Розв’язок оцінено на більше 60

Розбаловка:

1 завдання — 7/20

2 завдання — 20/20

3 завдання — щось 25 з 30

4 завдання — теж десь 24 з 30

Зауваження:

1. В першому завданні має бути опис того як змінюється стан мережі(наскільки я розумію цей стан включає і позиції і переходи), при чому не за допомогою матриць(матричні рівняння не підходять) там де M\* = M + av, а за допомогою логіко алгебраїчних виразів чи щось таке. Здається. Короче повинно бути більше.
2. В другому завданні просила пояснити які елементи існують в програмному забезпеченні імітації – в PetriObjModel є позиція, дуга, перехід. В Arena є групи елементів як от: discrete processing, decision і так далі.
3. В третьому, не дописав формулу для вихідної характеристики у звіт тому зняли бал, ще, в переходах які позначають обслуговування клієнтів треба додати одноканальність. все інше ок.
4. Забув скопіювати метод петрі мережі в звіт(save net as method), крім того на скрінах де я в редакторі налаштовую параметри переходів, назви переходів не збігаються з тими що вказані в схемі знизу(бо я їх потім перейменовував 10 раз). Також питання по наведеному коду, тому що в самому коді не видно рядок в якому відбувається просування модельного часу (this.setCurrentTime() де викликається).

В коді повинно бути видно:

* Просування модельного часу(де оновлюється значення часу моделі в основному циклі)
* Просування моделі в часі(де змінюється стан моделі, тобто де викликаються методи для входу маркерів в активні переходи, та виходу маркерів, де ці виклики в основному циклі)
* Збір статистики( де в основному циклі викликається метод для збору статистики, doStatistics в моєму випадку)

A close-up of a document

Description automatically generated

1. **Математичний опис стохастичної мережі Петрі.**

Математичний опис мережі складається з матриці входів, матриці виходів, вектора маркірування.

N = (P, T, A, W, K, R)

P={P} – множина позицій;

T={T} – множина переходів;

PT=∅

A ⊆ P × T ∪ T × P – множина дуг;

W: A → – множина натуральних чисел, що задають кратності дуг (кількість зв’язків);

K={(cT ,bT |T ∈ T, cT ∈ ,bT ∈[0;1] – множина пар значень, що задають пріоритет та ймовірність запуску переходів;

R: T → R+– множина невід’ємних чисел, що характеризують часові затримки;

T , T - множина вхідних та множина вихідних позицій перехода Т;

P, P - множина вхідних і множина вихідних переходів позиції Р;

