Documentatie de descriere solutie

Tehnici de Programare

Queues Simulator

Coblisan George, grupa 30225

An academic: 2020 – 2021

Abstract

Acest document urmareste descrierea programului, solutia abordata, tehnicile de programare folosite in elaborarea aplicatiei, utilizarea aplicatiei, analiza problemei, rezultate.

Cuprins

1. Cerinte functionale

Aceasta aplicatie dezvolta o simulare care vizeaza analiza sistemelor bazate pe cozi, determinarea si minimizarea timpului de asteptare al clientilor. Cozile sunt utilizate in mod obisnuit pentru modelarea domeniilor din lumea reala.

Obiectivul principal al unei cozi pentru acest caz, dar si pentru un caz general este sa ofere un loc unde un client sa astepte inainte de a fi servit. Functionalitatea unei cozi este bazata pe minimizarea timpului pe care clientii lor il petrec in coada (asteapta pentru a fi serviti).

E de dorit ca aplicatia sa aiba o interfara usor de utilizat si prietenoasa astfel incat orice tip de utilizator sa o foloseasca cu usurinta si cu drag.

2. Constrangeri de implementare

Din punctul meu de vedere, constrangerile fundamentale ale acestei aplicatii sunt urmatoarele:

- Generarea aleatoare a clientilor in cozi, dar sortarea dupa un ID in ordine crescatoare, de la 1 la numarul total de clienti
- Multithreading, cate un thread pentru fiecare coada
- Folosirea colectiilor din java (List) in schimbul utilizarii array-urilor in cat mai multe cazuri posibile
- Folosirea buclei foreach in schimbul buclei clasice (for int i=0...)
- Implementarea claselor sa contima maxim 300 de linii (cu exceptia claselor UI) si a metodelor sa contina maxim 30 de linii (cu anumite exceptii)
- Afisarea logurilor simularii intr-un fisier .txt pentru o vizulizare mai rapida si cu usurinta
- 2 ferestre de interfata: una pentru a introduce datele necesare simularii si una pentru afisarea evolutiei aplicatiei la fiecare secunda
- Afisarea timpului mediu de asteptare, timpului mediu de servire si a orei de varf in fisierul .txt sau in consola

3. Obiectivul aplicatiei

Principalul obiectiv al acestei aplicatii este ca interfata sa fie usor de folosit, prietenoasa si sa functioneze corect alaturi de afisarea in timp real a evolutiei aplicatiei (modificarea cozilor). Totodata, este esential ca aplicatia sa dezvolte o arhitectura impartita pe clase specifice si in pachete pentru o mai buna organizare a codului scris si o eleganta a acestuia, dar si pentru a opera cu solutii moderne si demne de un programator.

Aplicatia ar trebui sa functioneze prin definirea urmatoarelor: definirea unui timp de simulare maxim, o serie de N clienti sositi pentru a fi serviti intrand intr-un numar de Q cozi, asteptand, fiind serviti si in cele din urma parasind cozile. Toti clientii sunt generati la pornirea simularii si sunt caracterizati prin 3 parametri: ID (un numar intre 1 si N), T arrival (timpul de simulare cand clientii sunt gata sa mearga la coada, adica ora cand clientul a terminat cumparaturile) si T service (intervalul de timp sau durata necesara pentru a servi toate nevoile clientului, adica timpul de asteptare cand clientul se afla in varful cozii). Fiecare client este adaugat la coada cu timpul minim de asteptare cand timpul sau T arrival este mai mare sau egal cu timpul curent din simulare.

