

**UNIVERZITET U BEOGRADU  
SAOBRAĆAJNI FAKULTET  
ODSEK ZA POŠTANSKI SAOBRAĆAJ I MREŽE**

# **Završni rad**

## **OBJEKTNO ORIJENTISANA SIMULACIJA I ANIMACIJA SISTEMA ZA RAZVRSTAVANJE PAKETA**

Mentor:  
Prof. dr Milorad Stanojević, dipl. inž.

Student:  
Kristian Iker

Beograd, 2014. godina

**UNIVERZITET U BEOGRADU  
SAOBRAĆAJNI FAKULTET  
ODSEK ZA POŠTANSKI SAOBRAĆAJ I MREŽE**

**Z A V R Š N I   R A D**

Tema: OBJEKTNO-ORIJENTISANA SIMULACIJA I ANIMACIJA SISTEMA ZA RAZVRSTAVANJE PAKETA

Teze rada:

1. Dati osnovne pojmove o modeliranju i simulaciji;
2. Objasniti značaj i mogućnost primene objektno orijentisane simulacije;
3. Implementirati i verifikovati simulacioni model sistema za razvrstavanje paketa u RPLC Beograd primenom objektno orijentisane simulacije;
4. Izvršiti komparativnu analizu objektno orijentisanog simulacionog modela sa modelom realizovanim u simulacionom jeziku GPSS/FON;
5. Objasniti značaj i primenu animacije simulacionog modela;
6. Realizovati animaciju korišćenjem objektno orijentisanog simulacionog modela sistema za razvrstavanje paketa.

Mentor:

Prof. dr Milorad Stanojević, dipl. inž.

Student:

Kristian Iker PS 100142

\_\_\_\_\_  
Članovi komisije:

\_\_\_\_\_  
Datum odbrane rada:

\_\_\_\_\_  
Oцена: \_\_\_\_\_

## **Rezime**

*U radu je razvijen objektno orijentisani simulacioni model sistema za razvrstavanje paketa u pošti 11200 Beograd, koji je napisan u programskom jeziku C++. Dati model se zasniva na modelu datog sistema koji je bio realizovan u simulacionom jeziku GPSS/FON. Uporednom analizom rezultata oba simulaciona modela prikazane su sličnosti i razlike, kao i prednosti i mane, ova dva pristupa rešavanju istog problema.*

*Detaljno je objašnjen objektno orijentisani pristup simulaciji koji koristi koncept objekata koji se sastoje od podataka i opisa ponašanja.*

*Korišćenjem HTML5 Canvas i JavaScript internet tehnologija grafički je animiran je celokupan proces, korišćenjem internet pretraživača.*

*U prilogu je dat model sistema realizovan u C++ jeziku, kao i rezultati simulacije.*

## **Abstract**

*The paper presented an object oriented simulation model of a system for sorting packages in a Belgrade's 11200 post station, coded in programming language C++. The shown model is based on a simulation model of the same system created in simulation language GPSS/FON. Comparing results from both models we wanted to show similarities and differences between the two, as well as pros and cons of those two different approaches to the same problem.*

*Objected oriented aproach is explained in details. Objected oriented simulation is a simulation that uses concept of an object which contains both data and specific behavior.*

*Using HTML5 Canvas and JavaScript internet technologies, we have animated a simulation model. Simply by using web browser you are allowed to see the whole process from beginning to end.*

*The model of the system coded in C++ language, as well as the simulation results are presented in the appendix.*

# SADRŽAJ

UVOD .....	1
1. SIMULACIJA .....	3
1.1. Modeliranje i model .....	3
1.2. Vrste modela .....	3
1.3. Podela i vrste simulacionih modela.....	4
1.4. Verifikacija i validacija modela .....	5
1.5. Simulacioni proces .....	5
1.6. Simulacija diskretnih događaja .....	7
1.6.1. Razvoj simulacije diskretnih događaja .....	7
1.7. Programski jezici za izvođenje simulacije .....	9
2. SIMULACIJA U JEZICIMA GPSS I C++ .....	11
2.1. Simulacioni jezik GPSS .....	11
2.2. Osnovni koncept GPSS jezika .....	12
2.3. Vrste naredbi u GPSS-u .....	12
2.4. Objektno orijentisana simulacija.....	13
2.4.1. Vreme.....	13
2.4.2. Entiteti i resursi.....	14
2.4.3. Događaji.....	14
2.4.4. Stanja entiteta i redovi čekanja .....	14
2.4.5. Aktivnosti i granjanje aktivnosti.....	15
2.4.6. Događaji u simulacionoj strategiji raspoređivanja događaja .....	15
2.5. Programski jezik C++ .....	16
2.6. C++ kao objektno orijentisani jezik .....	17
2.7. Sličnosti i razlike C++ i GPSS jezika .....	18
3. ANIMACIJA SIMULACIONOG MODELA .....	22
3.1. Javaskript.....	23
3.2. Javaskript biblioteka za animaciju .....	23
3.3. Datoteka anim.h .....	25
4. SISTEM ZA POLUAUTOMATSKO RAZVRSTAVANJE PAKETA U POŠTI 11200 BEOGRAD .....	27
4.1. Sredstva mehanizacije sistema za razvrstavanje paketa.....	29
4.2. Terminali za usmeravanje paketa.....	30
4.3. Simulacija sistema za razvrstavanje poštanskih paketa .....	30
4.3.1. Ulazne veličine u sistemu .....	31
4.3.2. Operacije i funkcije u sistemu .....	31

5. REZULTATI SIMULACIJE.....	34
5.1. Generatori slučajnih brojeva i raspodele modela .....	35
5.2. Diskretna funkcija izbora kliznica .....	36
5.3. Klasa simulacija .....	37
5.4. Poređenje rezultata dobijenih realizacijama u GPSS i C++ .....	40
5.5. Realizacija animacije .....	45
6. ZAKLJUČAK.....	48
LITERATURA .....	49
PRILOG .....	50

#### Pregled Slika:

Slika 1. Dijagram toka simulacionog procesa	6
Slika 2. Struktura animacionog programa	24
Slika 3. Dijagram toka izvršavanja jednog animacionog okvira	24
Slika 4. Principiska šema za razvrstavanje paketa	28
Slika 5. Verovatnoće izbora pojedinih kliznica	32
Slika 6. Kumulanta verovatnoća izbora pojedinih kliznica	33
Slika 7. Prikaz rezultata simulacije realizovane u C++ jeziku	35
Slika 8. Statistike histograma frekvencije broja paketa u redu čekanja red	40
Slika 9. Broj paketa u redovima čekanja na manuelnom sortiranju	41
Slika 10. Broj ulazaka paketa u redove čekanja	42
Slika 11. Procenat paketa koji čekaju u redovima	42
Slika 12. Srednja iskorišćenost radnika na kodnim mestima i na ručnom sortiranju	43
Slika 13. Broj opsluženih paketa svakog radnika	43
Slika 14. Srednje vreme opsluge radnika	44
Slika 15. Broj paketa spuštenih niz svaku kliznicu	44
Slika 16. Sadržaj kolica	45
Slika 17: Prikaz animacije	46
Slika 18: Prikaz animacije bez pozadine, sa putanjama i mrežom	47

#### Pregled Tabela:

Tabela 1. Operacije datoteke anim.h	26
Tabela 2. Vreme zadržavanja paketa na trakastim transporterima	31
Tabela 3. Vreme zadržavanja paketa na segmentima	31
Tabela 4. Vremena kretanja paketa na sortirnom transporteru 5 od R1 do skretnica	31
Tabela 5. Vremena kretanja paketa na sortirnom transporteru 6 od R2 do skretnica	31
Tabela 6. Verovatnoća izbora kliznice	32

#### Pregled Algoritama:

Algoritam 1. Događaj koji kreira privremene entitete u modelu	15
Algoritam 2. Događaj sa tekućim entitetom koji oslobađa naredni entitet	16

#### Pregled Listinga:

Listing 1. Klasa simulacija C++	37-38
---------------------------------	-------

# UVOD

U radu je razvijen objektno orijentisani simulacioni model sistema za razvrstavanje paketa u pošti 11200 Beograd, koji je napisan u programskom jeziku C++. Data je uporedna analiza rezultata dobijenih objektno orijentisanom simulacijom [1], sa rezultatima prvobitno dobijenih simuliranjem sistema za razvrstavanje paketa realizovanog u simulacionom jeziku GPSS/FON.

Ovaj sistem se sastoji od niza trakastih transportera, tako raspoređenih da četiri sačinjavaju kružnu putanju, dva transportera uvode pakete u ovaj krug a preostala dva su sortirni i sa njih paketi izlaze na kliznice. Trakasti transporteri se koriste za transport paketa, koji se po njima kreću, tako što se paketi usmeravaju na određenu kliznicu u zavisnosti od toga gde su upućeni. U sistemu postoje dva kodna mesta na kojima se vrši pomenuto usmeravanje. Tako sortirani paketi se zatim pakuju u vreće, sačinjavanju se zaključci i vrši se njihovo otpremanje ka odgovarajućim poštanskim jedinicama.

Simulacija rada ovog sistema prvobitno je urađena u diplomskom radu [2] u jeziku za simulaciju diskretnih sistema GPSS (General Purpose Simulation System), tačnije u GPSS/FON verziji jezika. Korišćenjem postojećeg simulacionog modela realizovanog u GPSS jeziku u radu je korišćenjem strategije raspoređivanja događaja taj isti model realizovan i u programskom jeziku opšte namene C++. Takođe, izvršenje simulacije je animirano korišćenjem JavaScript i HTML5 Canvas internet tehnologija. Svi ovi programski jezici kao i same tehnologije će biti ukratko objašnjene kako bi se stvorila jasna slika o problemu koji se rešava.

GPSS (General Purpose Simulation System) je simulacioni jezik koji se koristi za simulaciju diskretnih stohastičkih sistema. U ovom jeziku se pomoću određenih blok naredbi definiše struktura modela i vrši simulacija. Tim naredbama se određuje redosled aktivnosti i operacija koje se tokom simulacije vrše u modelu.

C++ je programski jezik opšte namene koji može biti korišćen za rešavanje najrazličitijeg broja problema, od kojih je simulacija samo jedan od problema. C++ je pre svega objektno orijentisani programski jezik i kao takav biće korišćen za realizaciju objektno orijentisane simulacije. Za razliku od GPSS jezika koje se služi blokovima da bi se model izgradio, C++ jezik mora koristiti jednu od mnogih tačno definisanih strategija kako bi se događaji izvršavali po nekom određenom redosledu. Ove strategije zahtevaju drugačiji pristup pisanju samog programskog koda, i strategija koja se koristi jeste najjednostavnija za realizaciju, međutim sa druge strane događaji su u ovom slučaju i najkompleksiji.

HTML5 Canvas (HyperText Markup Language) predstavlja elemenat za iscrtavanje grafike, obično realizovane u JavaScript tehnologiji.

Javaskript je dinamički, skript programski jezik. Uglavnom se koristi na strani klijenta, u internet pretraživaču, za promenu sadržaja i stila dokumenta, interakciju sa korisnikom i asinhronu komunikaciju sa serverom.

Poštanski saobraćaj se menja i prilagođava novonastalim promenama u društvu i državi, glavni poštanski centar Beograd (GPC Beograd) danas nosi naziv regionalni poštansko-logistički centar Beograd (RPLC Beograd), i izmešten je sa stare lokacije, u samom centru

Beograda, na kojoj se opisani sistem nalazio, na novu lokaciju u Zemunu (Beograd). Ovakav sistem su danas zamenile najsavremenije mašine i proces celokupne prerade postaje gotovo potpuno automatizovan. Međutim ovaj model uz neke izmene može dalje biti usmeren ka nekim budućim rešenjima vezanim za novi RPLC Beograd.

Na samom početku rada dato je objašnjenje pojma simulacija, date su osnovne podele i definicije, način korišćenja, prednosti i nedostaci, kao i uvod u pojmove kao što su diskretni događaji.

U drugom poglavlju rada akcenat je stavljen na objektno orijentisanu simulaciju. Takođe je dat kratak uvod u simulacioni GPSS jezik, kao i u C++ jezik. Objašnjene su i prednosti i mane, kao i sličnosti i razlike ova dva jezika.

U trećem poglavlju je objašnjena animacija simulacionog modela. Dat je kratak uvod u HTML5 Canvas i JavaScript internet tehnologije, primena, kao i prednosti i mane. Takođe objašnjena je i biblioteka realizovana na Saobraćajnom fakultetu koja se može koristiti za bilo koji simulacioni model, a zatim prezentovati kroz animaciju, pomoću internet pretraživača.

Četvrto poglavlje rada odnosi se na sistem za poluautomatsko razvrstavanje paketa koji se nalazio u GPC 11200 Beograd. Date su tehničke karakteristike sistema, ulazne veličine kao i opis operacija i funkcija samog sistema.

Peto poglavlje rada odnosi se na rezultate simulacije, poređenje istih, kao i prezentaciju animacije. U ovom poglavlju su dati podaci prilagođeni C++ jeziku, kao i simulaciona klasa koja se koristi za izvršavanje modela.

U poslednjem poglavlju su data zaključna razmatranja.



# 1. SIMULACIJA

**Simulacija** predstavlja imitaciju nekog realnog sistema ili procesa u nekom vremenskom intervalu. Simulacija se danas koristi u mnogim sverama ljudske delatnosti, od testiranja sigurnosti sistema i rešavanja optimizacionih problema, do video igara. Sama simulacija se može vršiti i na nepostojećim sistemima i kao takva zahteva simulacioni model. Simulacioni model je ključni deo i osnovni uslov za mogućnost primene simulacije, jer se ista ne može vršiti na realnim sistemima [1].

