DOCUMENTAȚIE

Tehnici de programare

Tema 2

Giesswein Alexia
Grupa 30225

CUPRINS

Obiectivul temei

Obiectivul principal al temei este proiectarea și implementarea unei aplicații de gestionare a cozilor, care are ca scop analiza sistemelor bazate pe cozi de așteptare simulând o serie de N clienți care sosesc pentru servicii, care sunt atribuiți cozilor Q (așteaptă, sunt serviți și în cele din urmă părăsesc cozile) astfel încât timpul de așteptare este minimizat. De asemenea, se calculează timpul mediu de așteptare, timpul mediu de servire și ora de vârf (ora la care sunt cei mai mulți clienți la cozi).

Cozile sunt utilizate în mod obișnuit pentru a modela domeniile lumii reale. Obiectivul principal al unei cozi este de a oferi un loc unde un „client" să aștepte înainte de a primi un „serviciu". Managementul sistemelor pe bază de coadă este interesat să minimizeze timpul de așteptare în cozi al „clienților" lor înainte să fie serviți. O modalitate de a minimiza timpul de așteptare este prin adăugarea mai multor servere, adică adăugarea mai multor cozi în sistem (se consideră că fiecare coadă are un procesor asociat), dar această abordare crește costul furnizorilor de servicii.

Aplicația de gestionare a cozilor ar trebui să simuleze (prin definirea unui timp de simulare $t_{simulation}$ – time limit) o serie de N clienți care sosesc pentru a fi serviți, intră în Q cozi, așteaptă, sunt serviți și în cele din urmă părăsesc cozile. Toți clienții sunt generați atunci când simularea este pornită și sunt caracterizați de trei parametrii: ID (un număr între 1 și N), $t_{arrival}$ (timpul de simulare când sunt gata să intre în coadă) și $t_{service}$ (intervalul de timp sau durata necesară pentru a servi clientul, adică timpul de așteptare când clientul este primul din coadă). Aplicația urmărește timpul total petrecut de fiecare client în coadă și calculează timpul mediu de așteptare. Fiecare client este adăugat la coada cu timpul minim de așteptare, când timpul său $t_{arrival}$ este mai mare sau egal cu timpul de simulare ($t_{arrival} >= t_{simulation}$).

Datele de intrare (care trebuie adăugate în interfață pentru ca utilizatorul să le poată insera):

- Numărul de clienți;
- Numărul de cozi;
- Timpul de simulare ($t_{simulation}$ –time limit);
- Timpul de sosire minim și maxim ($t_{arrival}^{MIN} <= t_{arrival} <= t_{arrival}^{MAX}$);
- Timpul de servire minim și maxim ($t_{service}^{MIN} <= t_{service} <= t_{service}^{MAX}$);

Obiectivele secundare sunt:

- Analizarea problemei și identificarea cerinței.
- Proiectarea aplicației de gestionare a cozilor.
- Implementarea aplicației de gestionare a cozilor.
- Folosirea threadurilor.
- Integrarea interfeței grafice.
- Testarea aplicației de gestionare a cozilor.

Analiza problemei

Această aplicație ar trebui să simuleze clienții care așteaptă să primească un serviciu, la fel ca în lumea reală, ei trebuie să aștepte la cozi, fiecare coadă procesând clienți simultan. Ideea este de a analiza câți clienți pot fi deserviți într-un anumit interval de simulare, prin introducerea parametrilor într-o interfață grafică a aplicației ușor de utilizat. Cienții sunt generați random, fiecare având timpul propriu de servire și de simulare (când pot intra în coadă).

Aplicația de gestionare a cozilor trebuie să îndeplinească următoarele cerințe:

- Utilizatorul trebuie să poată introduce (în interfața grafică) numărul de clienți care trebuie serviți, numărul de cozi care sunt disponibile pentru a servi clienții, timpul de sosire minim și maxim al clienților (fiecare are un timp când poate intra în coadă pentru a fi servit), timpul de servire minim și maxim (fiecare client are un timp de servire, câte secunde îi trebuie clientului să fie servit), strategia (dacă se sortează după timpul minim sau după lungimea minimă a cozii) și timpul de simulare (time limit –când se termină simularea).