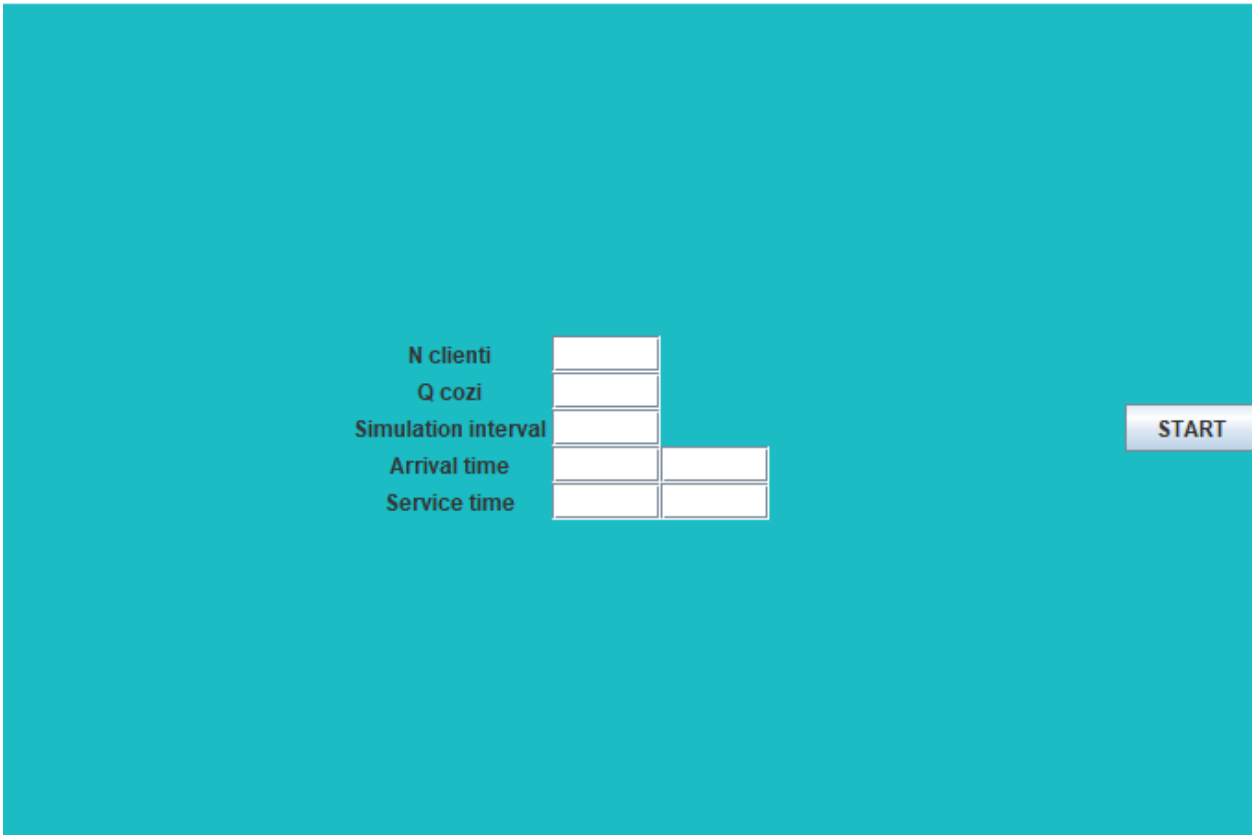
Bineinteles ca un alt obiectiv important este ca toate operatiile sa fie functionale pe toate cazurile posibile si sa informeze utilizatorul in cazul in care a introdus date de intrare invalide sau nu a introdus toate datele necesare pentru functionarea simulatorului cu un mesaj de eroare si un mesaj informativ.

Consider ca obiectivul pentru dezvoltator este sa analizeze toate cazurile posibile si sa adapteze problema la nivel de cod in cel mai simplu mod posibil deoarece acest mediu de dezvoltare ofera o multitudine de avantaje si o infinitate de moduri de a aborda problema, iar pentru utilizator sa inteleaga modul de functionare al aplicatiei si sa faca diferenta daca rezultatul afisat este cel asteptat sau nu.

