UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

AUTOMATICĂ ȘI CALCULATOARE, AN II

Lucrare laborator – Assignment II

Aplicatie care simuleaza analiza unui sistem bazat pe cozi de minimizare a timpului de asteptare

Profesor curs: Prof. Dr. Ing. Cristina Pop

Profesor laborator: Teodor Petrican

Student: Jitaru Andrei

Grupa: 30229

CUPRINS

1. Introducere – obiectivul proiectului

Obiectivul acestui proiect este de a verifica constintele studentului referitoare la utilizarea si sincronizarea thread-urilor cu scopul obtinerii de rezultate valide si dorite. Intregul proiect ofera posibilitatea de a ne familiariza cu diferitele caracteristici și proprietăți ale conceptelor care produc realizarea sa. Privind mai in detaliu tema s-a dorit implementarea in limbaj de programare Java a unei aplicatii care sa simuleze intrarile si iesirile unui anumit numar de persoane din mai multe cozi, urmand o anumita strategie care sa fie cat mai eficace in acest sens. Proiectul a fost realizat cu scopul de a demonstra atât înțelegerea asupra temei și cerintelor date cât și etalarea aptitudinilor de implementare a acestora în contextul limbajului de programare mai sus menționat. Utilizarea thread-urilor a fost ceruta cu scopul obtinerii unui comportament concurent.
Toate aspectele enumerate anterior au fost verificate prin implementarea unei aplicații simple. Aceasta a fost realizata conform cerintelor temei, fiind adoptat un stil cat mai minimalist pentru o profunda intelegere a utilizatorului.

2. Analiza problemei

Proiectul, care în speță reprezintă un sistem care permite simularea evenimentelor care pot avea loc la nivelul mai multor cozi (intrari in coada, momente de asteptare pentru a fi procesat clientul, iesiri din coada), sugereaza un proces complex care neccesita atentie la fiecare pas. O astfel de abordare este absolut necesara deoarece fiecare operatie este unica in privinta functionarii sale.

a. Asumpții

Pentru a obine rezultate optime in urma utilizarii aplicatiei este necesara impunerea unor asumptii. In acest sens s-a decis faptul ca evitarea completarii casutelor de input din interfata este interziza. Desigur, completarea incorecta a acestora poate conduce la aparitia diferitor erori care au fost tratate in cod sub

forma exceptiilor (vom reveni la acest subiect in capitolul dedicat exceptiilor). De asemenea, timpul maxim de sosire in coada (Max Arrival Time) si timpul maxim de procesare (Max Service Time) nu poate fi mai mare decat durata simularii (Simulation Interval).

b. **Modelare**

Cerința temei și anume realizarea unei aplicatii care sa simuleze un sistem bazat pe cozi de procesare pentru a determina si minimiza timpul de asteptare al clientilor prezinta un amplu proces de modelare.

In acest sens, a fost aleasa o abordare cat mai simplista si, in acelasi timp, cat mai eficienta din punct de vedere al resurselor folosite dar si al timpilor de executare. Functionarea aplicatiei se bazeze pe conceptul de thread. Acesta defineste cea mai mica unitate de procesare ce poate fi programata spre executie de catre sistemul de operare. Un astfel de concept este folosit in programare pentru a eficientiza executia programelor, executand portiuni distincte de cod in paralel in interiorul aceluiasi proces. Desigur, cateodata aceste portiuni de cod nu sunt complet independente si in anumite momente ale executiei se poate intampla ca un thread sa trebuiasca sa astepte executia unor instructiuni din alt thread pentru a putea continua executia propriilor instructiuni. Aceasta tehnica prin care un thread asteapta executia altor thread-ri inainte de a continua propria executie poarta denumirea de sincronizare a threadurilor.