- Dacă datele de intrare nu sunt introduse corect, programul va afișa un mesaj de eroare.

- Trebuie calculate: ora de vârf (când sunt cei mai mulți clienți la cozi), timpul mediu de așteptare și timpul mediu de servire.

- Când utilizatorul apasă butonul start, începe simularea cu datele introduse de utilizator. Programul afișează în cadrul interfeței grafice timpul curent de simulare (la ce secundă am ajuns cu simularea), clienții în aștepare (toți clienții au un ID, un timp de sosire – momentul în care vor intra în coadă și un timp de servire –câte secunde durează să fie serviți din momentul în care au ajuns primii în coadă), toate cozile, la fiecare fiind afișati clienții care așteaptă în coada respctivă, ora de vârf, timpul mediu de așteptare și timpul mediu de servire.

- Datele afișate în cadrul interfeței grafice sunt de asemenea scrise într-un fișier.

Proiectare

Pentru proiectarea aplicației de gestionare a cozilor am implementat trei pachete (după arhitectura MVC): pachetul Controller (care conține clasa Controller), pachetul View (care conține clasele View și QueueView) și pachetul Model (care conține clasele Task, Server, Scheduler, SimulationManager, ConcreteStrategyTime, ConcreteStrategyQueue, interfața Strategy și enumul SelectionPolicy).

Pachetul Model reprezintă structura logică de bază a datelor dintr-o aplicație software și clasa asociată cu acesta. Acest model de obiect nu conține nicio informație despre interfață sau

vreo conexiune cu utilizatorul. Acesta este ”creierul” aplicației, care conține clasele în care se modelează problema.

În clasa Task: fiecare task reprezintă de fapt un client.

În clasa Server: este o coadă și un timp de așteptare. Fiecare coadă are un thread. Threaurile execută codul din metoda run(). Aici se procesează taskurile și se modifică perioada de așteptare.

În clasa Scheduler: trimite taskurile (clienții) la servere (cozi), după strategia dorită. Se creează cozile și câte un thread pentru fiecare coadă.

Interfața Strategy este implementată de 2 clase: clasa ConcreteStrategyQueue și clasa ConcreteStrategyTime. Ea conține 2 metode addTask (implementată în funcție de strategie) și getTotalWaitingTime (pentru timpul de așteptare mediu).

În clasele ConcreteStrategyQueue și ConcreteStrategyTime se implementează metoda addTask (din interfața Strategy): în ConcreteStrategyTime se caută timpul minim și în funcție de acesta se adaugă taskul la coada potrivită, iar în ConcreteStrategyQueue se caută coada minim, iar taskul este adăugat la coada respectivă.

În enumul SelectionPolicy se găsesc cele 2 posibile strategii: după timpul minim sau după coada minimă.

În clasa SimulationManager: se generează taskuri (clienți) random (timpul de sosire, timpul de servire și ID-ul clientului sunt generate random). Se calculează timpul curent și în funcție de acesta se adaugă taskurile în cozi, iar cele deja adăugate se șterg din linia de așteptare. Fiecare coadă are un thread. Apoi se actualizează interfața grafică.

Pachetul View reprezintă o colecție de clase care reprezintă elementele de interfață (interacțiunea cu utilizatorul, toate lucrurile pe care utilizatorul le poate vedea și la care poate

răspunde pe ecran). Aici se primește comanda de la Controller și se afișează pe ecran.