Sam razvoj simulacije i široka primena se usko vezuje za nastanak i razvoj računara, tačno neke proste probleme možemo često simulirati i na primer korišćenjem fizičkih ili matematičkih modela, ali jasno je da kada govorimo o simulaciji u našem slučaju govorimo o simulaciju primenom računara. Moć koju današnji računari poseduju omogućava nam da veoma složene realne sisteme predstavljene odgovarajćim modelima, analiziramo u gotovo nekoliko sekundi, a da zatim za potpuno nove ulazne podatke, neke nove uslove, dobijemo nova rešenja samo nekoliko sekundi kasnije. O svemu ovome će biti više reči nešto kasnije, upoznaćemo se sa prednostima i manama same simulacije, vrstama problema koji se rešavaju, vrstama samih modela, osobinama istih itd.

Sam proces simulacije možemo posmatrati kroz dve osnovne faze [2]:

1. Izrada modela
2. Sprovođenje eksperimenata i analiza rezultata

Uspešnost simulacije zavisi od tačnosti i uspešnosti obe faze.

## 1.1. Modeliranje i model

**Modeliranje** je osnovni proces ljudskog uma. To je isplativo (u smislu troškova) korišćenje nečega (modela) umesto nečeg drugog (realnog sistema) sa ciljem da se dođe do određenog saznanja. Rezultat modeliranja je model [1].

**Model** je uprošćena i idealizovana slika realnosti. Model je apstrakcija realnog sistema, zadržava samo one osobine originala koje su bitne za izučavanje. Nivo apstrakcije utiče na validnost modela tj. na uspešnost predstavljanja realnog sistema preko modela. Isuviše složeni modeli su skupi i neadekvatni dok isuviše prosti modeli ne oslikavaju posmatrani sistem na pravi način [1].

## 1.2. Vrste modela

Postoje nekoliko podela modela, među najpoznatijim su sledeće .

Podela 1:

1. Mentalni (misaoni) modeli. Konstruiše ih ljudski um i na osnovu toga deluje. Omogućavaju komunikaciju među ljudima, planiranje aktivnosti itd.;
2. Verbalni modeli su direktna posledica mentalnih modela, njihov izraz u govornom jeziku. Uobičajeno se predstavljaju u pisanom obliku i spadaju u klasu neformalnih modela;

3. Fizički modeli predstavljaju umanjene modele realnog sistema. Ponašaju se kao njihovi originali a prave se na osnovu teorije sličnosti ili fizičkih zakona sličnosti;
4. Matematički modeli se javljaju ako su veze između objekata opisane matematičkim (numeričkim) relacijama. Polazi se od verbalnog modela koji se transformiše u stanje koje se može opisati matematičkim jezikom. Spadaju u klasu apstraktnih modela a primenjuju se u naučnim i inženjerskim disciplinama;

Različiti fizički modeli mogu imati iste matematičke modele pa se kaže da između njih postoji matematička analogija (analogija u ponašanju). To pruža mogućnost da se neki od fizičkih objekata jednog modela koristi za analizu drugog modela i tada se onda naziva analogni model.

5. Konceptualni modeli nastaju na osnovu strukture, logike rada sistema. Zovu se još i strukturni modeli pošto u grafičkom obliku ukazuju na strukturu sistema te su zgodno sredstvo za komunikaciju. Predstavljaju osnovu za izradu računarskih modela;
6. **Računarski (simulacioni) modeli** su prikaz konceptualnih modela u obliku programa za računar korišćenjem programskih jezika i usko su vezani za razvoj računarske nauke [1].

Podela 2:

Modeli se često dele na materijalne (hemijska struktura molekula, model aviona) i simboličke (matematički, konceptualni, računarski, simulacioni) [1].

Podela modela prema opisu:

1. Neformalni opis modela daje osnovne pojmove o modelu i najčešće nije potpun i precizan. Zbog toga se vrši podela na:
  - a) objekte – to su delovi iz kojih se sastoji model;
  - b) opisne promenljive – opisuju stanje u kome se objekat nalazi u nekom vremenskom trenutku;
  - c) pravila interakcije objekata – definišu se kako objekti modela koji utiču jedni na druge i na opisne promenljive u cilju promena njihovog stanja.

Neformalni opis je dosta brz i lak te zbog toga može biti nekompletan (ne sadrži sve situacije koje mogu da nastupe), nekonzistentan (predviđanje dva ili više pravila za istu situaciju – kontradiktorne akcije), nejasan (ako nije definisan redosled akcija). Ovakve situacije se prevazilaze pravilima i konvencijama u komuniciranju zvanim formalizmi [1].

2. Formalni opis modela treba da obezbedi veću preciznost, potpunost u opisivanju modela. Omogućava i formalizovanje nekompletnosti, nekonzistentnosti i nejasnosti kao i usmeravanje pažnje na karakteristike objekata koje su od najvećeg značaja za istraživanje (apstrakcija) [1].

### 1.3. Podela i vrste simulacionih modela

Često se modeli dele prema vrsti promenljivih (deterministički i stohastički modeli) i prema načinu na koji se stanje modela menja u vremenu (diskretni i kontinualni) [1].

1. Deterministički modeli su modeli čije stanje možemo predvideti, tj. novo stanje je potpuno određeno prethodnim;
2. Stohastički modeli su modeli čije se ponašanje ne može unapred predvideti, ali se mogu predvideti verovatnoće promene stanja;
3. Diskretni modeli su modeli u kojima se stanje sistema menja u tačno određenim vremenskim trenucima, a te promene nazivamo događajima;
4. Kontinualni modeli su modeli u kojima se promenjive stanja menjaju kontinualno u vremenu. Na računarima se ne mogu izvoditi kontinualne promene veličina već se moraju aproksimirati skupom diskretnih vrednosti.

Sa druge strane simulacioni modeli se dele na četiri osnovne vrste [1]:

1. Monte Karlo, statička simulacija kod koje se u rešavanju problema koristi stvaranje uzoraka iz raspodela slučajnih promenljivih, a problemi mogu biti i determinističkog i stohastičkog oblika;
2. Kontinualna simulacija (dinamička), se koristi za dinamičke probleme kod kojih se promenjive stanja menjaju kontinualno u vremenu;
3. Simulacija diskretnih događaja (dinamička), se bavi modeliranjem sistema koji se mogu predstaviti skupom događaja. Događaj je diskretna promena stanja entiteta sistema. Nastupa u određenom trenutku a te promene stanja entiteta se dešavaju diskontinualno u vremenu, tj. samo u nekim trenucima. Između dva događaja stanje sistema se ne menja;
4. Mešovita simulacija (hibridna, kontinualno-diskretna), se uvodi da bi se modelirali oni sistemi koji sadrže procese i događaje. Procesu teku kontinualno dok događaji dovode do diskontinuiteta u ponašanju sistema.

## 1.4. Verifikacija i validacija modela

Validacija i verifikacija su postupci kojim ispitujemo koliko verno i precizno jedan model predstavlja realni sistem. One se konceptualno razlikuju ali se najčešće simultano sprovode, odnosno kaže se da su u dinamičkoj povratnoj sprezi [1].

**Verifikacija** se odnosi na proveru da li je simulacioni program (računarski kod) bez grešaka i konzistentan sa modelom (konceptijom) [1].

**Validacija** se odnosi na proveru da li je model precizna reprezentacija realnog sistema. To je interaktivna procedura kojom se poredi ponašanje modela i realnog sistema sve dok se ne dobije tačnost modela koja zadovoljava [1].

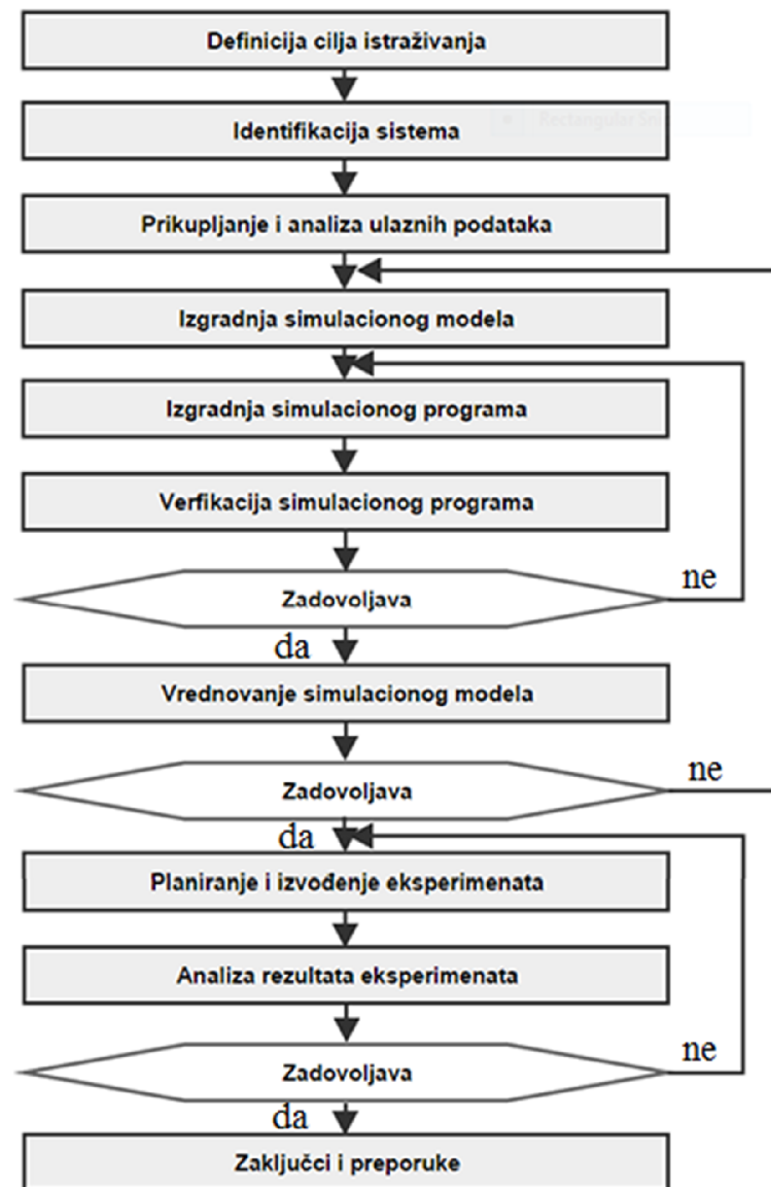
## 1.5. Simulacioni proces

Simulacioni proces je struktura rešavanja stvarnih problema pomoću simulacionog modeliranja. Sastoji se iz više koraka i nije strogo sekvencijalna, moguć je povratak na korake procesa (slika 1).

Koraci simulacionih procesa [1]:

1. Definicija cilja simulacione studije;
2. Identifikacija sistema (opis komponenata, način rada, formalni prikaz sistema);
3. Prikupljanje podataka o sistemu i njihova analiza;

4. Izgradnja simulacionog modela (stvaranje konceptualnog modela koji adekvatno opisuje sistem).;
5. Izgradnja simulacionog programa (izbor programskog jezika i pisanje simulacionog koda);
6. Verifikacija simulacionog programa (da li nam program verno prenosi model);
7. Validacija (vrednovanje) simulacionog modela (da li model adekvatno predstavlja realni sistem);
8. Planiranje simulacionih eksperimenata i njihovo izvođenje;
9. Analiza rezultata eksperimenata (najčešće statistička analiza);
10. Zaključci i preporuke.



*Slika 1. Dijagram toka simulacionog procesa*

## 1.6. Simulacija diskretnih događaja

Simulacija diskretnih događaja je metoda simulacionog modeliranja sistema kod kojih se diskretne promene stanja u sistemu ili njegovom okruženju događaju diskontinualno u vremenu. Koristi se uglavnom za analizu dinamičkih sistema sa stohastičkim karakteristikama. Događaj je diskretna promena stanja entiteta sistema i nastupa u određenom trenutku vremena. Sistemi sa diskretnim događajima su: banke, auto servisi, pošte, samoposluge jer se stohastički karakter ogleda u slučajnoj prirodi veličina (događaja) za opis ovakvog sistema. Prema Šanonu, opis diskretnih događaja je sledeći: Entiteti, koji se opisuju atributima i uzajamno deluju u aktivnostima, pod određenim uslovima stvaraju događaje koji menjaju stanje sistema [1].

Kod modela sa diskretnim događajima, pored koncepata koji opisuju strukturu kao što su objekti, relacije između njih i njihovi atributi, uvedeni su i koncepti za opis dinamike: **događaj, aktivnost i proces** [1].

**Događaj** predstavlja diskretnu promenu stanja entiteta u sistemu ili njegovom okruženju. Između dva uzastopna događaja stanje sistema se ne menja. Događaj može nastupiti zbog ulaska/izlaska privremenog entiteta u/iz sistema (dolazak/odlazak klijenta) – eksterni; ili zbog promene atributa pojedinih objekata sistema (opslužilac u sistemu postaje slobodan ili zauzet) - interni. Uslovni događaji mogu nastupiti kada je ispunjen uslov njihovog nastupanja. Obično su povezani sa zauzimanjem nekog resursa. Bezuslovni događaji su oni čiji je jedini uslov da tekuće vreme simulacije bude jednako vremenu njegovog nastupanja. Obično su povezani sa oslobađanjem nekog resursa odnosno planiranim završetkom neke aktivnosti [1].