4. Utilizarea aplicatiei

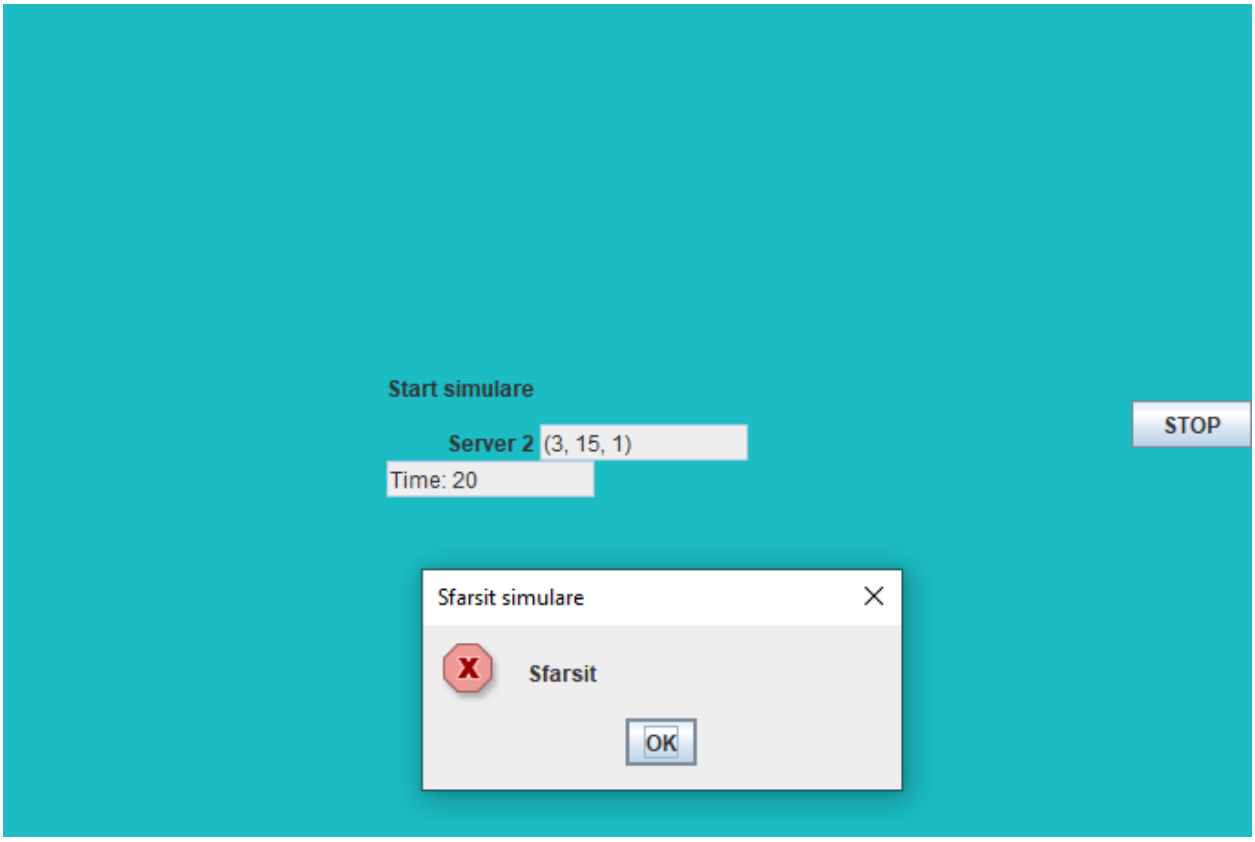
Meniul aplicatiei este unul primitor, simplu de folosit pentru orice tip de utilizator, contine 7 text field-uri pentru a introduce datele necesare pentru simulare si un buton de start pentru a porni aplicatia, respectiv simularea.

Poza urmatoare reprezinta meniul/interfata aplicatiei pentru introducerea datelor necesare pentru simulare:



A doua interfata contine un timer care numara secundele petrecute in aplicatie si totodata timpul necesar pentru finalizarea simularii, un buton Stop pentru a opri simularea in cazul in care apare vreo problema la utilizare sau pur si simplu se doreste terminarea brusca a aplicatiei si mai multe text field-uri pentru cozi care apar si dispar alaturi de clientii care sunt in acel moment in cozi si toate informatiile despre ei.

