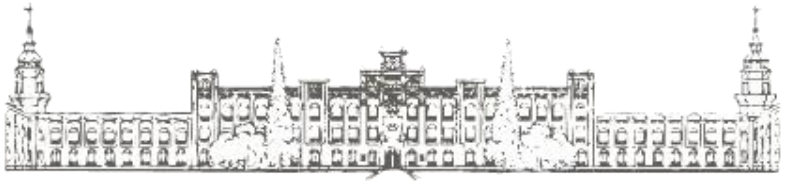

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту
«Моделювання систем. Курсова робота»
Тема: Імітаційна модель роботи посадових смуг аеропорту на основі формалізму мережі Петрі

Керівник:
Викладач Дифучин Антон Юрійович
«Допущено до захисту»
_____

«___» _____ 2024 р.

Захищено з оцінкою
_____

Члени комісії:

_____

_____

Виконавець:
Петров Ігор Ярославович
студент групи ІП-13
залікова книжка № ІП-1327
_____

«21» грудня 2024 р.

Інна СТЕЦЕНКО

Антон ДИФУЧИН

Київ – 2024

ЗАВДАННЯ

Завдання В12. Потік літаків, що потребують посадки в аеропорті, пуассонівський[2] з інтенсивністю 10 літаків за годину. В аеропорті 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 ± 10 хвилин, то виникає необхідність дозаправки літака, на що аеропорту доведеться витратити 1000 ± 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, прибуток становить

1500 ± 100 грошень. Витрати на будівництво додаткової посадкової смуги складають 3 000 000 грошових одиниць. В аеропорті 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:

$$T = \frac{S}{p}$$

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 сфомулювати 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

Петров Ігор Ярославович. Курсова робота на тему «Імітаційна модель роботи посадових смуг аеропорту на основі формалізму мережі Петрі».

Текстова частина курсової роботи складається із вступу, 5 розділів, висновків, списку використаних джерел з 5 найменувань та 1 додатку, які викладено на 66 сторінках. В тексті роботи оформлено 15 рисунків, 4 таблиць.

КЛЮЧОВІ СЛОВА: МЕРЕЖІ ПЕТРІ, ІМІТАЦІЙНА МОДЕЛЬ, ДИСПЕРСІЙНИЙ АНАЛІЗ, PetriObjModelPaint[1].

ЗМІСТ

ВСТУП

Сучасні аеропорти є важливими вузлами транспортної інфраструктури, що забезпечують зв'язок між регіонами, країнами та континентами. Ефективна організація роботи аеропортів є ключовим фактором для забезпечення стабільного функціонування повітряного транспорту. Одним із найважливіших аспектів є управління потоками літаків, які потребують злету і посадки.

Забезпечення достатньої кількості посадкових смуг є складним завданням. Недостатня кількість смуг призводить до затримок, додаткових витрат на обслуговування та втрати прибутку через перенаправлення рейсів. Водночас надлишкова кількість смуг може спричинити значні фінансові витрати. Тому важливо знайти оптимальний баланс.

У цій роботі розглядається задача визначення оптимальної кількості посадкових смуг в умовах, коли потік літаків має пуассонівський розподіл із заданою інтенсивністю. Враховуються витрати на будівництво, прибуток від обслуговування літаків і можливі додаткові витрати в разі затримок.

Метою роботи є моделювання роботи аеропорту для визначення кількості посадкових смуг, при якій інтервал окупності додаткових смуг буде мінімальним.

1 КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ

1.1 Опис призначення системи

Система призначена для моделювання роботи аеропорту, зокрема взаємодії літаків із посадковими смугами, з метою аналізу та оптимізації використання ресурсів. Основною функцією є оцінка впливу кількості посадкових смуг на ефективність обслуговування авіатранспорту та фінансові показники аеропорту.

Система має дозволяти:

- Моделювати сценарії навантаження. Перевіряється, як різна кількість смуг впливає на частоту затримок, час очікування та кількість втрачених рейсів.
- Розраховувати фінансові витрати і доходи. Оцінюються витрати на дозаправки, а також дохід від обслуговування літаків із врахуванням їх очікування.
- Розрахувати загальний час окупності. Система має дозволяти знайти час окупності, визначений як відношення ціни на побудову смуг до чистого доходу за рік з відповідною кількістю смуг.

Згідно поставленого завдання на курсову роботу необхідно створити віртуальну імітаційну модель, що буде інструментом для прийняття рішень щодо розвитку інфраструктури аеропорту, визначення оптимальної кількості посадкових смуг, яка забезпечить мінімальні витрати та максимальну ефективність.

1.2 Вхідні змінні моделі

Для моделі передбачені наступні вхідні змінні, що прописані у таблиці 1.1.

Таблиця 1.1 – Вхідні змінні моделі

Назва	Значення
Кількість посадкових смуг	N([3; 12])
Витрати на дозаправку	1000 ± 200
Заробіток за посадку з затримкою	1500 ± 100
Заробіток за посадку без затримки	2000
Час очікування перед дозаправкою	70 ± 10 хв.
Час очікування перед перельотом	140 хв.
Час обсуговування літака на смузі	35 хв.
Інтенсивність надходження літаків	Poisson(6) хв.

1.3 Вихідні змінні моделі

Для моделі передбачені наступні вихідні змінні:

1. Кількість літаків, що здійснили перельот на інший аеропорт
2. Прибуток від обслуговування літаків
3. Втрати аеропорту на дозаправку
4. Час окупності для встановленої кількості смуг

1.4 Схематичне представлення процесу функціонування системи

Процес функціонування системи у вигляді блок-схеми зображений на рисунку 1.1.

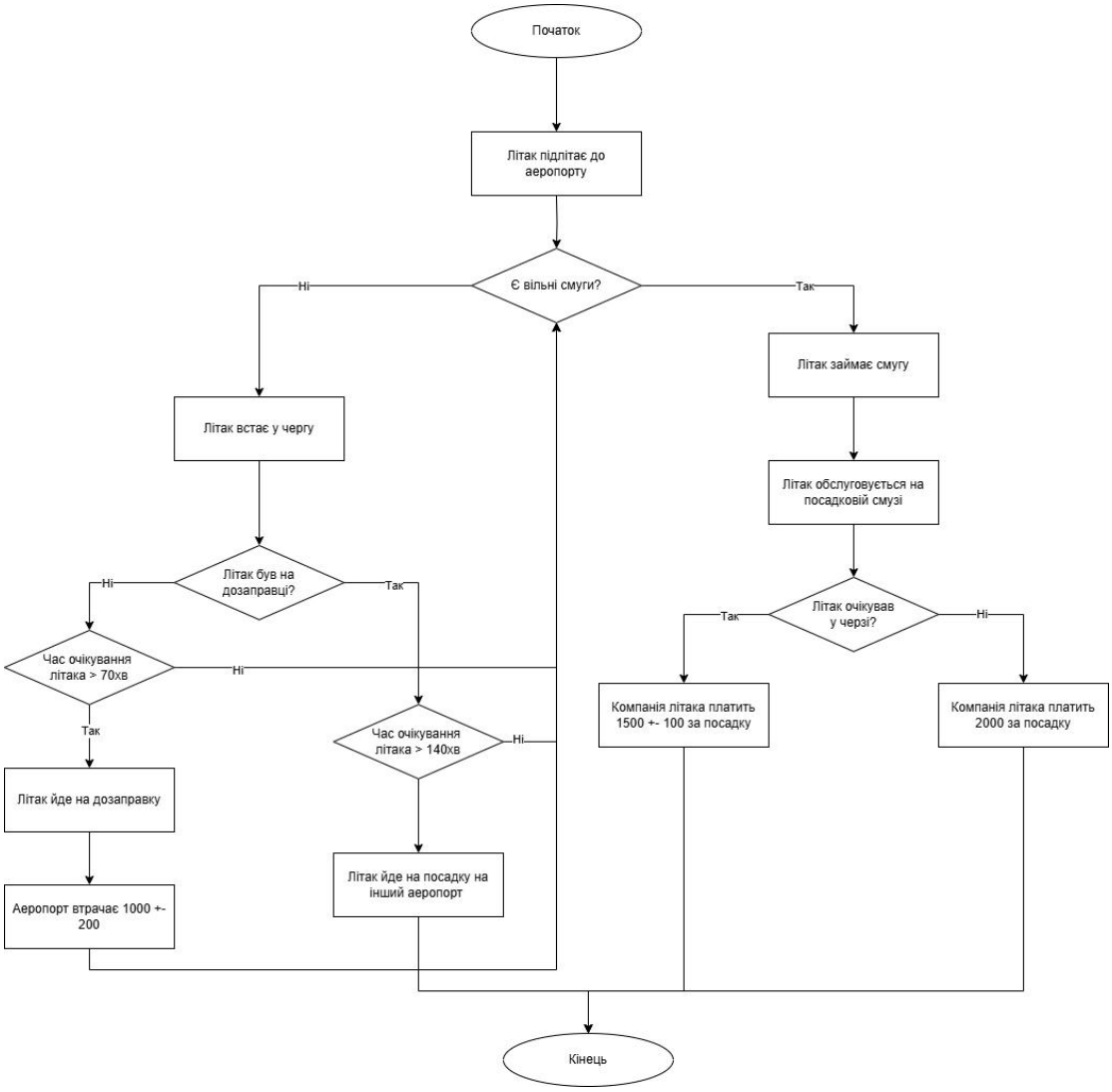

Рисунок 1.1 – Концептуальна блок-схема функціонування системи

1. Початок

   Схема починається з події: до аеропорту прибуває літак.

2. Перевірка наявності вільної смуги

   Після прибуття літака перевіряється, чи є в аеропорту вільна злітно-посадкова смуга.

   а) Якщо вільна смуга є:

      і. Літак займає цю смугу та проходить обслуговування (посадку).

      іі. Далі перевіряється, чи літак очікував у черзі до цього.

         1. Якщо літак не чекав у черзі, компанія сплачує 2000 (умовна одиниця) за посадку.

2. Якщо літак чекав у черзі, компанія сплачує 1500 ± 100 за посадку.

b) Якщо вільної смуги немає:

i. Літак стає в чергу для очікування.

ii. Далі з'ясовується, чи літак уже був відправлений на дозаправку раніше.

1. Якщо літак ще не був на дозаправці, то перевіряється час очікування:

a) Якщо час очікування більше 70 ± 10 хвилин, літак йде на дозаправку та повертається у чергу, аеропорт втрачає 1000 ± 200.

b) Якщо час очікування ≤ 70 ± 10 хвилин, літак продовжує очікувати в черзі, якщо звільниться смуга літак піде на посадку.

2. Якщо літак уже був на дозаправці, перевіряється інший граничний час очікування:

a) Якщо час очікування більше за 140 хвилин, літак відправляється на посадку в інший аеропорт.

b) Якщо час очікування ≤ 140 хвилин, то коли звільняється смуга, літак заходить на посадку.

3. Кінець

Після вирішення питання з посадкою (чи перенаправленням до іншого аеропорту), логічна гілка процесу завершується.

Отже, схема відображає логіку розподілу злітно-посадкових смуг, управління чергою, обробку затримок, а також економічні наслідки (оплати або втрати) в залежності від часу очікування літака та доступності інфраструктури аеропорту.

2 ФОРМАЛІЗОВАНА МОДЕЛЬ СИСТЕМИ

Створення формалізованої моделі є важливим етапом розробки імітаційної моделі роботи аеропорту з обслуговування літаків. Для аналізу функціонування системи необхідно відобразити основні процеси, такі як прибуття літаків, їхнє очікування, дозаправка, посадка, а також перенаправлення до іншого аеропорту у разі надмірної затримки. Завдання моделювання передбачає побудову імітаційної моделі на основі мережі Петрі, яка враховуватиме часові характеристики операцій, інтенсивність прибуття літаків, кількість та пропускну спроможність злітно-посадкових смуг, а також витрати, пов’язані з простоєм, дозаправкою й перенаправленням літаків.