**Aktivnost** je skup događaja koji menjaju stanje jednog ili više entiteta. Trajanje aktivnosti može biti unapred definisano (determinističko) ili da zavisi od ispunjenja uslova pa je vreme završetka takve aktivnosti nepoznato (stohastičko) [1].

**Proces** je niz uzastopnih, logički povezanih događaja kroz koje prolazi neki privremeni objekat. To je hronološki uređena sekvenca događaja koja opisuje jednu pojavu od nastajanja do terminiranja. On može da obuhvati deo ili celokupan život privremenog entiteta [1].

### 1.6.1. Razvoj simulacije diskretnih događaja

Ključni elementi razvoja simulacije diskretnih događaja su [1]:

1. **Mehanizam pomaka vremena.**
2. **Generisanje događaja.**
3. **Strategije simulacije.**

#### Mehanizam pomaka vremena

U simulaciji diskretnih događaja koriste se dva osnovna mehanizma pomaka vremena [1]:

1. *Pomak vremena za konstantni priraštaj* - Vreme u simulacionom modelu se menja tako da se uvek dodaje konstantan priraštaj  $\Delta t$ . Nakon svakog pomaka vremena, odnosno ažuriranja vrednosti simulacionog sata, ispituje se da li je u prethodnom intervalu vremena trebalo da dođe do nastupanja nekih

događaja. Ukoliko jeste, tada se ti događaji planiraju za kraj intervala. Nedostatak je u tome što se kod pomeranja događaja na kraj vremenskog intervala uvodi greška u simulaciji. Događaji koji nisu istovremeni u ovom se pristupu prikazuju kao istovremeni, a potom se određuje redosled njihovog izvođenja (koji se može razlikovati od stvarnog redosleda).

Smanjenjem vremenskog prirasta te se greške smanjuju, ali se povećava vreme koje se troši na izvođenje simulacije kao i porast broja vremenskih intervala u kojima nema događaja.

2. *Pomak vremena na naredni događaj* - Simulacioni sat se pomera na vreme u kom će nastupiti prvi naredni događaj (ili više njih). Simulacija se završava kada nema više događaja ili kada je zadovoljen neki unapred definisan uslov završetka simulacije. Na ovaj način se izbegava greška u vremenu izvođenja događaja a ujedno se preskaču intervali u kojima nema događaja. Ovaj princip je složeniji ali i efikasniji pa svi ključni simulacioni jezici koriste ovaj mehanizam. Simulacioni jezik GPSS, u kojem je realizovan diplomski rad [2], takođe koristi ovaj mehanizam. Iz gore navedenih razloga, simulacija realizovana u C++ jeziku će pratiti mehanizam pomaka vremena na naredni događaj, a kao primer ovi događaji mogu redom biti, dolazak paketa, čekanje u redu, put paketa na određenoj deonici, kao i sam završetak prerade tog jednog paketa, koji je kasnije nazvan uništenjem paketa, u smislu da je paket napustio model.

## **Generisanje događaja**

Događaj se opisuje sa više atributa, koji formiraju slog događaja. S obzirom na promenljiv broj događaja u vremenu, slogovi događaja se memorišu u listama događaja. Postoje dva pristupa generisanja događaja [1]:

1. *Definisanje događaja unapred* – Svi događaji su unapred poznati i definisani, a lista događaja sadrži slogove svih događaja;
2. *Definisanje narednog događaja* – Poznat je jedino prvi naredni događaj, a lista događaja sadrži samo jedan slog, slog poznatog događaja. Pri izvršavanju događaja, planira se i ubacuje u listu njegov naslednik.

Događaje možemo svrstati u dve kategorije u odnosu na mesto nastajanja (generisanja):

1. *Eksterni događaji* - su oni događaji koji ne zavise od modela i predstavljaju uticaj okoline na sistem (dolazak kupaca u samoposlugu, vozača u auto servis);
2. *Interni događaji* - zavise od modela i u njemu se generišu (dolazak kupaca u red na kasu).

## **Strategija izvođenja simulacije**

Strategije su [1]:

1. *Raspoređivanje događaja* - podrazumeva da se događaji planiraju unapred i drže u listi budućih događaja, najčešće sortirani po vremenu nastupanja i prioritetu. Procedura planiranja događaja: generiše se slog događaja; dodele se vednosti njegovih atributa (vreme nastajanja, prioritet); događaj se stavlja u listu budućih događaja koja je uređena po vremenu nastupanja događaja i prioritetu.

Funkcionisanje simulatora: sa liste budućih događaja uzima se prvi; ažurira se simulacioni sat na vreme njegovog nastupanja; na osnovu tipa izabranog događaja poziva se odgovarajuća procedura koja izvršava sva ažuriranja u modelu i simulatoru; kada se izvrše svi događaji koji imaju isto vreme nastupanja, simulacioni sat se ažurira na vreme sledećeg događaja iz liste budućih događaja.

2. *Skaniranje aktivnosti* - podrazumeva da se događaji implicitno raspoređuju tako da se promena stanja izvršava preko funkcija koje se nazivaju aktivnosti. Svaka aktivnost ima **uslov** i **akciju**. Za svaki vremenski korak  $\Delta t$ , aktivnosti se skaniraju i traži se prva aktivnost koja ima zadovoljen uslov. Tada se izvršava odgovarajući programski segment koji specificira akciju za zadatu aktivnost. Proces skaniranja traje sve dok sve aktivnosti ne budu blokirane. Tada i samo tada se simulacioni sat ažurira na sledeći vremenski korak. Skaniranje aktivnosti je bolje od raspoređivanja događaja kada je broj aktivnosti u modelu mali, a broj događaja u okviru aktivnosti veliki, jer se štedi na računarskim operacijama koje se odnose na listu i njegovo skidanje sa liste događaja.

3. *Interakcija procesa* - predstavlja tehniku simulacije koja je nastala kombinacijom raspoređivanja događaja i skaniranja aktivnosti. Proces možemo posmatrati kao skup isključivih aktivnosti, povezanih tako da terminiranje jedne aktivnosti dozvoljava inicijalizaciju neke druge aktivnosti iz skupa aktivnosti procesa. Glavni problem je sinhronizacija procesa jer je model sistema skup paralelnih procesa od kojih neki mogu biti uzajamno isključivi. Da do ovoga ne bi došlo uvode se dve naredbe WAIT i DELAY i to i u uslovnom i u безусловnom kontekstu. WAIT naredba u uslovnom kontekstu ima oblik: WAIT UNTIL <uslov> – ako je uslov zadovoljen, proces koji čeka na taj uslov nastavlja svoj tok, a u suprotnom biće blokiran. Tada se ispituje koji od ostalih procesa može da se aktivira i on se izvršava. DELAY naredbom se odlaže nastavak procesa za određeni broj vremenskih jedinica. U tom slučaju, ispituje se koji od ostalih procesa može da ostvari svoj tok. U безусловnom kontekstu naredba DELAY označava se i sa ADVANCE <broj vremenskih jedinica>.

## 1.7. Programski jezici za izvođenje simulacije

Postavlja pitanje kako izabrati programski jezik kojim ćemo prezenzovati model. Simulaciju je moguće izvesti pomoću programskih jezika opšte namene, kao što su, FORTRAN, PASCAL, BASIC, C jezici i drugi. Sa druge strane, model, možemo opisati simulacionim programskim jezicima koji su specijalizovani za ovaj tip problema. Izbor jezika zavisi i od same simulacije. Većina programskih jezika opšte namene, kao C++, su dobro standardizovani, imaju kompajlere za sve tipove računara i razvojno okruženje je dostupno na gotovo svim operativnim sistemima, dostupe su biblioteke sa gotovim programima i potprogramima, logaritmima i skladištima podataka, itd. Međutim simulacije napisane u programskim jezicima opšte namene, su relativno duge i glomazne. Svaka promena u simulacionom modelu često zahteva dosta dug vremenski period prepravljanja samog koda, a često isti može biti i napisan na najrazličitiji broj načina što otežava posao, onom, koji se bavi modifikovanjem. Sa druge strane simulacioni jezici su uglavnom namenjeni za rešavanje jedne klase problema, i često se javlja potreba za nekim dodatnim uslovom, ili nekom dodatnom statistikom, takvom da je njeno realizovanje jako komplikovano ili čak i nemoguće [2].

Simulacioni program je moguće realizovati u specijalizovanim simulacionim jezicima (GPSS, SIMSCRIPT II/III, SIMUL8,...), simulacionim paketima (Arena, Automod, Flexsim, Witness, ...) ili u nekom programskom jeziku opšte namene (Pascal, C, C++, Java, Python,...). U programskom jeziku moguće je iskoristiti neku od postojećih simulacionih biblioteka (sim (C++), CSIM 19 (C), baseSim (Delphi), C++ SIM, JavaSim, SimPy (Python), itd.) ili napraviti simulacioni program od početka [3].

GPSS – General Purpose Simulation System je jezik koji se naručito koristi za simulaciju modela sistema masovnog opsluživanja, a posebno kod sistema gde je matematička i statistička analiza veoma složena ili ne daje zadovoljavajuće rezultate. GPSS jezik je razvio IBM 1961. godine, a to je bila verzija GPSS-I. Neke verzije ovog jezika su: GPSS/360, GPSS V, GPSS PC (verzija za personalne računare), GPSS/H (verzija za personalne računare pod UNIX ili DOS operativnim sistemom, ova verzija omogućava vezu iz FORTRAN i C jezika) [2].

U Beogradu postoji GPSS/FON verzija, razvijena na Fakultetu organizacionih nauka od strane profesora dr. Božidara Radenković. Ova verzija jezika GPSS realizovana je u PASCAL-u. Na njoj se rade vežbe na Saobraćajom fakultetu.



## 2. SIMULACIJA U JEZICIMA GPSS I C++

### 2.1. Simulacioni jezik GPSS

GPSS (General Purpose Simulation System) predstavlja **interpreterski** jezik za simulaciju diskretnih – stohastičkih sistema. Struktura modela se definiše a simulacija izvršava pomoću naredbi ugrađenog jezika [1].

Nakon simulacije na raspolaganju su statistički pokazatelji o ponašanju modela u toku simulacije. GPSS je jezik orijentisan na procese, model se predstavlja dijagramom toka koji se konstruiše uz pomoć blokova. Svakom bloku odgovara jedna linija. Blokovi su statički objekti.

Entitete (objekte modela) možemo podeliti u četiri klase [1]:

1. **Dinamički entiteti** - Transakcije su dinamički objekti GPSS jezika koje se kreću kroz blokove. One imaju niz karakteristika koje nazivamo parametrima. One su aktivni objekti i može postojati više transakcija u svakom trenutku na različitim mestima u blok dijagramu. Predstavljaju saobraćajne jedinice u modelu (klijenti na opsluživanju, telefonski pozivi, vozila na servisu). Transakcije mogu da rezervišu određene resurse u modelu.
2. **Statički entiteti** - To su elementi opreme, pasivni su i predstavljaju resurse na koje se deluje transakcijama. Tu spadaju: uređaji (mogu da usluže samo jednu transakciju u nekom vremenskom trenutku), skladišta (mogu da usluže više transakcija istovremeno u zavisnosti od kapaciteta) i logički prekidači (stanje prekidača se postavlja prolaskom transakcije kroz njega; mogu biti u tri stanja uključeno, isključeno i invertovano; služe za testiranje stanja opreme i zavisno od toga transakcija se preusmerava).
3. **Statistički entiteti** - U ovu grupu spadaju redovi i tabele. Ako je uređaj zauzet ili skladište puno tada transakcija formira red. Na osnovu statistike GPSS automatski formira tabele sa vrednostima. Korisnik može i sam da formira tabele sa statističkim vrednostima koje su mu potrebne. Ona sadrži frekvencijske klase sa numeričkim vrednostima nekog atributa.
4. **Entiteti operacija** - Oni se nazivaju blokovima i pružaju logiku sistema, dajući instrukcije transakcijama gde treba da idu i šta dalje da rade. Blok dijagram predstavlja operacije koje se vrše unutar datog sistema. Blok pokazuje tip operacije koju obavlja a linije između blokova ukazuju na tok saobraćaja. Unutar blok dijagrama mogu da nastanu četiri osnovna tipa događaja:
  - a) Stvaranje ili uništavanje transakcija;
  - b) Promena numeričkih atributa entiteta;
  - c) Kašnjenje transakcije za određeno vreme;
  - d) Promena toka transakcije kroz blok dijagram.

## 2.2. Osnovni koncept GPSS jezika

Transakcije generiše blok GENERATE. Ona se kreće kroz model sve dok ne naiđe na blok koji nema uslova da je primi ili ne naiđe na blok TERMINATE koji uklanja transakciju iz modela. Takođe, postoje blokovi koji zadržavaju transakciju za određeni period simulacionog vremena. Ako neki blok nema uslova da primi transakciju, onda ona čeka da se ispuni uslov daljeg kretanja kroz model. Prolaz transakcija kroz određene blokove izaziva promene koje utiču na stanje modela i njegovo okruženje [1].

Transakcije u GPSS-u se čuvaju u listi tekućih događaja (LTD) i listi budućih događaja (LBD) [1].

Transakcije iz LTD nastoje da se kreću kroz blok dijagram. Transakcije iz LBD nisu spremne za kretanje, već kasnije, kada se ažurira simulacioni sat (ispunjeni uslovi kretanja), ove transakcije se prebacuju u LTD, kako bi nastavile kretanje kroz model [1].