Приклад того як виконується

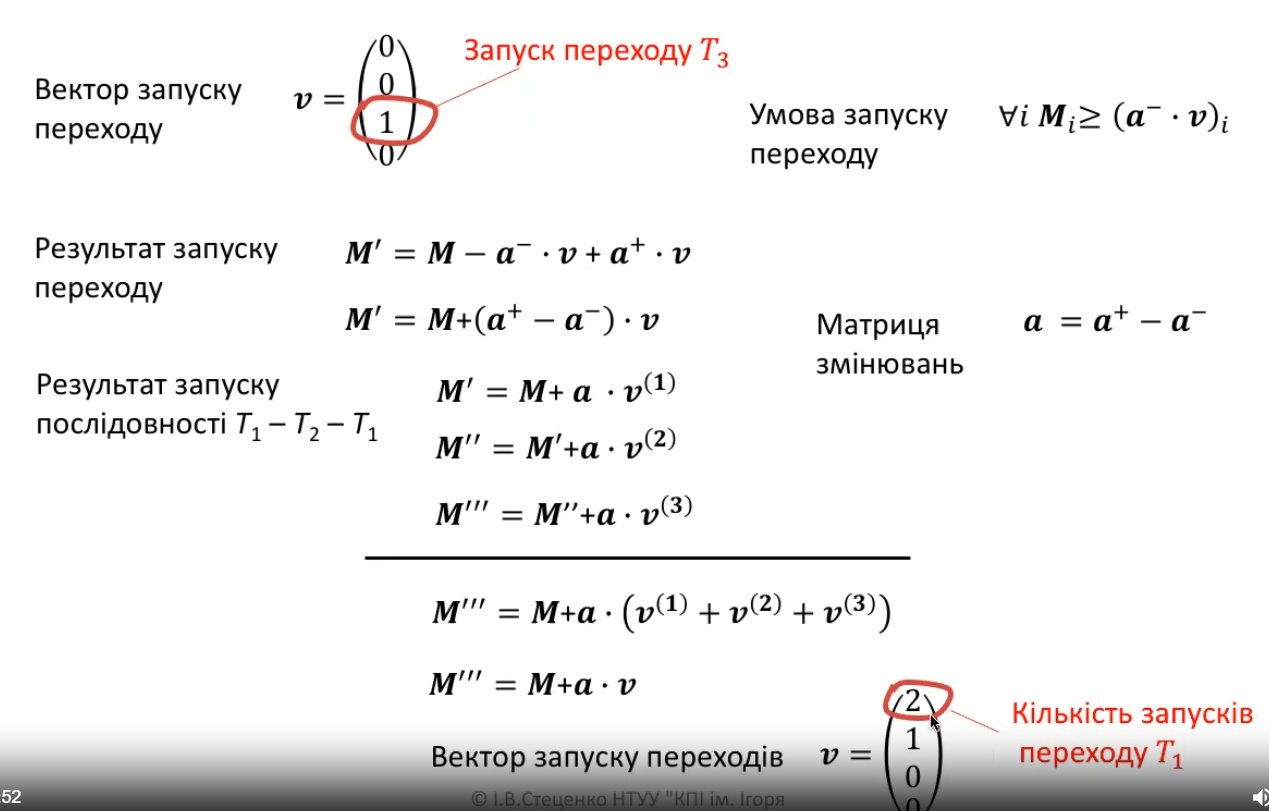
A close-up of a math problem

Description automatically generated

**Added later for myself: (after ticket was already handed in):**

Тут немає опису стану стохастичної мережі петрі в момент T. (S(T))

Від класичної мережі петрі відрізняється тим що до опису додаються кратність дуг, параметрів для розв’язання конфліктів(Kn), інформаційні дуги, часова затримка(дійсне невід’ємне число)



A screenshot of a computer

Description automatically generated

1. **Загальні складові сучасного програмного забезпечення з імітаційного моделювання систем.**

Якщо нам потрібно якісь специфічні штуки то звичайно краще брати універсальною мовою програмування.

Для імітаційного моделювання існує багато програмних застосунків. Протягом семестру ми використовували деякі з них, наприклад, для мереж Петрі ми використовували PetriObjectModelPaint. Також власноруч реалізовували ПЗ, якого вистачало для роботи з мережами Петрі та СМО. Окрім цього розглядали такі програмні застосунки, як Arena і GPSS

Для всіх цих застосунків характерними є такі компоненти:

* Комплекс елементів для складання моделей.
* Графічний редактор для складання, редагування та збереження моделі з набору елементів.
* Модуль для збору статистичних даних (стандартний або з можливістю налаштування).
* Модуль для управління збором статистики (час моделювання, час розгону моделі, налаштування набору даних, по яких збирається статистика).
* Модуль для проведення експериментальних досліджень з моделлю (дослідження відгуку в часі, факторний експеримент, експеримент з метою оптимізації).
* Анімація процесу функціонування моделі (2D, 3D).
* Тестові моделі для демонстрації імітаційного моделювання.
* Модуль для аналізу властивостей моделі.