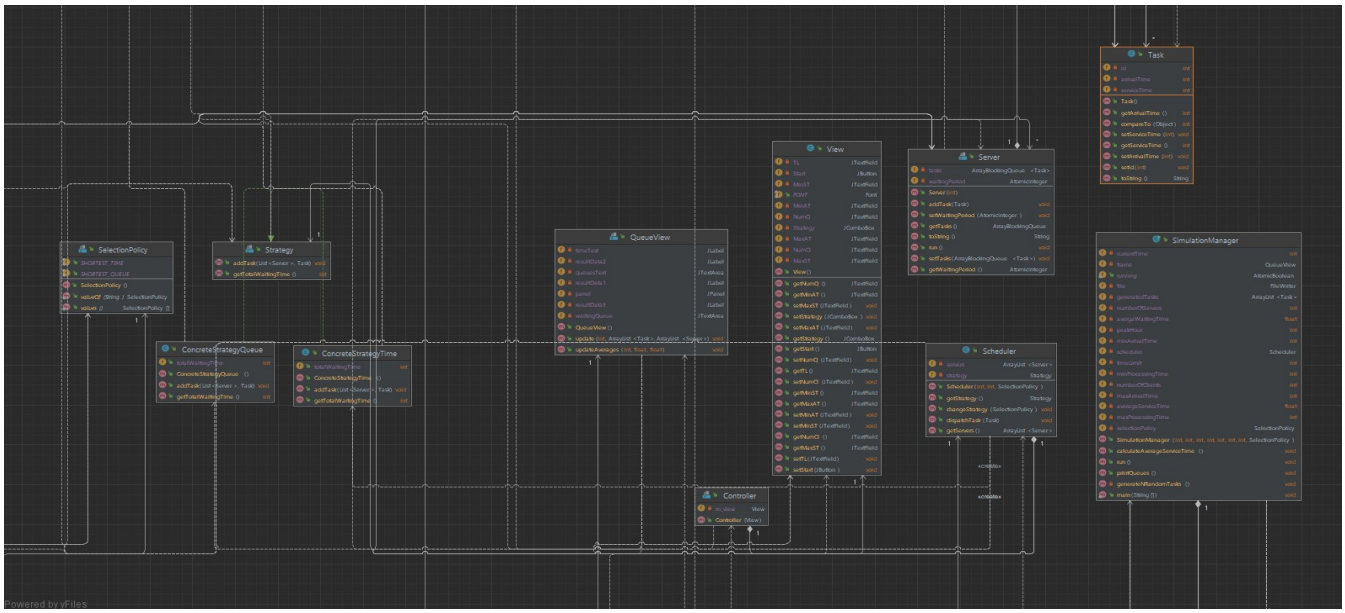
În clasa View este interfața grafică pentru citirea datelor (unde utilizatorul introduce datele).

În clasa QueueView este interfața grafică pentru afișarea datelor (după ce utilizatorul introduce datele, se afișează rezultatele: cozile, timpul mediu de așteptare, timpul mediu de servire și ora de vârf).

Pachetul Controller reprezintă clasele care conectează Model și View și este folosit pentru a comunica între clasele din Model și View. El primește ce trebuie să facă programul de la Model și ii transmite la View pentru a afișa răspunsul.

În clasa Controller se preia informația de la Model și se transmite către View. Am Implementat un ActionListener pe butonul de start, când utilizatorul apasă butonul, se citesc Datele introduse și se afișează rezultatele.

Diagrama UML



Implementare

Pachetele implementate sunt: Model, View, Controller.

Pachetul Model conține clasele: Task, Server, Scheduler, ConcreteStrategyTime, ConcreteStrategyQueue, SimulationManager, interfața Strategy și enumul SelectionPolicy.

Clasa Task are trei câmpuri: ID (fiecare client are un ID), arrivalTime (timpul la care Clientul intră în coadă) și serviceTime (timpul cât durează (în secunde) ca un client să fie servit). Aceasta implementează Comparable pentru a putea folosi metoda compareTo ca să sortăm clienții în ordinea crescătoare a timpului de sosire (arrivalTime). Am implementat și o metodă toString pentru a afișa frumos clienții.

Clasa Server implementează Runnable pentru că folosește threaduri (fiecare coadă are un thread pe care rulează). Aceasta are două câmpuri: tasks (o coadă cu taskuri –unde vor fi clienții) și waitingPeriod (timpul cozii –cât mai durează până la terminarea cozii). Am implementat un constructor (la care i se dă ca parametru numărul de clienți) care creează o nouă coadă cu taskuri și inițializează waitingPeriod cu 0. Apoi, am implementat metoda addTask care adaugă în coadă un nou client și crește waitingPeriod cu timpul de servire al clientului adăugat. Am implementat metoda run (pe care o execută threadurile), unde: dacă simularea merge, se ia primul task (primul client) din coadă și dacă coada nu este nulă, threadul se oprește pentru o secundă și în fiecare secundă timpul de servire al clientului curent tot scade până este scos din coadă. La final, am implementat o metodă toString pentru a afișa frumos coada.