U svakoj tački simulacionih vremena (simulacioni sat zadat numeričkim atributom C1 uzima samo celobrojne vrednosti), GPSS skanira sve transakcije u LTD i svaka koja ima uslov daljeg kretanja to i čini [1].

Kretanje transakcija se prekida zbog jednog od tri razloga [1]:

1. transakcija je uništena;
2. blok odbija da primi transakciju;
3. transakcija je ušla u blok koji je zadržava izvestan period simulacionog vremena.

Ukoliko je pitanju treći razlog, transakcija se prebacuje iz LTD u LBD. Ukoliko je kretanje transakcije izazvalo promenu stanja modela, GPSS ponovo obavlja skaniranje, ali od početka LTD; u suprotnom naredna transakcija iz LTD se skanira [1].

Kada nastupi trenutak da nijedna transakcija iz LTD ne može da se kreće, GPSS ispituje LBD. LBD je uređena tako da se na njenom početku nalaze transakcije koje uslov za kretanje kroz model stižu ranije, dok su na kraju one transakcije koje moraju duže da čekaju. Kada nijedna transakcija iz LTD ne može da nastavi kretanje, GPSS ažurira simulacioni sat na vreme prve transakcije iz LBD. Sve transakcije iz te liste, koje su sada stekle uslov za kretanje, kopiraju se iz LBD u LTD i skaniranje LTD se ponavlja [1].

Svaka transakcija ima svoj prioritet, koji se može menjati sa protokom simulacionog vremena i trenutnim položajem transakcije u blok dijagramu. LTD je sortirana po opadajućem redosledu prioriteta, tako da se transakcije sa većim prioritetom kreću pre onih čiji je prioritet manji u svakom trenutku simulacionog vremena [1].

## 2.3. Vrste naredbi u GPSS-u

1. Deklaracione naredbe entiteta definišu attribute pojedinih permanentnih entiteta u programu. Tu spadaju naredbe za deklaraciju uređaja, skladišta, tabela (histograma), funkcija i dr. Mogu se pisati bilo gde ali je uobičajeno da se pišu na početku programa. U polju LOKACIJE specificira se posebno definisano skladište, tabela i sl. a u polju VARIJABLE mogu da budu upisani različiti argumenti.

2. Blok naredbe služe za specifikaciju modela sistema. One se izvršavaju kada transakcija prilikom kretanja kroz model naiđe na blok. Model se sastoji od niza blok naredbi povezanih u obliku blok dijagrama. Svaka blok naredba može da ima identifikacioni broj, koji se piše u polju lokacije. Ako korisnik želi da se pozove na neki blok on može da u polje LOKACIJE unese određeni mnemonički simbol. U polju OPERACIJE piše se tip blok naredbe, a u polju VARIJABLE pišu se argumenti.
3. Kontrolne naredbe služe za kontrolu izvršenja simulacije, a mogu imati uticaj na statistiku o ponašanju entiteta u toku simulacije. Simulator zahteva neke informacije upravljanja kao što su dužina simulacije, kada je kraj naredbi programa i dr. U polju OPERACIJA piše se tip naredbe, u polje VARIJABLE unose se argumenti specificirane naredbe a polje LOKACIJE se ne koristi. To su naredbe SIMULATE,

START, RESET, CLEAR, END.

Trajanje simulacije može biti ograničeno brojem transakcija koje su prošle kroz model, što se kontroliše pogodnom upotrebom naredbi START i TERMINATE ili vremenski (fiksno), što je određeno tzv. Tajmerom koji čini par naredbi GENERATE i TERMINATE [1].

## **2.4. Objektno orijentisana simulacija**

Objektno orijentisana simulacija je simulacija koja koristi koncept objekata koji se sastoje od podataka sa jedne strane i opisa ponašanja sa druge strane. Kreiraju se šabloni koje predstavljaju uopštenu sliku nekog dela podsistema ili celog sistema određenog simulacionog modela. Na primer takav jedan sistem može predstavljati univerzalni poštanski šalter koji vrši ulogu opsluživanja korisnika u pošti. Šablon se kreira tako da se u napred predvide veličine i osobine koje bi ovakav sistem zahtevao. Sa druge strane i sam korisnik može biti predstavljen nekim novim šablonom, sa nekim novim karakteristikama. Ovakav pristup simulaciji omogućava da se korišćenjem istog šablona kreiraju različiti objekti koje će imati iste ili slične osobine kao i sam šablon, ponašaće se slično i u modelu će biti realizovani na sličan način. Drugim rečima to znači da iz istog šablona za univerzalni šalter možemo napraviti dva potpuno različita objekta (šalter 1 i šalter 2) sa drugačijim vremenima opsluge, međutim iako različita, ova dva objekta će se slično ponašati u smislu opsluge korisnika. Na ovaj način, jednostavnim kreiranjem i uništavanjem objekata mi možemo kreirati novog korisnika, novi šalter itd. sa potpuno jedinstvenim karakteristikama a sličnim ponašanjem. Terminologija je ovde namerno izostavljena radi lakšeg razumevanja samog pristupa objektno orijentisanoj simulaciji, i biće detaljno objašnjena u daljem delu rada.

Simulacione biblioteke i programi treba da sadrže određene elemente neophodne za realizaciju diskretne stohastičke simulacije. Te elemente sačinjavaju: simulaciono vreme, entiteti, stanja i aktivnosti entiteta, resursi, događaji, redovi čekanja [3].

### **2.4.1. Vreme**

Stanje modela sistema zasnovanog na diskretnoj stohastičkoj simulaciji menjaju događaji koji se odigravaju u diskretnim vremenskim trenucima. Svaki momenat u vremenu u kome se jedan ili više događaja odigravaju naziva se vremenskim trenutkom [3].

Razlikujemo dva vremena simulatora: relativno i apsolutno. Relativno vreme simulacije je vreme od početka do završetka neke faze u toku simulacije (npr. radni dan) nakon čega se vreme resetuje. Apsolutno vreme simulacije je vreme od početka do kraja simulacije [3].

Za strategiju koja je ovde korišćena vreme simulacije se uvek pomera iz vremena izvršenja prethodnog u vreme izvršenja prvog narednog događaja (mekanizam pomaka na naredni događaj) [3].

#### **2.4.2. Entiteti i resursi**

Objekti čije se aktivnosti modeliraju u simulacionom programu nazivaju se entitetima, koji su opisani atributima (svojstvima). Atributa može biti više, ali dva su osnovna: vreme odigravanja narednog događaja i naredni događaj [3].

Resursi su objekti simulacionog modela čija je osnovna uloga da ograniče aktivnosti entiteta. Resursi imaju na raspolaganju jedno ili više mesta u koje primaju entitete. Entiteti zauzimaju i oslobađaju mesta u resursu [3].

Postoji razlika između permanentnih i privremenih (ili prelaznih) entiteta. Permanentni entiteti se kreiraju na početku simulacije i do kraja simulacije se ne uništavaju, dok privremeni entiteti se kreiraju po potrebi u toku simulacije i uništavaju se onda kada više nisu neophodni [3].

#### **2.4.3. Događaji**

Događaj je diskretna promena stanja entiteta u određenom vremenskom trenutku. Između dva događaja stanje se ne menja. Razlikujemo dve vrste događaja [3]:

1. bezuslovni događaj (planirani događaj) je onaj čije se odigravanje može predvideti i zato se unapred planira;
2. uslovni događaj (neplanirani događaj) je onaj čije odigravanje zavisi od ispunjenja određenog uslova (npr. raspoloživosti resursa).

Onda kada entitet učestvuje u nekom planiranom događaju, vreme odigravanja planiranog događaja se pamti u entitetu. Događaj se aktivira onda kada je vreme nastupanja događaja jednako vremenu simulacije. Postoji jedna značajna vrsta planiranih događaja kojim se realizuju dolasci privremenih entiteta u sistem. Svaki put kada se taj događaj odigra on stvara naredni entitet koji ulazi u sistem i postavlja vreme njegovog dolaska [3].

#### **2.4.4. Stanja entiteta i redovi čekanja**

Jednom kada se kreira entitet može da bude u jednom od sledećih stanja [3]:

1. aktivan - događaj entiteta se upravo izvršava;
2. planiran - događaj entiteta je planiran i entitet čeka da se izvrši;
3. blokiran - događaj entiteta čeka da se izvrši sve da se ne ispuni neki uslov;
4. uništen - entitet je predviđen za uništavanje.

Oni entiteti koji se nalaze u blokiranom stanju smeštaju se u redove čekanja. Redovi čekanja mogu imati bilo koju disciplinu opsluživanja, ali naj češće su FIFO tipa.

### 2.4.5. Aktivnosti i granjanje aktivnosti

Aktivnost čini skup događaja koji menjaju stanje entiteta. Obično započinju uslovnim događajem, a završavaju se planiranim graničnim događajem. Takve aktivnosti se nazivaju "zadržavanjem" i njihovo vreme trajanja nije unapred poznato. Nekada je vreme trajanja aktivnosti moguće i unapred planirati sa jednim ili više bezuslovnih događaja [3].

U sistemu entiteti ponekad treba da izaberu između različitih putanja. Granjanje se može odigrati bilo iza planiranog ili uslovnog događaja. Postoje različiti kriterijumi za izbor rute [3]:

1. izbor putanje je slučajan i ne može se unapred predvideti,
2. izbor putanje zavisi od neke karakteristike entiteta,
3. izbor putanje zavisi od stanja resursa, dužine reda čekanja, itd.

### 2.4.6. Događaji u simulacionoj strategiji raspoređivanja događaja

Kada su događaji u simulacionim strategijama u pitanju moguće je uspostaviti jedno pravilo, a to je da što je simulaciona strategija jednostavnija to su događaji složeniji. Tako možemo reći da su za interakciju procesa koju je najteže realizovati događaji najjednostavniji, dok su događaji za raspoređivanje događaja najsloženiji [3].

Strategija raspoređivanja događaja ima samo bezuslovne događaje tako da je u okviru tih događaja potrebno realizovati i ono što se odigrava u uslovnim događajima druge dve strategije. U bezuslovnom događaju koji kreiraju privremene entitete u modelu se ukoliko entitet treba da zauzme neki resurs prvo taj entitet smešta u red čekanja, a nakon toga se proverava zauzeće resursa, oslobađanje iz reda i zauzimanje mesta ukoliko je slobodan i raspoređivanje za opslugu (algoritam 1). U tom događaju se kreira novi entitet i raspoređuje mu se vreme dolaska. Kod ostalih događaja je bitno naglasiti da ukoliko u prethodnom bezuslovnom događaju ima smeštanja u redčekanja i zauzimanja resursa, neophodno je da onaj entitet koji je izašao iz resursa kasnijem događaju oslobodi naredni entitet reda da zauzme njegovo mesto u resursu (algoritam 2) [3].

```
{ Bezuslovni događaj }  
procedure DolazakEntiteta  
begin  
    Postavi resurs u red dačeka;  
    if resurs raspoloživ then  
        Primi prvi entitet iz reda;  
        Zauzmi jedno mesto u resursu;  
        Postavi vreme za kraj opsluge entiteta;  
    end  
    { Kreiranje i raspoređivanje novog entiteta }  
    Kreiraj sledeći entitet;  
    Postavi vreme dolaska sledećeg entiteta;  
end
```

*Algoritam 1. Događaj koji kreira privremene entitete u modelu*

```
{ Bezuslovni događaj }  
procedure NekiBezuslovniDogađaj  
begin
```

```

...
{ Entitet oslobađa naredni entitet }
if red nije prazan then
    Primi prvi entitet iz reda;
    Zauzmi jedno mesto u resursu;
    Postavi vreme za kraj opsluge entiteta;
end
end

```

*Algoritam 2. Događaj sa tekućim entitetom koji oslobađa naredni entitet*

## 2.5. Programski jezik C++

C++ je jezik **opšte namene** (*eng. general-purpose*), **srednjeg nivoa apstrakcije** (*eng. middle-level language*), podržava **objektno orijentisano** (*eng. objected oriented programming*), **proceduralno** (*eng. procedural programming*) i **generičko** (*eng. generic programming*) programiranje. To je jezik koji se kompajlira (*eng. compiled language*), jezik koji **pravi razliku između malih i velikih slova** (*eng. case-sensitive language*) i jezik koji je **slobodne forme** (*eng. free-form language*). C++ je takođe jezik **statičkog tipa** (*eng. statically typed language*) [4-6].

Ovakav opis programskog jezika programerima daje okvirnu sliku o njegovoj složenosti, pravilima pisanja i mogućnostima. Svaka od gore navedenih osobina ima svoje značenje i redom će biti objašnjene.

Kada kažemo da je neki programski jezik, jezik opšte namene, mislimo na to da se taj jezik može koristiti za kreiranje aplikacija u najrazličitijem broju domena, drugim rečima to znači da jezik kao C++ može biti korišćen za izradu aplikacija koje bi se bavile upravljanjem hardverskih komponenti, animacijom, simulacijom, izradom teksta, izradom internet stranica, audio i video tehnologijama, video igricama i mnogim drugim aplikacijama. Kao primer, za razliku od C++, GPSS nema mogućnost izrade animacije i zato kažemo da GPSS nije jezik opšte namene, već tačno određen za specifičnu vrstu problema [4-6].