In cadrul modelarii aplicatiei a fost necesara utilizarea thread-urilor in 3 cazuri: la adaugarea unui client in coada, la scoaterea unui client din coada si la update-ului constat al interfetei grafice pentru a putea urmari evolutia evenimentelor din cadrul cozilor in timp real. In functie de desfasurarea threadurilor respective anumiti parametrii ai aplicatiei vor suferi schimbari pentru a determina media timpului de asteptare, media timpului in care cozile au fost goate, media timpului de procesare a clientilor si ora de varfa(momentul in care in cozi se afla numarul maxim de persoane).

c. Scenarii

O serie de etape trebuie urmate pentru ca aplicația să fie capabilă de a returna rezultate clare și corecte. Totalitatea acestor etape dau naștere unei serii de scenarii care definesc funcționalitatea programului. În acest sens, pentru a obține un anumit rezulat, diferite evenimente trebuie să aibă loc. Interfata simplista este alcatuita din 6 campuri in care putem introduce datele de intrareale aplicatiei, ui buton de start care sa faciliteze inceperea simularii si un slider care sa permita modificarea vitezei de simulare a evenimentelor ce au loc la nivelul cozilor.

Pasi de utilizare ai aplicatiei:

- Se introduc datele de intrare necesare functionarii aplicatiei (pot fi introduse in orice ordine)
- Se apasa butonul START pentru activarea aplicatiei
- In timp ce simularea are loc, in functie de dorinta utilizatorului, acesta are posibilitatea de a modifica viteza de simulare a evenimentelor ce au loc la nivelul cozilor prin glisarea orizontala a slider-ului Speed (daca slider-ul va fi glisat la stanga viteza va scadea iar daca slider-ul va fi glisat la dreapta viteza va creste);

Evolutia evenimentelor poate fi urmarita la orice moment de timp aruncand o privire asupra logger-ului de evenimente care prezinta in timp real fiecare actiune care are loc la nivelul fiecarei cozi. De asemeanea intrarile si iesirile din fiecare coada sunt prezentate vizual in zona de text corespunzatoare fiecarei cozi.

d. Cazuri de utilizare

Acest proiect este realizat cu scopul de a simula un sistem bazat pe cozi de procesare pentru a determina si minimiza timpul de asteptare in cadrul acestora. Luand in considerare aceste detalii putem deduce ca aplicatia ar putea fi utilizata fara dubii in scopuri practice. De exemplu, o societate comerciala de tip magazin ar putea utiliza aceasta aplicatia pentru a simula eficienta angajatilor proprii.

a. Erori

Anumite evenimente pot determina aparitia unor exceptii care au fost tratate la implementare sub forma unor eror. Aparitia acestor evenimente produce afisarea une casute de dialog in care este prezentata eroare pentru a fi mai facil de identificat. Erorile, in cazul aplicatiei de fata pot fi determinate doar din introducerea unor date de intrare necorespunzatoare.

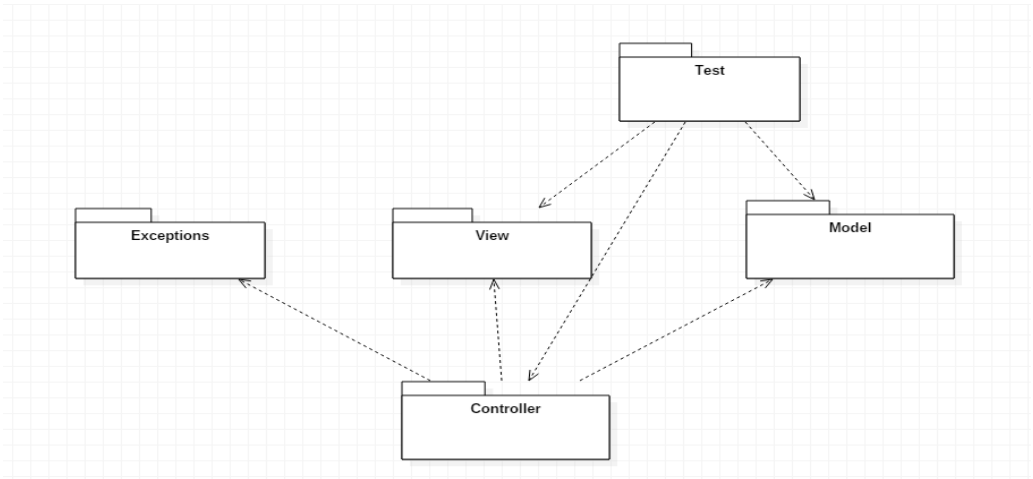
Cazuri care pot conduce la aparitia erorilor:

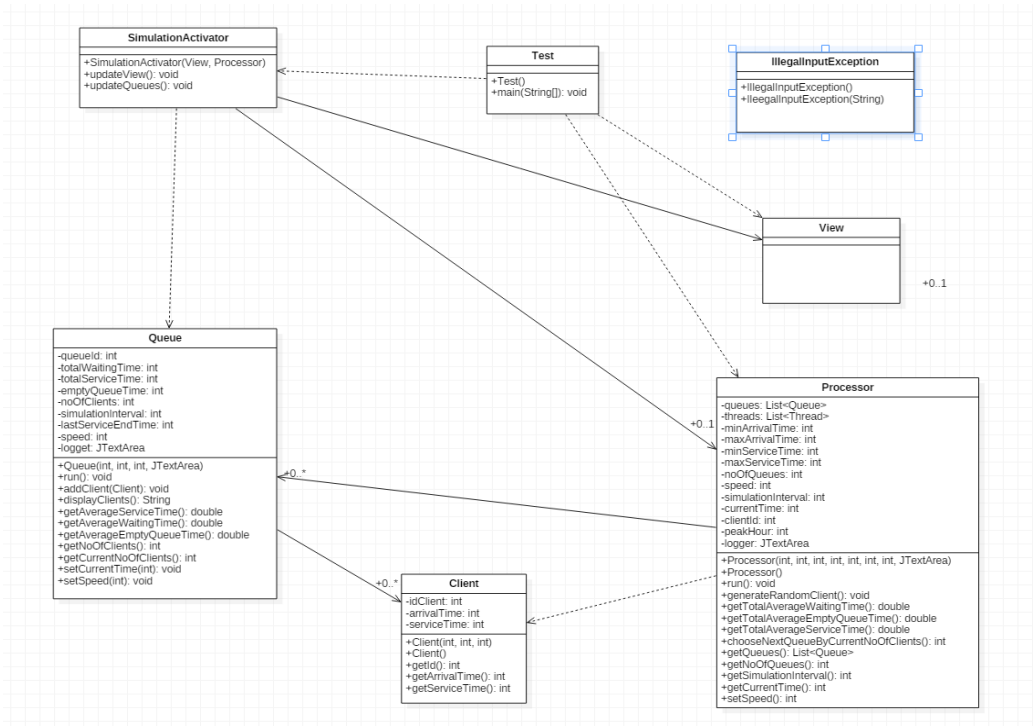
- Daca timpul maxim de sosire este mai mic decat timpul minim de sosire
- Daca timpul maxim de sosire este mai mare decat intervalul de simulare
- Daca timpul minim de sosire este mai mic decat unu
- Daca timpul maxim de servire este mai mic decat timpul minim de servire
- Daca timpul maxim de servire este mai mare decat intervalul de simulare
- Daca tipul minim de servire este mai mic decat unu
- Daca numarul de cozi nu se afla in intervalul [1, 6]

3. Proiectare

a. Diagrame UML

În diagramele UML de mai jos sunt prezentate toate pachetele si clasele, cu atributele și relațiile dintre acestea. O analiză detaliată a acestor diagrame este prezentată în capitolul 4 al acestei documentații.





b. Structuri de date utilizate în implementare

Funcționarea aplicatiei este strans legata de utilizarea de structuri de date in cadrul implementarii acesteia. In acest sens, a fost utilizata o structura de date de tip LinnkedBlockingQueue pentru reprezenta o coada. Utilizarea exclusiva a acestui tip de structura este foarte importanta datorita faptului ca la nivleul cozilor lucram cu thread-uri fapt pentru care avem nevoie de o structura de date care sa fie thread-safe. De asemenea cozile si thead-urile au fost stocate in niste ArrayList-uri care sa permita manipularea mai usoara a acestora.

a. Proiectare clase

Proiectul, ca un tot unitar, este divizat in 14 clase care permit transpunerea cerinței și temei în limbaj de programare Java. Clasele au dimensiuni, alcătuiri și roluri variate.