Clasa Scheduler are două câmpuri: servers (o listă de servere (cozile)) și strategy (strategia după care adăugăm clienții în cozi). Am implementat un constructor (la care i se dă ca Parametru numărul de cozi, numărul de clienți și tipul de strategie) care se execută metoda changeStrategy, se creează o nouă listă de servere și pentru câte cozi avem, se creează un server

nou, se adaugă la lista de servere, se creează un thread pentru fiecare server (coadă) și se pornește. Apoi, am implementat metoda changeStrategy care alege strategia care urmează să fie folosită (adăugarea clienților în coadă după timpul minim sau după coada minimă). Apoi, am implementat metoda dispatchTask, care adaugă un client într-o coadă în funcție de strategia aleasă.

Interfața Strategy are două metode: addTask, implementată diferit de cele două clase ConcreteStrategyTime și ConcreteStrategyQueue și getTotalWaitingTime care returnează timpul mediu de așteptare.

Clasa ConcreteStrategyTime implementează Strategy. Are un singur câmp: totalWaitingTime, pentru calcularea timpului mediu de așteptare. Am implementat metoda addTask din Strategy, dar pentru timpul minim: căutăm timpul minim de așteptare în coadă și adăugăm clientul la coada cu timpul minim, iar în același timp calculăm timpul mediu de așteptare.

Clasa ConcreteStrategyQueue implementează Strategy. Are un singur câmp: totalWaitingTime, pentru calcularea timpului mediu de așteptare. Am implementat metoda addTask din Strategy, dar pentru coada minimă: căutăm coada cu cei mai puțini clienți și adăugăm clientul la coada cu cei mai puțini clienți, iar în același timp calculăm timpul mediu de așteptare.

Enumul SelectionPolicy conține cele două tipuri de strategie: SHORTEST_TIME și SHORTEST_QUEUE.

Clasa SimulationManager implementează Runnable, are 17 câmpuri: timeLimit (timpul cât durează simularea), minProcessingTime și maxProcessingTime (timpul minim și maxim de servire al clienților), minArrivalTime și maxArrivalTime (timpul minim și maxim de sosire al

clienților), numberOfClients (numărul de clienți), currentTime (timpul curent al simulării), selectionPolicy (strategia aleasă), running (care spune dacă simularea merge sau s-a terminat), averageWaitingTime (timpul mediu de așteptare), averageServiceTime (timpul mediu de servire), peakHour (ora de vârf), scheduler (care conține lista de servere și strategia), frame (interfața grafică unde se afișează rezultatele), generatedTasks (lista de clienți generați random) și file (fișierul în care se scriu rezultatele). Am implementat un constructor în care inițializez atributele, creez un nou scheduler (cozile și clienții), apelez changeStrategy, apoi generez taskurile random (clienții sunt generați random), creez interfața grafică unde se vor afișa rezultatele și creez un nou fișier. Apoi am implementat metoda generateNRandomTasks care generează N clienți random (cu ID, timp de servire și timp de sosire random) și adaugă clientul la lista de clienți (generatedTasks). Tot în această metoda se apelează și metoda calculateAverageServiceTime, care calculează timpul mediu de servire. Am implementat metoda run, unde: inițializăm currentTime cu 0, maxClientsInQueue (pentru calcularea peakHour), cât timp simularea merge, se iau taskurile, se adaugă la scheduler și cele care au fost deja adăugate în cozi sunt șterse din lista de așteptare. Apoi calculăm peakHour (numărul maxim de clienți la un moment dat în coadă), actualizăm interfața grafică, calculăm averageWaitingTime și afișăm pe interfața grafică, apoi scriem în fișier. Am implementat metoda calculateAverageServiceTime care calculează timpul mediu de servire.

Pachetul View conține clasele: View și QueueView.

Clasa View conține interfața grafică în care utilizatorul introduce datele de intrare pentru a începe simularea.

Clasa QueueView conține interfața grafică care afișează rezultatele obținute și metoda update care actualizează datele afișate.

Pachetul Controller conține clasa Controller, unde am implementat un ActionListener pe butonul start, care ia datele introduse de utilizator, creează un nou SimulationManager și afișează rezultatele.