Головною метою моделювання є визначення оптимальної кількіості злітно-посадкових смуг. Застосування мережі Петрі дозволить формалізувати дискретні події, такі як прибуття літака, його постій, початок та завершення посадки, операції дозаправки та рішення про перенаправлення. Це дасть змогу детально відтворити поведінку системи за різних умов, порівняти наслідки різних сценаріїв (наприклад, збільшення кількості смуг, або залишення їх у початковій кількості) та оцінити мінімальний час окупності додаткових інвестицій.

Виходячи з попереднього концептуального опису, варто моделювати багатоканальну стохастичну мережу Петрі, оскільки одночасно можуть обслуговуватися кілька літаків (залежно від кількості смуг), а часові затримки операцій підпорядковуються різноманітним ймовірнісним законам розподілу. Крім того, при розробці алгоритму імітації цієї мережі важливо врахувати пріоритети виконання переходів, аби коректно моделювати ситуації з чергами, позачерговими посадками чи пріоритетною дозаправкою.

Таким чином, створена модель дасть змогу оцінити ефективність роботи аеропорту за різних умов та сприятиме пошуку оптимального рішення для зниження витрат і мінімізації часу очікування літаків.

Побудуємо формалізовану модель за допомогою графічного інтерфейсу, вона представлена на рисунку 2.1.

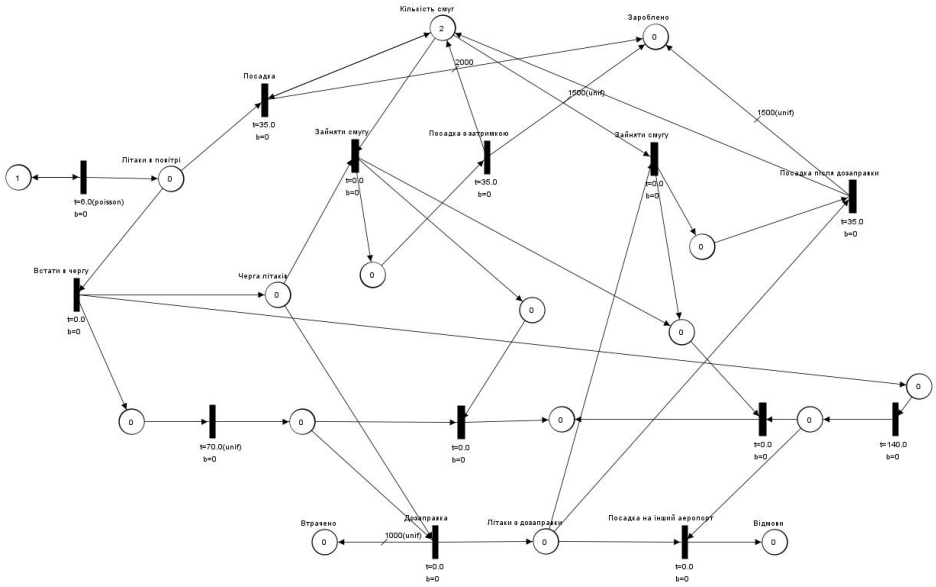

Рисунок 2.1 – Формалізована схема побудованої мережі

Модель використовує стандартні елементи мережі Петрі (рисунок 2.2).

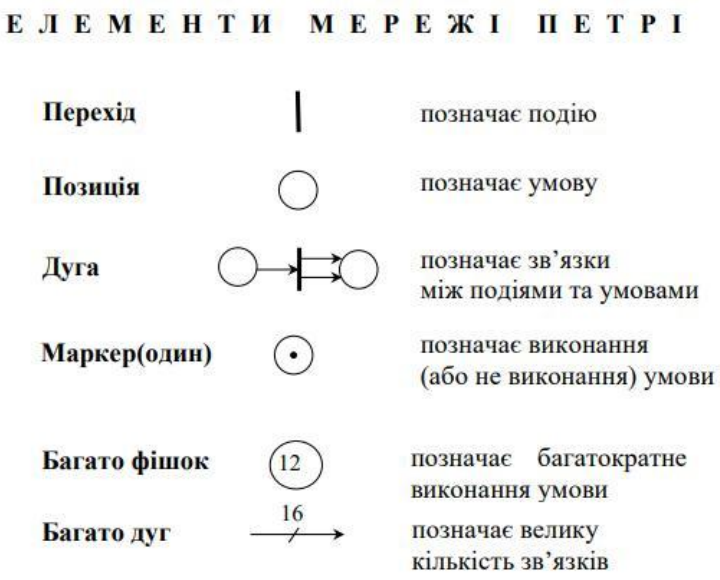

Рисунок 2.2 – Елементи мережі Петрі[3]

Також було використано специфічне позначення ваги дуг (рисунок 2.3). При такому позначенні вага дуги розподілена за законом у дужках, це необхідно для досягнення випадкових величин оплати, що вказані у завданні , наприклад 1000 ± 200.

Рисунок 2.3 – Багато дуг з рівномірним розподілом навколо 1000

Тепер детально опишемо кожний логічний модуль побудованої мережі.

Для надходження літаків використовується генерація випадкового цілого числа за Пуасонівським розподілом з $\lambda = 6$. Літак підлітає до аеропорту і має два варіанти: одразу піти на посадку, де пройде обслуговування, що триває 35 хв, або встати в чергу літаків, якщо усі злітно-посадкові смуги зайняті. Ці дії зображені на рисунку 2.4.

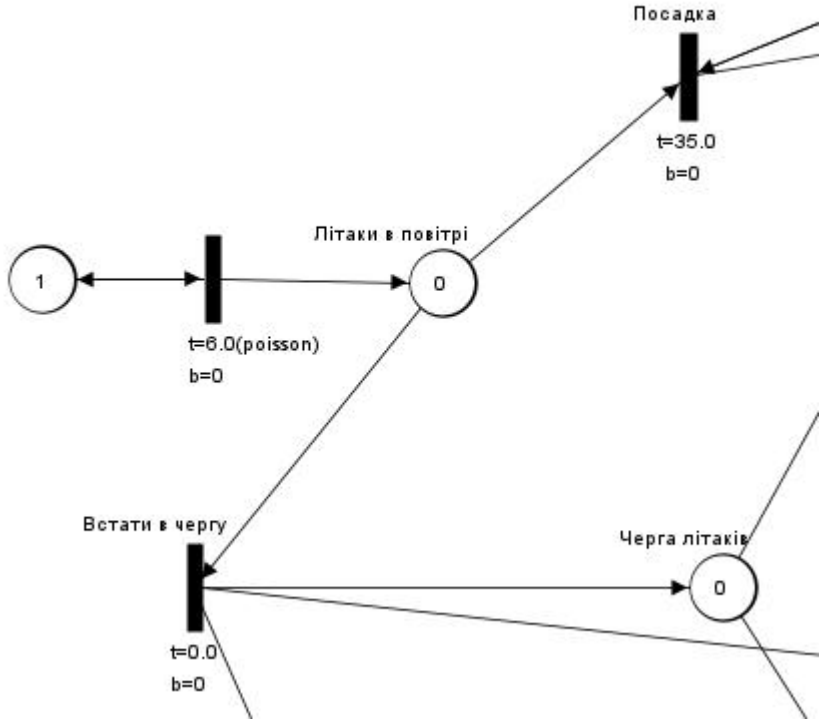

Рисунок 2.4 – Літак одразу йде на посадку або встає у чергу

Можливість посадки визначається наявністю вільних посадкових смуг, що визначена у позиції “Кількість смуг”. Загальна кількість визначається параматром моделі заздалегіть. Після посадки літак звільняє смугу і приносить 2000 умовних грошових одиниць аеропорту. Ці дії зображені на рисунку 2.5.

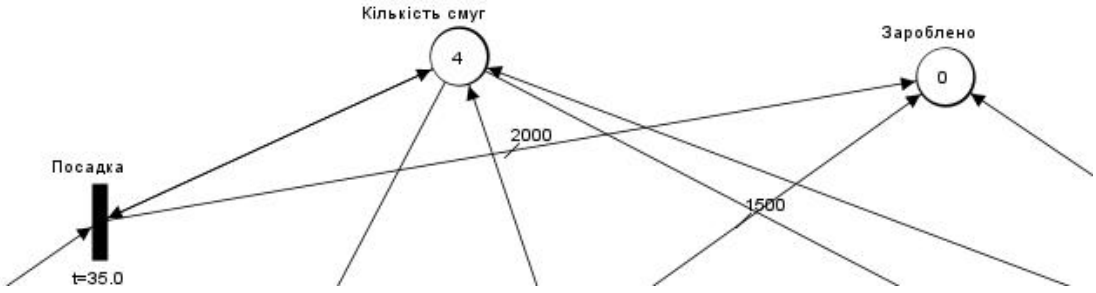

Рисунок 2.5 – Перевірка наявності вільної смуги і надходження 2000 до позиції “Зароблено”

Потрапляючи у чергу, ми також відправляємо мітку у таймер на 70 ± 10 хв. і 140 хв. (рисунок 2.6.).

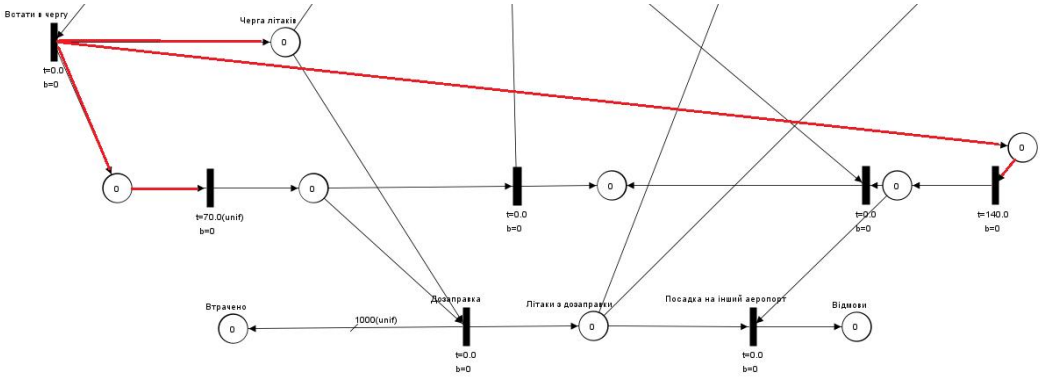

Рисунок 2.6 – Відправлення мітки до черги літаків та початок відліку 70 ± 10 хв. і 140 хв.

Зайняття смуги після очікування є більш пріоритетною задачею ніж пряма посадка після надходження. Під час очікування у черзі літак може зайняти вільну смугу, і , відповідно, зробити посадку з затримкою, що принесе аеропорту 1500 ± 100 умовних грошових одиниць. Також у такому разі таймер на 70 ± 10 і 140 хв. інформується, про те що літак вже здійснив дозаправку і він не буде відправлений на дозаправку або на інший аеропорт. (рисунок 2.7.)

