRunways,
    planeQueue_refueled,
    planeQueue,
    landing1Place,
    landing2Place,
    landedPlanes,
    income1,
    income2
]) ;

model.Run(1440);

Console.WriteLine("Refuelings expenses: " + refuelingExpenses.TokenCount);
Console.WriteLine("Leaves: " + leaves.TokenCount);

Console.WriteLine("Income With waiting: " + income1.TokenCount);
Console.WriteLine("Income Without waiting: " + income2.TokenCount);

```

```
var profit = (income1.TokenCount + income2.TokenCount -  
refuelingExpenses.TokenCount) * 365;  
Console.WriteLine("Profit: " + profit);
```

## ДОДАТОК Б. ЛІСТИНГ КОДУ ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ

```

double RunModel(int n)
{
    var values = new List<double>();
    for (int i = 0; i < 20; i++)
    {
        var (model, paybackFunc) = ModelFactory.GetNet(n);

        model.Run(1440);
        var paybackTime = paybackFunc(365);

        if (paybackTime != Double.PositiveInfinity)
            values.Add(paybackTime);
    }
    var mean = values.Mean();

    Console.WriteLine($"N={n}: Mean payback time = {mean}");
    return mean;
}

int FibonacciSearch(int left, int right, Func<int, double> func)
{
    List<int> fib = new List<int> { 1, 1 };
    while (fib[fib.Count - 1] < (right - left + 1))
        fib.Add(fib[fib.Count - 1] + fib[fib.Count - 2]);

    int k = fib.Count - 1;

    while (left < right)
    {
        if (k < 2)
            break;

        int x1 = left + (fib[k - 2] * (right - left)) / fib[k];

```

```
int x2 = left + (fib[k - 1] * (right - left)) / fib[k];

if (x1 == x2)
    x2++;

if (x2 > right) x2 = right;
if (x1 < left) x1 = left;

double f1 = func(x1);
double f2 = func(x2);

if (f1 > f2)
{
    left = x1 + 1;
}
else
{
    right = x2 - 1;
}

k--;
}

return left;
}

var bestN = FibonacciSearch(3, 12, RunModel);
Console.WriteLine("Best number of runways: " + bestN);
```