Найбільш відомі програмні продукти для імітаційного моделювання:

● Arena Simulation

● CPNTools

● AnyLogic

Тенденції:

* Підтримка паралельної реалізації алгоритму імітації (одно чи багатопроцесорної)
* Підтримка віддаленої роботи з моделлю системи (хмарні технології)
* Предметно-орієнтовані середовища моделювання (VisSim, Network Simulator)

1. **Складіть мережу Петрі, шо описує функціонування такої системи: Супермаркет з трьома касами обслуговує клієнтів протягом робочого дня. Інтервали часу між надходженням кліситів до кас складають в середньому 4 хвилини. Час обслуговування клієнтів на першій, ругій на третій касі с випадковоїо величиною з середнім значенням 7, 5 та12 хвилин. Клієнт, який надійшов на обслуговування, обирає найкоротшу чергу. Метою моделювання с визначення середньої довжини черги до кожного касира.**

A graph paper with a diagram

Description automatically generated

**Added later for myself: (after ticket was already handed in):**

Якщо в чергах 1 і 2 більше ніж в черзі 3 ідемо в 3, якщо в 1 більше ніж 2 а в 3 стільки ж або більше ідемо в 2, інакше в 1

Обчислення інтенсивності навантаження:

Для визначення середньої довжини черги до кожного касира ми можемо відстежувати кількість токенів у Ч1, Ч2, Ч3 упродовж усього часу моделювання. Оскільки в нашому випадку не однакові, то маємо використати наступну формулу для розрахунку вихідного параметру:

де — довжина черги у проміжку

**4) Складіть алгоритм імітації такої системи : Розподілений банк даних організований на базі комп'ютерів, сполучених дуплексним каналом зв'язку. Запит, що надходить, обробляється на першому комп'ютері і з ймовірністю 7336 необхідна інформація внявляється на місці. У противному випадку необхідне посилання запиту в другий комп'ютер. Запити надходять через 21Ї с, попередня обробка запиту на першому комп'ютері та визначення чи є інформація на ньорму займає 2 с, пошук інформації потребує 3+-1 є на обох комп'ютерах, обмін даними міжкомп'ютерами по канаклу зв'язку потребує 1 с. Метою моделювання є внапачення середньої кількості запитів я очікуванні обробки.**

**Added later for myself: (after ticket was already handed in):**

Алгоритм імітації має наступні кроки:

• здійснюється очистка списку conflictObj з попереднього кола циклу;

• знаходиться найближча подія для виконання;

• для кожного об'єкта визначається час до виконання події, і за потреби

відображається статистика;

• здійснюється просування в часі;

• якщо поточний час не перевищує час моделювання, то симуляція

продовжується;

• здійснюється пошук конфліктних елементів, у яких подія відбувається в цей же

проміжок часу;

• якщо таких елементів декілька, то здійснюється сортування та вибір за

пріоритетом елементу з найвищим пріоритетом (якщо пріоритети рівні – обирається

випадковий елемент;

• виконання події;

• відбувається сортування елементів за пріоритетами;

• рухаємо маркери на оновлені позиції;

• повторюємо цикл

В алгоритмі імітації ми будемо використовувати просування моделі в часі За принципом найближчої події.

Для виконання завдання було створено алгоритм імітації петрі-мережі, що включає наступні частини.

На кожній ітерації протягом часу виконання ми виконуємо наступні кроки:

1. Визначити момент найближчої події в системі:

В коді це відбувається в головному методі програми go, в циклі:

1. public void go(double timeModeling) {  
    double min;  
    this.setSimulationTime(timeModeling);   
    this.setCurrentTime(0.0);   
      
    getListObj().sort(PetriSim.*getComparatorByPriority*()); //edited 9.11.2015, 12.10.2017  
    for (PetriSim e : getListObj()) { //edited 9.11.2015, 18.07.2018  
    e.input();  
    }  
    if (isProtocolPrint() == true) {  
    for (PetriSim e : getListObj()) {  
    e.printMark();  
    }  
    }  
    ArrayList<PetriSim> conflictObj = new ArrayList<>();  
    Random r = new Random();  
     
    while (this.getCurrentTime() < this.getSimulationTime()) { // edited 18.07.2018  
     
    conflictObj.clear();  
     
    min = getListObj().get(0).getTimeMin(); //пошук найближчої події  
     
    for (PetriSim e : getListObj()) {  
    if (e.getTimeMin() < min) {  
    min = e.getTimeMin();  
    }  
    }
2. після того як ми знайшли мінімальний час наступної події, ми маємо розв’язати конфлікт переходів:

if (this.getCurrentTime() <= this.getSimulationTime()) {  
  
 for (PetriSim sim : getListObj()) {  
 if (this.getCurrentTime() == sim.getTimeMin()) // розв'язання конфлікту об'єктів рівноймовірнісним способом  
 {  
 conflictObj.add(sim); //список конфліктних обєктів  
 }  
 }  
 int num;  
 int max;  
 if (isProtocolPrint() == true) {  
 System.*out*.println(" List of conflicting objects " + "\n");  
 for (int ii = 0; ii < conflictObj.size(); ii++) {  
 System.*out*.println(" K [ " + ii + " ] = " + conflictObj.get(ii).getName() + "\n");  
 }  
 }  
  
 if (conflictObj.size() > 1) { //вибір об'єкта, що запускається  
 max = conflictObj.size();  
 conflictObj.sort(PetriSim.*getComparatorByPriority*());  
 for (int i = 1; i < conflictObj.size(); i++) { //System.out.println(" "+conflictObj.get(i).getPriority()+" "+conflictObj.get(i-1).getPriority());  
 if (conflictObj.get(i).getPriority() < conflictObj.get(i - 1).getPriority()) {  
 max = i - 1;  
 //System.out.println("max= "+max);  
 break;  
 }  
  
 }  
 if (max == 0) {  
 num = 0;  
 } else {  
 num = r.nextInt(max);  
 }  
 } else {  
 num = 0;  
 }

1. Після того як знайдено перехід який буде запускатись, ми запускаємо перехід:

for (PetriSim sim: getListObj()) {  
 if (sim.getNumObj() == conflictObj.get(num).getNumObj()) {  
 if (isProtocolPrint() == true) {  
 System.*out*.println(" time = " + this.getCurrentTime() + " Event '" + sim.getEventMin().getName() + "'\n"  
 + " is occuring for the object " + sim.getName() + "\n");  
 }  
 sim.doT();  
 sim.output(); // added by Inna 11.07.2018  
 }  
}  
if (isProtocolPrint() == true) {  
 System.*out*.println("Markers output:");  
 for (PetriSim sim : getListObj()) //ДРУК поточного маркірування  
 {  
 sim.printMark();  
 }  
}  
  
Collections.*shuffle*(getListObj()); // added by Inna 11.07.2018, need for correct functioning of Petri object's shared resource   
  
getListObj().sort(PetriSim.*getComparatorByPriority*());  
  
for (PetriSim e : getListObj()) {  
 //можливо змінились умови для інших обєктів  
 e.input(); //вхід маркерів в переходи Петрі-об'єкта  
  
}  
if (isProtocolPrint() == true) {  
 System.*out*.println("Markers input:");  
 for (PetriSim e : getListObj()){ //ДРУК поточного маркірування  
 e.printMark();  
 }  
}

1. Варто також розглянути як відбувається збір статистики. Він відбувається всередині PetriSim. За вивід статистики в консоль відповідає PetriObjModel з одноіменним методом doStatistics

public void doStatistics(double dt) {  
 if (dt > 0) {  
 for (PetriP position : listPositionsForStatistica) {  
 position.changeMean(dt);  
 }  
 }  
 if (dt > 0) {  
 for (PetriT transition : listT) {  
 transition.changeMean(dt);  
 }  
 }  
}

Метод для оновлення середнього значення позиції:

public void changeMean(double a) {//if(buffer>0)

// mean=mean+buffer\*a;

mean = mean + (buffer - mean) \* a;