C++ je jezik srednjeg nivoa apstrakcije i ako pruža mogućnosti i nižih i viših nivoa. Da bi objasnili šta znači srednji nivo apstrakcije prvo ćemo objasniti niži i viši nivo apstrakcije. Niži (*eng. low-level language*) je programski jezik koji je blizak mašinskom jeziku, udaljen je od korisnika u smislu razumljivosti i načina na koji se jezik piše i/ili čita. Niži programski jezici omogućavaju direktan pristup memoriji, brzi su u pogledu izvršavanja instrukcija, i koriste se za pisanje softvera koji upravljaju hardverom, kao na primer operativni sistemi ili se koriste za izgradnju kompajlera jezika koji se nalaze na višim nivoima. Sa druge strane jezici višeg nivoa apstrakcije su jezici koji su jako bliski korisnicima (*eng. users*), kod koji se piše je razumljiv, same komande su najčešće na engleskom jeziku i razumljive su. To su jezici koji često imaju ugrađene algoritme koje vode računa o zauzetoj memoriji, kako ne bi došlo do njenog potpunog zauzimanja, nekim podacima koji se više ne koriste i time usporavali sam računar koja izvršava program. Konačno, jezici srednjeg nivoa predstavljaju nešto između, u smislu da je kod dovoljno jasan i razumljiv za kreiranje i održavanje složenih aplikacija, ali isto tako, sama apstrakcija nije gurnuta do krajnjih granica. Drugim rečima ako bi uporedili programske jezike C++ i Ruby (jezik višeg nivoa), mogli bi reći da su oba jezika objektno orijentisana, međutim Ruby je u ovom slučaju potpuno objektno orijentisani jezik i sve se može posmatrati kao objekat, dok sa druge strane u C++ jeziku postoje osnovni tipovi podataka,

koji se predstavljaju direktno u memoriji, kao što su celi brojevi (`int`), karakteri (`char`) itd. [4-6].

U klasičnom programiranju, kod i podaci su odvojeni. Objektno orijentisano programiranje menja taj odnos i za razliku od struktuiranog programiranja, pruža mogućnosti korisnicima da pakujemo kod i podatke zajedno u nešto što se zove klasa (eng. *class*), u tom smislu klase mogu imati neko stanje ili vrednost (eng. *data fields - attributes*) i ponašanje (eng. *procedures - methods*). Kad klasu jednom deklariramo, možemo je koristiti za formiranje pojedinačnih stavki podataka – objekata (eng. *instance*). Svaki objekat ili instanca klase ima sopstvene vrednosti ali ima zajedničko ponašanje koje je programirano u klasi [4-6].

Generičko programiranje se odnosi na vrstu programiranja gde se algoritmi i kontejneri podataka pišu tako da se tip podataka koji koriste specificira kasnije. Ovo pruža programerima više mogućnosti prilikom pisanja programa, jer isti algoritam mogu primeniti na više različitih kontejnera [4-6].

Programski jezik slobodnog tipa je onaj jezik u kome pozicija ispisanog teksta tj. programskog koda na ekranu ne utiče na sam program, drugim rečima lokacija karaktera na strani programskog teksta nije značajna [4-6].

Statički i dinamički tip programskog jezika se odnosi na to da, u slučaju statičkog programiranja tipovi podataka su zadati u fazi kompajliranja, dok se kod dinamičkog tipa programiranja, tipovi podataka zadaju u fazi izvršenja programa (eng. *run-time*). Ovako dat opis svrstava C++ u obe kategorije ali se po standardizaciji C++ svrstava u statički tip [4-6].

C++ jezik je kreirao profesor **Bjarne Stroustrup** 1979. godine, baziran je na C jeziku i prvobitno nazvan C sa klasama, 1983. godine dobija naziv C++.

## 2.6. C++ kao objektno orijentisani jezik

Objektno-orijentisano programiranje (OOP) je deo objektno paradigme koja obuhvata osnovne objektno koncepte, od kojih su neki [3-6]:

1. apstraktni tipovi podataka (eng. *abstract data types*): tip koji je definisao programer, za koji se mogu kreirati primeri (instance) i koji je predstavljen strukturom i ponašanjem;
2. enkapsulacija (eng. *encapsulation*): deo softvera ima jasno definisan interfejs i implementaciju; interfejs je svima dostupan, implementacija je nedostupna;
3. nasleđivanje (eng. *inheritance*): jedan tip može da nasledi drugi, sa značenjem da su njegove instance jedna vrsta instanci osnovnog tipa;
4. polimorfizam (eng. *polymorphism*): isti interfejs se koristi za različite tipove podataka.

Kada govorimo o OOP, neizostavni element je klasa (u C++ jeziku postoje dva takva elementa, klasa (eng. *class*) i struktura (eng. *struct*), oba su jako slična i u daljem tekstu ćemo se služiti samo klasom kao osnovnim elementom, ne ulazeći u dalje razmatranje o razlikama i sličnostima istih) [3-6].

Klasa je osnovna organizaciona jedinica programa u OOP jezicima. Klasa predstavlja strukturu u koju su grupisani podaci (promenljive) i funkcije [3-6].

Klasom se definiše novi korisnički tip za koji se mogu kreirati instance (primerci, promenljive). Klasa može da predstavlja realizaciju apstrakcije iz domena problema. Instance klase nazivaju se objekti (*eng. objects*). Svaki objekat ima sopstvene elemente koji su navedeni u deklaraciji klase. Ovi elementi klase nazivaju se članovi klase (*eng. class members*). Članovi klase dostupni spolja (iz druge klase ili funkcije) nazivaju se javnim članovima (*eng. public*). Sa druge strane, članovi koji su nedostupni korisnicima klase (ali ne i članovima klase) i nazivaju se privatnim (*eng. private*) [3-6].

Da bi objekat inicijalizovali u klasi se definiše posebna funkcija koja se implicitno (automatski) poziva kada se objekat kreira. Ova funkcija se naziva konstruktor (*eng. constructor*) i nosi isto ime kao i klasa. Moguće je definisati i funkciju koja se poziva uvek kada objekat prestaje da živi. Ova funkcija naziva se destruktor (*eng. destructor*) [3-6].

Nasleđivanje je osobina klase da nasleđuje članove (funkcije i promenljive) neke druge klase. Nasleđivanjem članovi zadržavaju dostupnost koja je postojala i u roditeljskoj klasi prema korisnicima klase.

Funkcija članica koja će u izvedenim klasama imati nove verzije deklarise se u osnovnoj klasi kao virtuelna funkcija (*eng. virtual function*). Izvedena klasa može da ima svoju definiciju virtuelne funkcije, ali i ne mora. Apstraktna klasa je klasa koja sadrži makar jednu apstraktnu, čistu virtuelnu funkciju (*eng. pure virtual function*), tj. virtuelnu funkciju čija je deklaracija izjednačena sa 0 (u C++). Ona klasa kod koje su sve funkcije apstraktne naziva se interfejsom. Naziva se tako jer je to klasa koja izlaže svoj interfejs a ne i implementaciju. Zbog toga ne mogu da se prave objekti od apstraktnih klasa već samo od klasa koje ih nasleđuju i implementiraju ponuđeni interfejs. U Javi na primer ne postoji apstraktna klasa već samo interfejs kao poseban tip klase [3-6].

Svojstvo da se odaziva prava verzija funkcije klase čiji su naslednici dali nove verzije naziva se polimorfizam (*polymorphism*). Ostvaruje se korišćenjem virtualnih funkcija koje se preklapaju (*eng. overriding*) [3-6].

Šablonske (*eng. template*) klase su klase koje koriste jedan ili više nespecifiranih tipova. Umesto da se pravi više klasa, pravi se jedna klasa koja onda može da se koristi sa više različitih tipova. Na ovaj način je, takođe, moguće ostvariti svojstvo polimorfizma [3-6].

## 2.7. Sličnosti i razlike C++ i GPSS jezika

Za razliku od GPSS jezika koji se koristi blokovima kako bi se formirao model, C++ je kao što je već rečeno jezik opšte namene i nema tačno definisani kriterijum po kome bi se model izradio. Šta više model je u ovom slučaju nešto daleko i apstraktno, pa se postavlja pitanje, kako formirati model.

Kako bi što lakše objasnili funkcionisanje C++ jezika vršićemo analogiju sa GPSS-om.

Svaki program u GPSS jeziku započinje i završava se ključnim rečima (*eng. keyword*) SIMULATE i END. Ceo programski kod, ceo model je smešten tačno između ove dve ključne reči. Sa druge strane C++ kod se može nalaziti bilo gde na ekranu, u smislu čak i



da postoji `SIMULATE` i `END`, deo modela bi mogao biti napisan pre i posle ovih ključnih reči. Međutim svaki programski kod mora imati ulaz (*eng. entry point*) i izlaz (*eng. exit point*). U C++ jeziku ulaz u program se ostvaruje korišćenjem funkcije **main**. Funkcija `main` je jedna jedina, i svaki C++ program je mora imati, ona se prva izvršava i upravlja programom. Vrednost koju vraća funkcija `main` jeste sam izlaz iz programa (izlaz ne mora biti jedan jedini, i ne mora biti ostvaren u funkciji `main`). Drugim rečima ono što bi u GPSS-u bilo `SIMULATE` i `END` to je u C++ sledeće [1,4-6]:

```
#include <iostream>
int main(){                                     // SIMULATE
    std::cout << "Pozdrav";
    return 0;                                   // END
}
```

Ovakav C++ kod je sasvim validan, i njegova svrha je da po pokretanju na ekran ispiše tekst *Pozdrav*. Ovim primerom je objašnjeno nekoliko glavnih razlika između C++ i GPSS-a a prva koju primećujemo je ta da C++ ne vodi računa o poziciji teksta na ekranu kako je gore objašnjeno, što nije slučaj sa GPSS gde se na tačno određenoj lokaciji piše određena komanda. Zatim, funkcija `main` se piše na sledeći način [4-6]:

```
tip_funkcije ime_funkcije ( parametri_funkcije ) {
telo_funkcije }
```

Pri tome vodi se računa da je vrednost koju funkcija vraća `return 0;` u ovom slučaju mora odgovarati tipu same funkcije, koji je u ovom slučaju `int` (celobrojni tip). `#include <iostream>` naredba dodaje direktivu `iostream` u kojoj se nalazi objekat `cout` i koji predstavlja konzolu (*eng. command prompt*). Primetimo takođe da se svaka naredba u C++ završava znakom `;`. Konačno ostaje da objasnimo naredbu `std::` koja u stvari predstavlja dve naredbe, `std` naredba predstavlja ime domena (*eng. namespace*) u kome se određeni objekat ili funkcija nalazi, `::` koji predstavlja operator dosega, kao što bi `+` predstavljao operator sabiranja. Napomenimo da je "Pozdrav" tekstualni tip podataka čije ime je `string` ili znakovni niz [4-6].

Na konkretnom primeru objektno orijentisano programiranje mozemo objasniti na primer korišćenjem klase tačka.

```
#include<iostream>
// Naredbom using ukljucujemo ceo prostor imena.
using namespace std;
class tacka{
    // Ključna rec public oznacava da je kod ispod javan i
    // direktno dostupan za pristup.
    public:
        // Atributi klase tacka x i y celobrojnog su tipa.
        int x,y;
};
int main(){
    // Kreiramo instancu (objekat) klase tacka i nazivamo je t.
    tacka t;
    // Dodeljujemo vrednost atributu x
    t.x = 10;
    // Dodeljujemo vrednost atributu y
    t.y = 20;
    // Ispisujemo vrednosti atributa x i y, objekta t, klase
    // tacka.
    cout << "t.x = " << t.x << "\nt.y = " << t.y;
```

```

        return 0;
    }

```

Da bi kod bio što kraći i razumljivi za sada su korišćeni samo podaci klase, a rečeno je da klase mogu imati u svom opisu i kod koji se odnosi na zajedničko ponašanje objekata koje kreiramo iz iste. Na primeru to možemo objasniti kroz dve veoma važne metode klase koje su podrazumevane i svaka klasa ih ima, kompajler automatski generiše ove dve metode i u slučaju da ih korisnik ignoriše. Reč je o konstruktoru klase i destrukturu klase. Funkcije koje se pozivaju prilikom kreiranja i uništavanja instance. C++ nam nudi mogućnost pisanja naših jedinstvenih metoda koje će u tom slučaju biti korišćene umesto podrazumevanih. Da ne bi bilo zabune, C++ i to pogotovo C++11 standard nude posebne ključne reči i pravila koja se odnose na podrazumevane metode klase (kao što su na primer, konstruktor, destruktork, operator kopije itd.) kad će u kojim slučajevima koja metoda biti kreirana od strane kompajlera[4-6].

```

#include<iostream>
// Naredbom using ukljucujemo ceo prostor imena.
using namespace std;
class tacka{
    // Kljucna rec public oznacava da je kod ispod javan i
    direktno dostupan za pristup.
public:
    // Konstruktor je javan, i vrsi inicijalizaciju podataka
    instance.
    tacka(int xx, int yy){
        x = xx;
        y = yy;
        cout << "Objekat kreiran!" << endl;
    }
    // Destruktor klase
    ~tacka(){ cout << "Objekat unisten!" << endl; }
    // Atributi klase tacka x i y celobrojnog su tipa.
    int x,y;
};

int main(){
    // Kreiramo instancu (objekat) klase tacka pozivajuci kreirani
    konstruktor i nazivamo je t.
    tacka t(10,20);
    // Ispisujemo vrednosti atributa x i y, objekta t, klase
    tacka.
    cout << "t.x = " << t.x << "\nt.y = " << t.y << endl;
    return 0;
}