Rezultate

Pentru testare am implementat exemplele cerute în assigment.
Test1:
N = 4 (clienți)
Q = 2 (cozi)
$t_{simulation}\ MAX$ = 60 seconds (time limit)
$[t_{arrival}\ MIN\ ,t_{arrival}\ MAX\ ]$ = [2, 30] (timpul minim și maxim de sosire)
$[t_{service}\ MIN\ ,t_{service}\ MAX\ ]$= [2, 4] (timpul minim și maxim de servire)

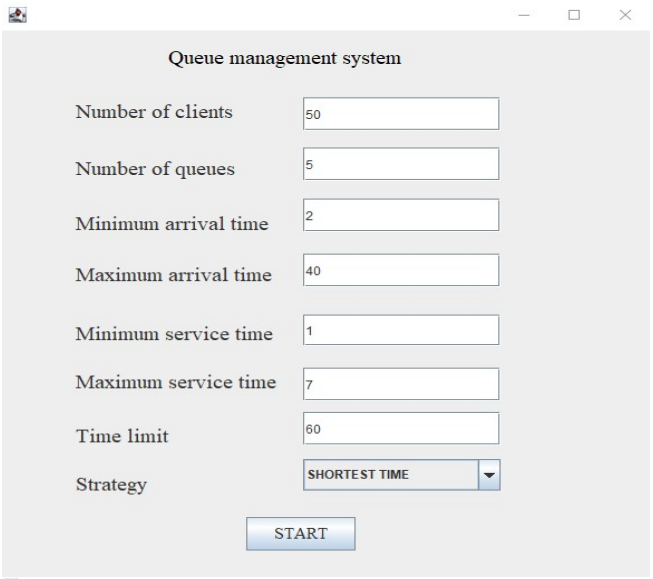
```
Time: 0
Waiting tasks: (3,2,4) (0,7,2) (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 1
Waiting tasks: (3,2,4) (0,7,2) (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 2
Waiting tasks: (0,7,2) (2,17,2) (1,28,2)
Queue 1: (3,2,4)
Queue 2: closed

Time: 3
Waiting tasks: (0,7,2) (2,17,2) (1,28,2)
Queue 1: (3,2,3)
Queue 2: closed

Time: 4
Waiting tasks: (0,7,2) (2,17,2) (1,28,2)
Queue 1: (3,2,2)
Queue 2: closed

Time: 5
Waiting tasks: (0,7,2) (2,17,2) (1,28,2)
Queue 1: (3,2,1)
Queue 2: closed

Time: 6
Waiting tasks: (0,7,2) (2,17,2) (1,28,2)
Queue 1: closed
```

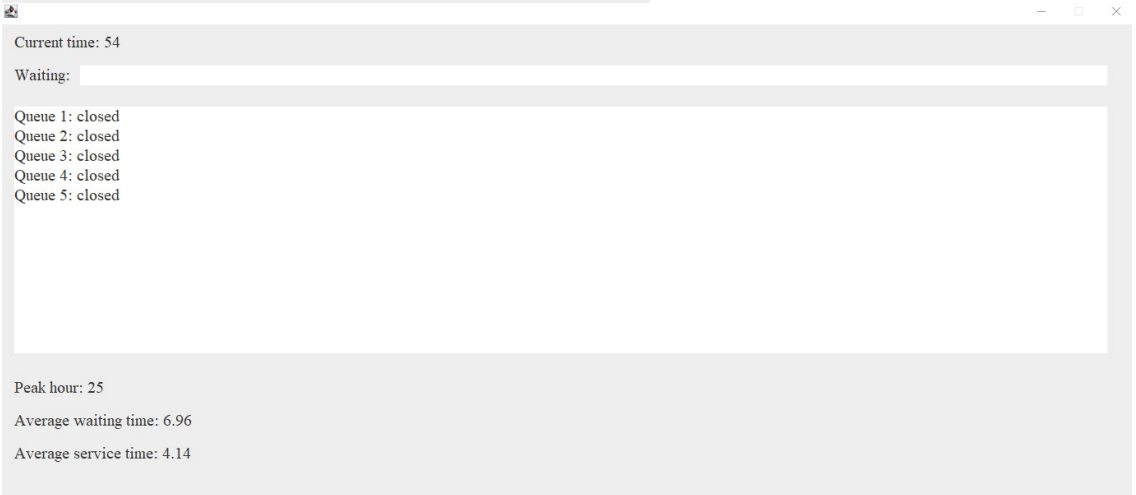
```
Queue 2: closed

Time: 7
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: (0,7,2)
Queue 2: closed

Time: 8
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: (0,7,1)
Queue 2: closed

Time: 9
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 10
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 11
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 12
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 13
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 14
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 15
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 16
Waiting tasks: (2,17,2) (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 17
Waiting tasks: (1,28,2)
Queue 1: (2,17,2)
Queue 2: closed
```

```
Time: 18
Waiting tasks: (1,28,2)
Queue 1: (2,17,1)
Queue 2: closed

Time: 19
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 20
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 21
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 22
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 23
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 24
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 25
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 26
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 27
Waiting tasks: (1,28,2)
Queue 1: closed
Queue 2: closed

Time: 28
Waiting tasks:
Queue 1: (1,28,2)
Queue 2: closed

Time: 29
Waiting tasks:
```