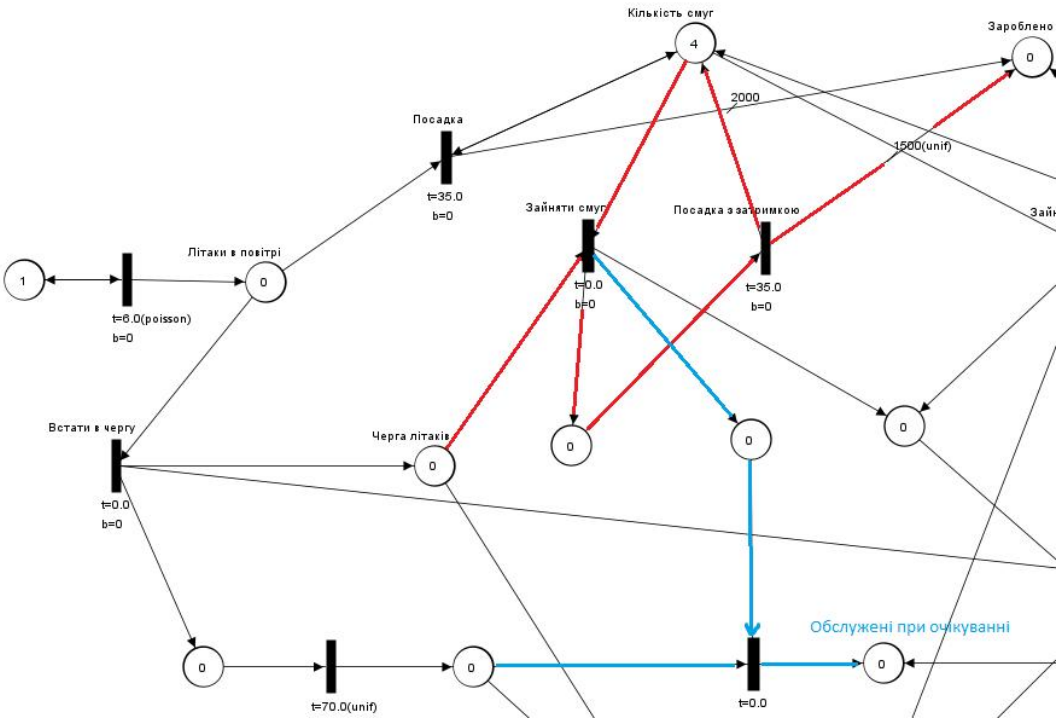

Рисунок 2.7 – Літак сідає на смугу після очікування до 70 ± 10 хв.

Якщо 70 ± 10 хв. сплинуло, а літак так і не зміг здійснити посадку, то він здійснює дозаправку, що поповнить місце “Втрачено” на 1000 ± 200, це і є витрати на дозаправку. Також літак з “Черги літаків” перейде до “Літаки після дозаправки”. (рисунок 2.8).

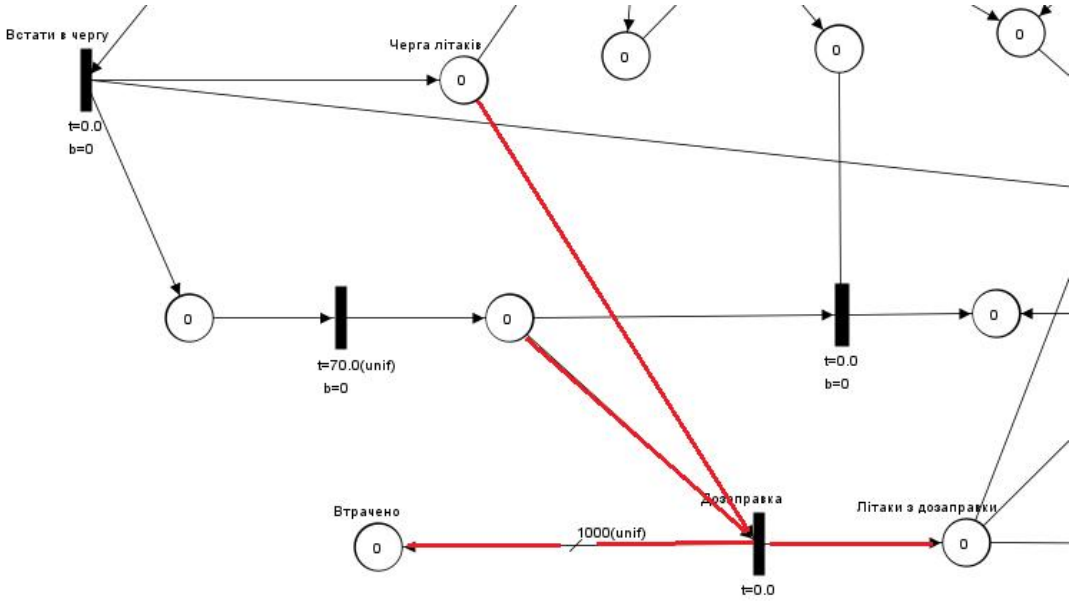

Рисунок 2.8 – Літак відправляється на дозаправку і встає у чергу після дозаправки

Зайняття смуги після дозаправки є більш пріоритетною задачею ніж посадка після очікування. Під час очікування у черзі літак може зайняти вільну смугу, і , відповідно, зробити посадку з затримкою, що принесе аеропорту 1500 ± 100 умовних грошових одиниць. Також у такому разі таймер на 140 хв. інформується, про те що літак вже здійснив посадку і він не буде відправлений на інший аеропорт. (рисунок 2.9.)

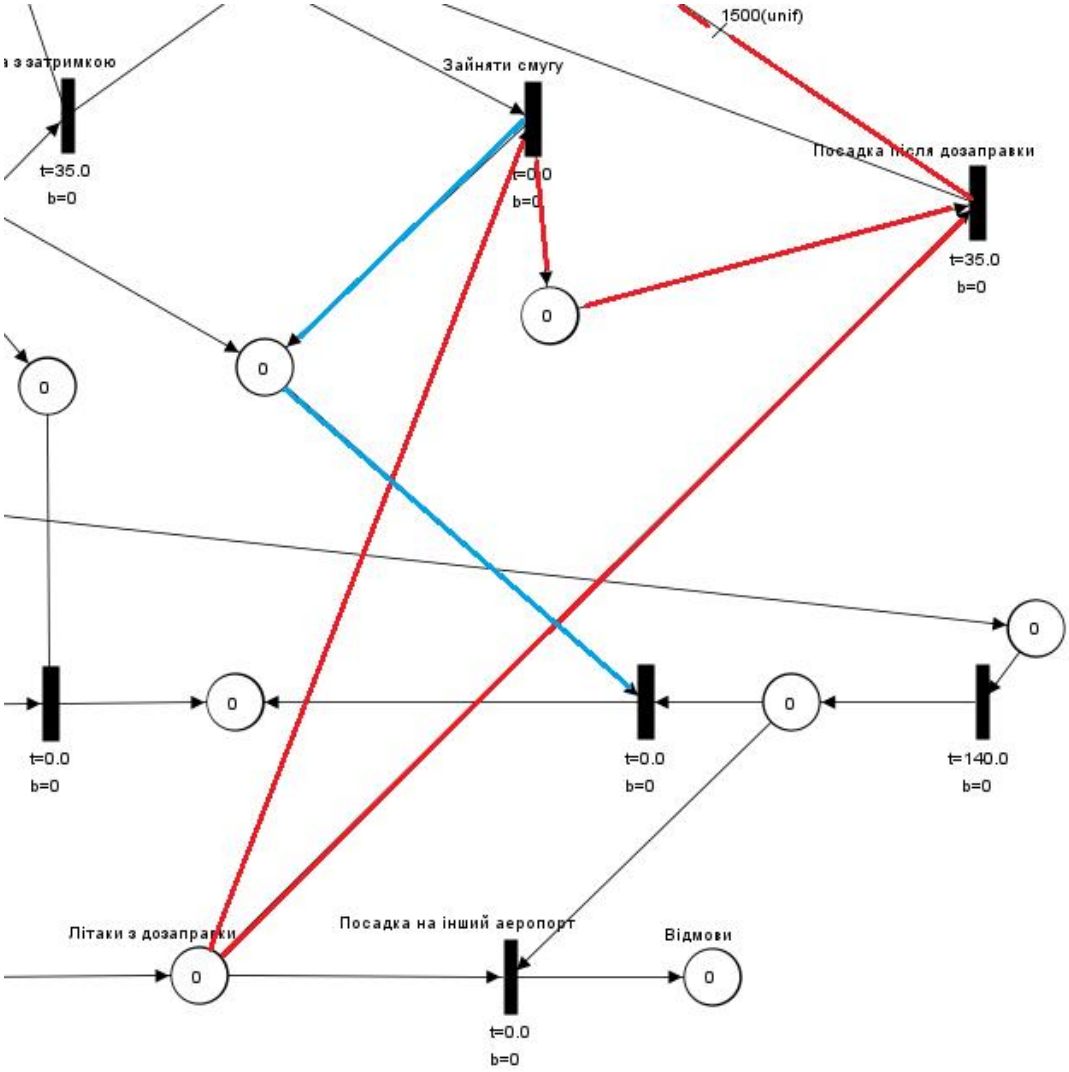

Рисунок 2.9 – Літак сідає на смугу після очікування до 140 хв.

Якщо 140 хв. сплинуло, а літак так і не зміг здійснити посадку, то він здійснить посадку на іншому аеропорті і поповнить місце “Відмови”. (рисунок 2.10).

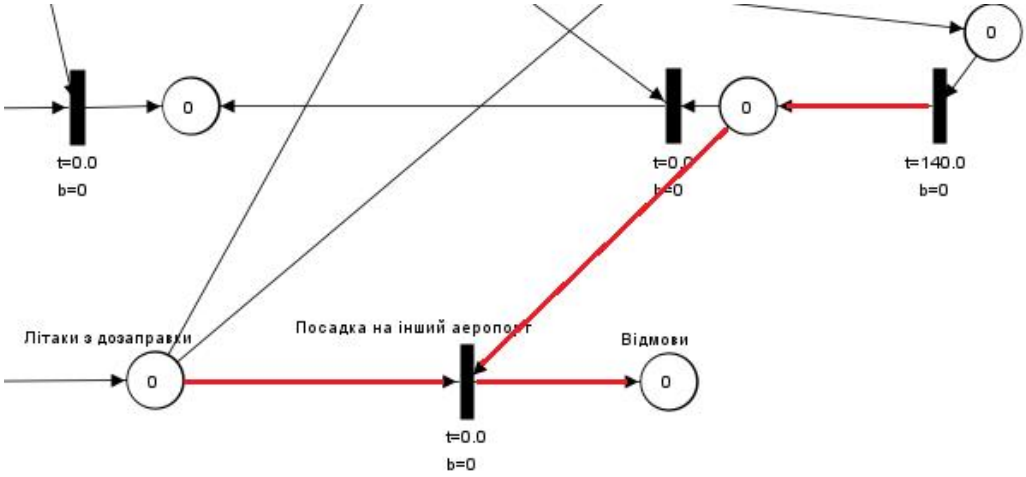

Рисунок 2.10 – Відмова, переліт літака на інший аеропорт.

Використовуючи значення з додаткових позицій, що збирають статистичну інформацію, можемо зібрати отримати вихідні змінні.

Кількість літаків, що здійснили перельот на інший аеропорт після моделювання буде визначено у місці “Відмови”.

Прибуток від обслуговування літаків після моделювання буде визначено у місці “Зароблено”.

Втрати аеропорту на дозаправку після моделювання будуть визначені у місці “Втрачено”.

Час окупності для встановленої кількості смуг можна розрахувати за формулою:

$$T = \frac{S}{p}$$

$S = 3000000 * \text{кількість додаткових смуг}$

$$p = A(\text{earned}) - A(\text{lost})$$

A(earned) - це прибуток аеропорту, A(lost) втрати на дозаправку

3 АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ

3.1 Опис ПЗ для реалізації моделі

Для побудови моделі на основі формалізму мереж Петрі та подальшого виконання алгоритму імітації в цій роботі було застосовано програмне забезпечення PetriObjModelPaint. Ця програма пропонує графічний інтерфейс, який дає можливість створити необхідну мережу та отримати статистичні дані, придатні для аналізу функціонування моделі системи.

Ключова ідея PetriObjModelPaint полягає у використанні стохастичної багатоканальної мережі Петрі з ймовірнісними переходами. Така концепція добре відповідає вимогам опису поведінки розглянутої системи і базується на об’єктно-орієнтованих принципах проектування.

Програмне забезпечення розроблено мовою Java[4] із застосуванням бібліотеки Swing[5] для реалізації графічного інтерфейсу та спеціалізованого пакету PetriObj. Цей пакет містить реалізацію основної логіки: створення мережі Петрі та алгоритму її імітації. Коротка характеристика основних класів пакету PetriObj, які були явно використані у роботі наведена у таблиці 3.1.