```

Gore navedeni primer je potpuno isti kao i prethodni sa tom razlikom što sada inicijalizaciju instance klase tačka vršimo pri samom kreiranju objekta tako što vrednosti `t.x` i `t.y` inicijalizujemo prosleđivanjem parametara 10 i 20. Takođe dodate su poruke koje se ispisuju na ekran u trenutku kreiranja i uništenja objekta. Napomenimo da naredba `endl` predstavlja novi red i možemo reći da ima istu ulogu kao i `\n` (međutim i tu postoji razlika). Postavlja se pitanje šta se to ispisuje na konzoli po pokretanju programa, jasno je da se objekat kreira, ispisuju se vrednosti podataka, međutim ni u jednom trenutku nije napisana naredba uništenja objekta u samom kodu. Da li je ovde reč o takozvanom problemu curenja memorije (*eng. memory leak*) koji je spomenut prilikom objašnjavanja nivoa programskih jezika. Da bi objasnili koncept memorije koju program koristi objasnićemo šta su to **stek** (*eng. stack*) i **hip** (*eng. heap*). Tokom kreiranja aplikacije, zauzima se određeni deo memorije, jedan deo odlazi na sam kod koji se izvršava, jedan

deo odlazi na globalne i statičke promenjive i jedan deo odlazi na stek. Stek je onaj deo memorije koje funkcije koriste za svoje izvršavanje, tu spadaju sve promenjive koje funkcija koristi (na primer lokalna celobrojna promenjiva koja služi kao brojač u petlji). U ovom slučaju objekat klase tačka biće kreiran unutar funkcije main, i kao takav biće kreiran na steku. Sva memorija alocirana tj. zauzeta na steku po izvršenju date funkcije biva oslobođena tj. vraćena operativnom sistemu. Drugim rečima po završetku funkcije main pošto je instanca tačke kreirana na steku poziva se destruktorklase kako bi se izvršilo uništenje instance i oslobođenje memorije. U slučaju da je konstruktor u ovom slučaju definisan kao privatni, on ne bi bio dostupan za pozivanje i kompajler bi javio grešku. Ovo ujedno i daje odgovor na pitanje da li se javlja curenje memorije, odgovor je ne. Stvari se komplikuju ako u razmatranje uključimo sada i dinamičku memoriju, tj. hip. Ponekad su klase veoma velike, pa samo kreiranje objekata zahteva veliki memoriski prostor, koji nije predviđen stekom, ili je potrebno sačuvati podatke o nekom objektu ili nekoj promenljivoj i po završetku neke funkcije, u tom slučaju se koristimo dinamičkom memorijom. C++ nam u tu svrhu uvodi pokazivače (*eng. pointers*) i njihova primena je gotovo ključna u skoro svim aplikacijama kreirane C++ jezikom. Pokazivači su ogromna prednost C++ jezika i ceo naš model se temelji na njihovoj primeni. Na sledećem primeru je pokazana primena pokazivača [4-6].

```
int main(){
// Kreiramo instancu (objekat) klase tacka i nazivamo je t.
    tacka* t = new tacka(10,20);
// Ispisujemo vrednosti atributa x i y, objekta t, klase
tacka.
    cout << "t.x = " << t->x << "\nt.y = " << t->y << endl;
    return 0;
}
```

U ovom slučaju je nebitno da li je destruktorklase javan ili privatn, jer isti neće biti pozvan od strane samog programa po završetku izvršenja funkcije main i u ovom slučaju će se javiti problem curenja memorije. Napomenimo, da ne bude zabune, da stek ne funkcioniše tačno po principu jedan stek jedna funkcija, kako je gore objašnjeno radi lakšeg razumevanja, već je ceo pristup malo drugačiji i ne bi se toliko upuštali u objašnjavanje kako funkcioniše raspodela memorije već ćemo samo reći da je neki određen prostor (*eng. scope*) zadužen za određeni deo memorije [4-6].

```
int main(){
// Kreiranje objekta t
    tacka t(10,20);
{
// Kreiranje objekta t2
    tacka t2(5,-1);
// Unistenje objekta t2
}

    cout << "t.x = " << t.x << "\nt.y = " << t.y << endl;
    cout << "t2.x = " << t2.x << "\nt2.y = " << t2.y << endl;
// Greska, t2 ne postoji u ovom delu programa
    return 0; // Unistenje objekta t
}
```

### 3. ANIMACIJA SIMULACIONOG MODELA

Animacija je dinamički, grafički prikaz simulacionog modela. Služi da se uklone greške koje se mogu javiti prilikom razvoja modela i verifikuje ispravno ponašanje simulacionog modela. Takođe, animacijom se drugima prikazuju naše ideje i pretpostavke realizovane modelom [7].

Danas većina simulacionih jezika i alata omogućava korisnicima da kroz animaciju grafički reprezentuju svoj simulacioni model. Pojedini jezici i alati omogućavaju samo dvodimenzionalnu, dok drugi obezbeđuju i trodimenzionalnu animaciju. Neki od njih su: GPSS/H i SLX sa ProofAnimation-om, MATLAB/Simulink SimEvents, Arena, AnyLogic, WITNES itd [7].

U radu će biti korišćena biblioteka za animaciju [7], korišćenjem sledećih internet tehnologija HTML5, canvas (*eng. canvas*, platno) i Javaskript (*eng. JavaScript*). Ovaj način animacije je izabran zato što su ove tehnologije svima dostupne (dovoljno je instalirati neki savremeni internet pretraživač), nije potrebna kompilacija animacionog programa i ne zavise od operativnog sistema (zato što je animacioni program napisan u vidu jedne ili više datoteka koje se izvršavaju u pretraživaču) i ukoliko se simulacija izvršava na strani servera omogućava prikaz animacije na klijentskoj strani [7].

HTML5 je nova verzija jezika za označavanje HTML koja se koristi za semantičko opisivanje hipertekstulanih dokumenata na internetu. U odnosu na HTML 4.01, HTML5-om su uvedeni novi elementi jezika za označavanje, dok su pojedini elementi izbačeni. Jedan od novo uvedenih elemenata je canvas element. Canvas element se koristi za crtanje u dve dimezije korišćenjem programskog interfejsa u Javaskript jeziku [7].

Javaskript je dinamički, skript programski jezik. Uglavnom se koristi na strani klijenta, u internet pretraživaču, za promenu sadržaja i stila dokumenta, interakciju sa korisnikom i asinhronu komunikaciju sa serverom. Pored toga, Javaskript se koristi i za programiranje na strani servera i za razvoj mobilnih i desktop aplikacija. Ovaj programski jezik je veliki deo svoje sintakse nasledio iz Java, ali bez obzira na to ova dva jezika nisu u vezi i imaju različitu semantiku. Ovaj jezik podržava više programskih paradigmi i to: objektno-orijentisano, proceduralno i funkcionalno programiranje. Objektno-orijentisano programiranje u Javaskript jeziku se ostvaruje korišćenjem prototipova. Prototip je postojeći objekat koji se koristi za stvaranje novog objekta koji će naslediti svu funkcionalnost prototipa i eventualno dodati novu funkcionalnost [7].

Pored korišćenja canvas elementa, animaciju u HTML5 je moguće ostvariti korišćenjem kaskadnih stilova ili SVG (*eng. Scalable Vector Graphics*) vektorskih slika sa animacijom. Takođe je animaciju moguće sprovesti promenom (korišćenjem Javaskripta) DOM (*eng. Document Object Model*) modela, kao i promenom vrednosti svojstava kaskadnih stilova ili atributa SVG elemenata. Svi navedeni načini animacije imaju iste mogućnosti, međutim ono što izdvaja canvas u odnosu na kaskadne stilove i SVG je da većina savremenih pretraživača koristi hardver, kada je to moguće, za iscrtavanje na canvas elementu te je zbog toga animacija u pogledu performansi bolja u odnosu na druga dva navedena načina. Upravo to je razlog zbog čega se danas veliki broj internet video igara razvija korišćenjem HTML5 canvasa [7].

### 3.1. Javaskript

HTML5 oznaka (tag) koja je pridružena canvas elementu glasi `<canvas>`. Koristi se na sledeći način u HTML dokumentu [7].

```
<canvas id='canvas'>Canvas not supported</canvas>
```

Programski, canvas elementu se pristupa preko id atributa elementa upotrebom funkcije objekta dokumenta `getElementById()`. To se radi na sledeći način [7].

```
var canvas = document.getElementById('canvas')
```

Canvas element nema veliki programski interfejs. On se sastoji od dva atributa i tri funkcije. Dva atributa su `weight` i `height` koji vraćaju i postavljaju širinu i visinu canvas elementa. Od tri funkcije navešćemo samo jednu `getContext()`. Ova funkcija vraća kontekst u kome se vrši iscrtavanje u dve dimenzije. Kontekst dobijamo korišćenjem sledećeg segmenta koda [7].

```
var context = canvas.getContext('2d')
```

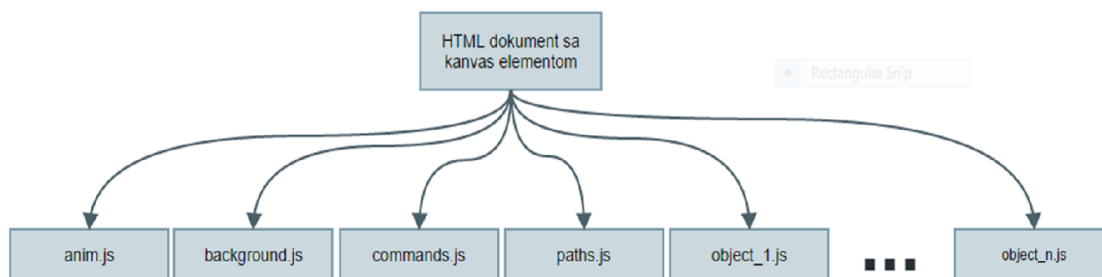
pri čemu vrednost '2d' koja se prosleđuje ka argument funkcije znači da se radi od dvodimenzionalnom kontekstu [7].

Za razliku od kanvasa, kontekst objekat ima puno atributa i funkcija različite namene [7].

Za iscrtavanje animacionog okvira koristi se funkcija `window.requestAnimationFrame()`. Ova funkcija kaže pretraživaču da želimo da izvršimo animaciju i zahteva od pretraživača da pozove zadatu funkciju u cilju ažuriranja animacije pre narednog iscrtavanja. Kao argumet ova metoda prima funkciju koja će biti pozvana pre narednog iscrtavanja. Funkcija vraća celobrojnu vrednost koja se može proslediti funkciji `window.cancelAnimationFrame()` u cilju zaustavljanja animacije. `window` je objekat koji predstavlja otvoreni prozor u pretraživaču [7].

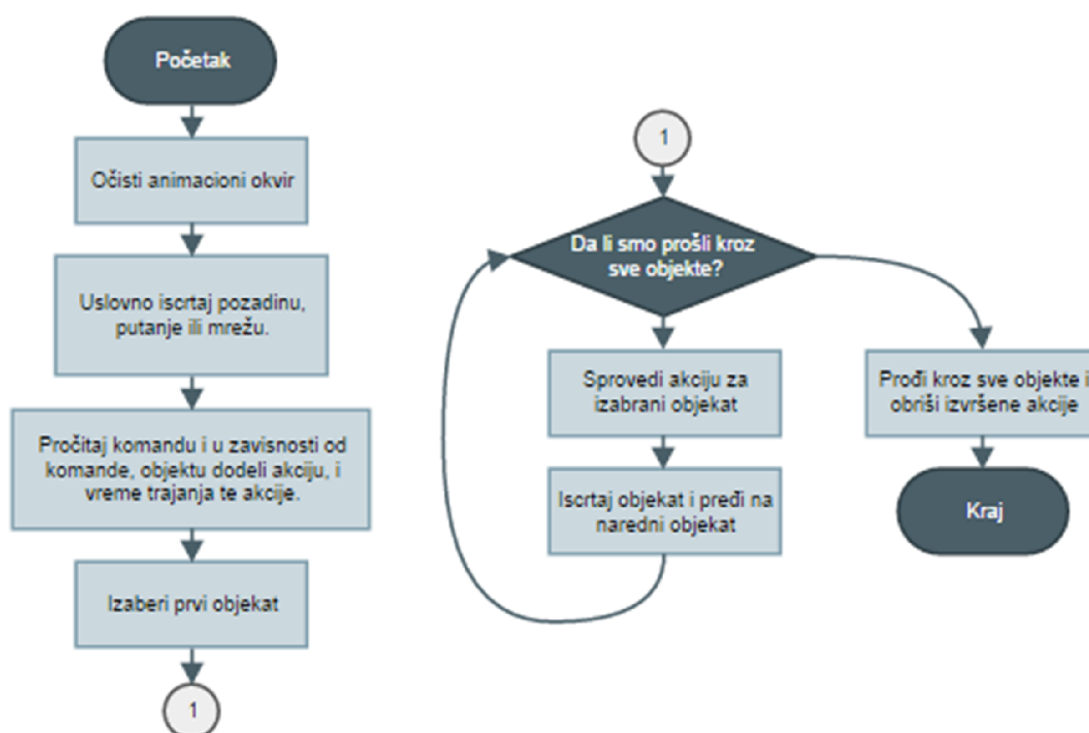
### 3.2. Javaskript biblioteka za animaciju

Ideja koja je vodila autore prilikom razvoja ove biblioteke je bila da njom omoguće animaciju u bilo kom simulacionom jeziku ili programskoj biblioteci koja poseduje mogućnost ispisa u tekstualnu datoteku. Ovaj princip, ispisa komandi u tekstualnu datoteku koji se kasnije koriste za animaciju, potiče iz animacionog programa `ProofAnimation`. Takođe, cilj je bio da se odvoji iscrtavanje pozadine, objekata koji se kreću i animacionih putanja. Zbog toga struktura animacionog programa izgleda kao na slici 2.