O coada a fost implementata sub forma unui thread. Astfel, clasa coada este o clasa care implementeaza Runnable. Aceasta detine parametrii specifici care sunt folositori pentru determinarea cerintelor temei: totalWaitingTime, totalServiceTime, emptyQueueTime, etc. Timpul curent dar si timpul simularii sunt determinati de atributere currentTime si simulationInterval. Acestia vor fi setati si actualizati de catre clasa Processor. Parametrlu logger este de asemenea de o importanta deosebita deoarece are rolul de a ajuta in afisarea evenimentelor pe interfata grafica.

Clasa Processor reprezinta piatra de temelie a aplicatiei. Aceasta contine lista de cozi si lista de thread-uri pe care in functie de momentul simularii le va porni sau opri. Atributele simulationInterval si currentTime mentionate in randurile de mai sus sunt prezente aici. Pe parcursul simularii currentTime va fi actualizat si transmis cozilor la fiecare moment. Clientii vor fi generati la nivelul acestei clase pentru ca mai tarziu, la momentul de timp potrivit, sa fie introdusi in cozi. De asemenea, sunt descrise operatiile care determina rezultate finale cerute de cerinta: determinarea timpului mediu de asteptare, determinarea timpului mediu in care cozile sunt goale, determinarea timpului mediu de servire si determinarea orei de varf. Apare din nou parametru logger care are de asemenea rolul de a ajuta in afisarea evenimentelor si timpilor pe interfata grafica.

Clasa client reprezinta, dupa cum ii sugereaza numele, implementarea in cod a unui client care produce schimbari asupra cozii. Aceasta contine parametrii definitorii unui client, cum ar fi clientId – prin care se diferentiaza clientii, arrivalTime – care reprezinta timpul in care vine un client fata de clientul anterior si serviceTime – care reprezinta timpul de servire al unui client.

Clasa simulationActivator reprezinta controller-ul aplicatiei. Interactiunea clasei Processor, care reprezinta si clasa model a aplicatiei, cu clasa View va avea loc aici. Evenimente precum inceperea simularii sau modificarea vitezei timpului de simulare sunt determinate de aceasta clasa.

Clasa View reprezinta interfata grafica care perminte interactiunea utilizatorlui cu sistemul.

c. Interfata utilizator

Interfața cu utilizatorul are un aspect minimalist pentru a putea fi utilizată cât mai ușor de orice persoană în parte. S-a optat doar pentru introducerea strictului necesar. Astfel, interfața este alcatuitadin 6 campuri de introducere a datelor de intrare care dermina detaliile de evolutie ale simularii, un buton care sa permita inceperea aplicatie, un slider prin care sa se modifice viteza de progres a evenimentelor, o zona de text unde sa urmarim evolutia timpilor simularii dar si a evenimentelor ce au loc si, in cele din urma, 6 zone de text specifice fiecarei cozi in parte pentru a putea urmari pe viu modul in care clientii intra, asteapta si ies din cozi.

4. Implementare

Buna functionare este oferita de o implementare simpla si usor de inteles a mecanismelor aplicatiei. Astfel, clasa pivot este clasa Procesor. Functionalitatea ei este oferita de metoda **run.** Prima etapa este creearea si inceperea cozilor si a threadurilor aferente. A doua etapa se ocupa de introducerea unui client in coada. In acest scop este utilizata o bucla for pentru a numara timpii de executie ai aplicatie. La fiecare moment timpul curent este actualizat pe toate cozile.

Introducerea unui client nou in coada se va determina in functie de momentul introducerii clientului anterior. De asemenea la fiecare moment va fi calculat numarul maxim de clienti care se afla in cozi pentru a determina ora de varf. A treia ertapa se refera la intrerupea threadurilor iar cea de-a patra la afisarea rezultatelor obtinute in logger-ul de evenimente.