Poza urmatoare reprezinta a doua interfata chiar la sfarsitul unei simulari:



In continuare voi prezenta pasii care trebuie efectuati de catre utilizator pentru folosirea in mod corect a aplicatiei.

- Dupa pornirea aplicatiei, primul pas este introducerea tuturor datelor (nu se poate omite niciun camp), altfel va fi generat un dialog cu un mesaj de eroare si aplicatia se va opri.
- Dupa introducerea tuturor datelor urmeaza pornirea aplicatiei, moment in care este necesar apasat butonul START.
- In continuare, utilizatorul nu mai are nimic de facut, ci doar sa vizualizeze simularea aplicatiei si la final sa apese un OK cand este afisat dialogul de finalizare.

5. Proiectare

Etapa de proiectare a aplicatiei consta in algoritmii din spate si modul de gandire a efectuarii tuturor operatiilor. In prima faza am implementat clasele Client si Coada, clientul fiind caracterizat de ID, arrival time si service time si clasa Coada contine un ArrayList de clienti si afisarea clientilor din cozi. Apoi, am implementat clasa Task si Operatii. Clasa Task extinde Thread si este folosita pentru pornirea fiecarui Thread necesar fiecarei cozi. Clasa Operatii contine un ArrayList de clienti si face toate operatiile necesare pentru aplicatie: genereaza n clienti random, porneste threadurile, gaseste minimul service time-ului, gaseste ora de varf,

calculeaza timpul mediu de astepare si timpul mediu de servire. Clasa Interface proiecteaza intreaga interfata si efectueaza toate operatiile mentionate anterior din clasa Operatii.

6. Implementare

In acest capitol voi prezenta fiecare clasa in parte alaturi de metodele acestora.

- **Main**

Aceasta clasa este necesara pentru executia programului si apeleaza metoda de creare a interfetei din clasa Interface.

- **Client**

Aceasta clasa caracterizeaza clientii care intra in cozi cu atributele ID, arrivalTime, serviceTime, contine setters si getters pentru aceste atribute, o metoda pentru afisarea unui client si o metoda care sorteaza un ArrayList de tipul Client in ordine crescatoare dupa arrivalTime si totodata le atribuie un ID clientilor de la 1 la N (numarul de clienti) tot in ordine crescatoare.

- **Coada**

Aceasta clasa contine ca atribute un ArrayList de tipul Client si doua varabile de tip integer pentru service time-ul unui client si indexul cozii. De asemenea, contine setters si getters pentru acestea, metoda care actualizeaza service time-ul cu unul dat ca si parametru, metoda care adauga un client, metoda care sterge primul client din coada, metoda care returneaza dimensiunea acestei cozi. Alte 2 metode importante sunt cea pentru calculul sumei tuturor service time-urilor si toString pentru a afisa clientul sub forma (ID, arrivalTime, serviceTime).

- **Task**

Aceasta clasa extinde Thread si contine 2 metode pentru a returna pozitia curenta a threadului, respectiv a o seta, moment in care o variabila booleana de tip start devine true, ceea ce inseamna ca threadul urmeaza sa se porneasca. Mai contine o metoda numita run() care functioneaza cat timp varabila start mentionata anterior este true si scade pozitia curenta a threadului.

- **Operatii**

Aceasta clasa contine tot un ArrayList de tipul Client si contine urmatoarele metode: generate, startThreads, findMin, findPeakHour, getAverageWating, getAverageService, getAService, checkIfEmpty.
Metoda generate primeste ca argumente toate informatiile introduse de utilizator: N numarul de clienti, arrival MIN, arrival Max, service MIN, service MAX si genereaza N clienti cu arrival Time intre arrival MIN si arrival MAX si service time intre service MIN si service MAX, apoi ii adauga in arrayList.