}

**Added later for myself: (after ticket was already handed in):**

Метод для входу маркерів в перехід:

public void input() {//вхід маркерів в переходи Петрі-об'єкта  
  
 ArrayList<PetriT> activeT = this.findActiveT(); //формування списку активних переходів  
  
 if (activeT.isEmpty() && isBufferEmpty() == true) { //зупинка імітації за умови, що  
 //не має переходів, які запускаються,і не має маркерів у переходах  
 timeMin = Double.*MAX\_VALUE*;  
 //eventMin = null; // 19.07.2018 by Sasha animation  
 } else {  
 while (activeT.size() > 0) {//запуск переходів доки можливо  
  
 this.doConflikt(activeT).actIn(listP, this.getCurrentTime()); //розв'язання конфліктів  
 activeT = this.findActiveT(); //оновлення списку активних переходів  
 }  
 this.eventMin();//знайти найближчу подію та ії час  
 }  
}

Метод для виходу маркерів з переходу:

public void output(){  
 if (this.getCurrentTime() <= this.getSimulationTime()) {  
 eventMin.actOut(listP, this.getCurrentTime());//здійснення події  
 if (eventMin.getBuffer() > 0) {  
 boolean u = true;  
 while (u == true) {  
 eventMin.minEvent();  
 if (eventMin.getMinTime() == this.getCurrentTime()) {  
 eventMin.actOut(listP,this.getCurrentTime());  
 } else {  
 u = false;  
 }  
 }  
  
 }  
 for (PetriT transition : listT) { //ВАЖЛИВО!!Вихід з усіх переходів, що час виходу маркерів == поточний момент час.  
   
 if (transition.getBuffer() > 0 && transition.getMinTime() == this.getCurrentTime()) {  
 transition.actOut(listP, this.getCurrentTime());//Вихід маркерів з переходу, що відповідає найближчому моменту часу  
 if (transition.getBuffer() > 0) {  
 boolean u = true;  
 while (u == true) {  
 transition.minEvent();  
 if (transition.getMinTime() == this.getCurrentTime()) {  
 transition.actOut(listP, this.getCurrentTime());  
 } else {  
 u = false;  
 }  
 }  
 }  
 }  
 }  
  
 }  
}

**Added later for myself: (after ticket was already handed in):**

Сам код для збору необхідної нам мережі має вигляд:

public static PetriNet CreateNetbackup() 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("P1",1));

d\_P.add(new PetriP("P2",0));

d\_P.add(new PetriP("P3",0));

d\_P.add(new PetriP("P4",0));

d\_P.add(new PetriP("P5",0));

d\_P.add(new PetriP("P7",0));

d\_P.add(new PetriP("P8",1));

d\_P.add(new PetriP("P9",0));

d\_P.add(new PetriP("P10",1));

d\_P.add(new PetriP("P11",1));

d\_P.add(new PetriP("P12",0));

d\_P.add(new PetriP("P13",1));

d\_T.add(new PetriT("incoming",2.0));

d\_T.get(0).setDistribution("unif", d\_T.get(0).getTimeServ());

d\_T.get(0).setParamDeviation(1.0);

d\_T.add(new PetriT("check\_first",2.0));

d\_T.add(new PetriT("pick\_first\_pc",0.0));

d\_T.get(2).setProbability(0.75);

d\_T.add(new PetriT("pick\_second\_pc",0.0));

d\_T.get(3).setProbability(0.25);

d\_T.add(new PetriT("send\_request",1.0));

d\_T.add(new PetriT("process\_2",3.0));

d\_T.get(5).setDistribution("unif", d\_T.get(5).getTimeServ());

d\_T.get(5).setParamDeviation(1.0);

d\_T.add(new PetriT("process\_1",3.0));

d\_T.get(6).setDistribution("unif", d\_T.get(6).getTimeServ());

d\_T.get(6).setParamDeviation(1.0);

d\_In.add(new ArcIn(d\_P.get(5),d\_T.get(5),1));

d\_In.add(new ArcIn(d\_P.get(9),d\_T.get(5),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(0),1));

d\_In.add(new ArcIn(d\_P.get(3),d\_T.get(6),1));

d\_In.add(new ArcIn(d\_P.get(6),d\_T.get(6),1));

d\_In.add(new ArcIn(d\_P.get(2),d\_T.get(3),1));

d\_In.add(new ArcIn(d\_P.get(4),d\_T.get(4),1));