Dupa cum am mentionat in capitolele urmatoare fiecare coada reprezinta defapt un thread a carui functionalitate este descrisa, de asemenea, in metoda run. Fiecare coada se ocupa cu scoaterea clientilor. Astfel, daca coada nu este goala atunci vom astepta un numar de secunde egal cu timpul de servire al clientului pe care dorim sa-l scoatem din coada dupa care il vom elimine efectiv din varful cozii. In caz contrar, vom adormi thread-ul cate o secunda. Cei trei timpi pe care dorim sa-i obtinem (timpul mediu in care cozile au fost goale, timpul mediu de servire si timpul mediu de asteptare) vor fi procesate in clasa Queue si calculate in clasa Processor.

Timpul mediu in care cozile au fost goale se obtine in urmatorul mod: in clasa Queue, incrementam variablia emptyQueueTime pentru a marca trecerea unei secunde in care coada respectiva a fost goala. Dupa care vom imparti timpul respectiv la numarul clientilor care au fost in coada pana la terminarea simularii, vom aduna acesti timpi pentru fiecare coada si vom imparti suma obtinuta la numarul de cozi care au fost active.

Timpul mediu de servire se obtine astfel: inainte sa scoatem un client din coada incrementam variabila totalServiceTime cu valoarea timpului de serviciu a clientului eliminat. In cele ce urmeaza vom imparti timpul respectiv la numarul clientilor care au fost in coada pana la terminarea simularii, vom aduna acesti timpi pentru fiecare coada si vom imparti suma obtinuta la numarul de cozi care au fost active.

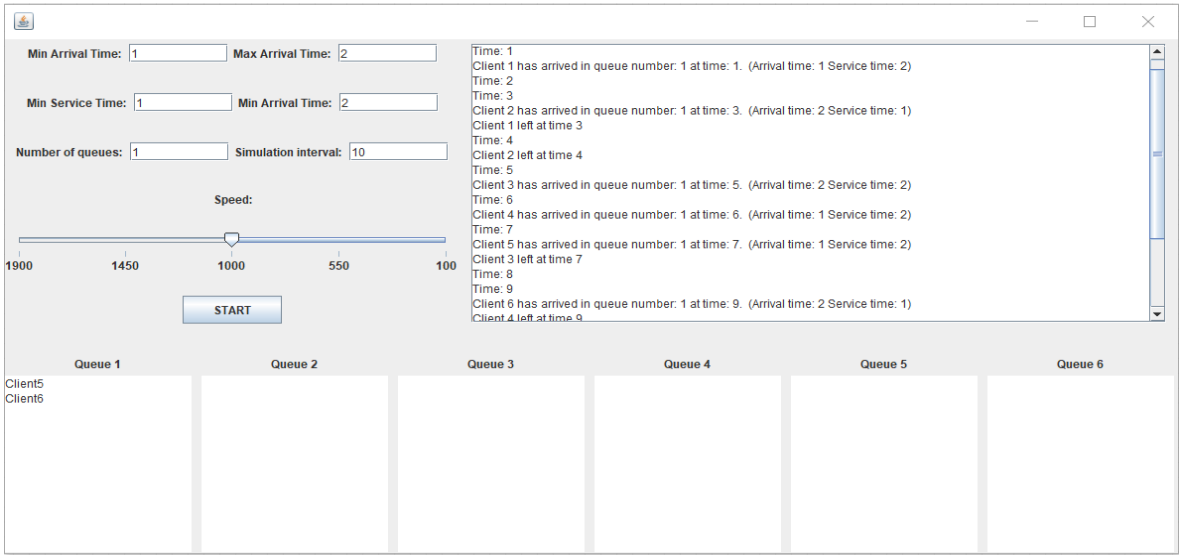
Timpul mediu de asteptare este determina astfel: in clasa Queue verificam daca coada este goala. In caz afirmativ incrementam variabila lastServiceEndTime. Apoi, daca la un moment dat intra un client in coada, calculam timpul sau de iesire, diferenta dintre timpul sau de iesire si timpul curent, iar aceasta diferenta o incrementam la variabila totalWaitingTime. De mentionat faptul ca timpul de asteptare al unui client este diferenta dintre