Metoda startThreads creeaza un arrayList de tipul clasei Task si primeste ca parametru numarul de cozi, urmand sa creeze threaduri pentru fiecare coada in parte si adaugandu-le in ArrayList, la final returnandu-l.

Metoda findMin primeste ca argument un Array List de cozi, de tipul Coada si calculeaza pentru fiecare coada care client are cel mai mic service time si returneaza indexul acelui client.

Metoda findPeakHour primeste ca argument un Array List de cozi, de tipul clasei Coada si secunda si calculeaza pentru fiecare coada suma tuturor clientilor existenti in aceasta, apoi i se va atribui variabilei globale secunda secunda ca si parametru daca si numai daca este mai mare decat cele anterioare.

Metoda getAverageService parcurge toti clientii existenti si le insumeaza service time-ul, apoi returneaza media, adica suma respectiva impartita la numarul de clienti.

Metoda getAService calculeaza pur si simplu suma tuturor service time-urilor pentru a le folosi ulterior in metoda getAverageWaiting care va face media timpului de asteptare.

Metoda checkIfEmpty verifica daca pe fiecare pozitie a unui Array List de tipul boolean se afla vreun false, aceasta metoda este folosita in Interface pentru a verifica diferite conditii.

- **Interface**

Clasa Interface este miezul acestei aplicatii si cea mai importanta clasa fara de care aplicatia nu ar putea functiona. O sa incerc sa o explic cat de scurt posibil si cat de bine de inteles.

Metoda addComponents() creeaza prima interfata intr-un panel de tipul GridBagLayout, metoda in care punem Action Listener pe butonul de Start, in care se preiau toate informatiile introduse de utilizator, se verifica anumite conditii de eroare, se genereaza clientii random cu ajutorul metodei descrise mai sus si se face o afisare a acestora atat in consola, cat si in fisier. Urmeaza sa cream numarul de cozi introduse de utilizator si text field-uri pentru acestea. Tot in aceasta metoda se ascund toate informatiile din interfata de introducere a datelor si urmeaza printr-un Action Listener sa adaugam campurile din noua interfata, in prima faza Timerul si pornim threadul pentru prima coada, procesam clientul, cautam minimul si il adaugam in coada daca timpul curent este timpul sau. Acesta metoda se finalizeaza cu apelarea metodei startSimulating.

Metoda startSimulating() efectueaza toate verificarile necesare intr-o coada si modifica in thread schimbarile facute si scade timpul de service daca clientul este procesat in acest moment. La sfarsit aceasta metoda apeleaza metoda refreshInterface care actualizeaza informatiile curente care trebuie afisate, adica serverul (numarul cozii) si clientii din coada in acel moment.

Metoda error afiseaza un dialog cu un mesaj de eroare daca este apelata in situatiile de eroare sau afiseaza in fisier timpul mediu de asteptare, timpul mediu de servire si ora de varf daca a fost apelata in acest scop.

Metoda createInterface creeaza un frame, adauga panelurile si seteaza toate datele necesare.

Metoda resetInterface reseteaza absolut toate campurile din interfata.

7. Rezultate asteptate de utilizator

In acest capitol voi prezenta cateva screenshoturi cu datele din fisier pentru cazurile de rulare din cerinta.

Test 1
4 clienti, 2 cozi, 60 de secunde timp maxim, [2, 30] arrival Time, [2, 4] service Time

```
Time =1
Waiting clients: (1, 3, 4) (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =2
Waiting clients: (1, 3, 4) (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =3
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (1, 3, 4)

Time =4
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (1, 3, 3)

Time =5
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (1, 3, 2)


Time =6
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (1, 3, 1)

Time =7
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0


Time =8
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =9
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =10
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0
```

```
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =12
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =13
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =14
Waiting clients: (2, 15, 2) (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =15
Waiting clients: (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (2, 15, 2)

Time =16
Waiting clients: (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: (2, 15, 1)

Time =17
Waiting clients: (3, 18, 3) (4, 18, 3)
Queue 1: 0
Queue 2: 0

Time =18
Waiting clients:
Queue 1: (4, 18, 3)
Queue 2: (3, 18, 3)

Time =19
Waiting clients:
Queue 1: (4, 18, 2)
Queue 2: (3, 18, 2)

Time =20
Waiting clients:
Queue 1: (4, 18, 1)
Queue 2: (3, 18, 1)

Average waiting time: 3.00
Average service time: 3.00
Peak hour: 18 seconds
```