Slika 2. Struktura animacionog programa

U `anim.js` datoteci najalze se dve klase: `Interval` i `Animation`. `Interval` klasa služi da periodično (svakih 500 ms) prikaže vreme simulacije i prosečni broj frejmova u sekundi, dok klasa `Animacija` čita prosleđene komande i u zavisnosti od komande dodeljuje objektima akcije koje ažuriraju objekat za iscrtavanje u animacionom okviru. Na slici 3 je prikazan dijagram toka izvršavanja jednog animacionog okvira [7].



Slika 3. Dijagram toka izvršavanja jednog animacionog okvira

U datoteci `paths.js` se nalazi Javaskript objekat `paths` u koji se upisuju putanje po kojima animacioni objekti mogu da se kreću. Putanje se sastoje iz segmenata koji mogu da budu linearni ili kružni. Linearni segmenti su zadati početnom i krajnjom tačkom, dok su kružni segmenti zadati pozicijom centra, radijusom, početnim i krajnjim uglom kružnog segmenta. Takođe, kod kružnog segmenta se zadaje da li je u smeru kazaljke na satu ili u suprotnom smeru. Vreme boravka na putanji se zadaje komandom objektu `duration` ili `speed`. Na ovaj način objekat zna koliko vremena mu treba da pređe celu putanju ili kolikom brzinom se kreće na putanji. Objekat se kreće od početka do kraja putanje. Pored ovih putanja postoje i kružne putanje, kao i putanje sa redovima čekanja. Kod kružnih putanja objekat kada dođe do kraja putanje, ponovo započinje kretanje od

početka, i to radi tako sve dok ga ne uništimo ili prebacimo na neku drugu putanju. Kod putanja sa redovima čekanja kada objekti stižu na kraj putanje oni staju u red čekanja jedan iza drugog. Uklanjanje objekta iz putanje dovodi do pomeranja objekata iza za jedno mesto napred. Animacioni objekat smeštamo na putanju korišćenjem komande `place_on` [7].

Komade animacione biblioteke se zadaju u `commands` objektu Javaskript datoteke `commands.js`. Ova datoteka se direktno generiše iz simulacionog programa ili biblioteke [7].

Pored zadavanja komandi u `commands.js` je moguće promeniti i dimenzije canvas elementa [7].

U `background.js` se nalazi funkcija `draw_background()` čijim pozivom se iscrtava statična pozadina animacionog okvira [7].

U Javaskript datotekama objekata se nalaze klase koje se koriste za instanciranje animacionih objekata. Svaka klasa mora da ima određena svojstva i metode. Svojstva koje klasa mora da ima su: `x`, `y`, `width`, `height`, `angle`, `actions`, `fillStyle`, `strokeStyle`, `lineWidth`, `speed`, `traveltime`, `directional` i `message`. Pored toga klase objekata moraju da imaju jednu metodu i to je metoda `draw()` [7].

### 3.3. Datoteka anim.h

`anim.h` datoteka sadrži makroe tako napisane da ih možemo koristiti u simulaciji gotovo kao funkcije. Korišćenje makroa objasnićemo kroz nekoliko primera.

```
char msg[10];
ANIM_TIME(VREME);
sprintf_s(msg, "%d", mem.kliz[4]);
ANIM_MESSAGE("Por4", msg);
```

Ovako napisane ove četiri naredbe vrše ažuriranje poruke koja na ekranu ispisuje broj paketa koji je skrenuo na kliznicu 4.

Makro naredbe u suštini predstavljaju naziv za neki određeni kod. Kada god u programskom kodu koristimo naziv, on će prilikom kompajliranja biti zamenjen određenim kodom, čije ime nosi. Makro naredbe se mogu ponašati i kao objekti i kao funkcije i pored neke osnovne primene radi povećanja čitljivosti programskog koda, i smanjenog obima pisanja, makro naredbe na primer, mogu služiti da prikrijemo ključne reči. Ako bi zamenili ove četiri linije koda sa originalnim kodom dobili bi nešto ovakvo.

```
char msg[10];
fanim_out << "{type:'time',value:{time:" <<
(simu.vrati_vreme_simulacije()) << "}}," << endl;
sprintf_s(msg, "%d", mem.kliz[4]);
fanim_out << "{type:'message',value:{id:" << "Por4" <<
"',message:" << msg << "'}}," << endl
```

`fanim_out` jeste objekat klase `fstream`, koji je inicijalizovan makro naredbom `ANIM_INIT` a u pozadini se krije sledeća linija koda.

```
fstream fanim_out;
```

Sličan ovom kodu jeste kod koji uklanja pakete sa pozadine.

```
char obj[100];
ANIM_TIME(VREME);
sprintf_s(obj, "%d", e->vrati_id());
ANIM_DESTROY(obj);
```

Jedina razlika je u tome što ANIM\_DESTROY(obj) u pozadini izgleda ovako.

```
fanim_out << "{type:'destroy',value:{id:'" << obj << "'}}," << endl;
```

Sve operacije datoteke anim.h su date sledećom tabelom.

*Tabela 1. Operacije datoteke anim.h*

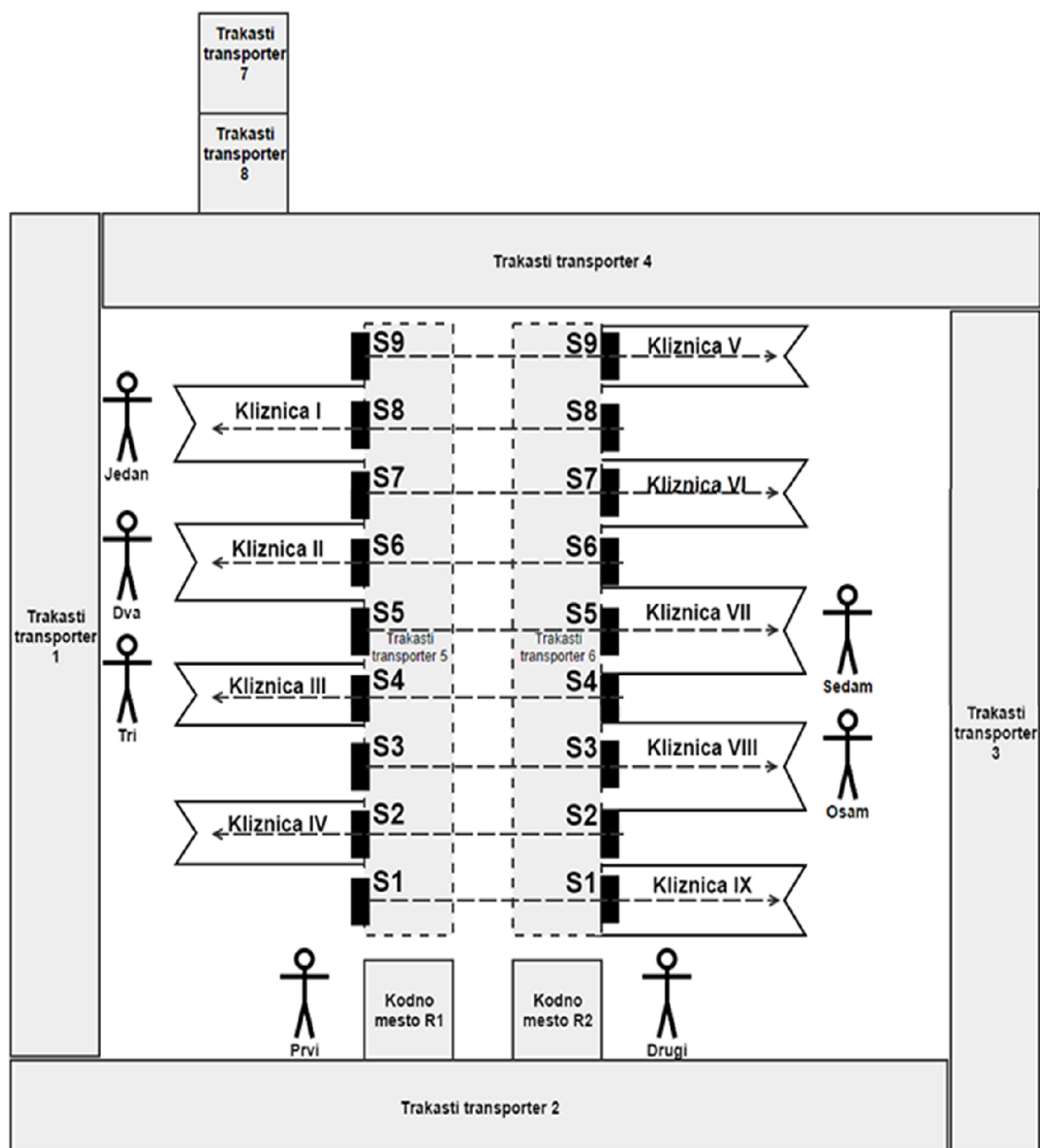
// Osnovne operacije	
ANIM_INIT	Inicijalizacija animacije
ANIM_START(dir,w,h)	Startovanje animacije
ANIM_END	Zaustavljanje animacije
// Vreme animacije	
ANIM_TIME(t)	Vreme animacije
// Stvaranje i uništavanje objekata	
ANIM_CREATE(class,id)	Stvaranje objekta
ANIM_DESTROY(id)	Uništavanje objekta
// Animiranje objekta	
ANIM_PLACE_AT(id,x,y)	Postavljanje objekta na koordinate x i y
ANIM_PLACE_ON(id,path)	Postavljanje objekta na putanju
ANIM_MOVE(id,x,y)	Pomeranje objekta
ANIM_ROTATE(id,angle)	Rotiranje objekta
// Svojstva objekta	
ANIM_FILL_STYLE(id,style)	Boji unutrašnjost objekta
ANIM_STROKE_STYLE(id,style)	Boji okvir objekta
ANIM_LINE_WIDTH(id,lw)	Menja se širina putanje
ANIM_SPEED(id,speed)	Postavlja brzinu
ANIM_DURATION(id,dur)	Postavlja trajanje
ANIM_MESSAGE(id,text)	Postavlja tekst
ANIM_DIRECTIONAL(id,dir)	Postavlja orijentaciju objekta na putnaji



## **4. SISTEM ZA POLUAUTOMATSKO RAZVRSTAVANJE PAKETA U POŠTI 11200 BEOGRAD**

Paketi u centar pristižu druskim transportnim sredstvima. Sam sistem sastoji se od 8 trakastih transportera, dva radna mesta za usmeravanje paketa, koji, su opremljeni terminalom sa ekranom na dodir (*eng. touch screen*), 9 kliznica, 18 skretnica (po 9 na svakom sortirnom transporteru) prikazanom na slici 4 [2].

Proces se odvija na sledeći način, paketi koji su pristigli druskim transportnim sredstvima istovaraju se direktno na transporter 7. Prilikom istovara paketa iz transportnog sredstva, paketi se broje i upoređuju sa spiskom razmene, to je tzv. sravnjavanje, isti postupak se radi i prilikom otpreme. Ovaj trakasti horizontalni transporter 7 zajedno sa kosim trakastim transporterom 8, uvodi pakete u kružni tok za razvrstavanje paketa. Transporter 8 penje pakete na visinu na kojoj se nalazi kružni sistem trakastih transportera 1,2,3 i 4, kao i sortirni transporteri 5 i 6. Paketi ulaze u sistem na transporter 4, zatim transporterima 1 i 2, paketi dolaze do kodnog mesta R1, gde postavljaju zahtev za opsluživanjem. Radnik na kodnom mestu R1 ako je slobodan, uzima paket i opslužuje ga, u suprotnom paket postavlja zahtev za opsluživanjem na kodnom mestu R2. Ako je radnik i na mestu R2 zauzet, paket ne biva opslužen u prvom pokušaju i nastavlja da se kreće transporterom 2, zatim transporterima 3,4,1 i ponovo postavlja zahtev za opsluživanje na kodnom mestu R1, i tako sve dok se ne opsluži. Paket koji se prihvati na opsluživanje razvrstava se na taj način što radnik pročita odredište paketa i usmeri ga pomoću terminala, transporterom 5 (ili 6), na odgovarajuću kliznicu. Paket se na kliznicu usmerava određenom skretnicom, koja se aktivira kada radnik pritisne na ekranu polje, koje označava odredište. Skretnice su označene sa S1,S2,...,S9. Svaki sortirni transporter 5 i 6, ima svojih 9 skretnica. Paketi se izbacuju u 9 kliznih kanala, označenih sa I,II,...IX. Sa bilo kog sortiranog transportera može izaći na svaku od kliznica. Kako je prikazano na slici paketi se na kliznicu I izbacuju skretnicama S8 transportera 5 i 6. Na kliznicu II pakete skreću skretnice S6 transportera 5 i 6. Na kliznicu III pakete skreću skretnice S4, a na kliznicu IV skretnice S2 transportera 5 i 6. Sa desne strane sortirnih transportera nalaze se kliznice, V, VI, VII, VIII i IX. Na kliznicu V paket izlazi posredstvom skretnica S9, na kliznicu VI posredstvom skretnica S7, na kliznicu VII pomoću skretnica S5, na kliznicu VIII pomoću skretnica S3 i konačno, na kliznicu IX pakete skreću skretnice S1 oba sortirna transportera [2].