Таблиця 3.1 – Опис класів

Клас	**Опис**
ArcIn	Дуга від місця до переходу, може бути звичайною або інформаційною
ArcOut	Дуга з переходу до місця, може

	бути звичайною або інформаційною

Продовження таблиці 3.1

FunRand	Містить статичні методи-генератори чисел за різними законами розподілу
PetriNet	Головний клас, який збирає усі інші елементи у одну мережу Петрі
PetriP	Позиція у Петрі моделі
PetriSim	Містить інструменти для симуляції мережі Петрі, протягом заданого часу.
PetriT	Перехід у мережі Петрі
StateTime	Поточний час симуляції
PetriObjModel	Головний клас для конструювання Петрі-об’єктної моделі.

Просування часу в імітації мережі Петрі в PetriObjModelPaint відбувається по принципу найближчої події, тобто використовується не реальний час, а імітація у часових одиницях, що зручно для поставленої задачі і дозволяє моделювати систему впродовж навіть віртуального року.

Також застосунок має графічний інтерфейс для мануальної побудови мережі петрі.

3.2 Опис модифікацій ПЗ

Задля виконання роботи було додано невелику модифікацію ArcIn і ArcOut. Для них було додано новий конструктор з вказанням розподілу, приклад у ArcOut:

```
public ArcOut(PetriT T, PetriP P, String distribution, double param1, double param2)
{
    this.numP = P.getNumber();
    this.numT = T.getNumber();
    this.distribution = distribution;
    this.param1 = param1;
    this.param2 = param2;
    this.useDistribution = true;
}
```

Також було додано вспоміжну функцію для отримання цілочисельного числового значення з урахунком розподілу.

```
private int calculateWeight() {
    double value = 1.0;
    try {
        switch (distribution.toLowerCase()) {
            case "exp":
                value = FunRand.exp(param1);
                break;
            case "unif":
                value = FunRand.unif(param1, param2);
                break;
            case "norm":
                value = FunRand.norm(param1, param2);
                break;
            case "poisson":
                value = FunRand.poisson(param1);
                break;
            default:
                throw new IllegalArgumentException("Unknown distribution type: " +
distribution);
        }
    } catch (Exception e) {
        System.err.println("Error calculating weight: " + e.getMessage());
    }
    return Math.max(1, (int) Math.ceil(value));
}
```

Трохи змінено getQuantity(), щоб враховувати випадок, коли дуга має розподіл ваги:

```
public int getQuantity() {
   if (useDistribution) {
      return calculateWeight();
   }
   return k;
}
```

Використання:

```
d_Out.add(new ArcOut(d_T.get(7),d_P.get(6),"unif", 800, 1200));
```

Модифікуємо клас PetriObjModel, що описує універсальний алоритм імітації для Петрі-об'єктної моделі[5]. Оскільки в цій роботі використано формалізм мережі Петрі, то задля пришвидшення роботи алгоритму функціонал буде спрощений.

Головні зміни:

1. Прибрано функціональність Petri-об'єктних зв'язків:

В оригінальному коді були посилання на спеціальний клас LinkByPlaces, а також можливість об'єднувати місця різних Petri-об'єктів (через метод linkObjectsCombiningPlaces). У спрощеній версії ці можливості повністю вилучені. Тепер PetriObjModel просто оперує списком PetriSim-об'єктів незалежно один від одного, без синхронізації їх місць.

2. Вилучено унікальний ідентифікатор id:

У первісному коді PetriObjModel мав поле id, яке могло бути корисним для ідентифікації моделі на сервері чи в розподіленій системі. У спрощеній версії це поле та всі пов'язані з ним методи та логіка видалені, оскільки для базової Petri-моделі це не є принципово необхідним.

3. Видалено логіку клонування та спільних місць:

Оригінальний код містив методи для клонування моделі (clone()) та відновлення зв'язків між спільними місцями при клонуванні. Ця логіка тепер не потрібна, оскільки ми більше не використовуємо «об'єктні» Petri-моделі та не об'єднуємо місця. Відповідно, клонування та пов'язані з ним операції вилучено.

4. Спрощено метод go():

У початковому коді метод go(double timeModeling) реалізовував складну логіку, зокрема пов'язану з відновленням станів, обробкою посилань тощо. У спрощеному варіанті зосереджено лише на базових кроках моделювання:

5. Прибрано зайві змінні та методи:

Оскільки більше не потрібно зберігати або відновлювати інформацію про посилання між Petri-об'єктами, видалено поля links та методи linkObjectsCombiningPlaces, clearLinks, printLinks тощо.

Також для верифікації та щоб мати змогу використовувати модель для тестування без графічного редактора в програмі PetriObjModelPaint є збережемо модель як метод у класі NetLibrary, цей метод дозволить передавати вхідні параметри у модель для верифікацій.

```
public static PetriNet VerificationModel(int stripesCount, int intensivity, int
serviceTime, int refillTime, int migrationTime) throws ExceptionInvalidNetStructure,
ExceptionInvalidTimeDelay {
    ArrayList<PetriP> d_P = new ArrayList<>();
    ArrayList<PetriT> d_T = new ArrayList<>();
    ArrayList<ArcIn> d_In = new ArrayList<>();
    ArrayList<ArcOut> d_Out = new ArrayList<>();
    d_P.add(new PetriP("",1));
    d_P.add(new PetriP("Літаки в повітрі",0));
    d_P.add(new PetriP("Кількість смуг",stripesCount));
```

```
d_P.add(new PetriP("Черга літаків",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Втрачено",0));
d_P.add(new PetriP("Літаки з дозаправки",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Відмови",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Зароблено",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_T.add(new PetriT("",intensivity));
d_T.get(0).setDistribution("poisson", d_T.get(0).getTimeServ());
d_T.get(0).setParamDeviation(0.0);
d_T.add(new PetriT("Посадка",serviceTime));
d_T.get(1).setPriority(1);
d_T.add(new PetriT("Посадка з затримкою",serviceTime));
d_T.add(new PetriT("Посадка після дозаправки",serviceTime));
d_T.get(3).setPriority(3);
d_T.add(new PetriT("Встати в чергу",0.0));
d_T.add(new PetriT("",refillTime));
d_T.get(5).setDistribution("unif", d_T.get(5).getTimeServ());
d_T.get(5).setParamDeviation(10.0);
d_T.add(new PetriT("",0.0));
d_T.get(6).setPriority(2);
d_T.add(new PetriT("Дозаправка",0.0));
d_T.add(new PetriT("",migrationTime));
d_T.add(new PetriT("",0.0));
d_T.get(9).setPriority(2);
d_T.add(new PetriT("Посадка на інший аеропорт",0.0));
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(11).setPriority(2);
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(12).setPriority(3);
d_In.add(new ArcIn(d_P.get(5),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(12),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(1),d_T.get(4),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(0),d_T.get(0),1));
d_In.add(new ArcIn(d_P.get(5),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(9),d_T.get(8),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(3),1));
d_In.add(new ArcIn(d_P.get(15),d_T.get(3),1));
```

```
    d_In.add(new ArcIn(d_P.get(13),d_T.get(2),1));
    d_In.add(new ArcIn(d_P.get(7),d_T.get(10),1));
    d_In.add(new ArcIn(d_P.get(8),d_T.get(10),1));
    d_In.add(new ArcIn(d_P.get(4),d_T.get(5),1));
    d_In.add(new ArcIn(d_P.get(8),d_T.get(9),1));
    d_In.add(new ArcIn(d_P.get(16),d_T.get(9),1));
    d_In.add(new ArcIn(d_P.get(1),d_T.get(1),1));
    d_In.add(new ArcIn(d_P.get(2),d_T.get(1),1));
    d_Out.add(new ArcOut(d_T.get(6),d_P.get(10),1));
    d_Out.add(new ArcOut(d_T.get(4),d_P.get(3),1));
    d_Out.add(new ArcOut(d_T.get(4),d_P.get(4),1));
    d_Out.add(new ArcOut(d_T.get(4),d_P.get(9),1));
    d_Out.add(new ArcOut(d_T.get(12),d_P.get(15),1));
    d_Out.add(new ArcOut(d_T.get(12),d_P.get(16),1));
    d_Out.add(new ArcOut(d_T.get(0),d_P.get(0),1));
    d_Out.add(new ArcOut(d_T.get(0),d_P.get(1),1));
    d_Out.add(new ArcOut(d_T.get(7),d_P.get(6),"unif", 800, 1200));
    d_Out.add(new ArcOut(d_T.get(7),d_P.get(7),1));
    d_Out.add(new ArcOut(d_T.get(8),d_P.get(8),1));
    d_Out.add(new ArcOut(d_T.get(11),d_P.get(12),1));
    d_Out.add(new ArcOut(d_T.get(11),d_P.get(13),1));
    d_Out.add(new ArcOut(d_T.get(11),d_P.get(16),1));
    d_Out.add(new ArcOut(d_T.get(3),d_P.get(14),"unif", 1400, 1600));
    d_Out.add(new ArcOut(d_T.get(3),d_P.get(2),1));
    d_Out.add(new ArcOut(d_T.get(2),d_P.get(2),1));
    d_Out.add(new ArcOut(d_T.get(2),d_P.get(14),"unif", 1400, 1600));
    d_Out.add(new ArcOut(d_T.get(10),d_P.get(11),1));
    d_Out.add(new ArcOut(d_T.get(5),d_P.get(5),1));
    d_Out.add(new ArcOut(d_T.get(9),d_P.get(10),1));
    d_Out.add(new ArcOut(d_T.get(1),d_P.get(2),1));
    d_Out.add(new ArcOut(d_T.get(1),d_P.get(14),2000));
    PetriNet d_Net = new PetriNet("model2",d_P,d_T,d_In,d_Out);
    PetriP.initNext();
    PetriT.initNext();
    ArcIn.initNext();
    ArcOut.initNext();
    return d_Net;
}
```

Також для верифікації було написано метод verify() для виконання 3х прогонів на визначених наборах даних:

```
public static void verify() throws ExceptionInvalidTimeDelay,
ExceptionInvalidNetStructure {
    int TIME = 525600;
    int S = 3000000;
    // Змінні параматери для верифікації
    int stripes = 4;
    int intensivity = 6;
```

```
    int serviceTime = 35;
    int refillTime = 70;
    int migrationTime = 140;
    for (int i = 0; i <= 3; i++) {
        ArrayList<PetriSim> list = new ArrayList<>();
        list.add(new PetriSim(NetLibrary.VerificationModel(stripes, intensivity,
serviceTime, refillTime, migrationTime)));
        PetriObjModel model = new PetriObjModel(list);
        model.setIsProtokol(false);
        model.go(TIME);
        int migratedFleets =
model.getListObj().get(0).getNet().getListP()[11].getObservedMax();
        int earned =
model.getListObj().get(0).getNet().getListP()[14].getObservedMax();
        int lost = model.getListObj().get(0).getNet().getListP()[6].getObservedMax();
        int profit = earned - lost;
        double profitTime = ((double) (S*stripes - S*2) / profit);
        System.out.println("Кількість літаків, що здійснили перельоти на інший
аеропорт: "+migratedFleets);
        System.out.println("Прибуток від обслуговування літаків: "+earned);
        System.out.println("Втрати аеропорту на дозаправку: "+lost);
        System.out.println("Час окупності: "+profitTime);
    }
}
```

3.3 Верифікація моделі

Перед початком експериментів над моделлю, варто перевірити правильність роботи створеного алгоритму імітації. Це гарантує, що модель правильно описує поведінку системи, не містить логічних помилок, «пасток» та забезпечує відповідність початковим вимогам. Вона дозволяє виявити й виправити проблеми на ранній стадії, підвищує надійність, безпечність та ефективність майбутньої системи.