d\_In.add(new ArcIn(d\_P.get(8),d\_T.get(4),1));

d\_In.add(new ArcIn(d\_P.get(1),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(11),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(2),d\_T.get(2),1));

d\_Out.add(new ArcOut(d\_T.get(5),d\_P.get(9),1));

d\_Out.add(new ArcOut(d\_T.get(5),d\_P.get(10),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(6),d\_P.get(7),1));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(6),1));

d\_Out.add(new ArcOut(d\_T.get(3),d\_P.get(4),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(5),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(8),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(11),1));

d\_Out.add(new ArcOut(d\_T.get(2),d\_P.get(3),1));

PetriNet d\_Net = new PetriNet("backup",d\_P,d\_T,d\_In,d\_Out);

PetriP.initNext();

PetriT.initNext();

ArcIn.initNext();

ArcOut.initNext();

return d\_Net;

}

Параметри переходів мережі було задано в графічному редакторі:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Обов’язково вирішуємо конфлікт переходів за допомогою встановлення імовірності(стохастична мережа петрі).

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Вигляд мережі після запуску:

A diagram of a diagram

Description automatically generated

A diagram of a diagram

Description automatically generated with medium confidence

A number of numbers and symbols

Description automatically generated with medium confidence

**Added later for myself: (after ticket was already handed in):**

ChatGPT generated:

Below is an explanation of **how PetriObjModel and PetriSim work together** and a **step-by-step outline** of the overall simulation algorithm. We’ll start with the high-level flow (from the perspective of PetriObjModel) and then highlight how each PetriSim contributes to the simulation.

**1. PetriObjModel: The High-Level Manager of PetriSim Objects**

PetriObjModel is responsible for orchestrating multiple PetriSim objects (i.e., multiple Petri nets). Its key method is:

public void go(double timeModeling) {

...

}

**What Happens Inside PetriObjModel.go(timeModeling)?**

1. **Initialization**
   * Sets the total simulation time (e.g., timeModeling) in a shared StateTime object.
   * Resets the current simulation time to 0.0 for *all* PetriSim objects in the model.
   * Sorts the PetriSim objects by priority.
2. **Initial Marking and input()**
   * For each PetriSim, calls input(). Internally, this means:
     + Each PetriSim checks which of its own transitions are *enabled* and which of those can immediately begin firing.
3. **Main Loop**
   * Continues **while** currentTime < timeModeling.
   * Finds the **earliest next event time** (min) among *all* PetriSims by looking at each PetriSim’s getTimeMin().
   * Advances the simulation clock to that earliest event time.
   * **Conflict resolution among PetriSim objects**:
     + If multiple PetriSim objects share the same earliest event time, chooses which one fires next.
       - PetriSim with highest priority is chosen. If there is a tie, it picks randomly.
   * Calls doT() **and** output() on the chosen PetriSim.
     + doT() is a placeholder for any *cross-model* token passing logic (i.e., if the transition that fires in PetriSim A wants to send tokens to PetriSim B).
     + output() causes that PetriSim to actually remove tokens from transitions that have completed firing.
   * **Shuffle** (for fairness) and re-sort the list of PetriSim objects, then call input() on *all* objects again. This ensures that newly enabled transitions in other PetriSim objects get triggered.
   * Repeat until currentTime >= timeModeling.
4. **Exit**
   * When currentTime >= timeModeling, the loop ends. All PetriSim objects are sorted by their original numbering (for nice output).
   * You can then print statistics (markers in each place, etc.).