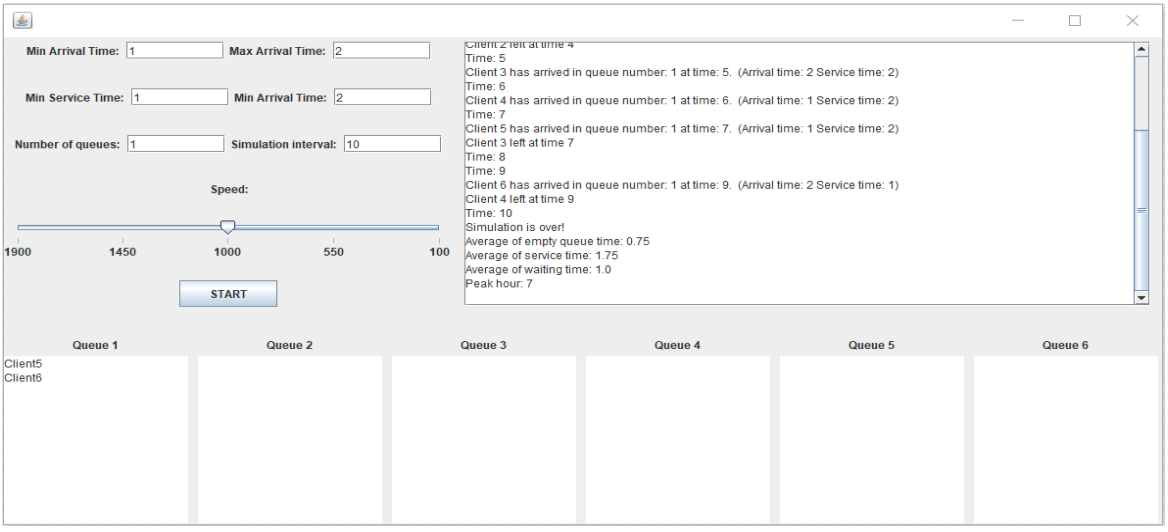
momentul in care un client intra in coada si momentul in care un client iese din coada. In cele ce urmeaza vom imparti timpul respectiv la numarul clientilor care au fost in coada pana la terminarea simularii, vom aduna acesti timpi pentru fiecare coada si vom imparti suma obtinua la numarul de cozi care au fost active.

5. Rezultate

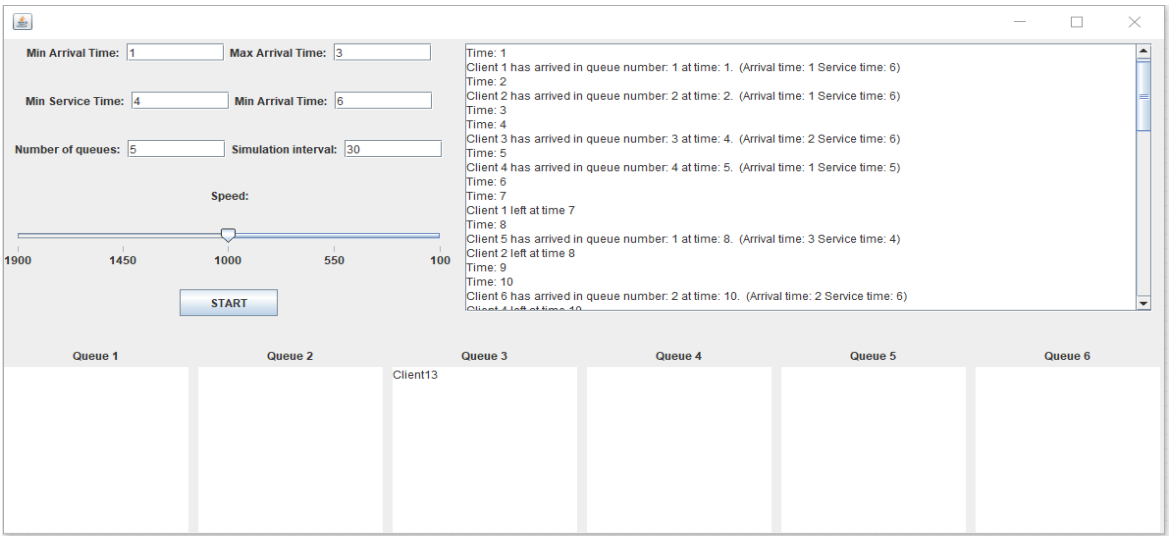
Au fost executate o serie de teste pentru a exemplifica diferitele rezultate care se pot obține în urma interacțiuniii utilizatorui cu interfața.