Для цього побудуємо таблицю верифікації і будемо змінювати вхідні змінні по одному параметру за одну пачку прогонів з 20 і будемо слідкувати за вихідними змінними. Усі прогони будуть проводитись на 525600 умовних одиниць часу, що відповідає року, тому що ми використовували хвилину, як одиницю часу, вхідні і вихідні змінні були визначені раніше.

Результати верифікації моделі наведено в таблиці 3.2.

Таблиця 3.2 – Верифікація моделі

A	B	C	D	E	F	G	H	I	J	K
Прогін	Кількість посадкових смуг	Інтенсивність надходження літаків	Час обслуговування літака на смузі	Час очікування перед дозаправкою	Час очікування перед перельотом		Кількість літаків, що здійснили перельот на інший аеропорт	Прибуток від обслуговування літаків	Втрати аеропорту на дозаправку	Час окупності
1	3	Poisson(6)	35	70 ± 10	140		42692	68740826	51233270	0.17135458541443477
2	4	Poisson(6)	35	70 ± 10	140		27328	85292046	32702370	0.1140908341021154
3	3	Poisson(5)	35	70 ± 10	140		60091	70366928	59834325	0.28482987538787896
4	3	Poisson(6)	30	70 ± 10	140		34973	81090090	35853312	0.0663177205060891
5	3	Poisson(6)	35	50 ± 10	140		42472	68066433	57216088	0.19063303001923743
6	3	Poisson(6)	35	70 ± 10	100		57051	59234323	51260160	0.2048343656456896

Першим прогін є контрольним і з ним уже будуть порівнюватись інші 5.

Щоб впевнитися у коректності роботи моделі потрібно провести аналіз результатів порівнюючи їх з початковим прогоном.

Перший експеримент проводився з підвищенням кількості посадкових смуг. Логічно, що від цього параметру залежать більшість інших параметрів. При підвищенні кількості смуг, значно покращується обслуговування: зменшується кількість відмов (перельотів на інший аеропорт), збільшується чистий прибуток, загалом так відбувається тому що менше літаків загалом йдуть у чергу, а ті що пішли, мають більше можливостей для посадки.

Другим був експеримент зі зменшенням інтенсивності надходження, через цю зміну аеропорт перестав справлятись з

об’ємом літаків, однак прибуток все одно виріс, більша обслужених літаків перекрила відмови і дозаправки.

Третій і 4й експерименти були над часом очікування у черзі. Тут очевидно, що при зменшенні часу очікування перед дозаправкою буде більше дозаправок і це збільшить втрати аеропорту. При зменшенні часу очікування до перельоту, маємо менше заробітку і більше відмов.

Загалом, можна зробити висновок, що модель працює правильно, зміна вхідних змінних логічно і закономірно змінює вихідні змінні. Також верифікація моделі показала, що з точки зору бізнесу будь-які навіть незначні зміни можуть сильно змінити результати.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ

4.1 Мета й опис експерименту

Головна мета проведення даного експерименту згідно завдання полягає у тому, щоб дізнатися оптимальну кількість посадкових смуг для аеропорту. Для цього буде проведено експеримент з кожною можливою кількостю смуги від 3 до 12. Також буде виконано декілька прогонів моделі для визначеннння точного значення необхідної кількості смуг.

Рисунок 4.1– Зміна кількості смуг аеропорту.

4.2 Тактичне планування експерименту

Проведемо тактичне планування експерименту. З завдання слідує, що час експерименту складає 1 рік. Для визначення перехідного періоду, щоб впевнитись що року достатньо для роботи моделі побудуємо графік залежності середнього значення “Черги літаків після дозаправки” у якості відгуку моделі від часу прогону, ця черга є найменш стабільною серед представлених у моделі. Зробимо 5 прогонів, визначеним часом моделювання від 1 дня до 1 року.

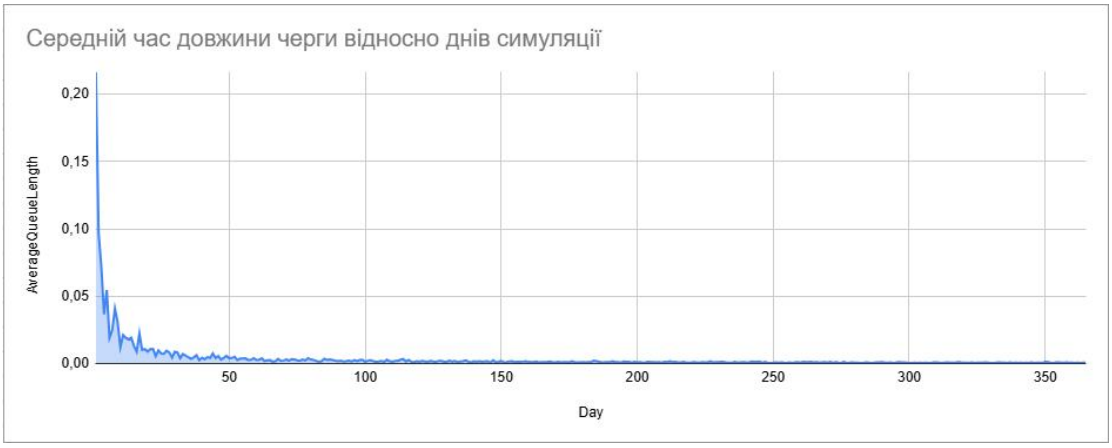

Рисунок 4.2 – Графік залежності

Бачимо, що починаючи з 50 дня (1440 * 50) симуляції значення відгуку прогонах стабілізується. Так як цільові вихідні змінні кумулятивні, ми не можемо відкинути частину статистики за рік, однак тепер ми можемо бути певні, що рік симуляції видає реальні результати.

Для визначення кількості прогонів використаємо нерівність Чебишева:

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

де σ^2 – дисперсія відгуку моделі, β – довірча ймовірність, ε – точність вимірювання відгуку моделі. β = 0,95, $\varepsilon = \sigma$. Звідси визначена кількість прогонів p = 20.

4.3 Проведення експерименту

Основною метою цього дослідження є визначення оптимальної кількості злітно-посадкових смуг, за якої час окупності буде найменшим. Кількість смуг у моделі для аналізу може змінюватися в межах від 3 до 12, дві смуги вже є за замовченням. Однією з обов’язкових умов є проведення симуляції протягом 1 року, де одна одиниця симуляції відповідає одній реальній хвилині. Таким чином.

Для проведення експерименту ми ініціалізуємо модель з необхідними вхідними параметрами. Кожного разу змінюється лише кількість злітно-посадкових смуг. У рамках експерименту реалізується цикл із 20 ітерацій, під час якого модель створюється заново для кожної ітерації з фіксованими параметрами. Після кожного запуску до загальної суми додається значення інтервалу окупності, отримане в ітерації. Після завершення 20 ітерацій середній інтервал окупності обчислюється шляхом ділення загальної суми на кількість ітерацій. Отримані результати у таблиці 4.1.

Таблиця 4.1 – Час окупності для кожної кількості смуг.

Кількість смуг	**Час окупності**
3	0,1202901774
4	0,09704830115
5	0,088753426
6	0,08557164058
7	0,08922402956
8	0,103403242
9	0,1199794867
10	0,1371629364
11	0,1541970805
12	0,1713148232

Також перенесемо ці дані на гістрограму для більшої наглядності (рисунок 4.3).

Рисунок 4.3 – Гістрограма часу окупності

З таблиці і діаграми бачимо, що найкращі результати при 6 смугах, така кількість легко пояснюється. При 6 смугах літаки, які генеруються Poisson(6) оброблюються практично без черги, а ті що все ж потрапили у чергу, оброблюються без дозаправки. Можемо впевнитись у цьому запустивши модель у графічному інтерфейсі з 6 смугами, рисунок 4.4.

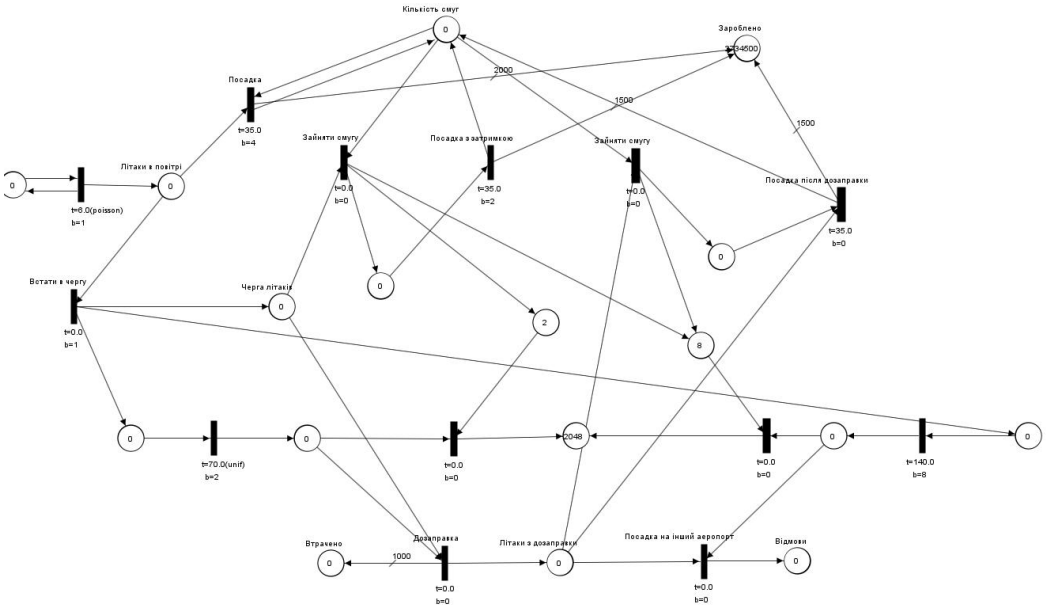

Рисунок 4.3 – Модель після симуляції

5 ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ

У результаті моделювання функціонування аеропорту з різною кількістю злітно-посадкових смуг вдалося досягти таких ключових висновків:

1. Оптимізація ресурсів

Найбільш ефективною конфігурацією є шість злітно-посадкових смуг. При такій кількості забезпечується мінімальний час окупності додаткових смуг, що підтверджується стабільно низьким рівнем черг та мінімальними витратами на дозаправки літаків.

2. Ефективність моделі

Модель виявила високу точність у прогнозуванні параметрів системи. Верифікація показала, що зміна кількості смуг та інших параметрів призводить до логічно обґрунтованих змін вихідних змінних.

3. Вплив черг та затримок

Зі збільшенням кількості смуг черги на обслуговування зменшуються, що безпосередньо впливає на зменшення витрат через дозаправку та перенаправлення літаків. Водночас надмірне збільшення смуг підвищує витрати на їх будівництво, що негативно впливає на час окупності.

4. Аналіз ризиків

При недостатній кількості смуг зростають витрати на перенаправлення літаків до інших аеропортів. У свою чергу, перевищення оптимальної кількості смуг не приносить додаткового прибутку, а лише збільшує витрати.

5. Рекомендації щодо покращення

Для забезпечення стабільного функціонування системи рекомендовано дотримуватись оптимального співвідношення кількості смуг до середньої інтенсивності прибуття літаків.

Для подальшої оптимізації роботи аеропорту можна дослідити:

- Вплив змінних, таких як різна швидкість обслуговування літаків чи непередбачувані затримки.
- Використання систем прогнозування для оптимізації використання смуг.
- Аналіз вартості перенаправлення літаків залежно від погодних умов чи іншого навантаження.

Результати моделювання демонструють ефективність розробленої імітаційної моделі та її здатність допомагати в ухваленні управлінських рішень щодо оптимізації роботи аеропорту.

ВИСНОВКИ

У процесі виконання роботи була створена імітаційна модель управління злітно-посадковими смугами аеропорту, яка базується на формалізмі мереж Петрі та реалізована за допомогою програмного забезпечення PetriObjModelPaint. Ця модель надає можливість дослідити ефективність роботи аеропорту в різних умовах та оцінити вплив ключових параметрів на продуктивність системи, таких як час обслуговування літаків, рівень затримок та економічна доцільність додаткових смуг.