Pentru urmatoarele exemple (fiind tot mai mari) voi pune doar o poza cu sfarsitul fisierului si voi include fisierele in folderul incarcat pe git lab.

Test 2
50 de clienti, 5 cozi, 60 de secunde timp maxim, [2, 40] arrival Time, [1, 7] service Time

```
Waiting clients:
Queue 1: 0
Queue 2: (50, 39, 3)
Queue 3: 0
Queue 4: (49, 39, 3)
Queue 5: 0


Time =44
Waiting clients:
Queue 1: 0
Queue 2: (50, 39, 2)
Queue 3: 0
Queue 4: (49, 39, 2)
Queue 5: 0


Time =45
Waiting clients:
Queue 1: 0
Queue 2: (50, 39, 1)
Queue 3: 0
Queue 4: (49, 39, 1)
Queue 5: 0

Average waiting time: 4.90
Average service time: 3.84
Peak hour: 37 seconds
```

Test 3
1000 de clienti, 20 de cozi, 200 de secunde timp maxim, [10, 100] arrival Time, [5, 9] service Time

```
Time =200
Waiting clients:
Queue 1: (641, 68, 8) (665, 71, 5) (690, 73, 9) (716, 76, 9) (744, 78, 6) (764, 79, 6) (785, 82, 8) (809, 84, 4) (826, 85, 7) (850, 88, 3) (861, 89
Queue 2: (640, 68, 8) (671, 71, 3) (685, 73, 3) (692, 74, 4) (703, 74, 8) (729, 77, 6) (755, 79, 7) (777, 81, 9) (806, 84, 6) (825, 85, 8) (855, 89
Queue 3: (634, 68, 2) (653, 70, 9) (684, 73, 8) (708, 75, 4) (719, 76, 9) (754, 79, 6) (771, 80, 7) (796, 83, 5) (808, 84, 7) (838, 87, 3) (846, 87
Queue 4: (637, 68, 2) (647, 69, 4) (660, 70, 7) (689, 73, 4) (697, 74, 5) (715, 75, 8) (739, 77, 5) (758, 79, 5) (776, 80, 4) (788, 82, 9) (817, 85
Queue 5: (633, 68, 4) (658, 70, 6) (678, 72, 5) (694, 74, 6) (714, 75, 7) (735, 77, 4) (753, 78, 3) (763, 79, 3) (770, 80, 7) (795, 83, 9) (824, 85
Queue 6: (632, 68, 5) (657, 70, 3) (664, 71, 4) (683, 73, 7) (702, 74, 8) (728, 77, 4) (743, 78, 7) (767, 79, 8) (794, 82, 3) (805, 83, 5) (821, 85
Queue 7: (623, 67, 2) (646, 69, 9) (677, 72, 9) (707, 75, 4) (718, 76, 4) (734, 77, 8) (762, 79, 6) (784, 82, 6) (803, 83, 6) (820, 85, 9) (854, 88
Queue 8: (636, 68, 3) (652, 70, 5) (663, 71, 5) (688, 73, 9) (713, 75, 8) (738, 77, 3) (752, 78, 8) (781, 82, 5) (793, 82, 3) (804, 83, 4) (816, 85
Queue 9: (644, 69, 4) (656, 70, 3) (662, 70, 4) (682, 73, 8) (706, 75, 5) (723, 76, 7) (748, 78, 6) (766, 79, 6) (787, 82, 8) (813, 84, 5) (831, 86
Queue 10: (639, 68, 9) (670, 71, 4) (687, 73, 9) (712, 75, 6) (733, 77, 3) (742, 78, 4) (757, 79, 3) (765, 79, 3) (780, 81, 6) (800, 83, 8) (823, 8
Queue 11: (631, 68, 3) (651, 69, 6) (669, 71, 3) (681, 73, 5) (696, 74, 4) (710, 75, 9) (737, 77, 3) (751, 78, 7) (775, 80, 4) (786, 82, 8) (812, 8