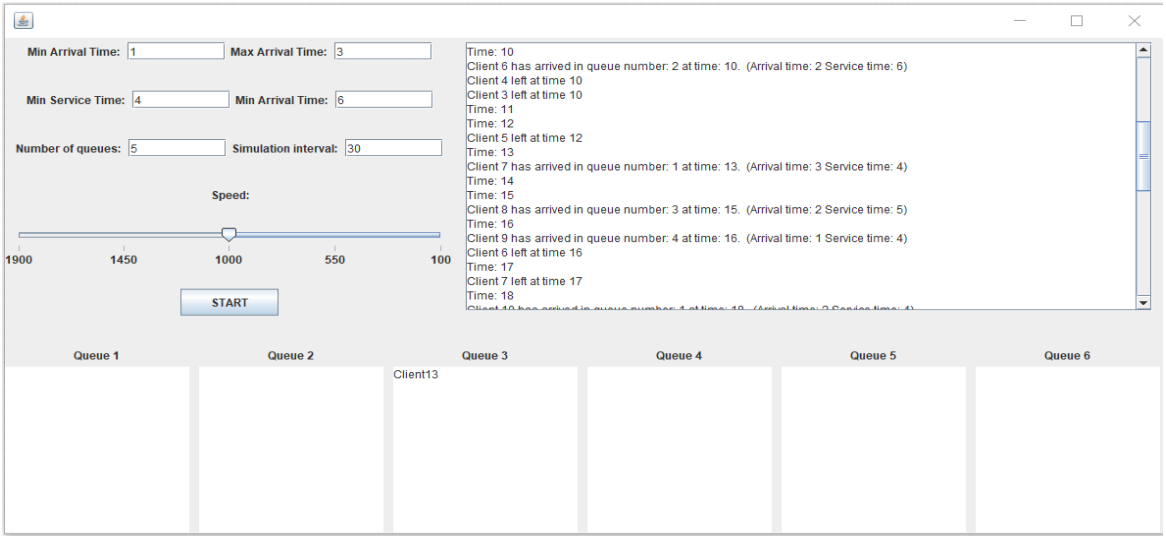
- Simulare realizata pentru o singura coada pe o perioada de 10 secunde:

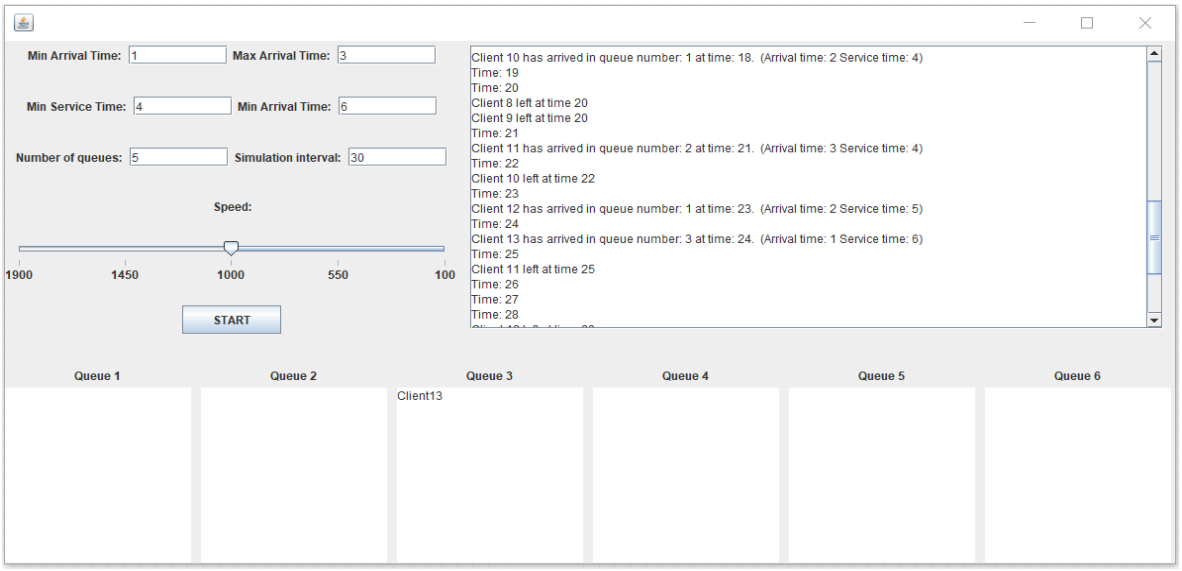


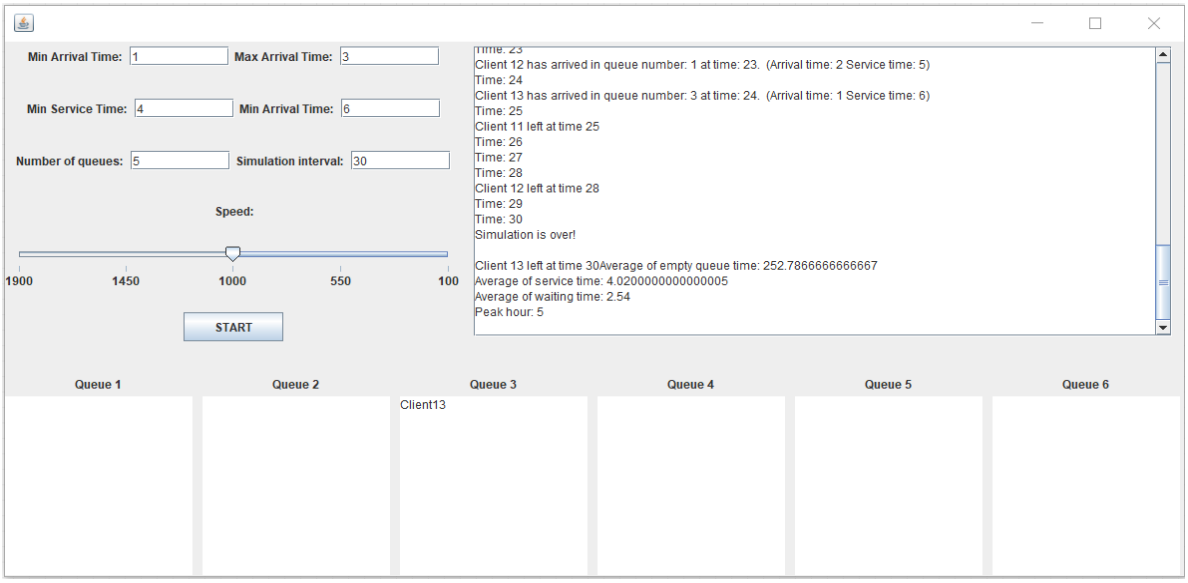


- Simulare realizata pentru o cinci cozi pe o perioada de 30 secunde:









6. Concluzii si dezvoltari ulterioare

Sunt de parere ca aplicația de față prezintă posibilitatea de a fi utilizată cu usurință de absolut orice persoană interesată de simularea evenimentelor care ar putea avea loc la nivelul unor cozi. Ca dezvoltari ulterioare sunt de parea ca ar putea fi implementate mai multe strategii de partajare a clientilor in cozi, in functie de diferite criterii. De asemenea numarul cozilor ar putea fi mai mare.