З метою підтвердження коректності розробленої моделі було проведено її перевірку. Аналіз показав, що поведінка моделі логічно відповідає очікуваним змінам характеристик при варіації вхідних параметрів. Експерименти підтвердили, що оптимальною кількістю злітно-посадкових смуг в заданих умовах є шість, оскільки це забезпечує баланс між витратами на розширення інфраструктури та ефективністю обслуговування авіатранспорту.

Моделювання також виявило, що зменшення кількості смуг веде до зростання черг та збільшення кількості літаків, що перенаправляються до інших аеропортів, тоді як надмірне збільшення кількості смуг не виправдовує додаткових витрат через незначний приріст ефективності. Це підтверджує необхідність оптимального підходу до розвитку інфраструктури.

Статистичний аналіз результатів продемонстрував значний вплив кількості смуг на показник часу окупності системи, що підкреслює важливість точного планування параметрів роботи аеропорту для досягнення найкращих економічних показників.

Розроблена модель підтвердила цінність імітаційного підходу до аналізу складних систем, дозволивши отримати цінні дані для прийняття управлінських рішень. Використання мереж Петрі дало

змогу детально відтворити процеси системи, а також швидко проводити тестування та аналіз.

Для подальшого розвитку моделі варто врахувати можливі додаткові фактори, наприклад, змінну інтенсивність авіапотоків, ризики технічних несправностей або вплив погодних умов. Врахування таких елементів допоможе підвищити точність прогнозування та адаптувати модель до реальних умов функціонування аеропорту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PetriObjModel. GitHub repo.

URL:https://github.com/StetsenkoInna/PetriObjModelPaint

2. Пуасонівский розподіл. Wikipedia.

URL:https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB_%D0%9F%D1%83%D0%B0%D1%81%D1%81%D0%BE%D0%BD%D0%B0

3. Моделі Петрі. Стеценко І.В., Дифучин А.Ю. Дистанційний навчальний курс моделювання систем, коротка назва ws04jg, сертифікат ДК № 0097, затверджений Методичною радою КПІ ім. Ігоря Сікорського, протокол № 8 від 02.06.2023 р

4. Java Platform, Standard Edition .

URL: https://docs.oracle.com/en/java/

5. Swing, Swing documentation

URL: https://docs.oracle.com/javase/8/docs/api///?javax/swing/package-summary.html

ДОДАТКИ

Додаток А. Лістинг коду

ArcIn.java

```
package PetriObj;

import java.io.Serializable;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 * This class for creating the arc between place and transition of Petri net
 * (and directed from place to transion) numP - number of place numT - number of
 * transition k - arc multiplicity inf - flag of information arc
 *
 *  @author Inna V. Stetsenko
 */
public class ArcIn implements Cloneable, Serializable {

    private int numP;
    private int numT;
    private int k;
    private String distribution;
    private double param1;
    private double param2;
    private boolean useDistribution = false;
    boolean inf = false;
    private String nameP;
    private String nameT;
    private static int next = 0;
    private int number;

    // whether k and inf are parameters; added by Katya 08.12.2016
    private boolean kIsParam = false;
    private boolean infIsParam = false;
    // param names
    private String kParamName = null;
```

```
private String infParamName = null;
private boolean takeAllAndPassOne = false;


/**
 *
 */
public ArcIn() {
    k = 1;
    number = next;
    next++;
}

/**
 * @param P number of place
 * @param T number of transition
 * @param K arc multiplicity
 *
 */
public ArcIn(int P, int T, int K) {
    numP = P;
    numT = T;
    k = K;
    inf = false;
    number = next;
    next++;
}

/**
 *
 * @param P number of place
 * @param T number of transition
 */
public ArcIn(PetriP P, PetriT T) {
    numP = P.getNumber();
    numT = T.getNumber();
    k = 1;
    inf = false;
    nameP = P.getName();
    nameT = T.getName();
```

```
    number = next;
    next++;
}

/**
 *
 * @param P number of place
 * @param T number of transition
 * @param K arc multiplicity
 */
public ArcIn(PetriP P, PetriT T, int K) {
    numP = P.getNumber();
    numT = T.getNumber();
    k = K;
    inf = false;
    nameP = P.getName();
    nameT = T.getName();
    number = next;
    next++;
}

public ArcIn(PetriP place, PetriT transition, boolean takeAllAndPassOne) {
    this.numP = place.getNumber();
    this.numT = transition.getNumber();
    this.inf = false;
    this.nameP = place.getName();
    this.nameT = transition.getName();
    number = next;
    next++;

    if (takeAllAndPassOne) {
        this.k = -1;
    } else {
        this.k = 1;
    }
}

public ArcIn(PetriP place, PetriT transition, String distribution, double param1, double param2) {
    this.numP = place.getNumber();
    this.numT = transition.getNumber();
```

```
        this.distribution = distribution;
        this.param1 = param1;
        this.param2 = param2;
        this.useDistribution = true;
    }

    private int calculateWeight() {
        double value = 1.0;
        try {
            switch (distribution.toLowerCase()) {
                case "exp":
                    value = FunRand.exp(param1);
                    break;
                case "unif":
                    value = FunRand.unif(param1, param2);
                    break;
                case "norm":
                    value = FunRand.norm(param1, param2);
                    break;
                case "poisson":
                    value = FunRand.poisson(param1);
                    break;
                default:
                    throw new IllegalArgumentException("Unknown distribution type: " + distribution);
            }
        } catch (Exception e) {
            System.err.println("Error calculating weight: " + e.getMessage());
        }
        return Math.max(1, (int) Math.ceil(value)); // Округляем вверх до целого и гарантируем минимум 1
    }

    /**
     *
     * @param P number of place
     * @param T number of transition
     * @param K arc multiplicity
     * @param isInf arc is informational
     */
    public ArcIn(PetriP P, PetriT T, int K, boolean isInf) {
```

```
    numP = P.getNumber();
    numT = T.getNumber();
    k = K;
    inf = isInf;
    nameP = P.getName();
    nameT = T.getName();
    number = next;
    next++;
}

public ArcIn(ArcIn arcIn) {
    this(arcIn.getNumP(), arcIn.getNumT(), arcIn.getQuantity());
    inf = arcIn.getIsInf();
}

public boolean kIsParam() {
    return kIsParam;
}

public boolean infIsParam() {
    return infIsParam;
}

public String getKParamName() {
    return kParamName;
}

public String getInfParamName() {
    return infParamName;
}

public void setKParam(String paramName) {
    if (paramName == null) {
        kIsParam = false;
        kParamName = null;
    } else {
        kIsParam = true;
        kParamName = paramName;
        k = 1;
    }
```

```
}

public void setInfParam(String paramName) {
   if (paramName == null) {
      infIsParam = false;
      infParamName = null;
   } else {
      infIsParam = true;
      infParamName = paramName;
      inf = false;
   }
}
/**
 * Set the counter of input arcs to zero.
 */
public static void initNext(){ //ініціалізація лічильника нульовим значенням
      next = 0;
}

/**
 *
 * @return arc multiplicity
 */
public int getQuantity() {
 if (useDistribution) {
      return calculateWeight();
   }
   return k;
}

/**
 *
 * @param K value of arc multiplicity
 */
public void setQuantity(int K) {
   k = K;
}

/**
 *
```

```
 * @return the number of place that is the beginning of the arc
 */
public int getNumP() {
    return numP;
}

/**
 *
 * @param n number of place that is the beginning of the arc
 */
public void setNumP(int n) {
    numP = n;
}

/**
 *
 * @return number of transition that is the end of the arc
 */
public int getNumT() {
    return numT;
}

/**
 *
 * @param n number of transition that is the end of the arc
 */
public void setNumT(int n) {
    numT = n;
}

/**
 *
 * @return transition name
 */
public String getNameT() {
    return nameT;
}

/**
 *
```

```
 * @param s transition name
 */
public void setNameT(String s) {
   nameT = s;
}

/**
 *
 * @return name of place that is the beginning of the arc
 */
public String getNameP() {
   return nameP;
}

/**
 *
 * @param s name of place that is the beginning of the arc
 */
public void setNameP(String s) {
   nameP = s;
}

/**
 *
 * @return true if arc is informational
 */
public boolean getIsInf() {
   return inf;
}

/**
 *
 * @param i equals true if arc must be informational
 */
public void setInf(boolean i) {
   inf = i;
}

/**
 *
```

```
     */
    public void print() {
        if (nameP != null && nameT != null) {
            System.out.println(" P=   " + nameP + ", T=   " + nameT + ", inf= " + getIsInf() + ", k= " +
getQuantity());
        } else {
            System.out.println(" P= P" + numP + ", T= T" + numT + ", inf= " + getIsInf() + ", k= " +
getQuantity());
        }
    }

    public void printParameters() {
        System.out.println("This arc has direction from  place with number " + numP + " to transition with
number " + numT
                + " and has " + k + " value of multiplicity, ");
        if (inf == true) {
            System.out.println(" and is informational.");
        }
    }

    /**
     *
     * @return ArcIn object with parameters which copy current parameters of
 this arc
     * @throws java.lang.CloneNotSupportedException if Petri net has invalid structure
     */
    @Override
    public ArcIn clone() throws CloneNotSupportedException {
        super.clone();
        ArcIn arc = new ArcIn(numP, numT, k);  // коректніть номерів дуже важлива!!!
        return arc;

    }
}
```