Queue 12: (638, 68, 9) (668, 71, 3) (680, 72, 7) (701, 74, 8) (727, 77, 9) (761, 79, 4) (774, 80, 6) (792, 82, 9) (822, 85, 6) (845, 87, 4) (859, 8
Queue 13: (630, 68, 3) (650, 69, 9) (679, 72, 8) (705, 74, 5) (722, 76, 5) (736, 77, 9) (769, 80, 7) (791, 82, 8) (819, 85, 6) (840, 87, 3) (852, 8
Queue 14: (627, 67, 6) (659, 70, 5) (676, 72, 7) (699, 74, 7) (721, 76, 3) (732, 77, 5) (750, 78, 3) (760, 79, 6) (783, 82, 6) (802, 83, 5) (815, 8
Queue 15: (635, 68, 2) (645, 69, 9) (675, 71, 8) (700, 74, 6) (720, 76, 6) (741, 78, 5) (759, 79, 5) (779, 81, 7) (801, 83, 9) (829, 86, 7) (857, 8
Queue 16: (625, 67, 3) (654, 70, 7) (674, 71, 7) (698, 74, 3) (709, 75, 7) (731, 77, 5) (749, 78, 9) (782, 82, 5) (799, 83, 6) (814, 84, 4) (828, 8
Queue 17: (643, 68, 4) (655, 70, 4) (667, 71, 5) (691, 74, 6) (704, 74, 6) (726, 76, 5) (740, 77, 9) (773, 80, 5) (789, 82, 7) (811, 84, 6) (836, 8
Queue 18: (621, 67, 3) (649, 69, 4) (661, 70, 3) (673, 71, 7) (695, 74, 9) (725, 76, 6) (747, 78, 7) (768, 79, 8) (798, 83, 7) (818, 85, 4) (835, 8
Queue 19: (629, 67, 2) (642, 68, 8) (666, 71, 4) (686, 73, 9) (711, 75, 6) (730, 77, 4) (746, 78, 9) (778, 81, 6) (797, 83, 5) (810, 84, 6) (834, 8
Queue 20: (620, 67, 2) (648, 69, 7) (672, 71, 6) (693, 74, 7) (717, 76, 3) (724, 76, 6) (745, 78, 3) (756, 79, 5) (772, 80, 6) (790, 82, 4) (807, 8

Average waiting time: 89.81
Average service time: 6.02
Peak hour: 100 seconds
```

8. Concluzii si dezvoltare ulterioara

Aplicatia poate fi imbunatatita din multe puncte de vedere, iar cele pe care as fi dorit eu sa le implementez sunt urmatoarele:

- Un sistem de informare pentru utilizator despre cum ar trebui sa foloseasca si sa introduca datele si un sistem care sa ii afiseze pe langa rezultatul din fisier si toti timpii de asteptare, servire si ora de varf tot in interfata.
- O interfata mai primitoare, cu un altfel de meniu si totodata usor de utilizat, dar si mai animata referitor la simularea clientilor in cozi.
- Un istoric al datelor introduse de catre utilizator si un mesaj daca pentru acele date aplicatia a functionat corect sau nu alaturi de restul informatiilor deja existente.
- Un sistem care ar aproxima de cate secunde mai era nevoie pentru a se finaliza simularea in cazul in care aceasta nu a reusit sa fie terminata.