**2. PetriSim: The Single Petri Net Simulator**

Each PetriSim manages **one** Petri net (i.e., one PetriNet object). Think of PetriSim as a wrapper that knows:

* **Places** (PetriP[] listP)
* **Transitions** (PetriT[] listT)
* **Incoming/Outgoing Arcs** (ArcIn[] listIn, ArcOut[] listOut)
* **Time** (via the shared StateTime)

A PetriSim primarily has three phases to manage transitions:

1. **input()**
   * Finds all *active* transitions (where the firing condition is satisfied) and tries to push tokens (“markers”) *into* each transition’s queue or buffer.
   * This uses a conflict resolution method **inside** the PetriSim (the doConflikt(...) method) if multiple transitions can consume the same tokens.
2. **output()** (or its near equivalent in step())
   * Fires the transition that is scheduled to complete at timeMin.
   * Removes tokens from the transition’s buffer (“actOut”) and places them in the correct output places.
   * If the transition still has tokens waiting in its buffer for the *same* current time, it continues to empty them until either the buffer is empty or the firing time changes.
3. **Time Calculation**
   * eventMin() scans all transitions to determine the earliest firing completion among them, storing that time in timeMin.
   * getTimeMin() returns that earliest firing completion time to the PetriObjModel.

**Key Methods in PetriSim**

* **go()** and **step()**
  + PetriSim.go() is for **standalone** usage if you simulate only one Petri net. It loops internally (using step()) until time is up.
  + In a *multi*-PetriSim scenario, PetriObjModel typically drives the timeline, so you don’t usually call PetriSim.go() directly. Instead, PetriObjModel.go() calls each PetriSim’s input(), picks a PetriSim to do doT() and output(), etc.
* **findActiveT()**
  + Checks each transition to see if it *can* fire right now. This includes verifying input places have enough tokens, and checking the transition’s probability/priority.
* **doConflikt(...)**
  + If multiple transitions want the *same tokens* from the same places, conflict resolution chooses which transition wins. This conflict resolution differs from the *inter-object conflict resolution* performed by PetriObjModel.
* **doT()**
  + A “hook” method intended for **extending** token passing to other PetriSim objects. By default, it doesn’t do anything. Subclasses often override it to pass tokens from a transition’s output places into another PetriSim’s input places.

**3. Putting It All Together: The Simulation Flow**

Here’s a **combined** perspective, step by step:

1. **Model Setup**
   * You create a list of PetriSim objects, each holding a PetriNet.
   * You place them into a PetriObjModel, which “owns” those PetriSim objects.
2. **Run PetriObjModel.go(totalSimulationTime)**
   * The model sets the clock to 0.
   * Sorts the PetriSim objects by priority.
   * For each PetriSim, calls input() once to preload transitions with tokens if possible.
   * If printing protocol is enabled, prints the initial marking.
3. **Main Event Loop**
   * **Find earliest next event time** among *all* PetriSim objects. Call it min.
   * **Collect statistics**, if enabled (how many tokens in each place, how many transitions active).
   * **Advance the clock** to min.
   * **Check which PetriSim(s)** have that same min time.
   * **Conflict resolution** among those PetriSim objects based on priority (or random if same priority).
   * **Fire transitions** on the chosen PetriSim:
     + doT() (if overridden, can pass tokens to other PetriSims),
     + output() (removes tokens from transitions that complete at time = min).
   * **Shuffle** (optional, for fairness with shared resources) and re-sort PetriSim objects by priority.
   * **Call input()** again on *all* PetriSim objects to see if any new transitions can start firing.
   * Repeat until currentTime >= totalSimulationTime.
4. **End of Simulation**
   * Sort PetriSim objects back by their original ID for final results.
   * Print or gather final stats (token counts in each place, transition usage, etc.).

**4. Summary**

* **PetriObjModel** is the higher-level controller that manages:
  1. *When* events happen across all PetriSim objects,
  2. *Which* PetriSim gets to fire next if multiple have events at the same time,
  3. *Synchronizing time* across all PetriSims,
  4. *Gathering overall statistics*.
* **PetriSim** manages:
  1. The individual Petri net’s places, transitions, and arcs,
  2. Internal conflict resolution among transitions that compete for the same tokens,
  3. Firing transitions (moving tokens from places into transitions and from transitions back into places),
  4. Calculating the next firing completion time (timeMin).

Together, they enable simulation of **multiple Petri nets** (PetriSim objects) in a single timeline, potentially interacting via shared places or passing tokens from one PetriSim to another (using doT() hooks).