ArcOut.java

```
package PetriObj;

import java.io.Serializable;

/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 * This class for creating the arc between transition and place of Petri
 * net (and directed from transion to place)
 *
 *  @author Inna V. Stetsenko
 */
public class ArcOut implements Cloneable, Serializable {

    private int numP;
    private int numT;
    private int k;
    private String distribution;
    private double param1;
    private double param2;
    private boolean useDistribution;
    private String nameT;
    private String nameP;
    private static int next = 0;
    private int number;

    // whether k is a parameter; added by Katya 08.12.2016
    private boolean kIsParam = false;
    // param name
    private String kParamName = null;

    /**
     *
     */
    public ArcOut() {
        k = 1;
        number = next;
        next++;
    }

    /**
     * @param T number of transition
     * @param P number of place
```

```
 * @param K arc multiplicity
 */
public ArcOut(int T, int P, int K) {
    numP = P;
    numT = T;
    k = K;
    number = next;
    next++;
}

/**
 *
 * @param T number of transition
 * @param P number of place
 * @param K arc multiplicity
 */
public ArcOut(PetriT T, PetriP P, int K) {
    numP = P.getNumber();
    numT = T.getNumber();
    k = K;
    nameP = P.getName();
    nameT = T.getName();
    number = next;
    next++;
}

public ArcOut(PetriT T, PetriP P, String distribution, double param1, double param2) {
    this.numP = P.getNumber();
    this.numT = T.getNumber();
    this.distribution = distribution;
    this.param1 = param1;
    this.param2 = param2;
    this.useDistribution = true;
}

public ArcOut(ArcOut arcOut) {
    this(arcOut.getNumT(), arcOut.getNumP(), arcOut.getQuantity());
}

public boolean kIsParam() {
```

```
    return kIsParam;
}

public String getKParamName() {
    return kParamName;
}

public void setKParam(String paramName) {
    if (paramName == null) {
        kIsParam = false;
        kParamName = null;
    } else {
        kIsParam = true;
        kParamName = paramName;
        k = 1;
    }
}


/**
 * Set the counter of output arcs to zero.
 */
public static void initNext() //ініціалізація лічильника нульовим значенням
{
    next = 0;
}

/**
 *
 * @return arc multiplicity
 */
public int getQuantity() {
    if (useDistribution) {
        return calculateWeight();
    }
    return k;
}

private int calculateWeight() {
    double value = 1.0;
```

```
    try {
        switch (distribution.toLowerCase()) {
            case "exp":
                value = FunRand.exp(param1);
                break;
            case "unif":
                value = FunRand.unif(param1, param2);
                break;
            case "norm":
                value = FunRand.norm(param1, param2);
                break;
            case "poisson":
                value = FunRand.poisson(param1);
                break;
            default:
                throw new IllegalArgumentException("Unknown distribution type: " + distribution);
        }
    } catch (Exception e) {
        System.err.println("Error calculating weight: " + e.getMessage());
    }
    return Math.max(1, (int) Math.ceil(value));
}

/**
 *
 * @param K arc multiplicity
 */
public void setQuantity(int K) {
    k = K;
}

/**
 *
 * @return the number of place that is end of the arc
 */
public int getNumP() {
    return numP;
}

/**
```

```
 *
 * @param n the number of place that is end of the arc
 */
public void setNumP(int n) {
    numP = n;
}

/**
 *
 * @return number of transition that is beginning of the arc
 */
public int getNumT() {
    return numT;
}

/**
 *
 * @param n number of transition that is beginning of the arc
 */
public void setNumT(int n) {
    numT = n;
}

/**
 *
 * @return name of transition that is the beginning of the arc
 */
public String getNameT() {
    return nameT;
}

/**
 *
 * @param s name of transition that is the beginning of the arc
 */
public void setNameT(String s) {
    nameT = s;
}

/**
```

```
     *
     * @return name of place that is the end of the arc
     */
    public String getNameP() {
        return nameP;
    }

    /**
     *
     * @param s name of place that is the end of the arc
     */
    public void setNameP(String s) {
        nameP = s;
    }

    /**
     *
     */
    public void print() {
        if (nameP != null && nameT != null) {
            System.out.println(" T=  " + nameT + ", P=  " + nameP + ", k= " + getQuantity());
        } else {
            System.out.println(" T= T" + numT + ", P= P" + numP + ", k= " + getQuantity());
        }
    }

    /**
     *
     * @return ArcOut object with parameters which copy current parameters of
this arc
     * @throws java.lang.CloneNotSupportedException if Petri net has invalid structure
     */
    @Override
    public ArcOut clone() throws CloneNotSupportedException {
        super.clone();
        ArcOut arc = new ArcOut(numT, numP, k); // коректніть номерів дуже важлива!!!
        return arc;

    }
```

```
    public void printParameters() {
        System.out.println("This arc has direction from  transition  with number " + numT + " to place with number " + numP
                + " and has " + k + " value of multiplicity");
    }

}
```

PetriObjModel.java(Спрощена версія)

```
/*
 * Спрощена модель Petri, яка оперує списком PetriSim.
 * Залишено використання StateTime для відліку часу.
 * Видалено функціонал Petri-об’єктних моделей.
 */
package PetriObj;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import javax.swing.JTextArea;

public class PetriObjModel implements Serializable {

    private ArrayList<PetriSim> listObj = new ArrayList<>();
    private boolean protocolPrint = true;
    private boolean statistics = true;
    private StateTime timeState;  // використання часу через StateTime

    public PetriObjModel(ArrayList<PetriSim> listObj) {
        this(listObj, new StateTime());
    }

    public PetriObjModel(ArrayList<PetriSim> listObj, StateTime timeState) {
        this.listObj = listObj;
        this.timeState = timeState;
        this.listObj.forEach(sim -> sim.setTimeState(timeState));
    }

    /**
     * Встановити потребу виведення протоколу.
```

```
 */
public void setIsProtokol(boolean b) {
    this.protocolPrint = b;
}

/**
 * Встановити потребу збору статистики.
 */
public void setIsStatistics(boolean b) {
    this.statistics = b;
}

public ArrayList<PetriSim> getListObj() {
    return listObj;
}

public void setListObj(ArrayList<PetriSim> List) {
    listObj = List;
    this.listObj.forEach(sim -> sim.setTimeState(timeState));
}

/**
 * Запустити моделювання до часу timeModeling.
 * Якщо ввімкнуто протокол, він буде виводитися у консоль.
 * Якщо ввімкнуто статистику, вона оновлюватиметься.
 */
public void go(double timeModeling) {
    setSimulationTime(timeModeling);
    setCurrentTime(0.0);

    // Сортуємо об'єкти за пріоритетом
    getListObj().sort(PetriSim.getComparatorByPriority());

    // Ініціалізуємо вхідні переходи
    for (PetriSim e : getListObj()) {
        e.input();
    }

    if (protocolPrint) {
        for (PetriSim e : getListObj()) {
```

```
            e.printMark();
        }
    }

    ArrayList<PetriSim> conflictObj = new ArrayList<>();
    Random r = new Random();

    while (getCurrentTime() < getSimulationTime()) {
        conflictObj.clear();
        double min = getListObj().get(0).getTimeMin();

        for (PetriSim e : getListObj()) {
            if (e.getTimeMin() < min) {
                min = e.getTimeMin();
            }
        }

        // Збір статистики
        if (statistics && min > 0) {
            double delta = Math.min(min - getCurrentTime(), getSimulationTime() - getCurrentTime());
            for (PetriSim e : getListObj()) {
                if (delta > 0) {
                    e.doStatistics(delta / min);
                }
            }
        }

        setCurrentTime(min);
        if (protocolPrint) {
            System.out.println(" Time progress: time = " + getCurrentTime() + "\n");
        }

        if (getCurrentTime() <= getSimulationTime()) {
            for (PetriSim sim : getListObj()) {
                if (getCurrentTime() == sim.getTimeMin()) {
                    conflictObj.add(sim);
                }
            }

            if (protocolPrint) {
```

```
    System.out.println(" Conflicting objects:");
    for (int ii = 0; ii < conflictObj.size(); ii++) {
        System.out.println(" K[" + ii + "] = " + conflictObj.get(ii).getName());
    }
}

int num;
if (conflictObj.size() > 1) {
    conflictObj.sort(PetriSim.getComparatorByPriority());
    int max = conflictObj.size();
    for (int i = 1; i < conflictObj.size(); i++) {
        if (conflictObj.get(i).getPriority() < conflictObj.get(i - 1).getPriority()) {
            max = i - 1;
            break;
        }
    }
    num = (max == 0) ? 0 : r.nextInt(max);
} else {
    num = 0;
}

PetriSim chosen = conflictObj.get(num);
if (protocolPrint) {
    System.out.println(" Selected object: " + chosen.getName());
}

for (PetriSim sim : getListObj()) {
    if (sim.getNumObj() == chosen.getNumObj()) {
        if (protocolPrint) {
            System.out.println(" time = " + getCurrentTime() +
                    " Event "' + sim.getEventMin().getName() + "' " +
                    "occurring for the object " + sim.getName());
        }
        sim.doT();
        sim.output();
    }
}

if (protocolPrint) {
    System.out.println("Markers output:");
```

```
            for (PetriSim sim : getListObj()) {
                sim.printMark();
            }
        }

        Collections.shuffle(getListObj());
        getListObj().sort(PetriSim.getComparatorByPriority());

        for (PetriSim e : getListObj()) {
            e.input();
        }

        if (protocolPrint) {
            System.out.println("Markers input:");
            for (PetriSim e : getListObj()) {
                e.printMark();
            }
        }
    }
  }

  getListObj().sort(PetriSim.getComparatorByNum());
}

public void printStatistics() {
  System.out.println("State of places and transitions:");
  for (PetriSim e : listObj) {
    e.printMark();
    e.printBuffer();
  }
  if (statistics) {
    for (PetriSim e : listObj) {
      System.out.println("\nMean values for " + e.getName() + ":");
      for (PetriP p: e.getNet().getListP()) {
        System.out.println(p.getName() + "  " + p.getMean());
      }
      for (PetriT tr: e.getNet().getListT()) {
        System.out.println(tr.getName() + "  " + tr.getMean());
      }
    }
```

```
    }
}

public void setSimulationTime(double t) {
    getTimeState().setSimulationTime(t);
    for (PetriSim sim: getListObj()) {
        sim.setSimulationTime(t);
    }
}

public double getSimulationTime() {
    return getTimeState().getSimulationTime();
}

public void setCurrentTime(double t) {
    getTimeState().setCurrentTime(t);
    for(PetriSim sim: this.listObj) {
        sim.setTimeCurr(t);
    }
}

public double getCurrentTime() {
    return getTimeState().getCurrentTime();
}

public StateTime getTimeState() {
    return timeState;
}

public void setTimeState(StateTime timeState) {
    this.timeState = timeState;
    this.listObj.forEach(sim -> sim.setTimeState(timeState));
}

public boolean isProtocolPrint() {
    return protocolPrint;
}

public boolean isStatistics() {
    return statistics;
```

```
    }

    public void printInfo(String info, JTextArea area) {
        if (protocolPrint) {
            area.append(info);
        }
    }

    public void printMark(JTextArea area) {
        if (protocolPrint) {
            for (PetriSim e : listObj) {
                e.printMark(area);
            }
        }
    }
}
```

Coursework.java(Запуск моделей з заданими параметрами)

```
package LibTest;
import PetriObj.*;
import LibNet.NetLibrary;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class Coursework {
    public static void main(String[] args) throws ExceptionInvalidTimeDelay,
ExceptionInvalidNetStructure {
//        verify();
// experiment();
        getMeanValuesTest();
    }
    public static void verify() throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {
        int TIME = 525600;
        int S = 3000000;

        // Змінні параматери для верифікації
        int stripes = 4;
        int intensivity = 6;
        int serviceTime = 35;
```

```
        int refillTime = 70;
        int migrationTime = 140;

        for (int i = 0; i <= 3; i++) {
            ArrayList<PetriSim> list = new ArrayList<>();
            list.add(new PetriSim(NetLibrary.VerificationModel(stripes, intensivity, serviceTime,
refillTime, migrationTime)));
            PetriObjModel model = new PetriObjModel(list);
            model.setIsProtokol(false);
            model.go(TIME);

            int migratedFleets = model.getListObj().get(0).getNet().getListP()[11].getObservedMax();
            int earned = model.getListObj().get(0).getNet().getListP()[14].getObservedMax();
            int lost = model.getListObj().get(0).getNet().getListP()[6].getObservedMax();
            int profit = earned - lost;
            double profitTime = ((double) (S*stripes - S*2) / profit);
            System.out.println("Кількість літаків, що здійснили перельоти на інший аеропорт:
"+migratedFleets);
            System.out.println("Прибуток від обслуговування літаків: "+earned);
            System.out.println("Втрати аеропорту на дозаправку: "+lost);
            System.out.println("Час окупності: "+profitTime);
        }
    }
    public static void experiment() throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {
        int TIME = 525600;
        int S = 3000000;

        int startJ = 3;
        int endJ = 12;
        int countJ = endJ - startJ + 1;

        double[] profitSum = new double[countJ];

        for (int i = 1; i <= 20; i++) {
            int index = 0;

            for (int j = startJ; j <= endJ; j++) {
                ArrayList<PetriSim> list = new ArrayList<>();
                list.add(new PetriSim(NetLibrary.CourseworkModel(j)));
                PetriObjModel model = new PetriObjModel(list);
```

```
            model.setIsProtokol(false);
            model.go(TIME);

            int earned = model.getListObj().get(0).getNet().getListP()[14].getObservedMax();
            int lost = model.getListObj().get(0).getNet().getListP()[6].getObservedMax();
            int profit = earned - lost;

            double profitTime = ((double) (S*j - S*2) / profit);

            profitSum[index] += profitTime;
            index++;
        }
    }

    System.out.println("Середній час окупності для кожної довжини смуги (за 20 прогонів):");
    int jVal = startJ;
    for (int i = 0; i < countJ; i++) {
        double avgProfit = profitSum[i] / 20.0;
        System.out.println("Довжина смуги " + jVal + ": " + avgProfit);
        jVal++;
    }
  }
  public static void getMeanValuesTest() throws ExceptionInvalidTimeDelay,
ExceptionInvalidNetStructure {
    int MAX_TIME = 525600;
    int MIN_TIME = 1440;
    int step = 1440;

    int stripes = 3;
    int intensivity = 6;
    int serviceTime = 35;
    int refillTime = 70;
    int migrationTime = 140;

    ArrayList<Double> averageQueueLengths = new ArrayList<>();

    for (int i = MIN_TIME; i <= MAX_TIME; i += step) {
        double totalQueueLength = 0.0;

        for (int run = 0; run < 20; run++) {
```

```
            ArrayList<PetriSim> list = new ArrayList<>();
            list.add(new PetriSim(NetLibrary.VerificationModel(stripes, intensivity, serviceTime, refillTime, migrationTime)));
            PetriObjModel model = new PetriObjModel(list);
            model.setIsProtokol(false);
            model.go(i);
            double queueLength = model.getListObj().get(0).getNet().getListP()[7].getMean();

            totalQueueLength += queueLength;
        }

        double averageQueueLength = totalQueueLength / 20.0;
        averageQueueLengths.add(averageQueueLength);
    }

    writeDataToCsv(averageQueueLengths, "average_queue_lengths.csv");
  }


  private static void writeDataToCsv(ArrayList<Double> data, String fileName) {
    try (FileWriter writer = new FileWriter(fileName)) {
      writer.write("AverageQueueLength\n");

      for (Double value : data) {
        writer.write(value + "\n");
      }

      System.out.println("Дані записано: " + fileName);
    } catch (IOException e) {
      System.err.println("Помилка: " + e.getMessage());
    }
  }
}
```