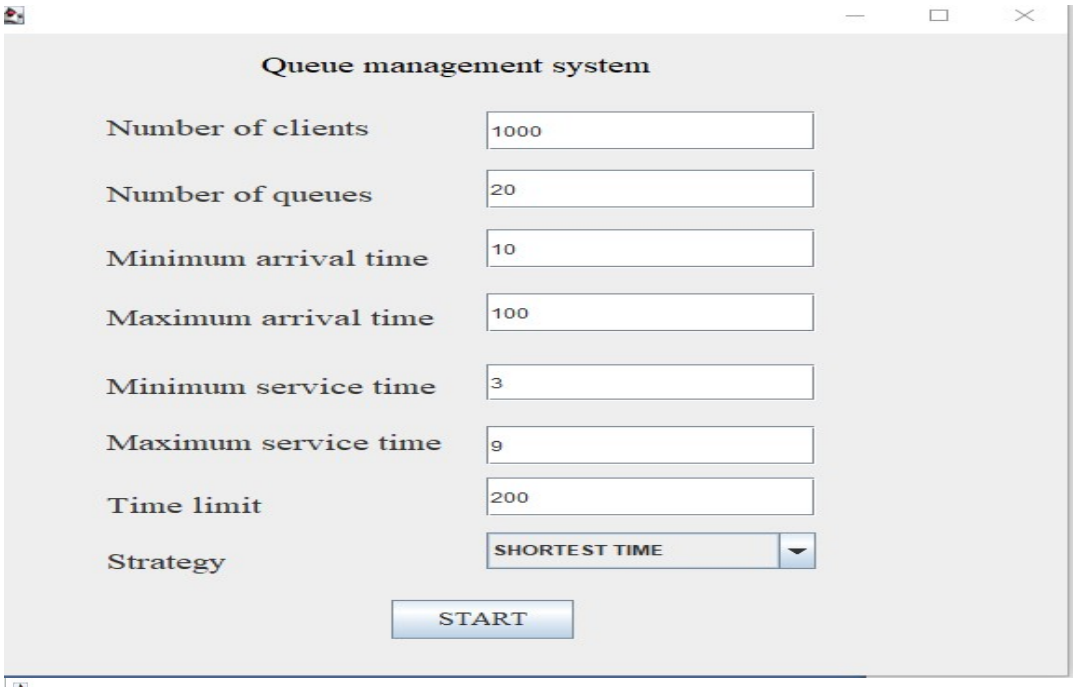
```
Queue 1: (1,28,1)
Queue 2: closed

Time: 30
Waiting tasks:
Queue 1: closed
Queue 2: closed

Peak hour: 2
Average waiting time: 0.0
Average service time: 2.5
```