Основна модель вже з заданими параметрами і її ж версія зі змінними параметрами для верифікації

```
        public static PetriNet VerificationModel(int stripesCount, int intensivity, int serviceTime, int refillTime, int migrationTime) throws ExceptionInvalidNetStructure, ExceptionInvalidTimeDelay {
                ArrayList<PetriP> d_P = new ArrayList<>();
                ArrayList<PetriT> d_T = new ArrayList<>();
```

```
ArrayList<ArcIn> d_In = new ArrayList<>();
ArrayList<ArcOut> d_Out = new ArrayList<>();
d_P.add(new PetriP("",1));
d_P.add(new PetriP("Літаки в повітрі",0));
d_P.add(new PetriP("Кількість смуг",stripesCount));
d_P.add(new PetriP("Черга літаків",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Втрачено",0));
d_P.add(new PetriP("Літаки з дозаправки",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Відмови",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("Зароблено",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",0));
d_T.add(new PetriT("",intensivity));
d_T.get(0).setDistribution("poisson", d_T.get(0).getTimeServ());
d_T.get(0).setParamDeviation(0.0);
d_T.add(new PetriT("Посадка",serviceTime));
d_T.get(1).setPriority(1);
d_T.add(new PetriT("Посадка з затримкою",serviceTime));
d_T.add(new PetriT("Посадка після дозаправки",serviceTime));
d_T.get(3).setPriority(3);
d_T.add(new PetriT("Встати в чергу",0.0));
d_T.add(new PetriT("",refillTime));
d_T.get(5).setDistribution("unif", d_T.get(5).getTimeServ());
d_T.get(5).setParamDeviation(10.0);
d_T.add(new PetriT("",0.0));
d_T.get(6).setPriority(2);
d_T.add(new PetriT("Дозаправка",0.0));
d_T.add(new PetriT("",migrationTime));
d_T.add(new PetriT("",0.0));
d_T.get(9).setPriority(2);
d_T.add(new PetriT("Посадка на інший аеропорт",0.0));
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(11).setPriority(2);
```

```
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(12).setPriority(3);
d_In.add(new ArcIn(d_P.get(5),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(12),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(1),d_T.get(4),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(0),d_T.get(0),1));
d_In.add(new ArcIn(d_P.get(5),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(9),d_T.get(8),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(3),1));
d_In.add(new ArcIn(d_P.get(15),d_T.get(3),1));
d_In.add(new ArcIn(d_P.get(13),d_T.get(2),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(10),1));
d_In.add(new ArcIn(d_P.get(8),d_T.get(10),1));
d_In.add(new ArcIn(d_P.get(4),d_T.get(5),1));
d_In.add(new ArcIn(d_P.get(8),d_T.get(9),1));
d_In.add(new ArcIn(d_P.get(16),d_T.get(9),1));
d_In.add(new ArcIn(d_P.get(1),d_T.get(1),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(1),1));
d_Out.add(new ArcOut(d_T.get(6),d_P.get(10),1));
d_Out.add(new ArcOut(d_T.get(4),d_P.get(3),1));
d_Out.add(new ArcOut(d_T.get(4),d_P.get(4),1));
d_Out.add(new ArcOut(d_T.get(4),d_P.get(9),1));
d_Out.add(new ArcOut(d_T.get(12),d_P.get(15),1));
d_Out.add(new ArcOut(d_T.get(12),d_P.get(16),1));
d_Out.add(new ArcOut(d_T.get(0),d_P.get(0),1));
d_Out.add(new ArcOut(d_T.get(0),d_P.get(1),1));
d_Out.add(new ArcOut(d_T.get(7),d_P.get(6),"unif", 800, 1200));
d_Out.add(new ArcOut(d_T.get(7),d_P.get(7),1));
d_Out.add(new ArcOut(d_T.get(8),d_P.get(8),1));
d_Out.add(new ArcOut(d_T.get(11),d_P.get(12),1));
d_Out.add(new ArcOut(d_T.get(11),d_P.get(13),1));
d_Out.add(new ArcOut(d_T.get(11),d_P.get(16),1));
d_Out.add(new ArcOut(d_T.get(3),d_P.get(14),"unif", 1400, 1600));
d_Out.add(new ArcOut(d_T.get(3),d_P.get(2),1));
d_Out.add(new ArcOut(d_T.get(2),d_P.get(2),1));
```

```
                d_Out.add(new ArcOut(d_T.get(2),d_P.get(14),"unif", 1400, 1600));
                d_Out.add(new ArcOut(d_T.get(10),d_P.get(11),1));
                d_Out.add(new ArcOut(d_T.get(5),d_P.get(5),1));
                d_Out.add(new ArcOut(d_T.get(9),d_P.get(10),1));
                d_Out.add(new ArcOut(d_T.get(1),d_P.get(2),1));
                d_Out.add(new ArcOut(d_T.get(1),d_P.get(14),2000));
                PetriNet d_Net = new PetriNet("model2",d_P,d_T,d_In,d_Out);
                PetriP.initNext();
                PetriT.initNext();
                ArcIn.initNext();
                ArcOut.initNext();

                return d_Net;
        }

        public static PetriNet CourseworkModel(int stripesCount) throws
ExceptionInvalidNetStructure, ExceptionInvalidTimeDelay {
                ArrayList<PetriP> d_P = new ArrayList<>();
                ArrayList<PetriT> d_T = new ArrayList<>();
                ArrayList<ArcIn> d_In = new ArrayList<>();
                ArrayList<ArcOut> d_Out = new ArrayList<>();
                d_P.add(new PetriP("",1));
                d_P.add(new PetriP("Літаки в повітрі",0));
                d_P.add(new PetriP("Кількість смуг",stripesCount));
                d_P.add(new PetriP("Черга літаків",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("Втрачено",0));
                d_P.add(new PetriP("Літаки з дозаправки",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("Відмови",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("Зароблено",0));
                d_P.add(new PetriP("",0));
                d_P.add(new PetriP("",0));
                d_T.add(new PetriT("",6.0));
                d_T.get(0).setDistribution("poisson", d_T.get(0).getTimeServ());
```

```
d_T.get(0).setParamDeviation(0.0);
d_T.add(new PetriT("Посадка",35.0));
d_T.get(1).setPriority(1);
d_T.add(new PetriT("Посадка з затримкою",35.0));
d_T.add(new PetriT("Посадка після дозаправки",35.0));
d_T.get(3).setPriority(3);
d_T.add(new PetriT("Встати в чергу",0.0));
d_T.add(new PetriT("",70.0));
d_T.get(5).setDistribution("unif", d_T.get(5).getTimeServ());
d_T.get(5).setParamDeviation(10.0);
d_T.add(new PetriT("",0.0));
d_T.get(6).setPriority(2);
d_T.add(new PetriT("Дозаправка",0.0));
d_T.add(new PetriT("",140.0));
d_T.add(new PetriT("",0.0));
d_T.get(9).setPriority(2);
d_T.add(new PetriT("Посадка на інший аеропорт",0.0));
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(11).setPriority(2);
d_T.add(new PetriT("Зайняти смугу",0.0));
d_T.get(12).setPriority(3);
d_In.add(new ArcIn(d_P.get(5),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(12),d_T.get(6),1));
d_In.add(new ArcIn(d_P.get(1),d_T.get(4),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(12),1));
d_In.add(new ArcIn(d_P.get(0),d_T.get(0),1));
d_In.add(new ArcIn(d_P.get(5),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(9),d_T.get(8),1));
d_In.add(new ArcIn(d_P.get(2),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(3),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(3),1));
d_In.add(new ArcIn(d_P.get(15),d_T.get(3),1));
d_In.add(new ArcIn(d_P.get(13),d_T.get(2),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(10),1));
d_In.add(new ArcIn(d_P.get(8),d_T.get(10),1));
d_In.add(new ArcIn(d_P.get(4),d_T.get(5),1));
d_In.add(new ArcIn(d_P.get(8),d_T.get(9),1));
d_In.add(new ArcIn(d_P.get(16),d_T.get(9),1));
```

```
        d_In.add(new ArcIn(d_P.get(1),d_T.get(1),1));
        d_In.add(new ArcIn(d_P.get(2),d_T.get(1),1));
        d_Out.add(new ArcOut(d_T.get(6),d_P.get(10),1));
        d_Out.add(new ArcOut(d_T.get(4),d_P.get(3),1));
        d_Out.add(new ArcOut(d_T.get(4),d_P.get(4),1));
        d_Out.add(new ArcOut(d_T.get(4),d_P.get(9),1));
        d_Out.add(new ArcOut(d_T.get(12),d_P.get(15),1));
        d_Out.add(new ArcOut(d_T.get(12),d_P.get(16),1));
        d_Out.add(new ArcOut(d_T.get(0),d_P.get(0),1));
        d_Out.add(new ArcOut(d_T.get(0),d_P.get(1),1));
        d_Out.add(new ArcOut(d_T.get(7),d_P.get(6),"unif", 800, 1200));
        d_Out.add(new ArcOut(d_T.get(7),d_P.get(7),1));
        d_Out.add(new ArcOut(d_T.get(8),d_P.get(8),1));
        d_Out.add(new ArcOut(d_T.get(11),d_P.get(12),1));
        d_Out.add(new ArcOut(d_T.get(11),d_P.get(13),1));
        d_Out.add(new ArcOut(d_T.get(11),d_P.get(16),1));
        d_Out.add(new ArcOut(d_T.get(3),d_P.get(14),"unif", 1400, 1600));
        d_Out.add(new ArcOut(d_T.get(3),d_P.get(2),1));
        d_Out.add(new ArcOut(d_T.get(2),d_P.get(2),1));
        d_Out.add(new ArcOut(d_T.get(2),d_P.get(14),"unif", 1400, 1600));
        d_Out.add(new ArcOut(d_T.get(10),d_P.get(11),1));
        d_Out.add(new ArcOut(d_T.get(5),d_P.get(5),1));
        d_Out.add(new ArcOut(d_T.get(9),d_P.get(10),1));
        d_Out.add(new ArcOut(d_T.get(1),d_P.get(2),1));
        d_Out.add(new ArcOut(d_T.get(1),d_P.get(14),2000));
        PetriNet d_Net = new PetriNet("model2",d_P,d_T,d_In,d_Out);
        PetriP.initNext();
        PetriT.initNext();
        ArcIn.initNext();
        ArcOut.initNext();

        return d_Net;
}
```