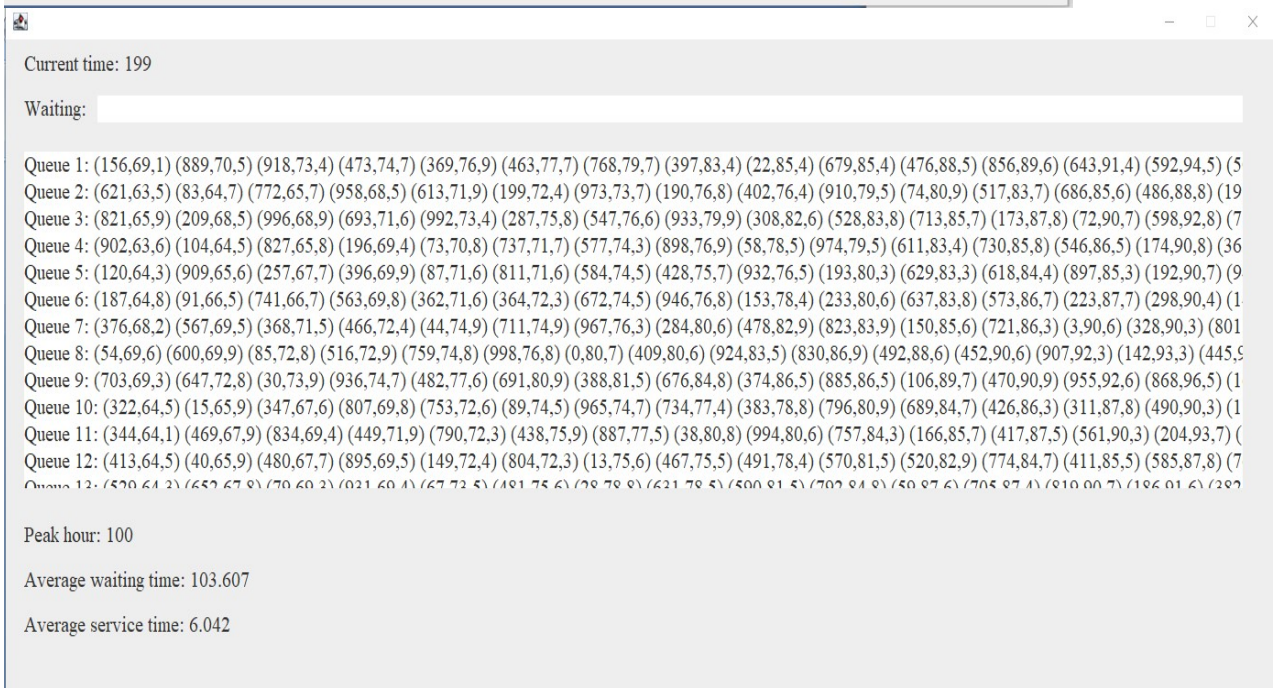
Test2:
N = 50 (clienți)
Q = 5 (cozi)
$t_{simulation}\ MAX$ = 60 seconds (time limit)
$[t_{arrival}\ MIN\ , t_{arrival}\ MAX\ ]$ = [2, 40] (timpul minim și maxim de sosire)
$[t_{service}\ MIN\ , t_{service}\ MAX\ ]$= [1, 7] (timpul minim și maxim de servire)





Test3:
N = 1000 (clienți)
Q = 20 (cozi)
$t_{simulation}\ MAX$ = 200 seconds (time limit)
$[t_{arrival}\ MIN\ , t_{arrival}\ MAX\ ]$ = [10, 100] (timpul minim și maxim de sosire)
$[t_{service}\ MIN\ , t_{service}\ MAX\ ]$= [3, 9] (timpul minim și maxim de servire)





Concluzii

În concluzie, această tema a fost un exercițiu excelent pentru familiarizarea cu limbajul de programare Java, pentru a învăța lucrul cu interfața grafică, pentru înțelegerea modelului MVC și pentru a aprofunda cunoștiințe despre limbajul Java: clasele, metodele, constructorii, lucrul cu liste. De asemenea, a fost un exercițiu foarte util, deoarece rezolvă o problemă din lumea reală.

Bibliografie

- https://docs.oracle.com/javase/tutorial/uiswing/
- https://beginnersbook.com/java-tutorial-for-beginners-with-examples/
- https://www.baeldung.com/java-tutorial
- https://google.github.io/styleguide/javaguide.html
- https://www.tutorialspoint.com/questions/category/Java