Slika 4. Principialna šema za razvrstavanje paketa

- Kliznica I – Šabac, Valjevo, Požarevac
- Kliznica II – Sremska Mitrovica, Pančevo
- Kliznica III – Mladenovac, Smederevska Palanka, Velika Plana
- Kliznica IV – Beograd
- Kliznica V – Putujuća pošta 11608 (Beograd-Bar)
- Kliznica VI – Putujuća pošta 11601 (Beograd-Niš)
- Kliznica VII – Kraljevo, Kragujevac, Užice
- Kliznica VIII – Novi Sad, Sombor, Subotica
- Kliznica IX – Poštanske jedinica sa područja Republike Srpske.

Treba napomenuti da se kliznica IX sada ne koristi, jer se paketi i ostale pošiljke za Republiku Srpsku usmeravaju i otpremaju kao pošiljke za inostranstvo [2].

Paketi koji su sortirani po kliznicama nastavljaju da se prerađuju ručno tako da se sačine zaključci za gore navedene poštanske jedinice. Sva odredišta koja nisu gore navedena, a sigurno je da se pojavljuju paketi sa drugim odredištima osim navedenih, usmeravaju se preko dgovarajućih pravaca kako je to određeno pravilnicima. Drugim rečima pored kliznica I, II, III, VII i VIII stoje radnici (jedan, dva, tri, sedam i osam) koji vrše manuelno razvrstavanje paketa [2].

#### **4.1. Sredstva mehanizacije sistema za razvrstavanje paketa**

Već je rečeno da se sistem sastoji od 8 trakastih transportera, 18 skretnica i 9 kliznica. Pored ovih, koriste se jos i prikolice i karete [2].

Trakasti transporteri 1 i 3:

1. Dužina transportera  $L = 26m$
2. Širina transportera  $B = 0.88m$
3. Širina trake  $b = 0.8m$
4. Brzina transportovanja  $v = 30m/min$
5. Snaga pogona  $N = 0.75kW$

Trakasti transporteri 2 i 4:

1. Dužina transportera  $L = 16.4m$
2. Širina transportera  $B = 0.88m$
3. Širina trake  $b = 0.8m$
4. Brzina transportovanja  $v = 30m/min$
5. Snaga pogona  $N = 0.75kW$

Trakasti transporteri 5 i 6:

1. Dužina transportera  $L = 22.5m$
2. Širina transportera  $B = 0.58m$
3. Širina trake  $b = 0.5m$
4. Brzina transportovanja  $v = 60m/min$
5. Snaga pogona  $N = 2.2kW$

Trakasti transporter 7:

1. Dužina transportera  $L = 20m$
2. Širina transportera  $B = 0.88m$
3. Širina trake  $b = 0.8m$
4. Brzina transportovanja  $v = 25m/min$
5. Snaga pogona  $N = 0.75kW$

Trakasti transporter 8:

1. Dužina transportera  $L = 12.4m$
2. Širina transportera  $B = 0.88m$
3. Širina trake  $b = 0.8m$
4. Brzina transportovanja  $v = 30m/min$

#### 5. Snaga pogona $N = 1.1kW$

Poštanska prikolica:

Prikolica ima 4 točka i priključnu rudu za karetu. Dimenzije tovarnog prostora su: dužina  $2m$ , širina  $1m$  i dubina  $0.5m$ . Visina dna tovarnog prostora od tla je  $0.55m$ . Paketi se slažu do visine  $1m$  iznad gornje ivice prikolice [2].

Elektrotraktor – Kareta:

1. Nosivost tereta mase  $10000 KN$
2. Brzina vožnje  $6-11 km/h$
3. Autonomija (akcioni radius)  $40 km$
4. Vlastita masa max  $7800 KN$
5. Savlađivanje uspona  $8\%$
6. Snaga elektro-motora  $2 KW$

### 4.2. Terminali za usmeravanje paketa

Za poluautomatsko usmeravanje paketa na izlazne kliznice postoje dva radna mesta. Ova mesta su opremljena terminalima za kodiranje paketa. Zapravo terminal ima ekran na dodir, tako da se komanda za usmeravanje paketa na odgovarajuću kliznicu zadaje dodirom polja na ekranu u kome je upisan broj i naziv pošte. Ovi terminali omogućavaju precizno kodiranje većeg broja pravaca, upravljanje transportnim trakama i daju osnovne informacije o radu sistema. Pogrešno usmeravanje se može poništiti pritiskom na drugo polje ili na polje *Clear*. Ako se aktivira polje *Reset* svi paketi zatečeni na sortirnom transporteru nastaviće sa kretanjem do izlaza na transporter 4, a potom će ponovo doći na kodiranje. Na ekranu se nalaze polja za upravljanje transporterima 1,2,...,8 [2].

### 4.3. Simulacija sistema za razvrstavanje poštanskih paketa

Sistem za preradu poštanskih paketa je složen sistem i predstavlja jedan deo većeg sistema – sistema poštanskog saobraćaja. Tehnološki proces prerade paketa odvija se pretežno na transportnim uređajima. Da bi se analizirao sistem potrebno je najpre [2]:

1. Definirati granice sistema i veze sistema za preradu paketa sa njegovom okolinom.
2. Utvrditi osnovnu strukturu sistema za preradu paketa.
3. Izvršiti identifikaciju elemenata, radnih operacija i funkcija u procesu prerade paketa.
4. Izabrati način opisivanja elemenata, operacija i funkcija u sistemu na osnovu provedene identifikacije.

Sve ove vrednosti sistema prezentovane u ovom radu su vrednosti dobijene u proračunu uzetom iz diplomskog rada [2]. Mi ih kao takve usvajamo i koristimo u našoj simulaciji. Ideja je da nam se rezultati analize na kraju poklope kako bi bili sigurni da je model rešen u C++ jeziku potpuno zadovoljio sve uslove simulacije [2].

Napomenimo da je diplomskom radu [2] menjan simulacioni model i analiziran sistem u različitim situacijama funkcionisanja. U ovom radu akcenat nije na samoj analizi različitičutih situacija, već je akcenat stavljen na objektno orijentisani pristup rešenju problema. Model koji je realizovan u C++ jeziku odgovara slučaju kada:

1. Broj paketa iznosi **1000**
2. Broj kanala opsluživanja sa trenutnim odredištima iznosi **2**
3. Brzina trake u krugu  $v = 20 \text{ m/min}$
4. Vremenska jedinica  $v.j. = 0.1 \text{ s}$

#### 4.3.1. Ulazne veličine u sistemu

Svaki elemenat sistema opisuje određeni broj parametara. Na osnovu tih parametara određuju se vremena zadržavanja paketa na svakom od elemenata sistema. Vrednosti su sledeće [2]:

*Tabela 2. Vreme zadržavanja paketa na trakastim transporterima*

Transporter	8	7	6	5	4	3	2	1
Vreme	48s	24.8s	22.5s	22.5s	32.8s	52s	32.8s	52s

*Tabela 3. Vreme zadržavanja paketa na segmentima*

Transporter 7 i 8	Prelazak do R1	Sa R1 do R2	Sa R2 do ulazka 8 u 4
72.8s	68.1s	6s	95.5s

Na osnovu ovih veličina i toka kretanja paketa, možemo definisati veličine koje direktno ubacujemo u simulacioni model [2].

*Tabela 4. Vremena kretanja paketa na sortirnom tranporteru 5 od R1 do skretnica*

I	II	III	IV	V	VI	VII	VIII	IX
16.0s	12.0s	8.0s	4.0s	18.1s	14.1s	10.1s	6.1s	2.1s

*Tabela 5. Vremena kretanja paketa na sortirnom tranporteru 6 od R2 do skretnica*

I	II	III	IV	V	VI	VII	VIII	IX
17.1s	13.0s	8.9s	4.8s	19.3s	15.3s	11.2s	7.2s	3.1s

Vreme za koje paket skrene na klizni kanal iznosi [2]: 1.4s

Usvojeno je da paket sklizne niz klizni kanal za vreme u intervalu od 2 do 4 sekunde. Ova vrednost se uzima sa ravnomernom raspodelom verovatnoća [2].

#### 4.3.2. Operacije i funkcije u sistemu

U sistemu za prerađu paketa odvija se proces transporta, skladištenja i podele paketa. Pojedine faze procesa, ako su mehanizovane ili automatizovane, nazivaju se funkcije toka materijala, a ako se obavljaju ručno, nazivaju se operacije. U sastav modela sistema za prerađu paketa uključen je samo proces prerade paketa. Dakle procesi skladištenja i transporta paketa nisu predmet simulacije i modeliranja u ovom radu. Proces transporta je razmatran samo toliko da se u zavisnosti od reda prevoza uoče intervali kada se

pojavljaju pošiljke za prerađu. Ulazne veličine modela sistema za prerađu poštanskih paketa, koje se odnose na operacije i funkcije u sistemu su sledeće [2]:

Operacija stavljanja paketa na ulazni transporter broj 7 okarakterisana je slučajnom veličinom  $T_u$ . Ovo vreme, prema pretpostavci, ima pomećenu eksponencijalnu raspodelu [2].

Srednja vrednost  $T_u$  iznosi 3s [2].

Ova veličina uzima vrednosti sa verovatnoćom po eksponencijalnom zakonu i ne može biti manja od 2 sekunde [2].

Funkcija kodiranja paketa, na terminalima, opisuje se vremenom kodiranja  $T_k$ . Ovo vreme, po pretpostavci, je eksponencijalno raspoređeno i ima srednju vrednost [2].

$$T_k = 3s$$

Operacija manuelnog razvrstavanja paketa na kraju kliznica, opisuje se vremenom trajanja te operacije  $T_r$ . Po pretpostavci, ovo vreme je, takođe, eksponencijalno raspoređeno i ima srednju vrednost [2].

$$T_r = 6s$$

Verovatnoća izbora odgovarajuće kliznice data je u tabeli 6 [2].

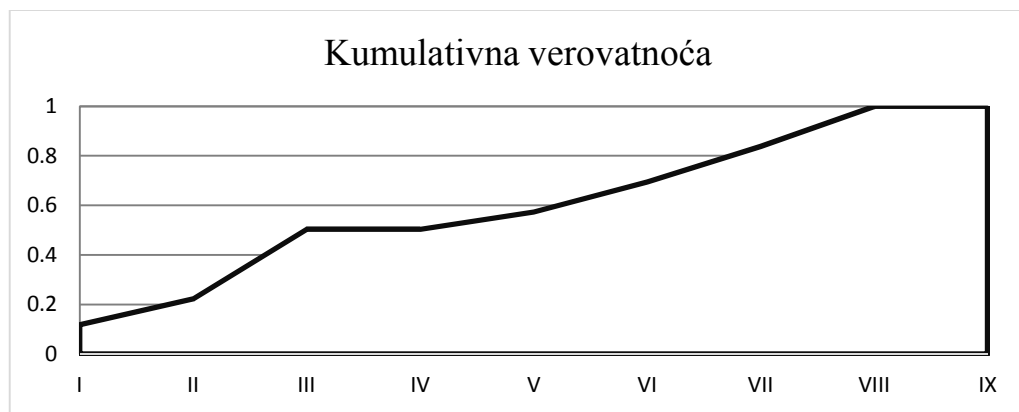
*Tabela 6. Verovatnoća izbora kliznice*

Kliznica (k)	I	II	III	IV	V	VI	VII	VIII	IX
Verovatnoća P(k)	0.119	0.104		0.281	0.070	0.121	0.144	0.161	
Kumulanta	0.119	0.223	0.504	0.504	0.574	0.695	0.839	1.000	1.000



*Slika 5. Verovatnoće izbora pojedinih kliznica*

Na slici 5 su grafički prikazane verovatnoće izbora pojedinih kliznica prikazane u tabeli 6 [2].



*Slika 6. Kumulativna verovatnoća izbora pojedinih kliznica*

Na slici 5 se vidi da se na kliznici IX se paketi ne pojavljuju, jer je saobraćaj sa Republikom Srpskom, u režimu inostranstva. Broj paketa za kliznicu III ulazi u broj paketa za područje Beograda [2].

## 5. REZULTATI SIMULACIJE

Rezultati koji su praćeni su sledeće veličine:

Statistike redova čekanja:

1. Broj entiteta koji je usao u red čekanja;
2. Preostali broj entiteta u redu čekanja;
3. Maksimalni broj entiteta u redu;
4. Srednje vreme čekanja u redu;
5. Srednji broj entiteta u redu;
6. Broj entiteta koji je prošao kroz red bez zadržavanja;
7. Procenat ulaza bez zadržavanja;
8. Srednje vreme čekanja u redu (isključujući entitete koji se nisu zadržali).

Statistike resursa:

1. Broj entiteta koji je ušao u resurs;
2. Preostali broj entiteta u resursu;
3. Maksimalni broj zauzetih kanala opsluge;
4. Srednje vreme opsluge;
5. Srednji broj zauzetih kanala;
6. Iskorišćenost.

Statistike prikupljene histogramima:

1. Broj ulaza u histogram;
2. Prosečna vrednost overflowa;
3. Frekvencija overflowa;
4. Verovatnoća overflowa;
5. Srednja vrednost histograma;
6. Standardna devijacija histograma;
7. Suma argumenata histograma.

Statistike kolica na kraju sortiranja:

1. Brojeve vrednosti svakog od pravaca na koji je paket upućen.

Rezultati simulacije su ispisani na standardnom izlazu aplikacije C++ jezika prikazanom na slici 7. Jasno je da je ceo ispis preveliki da se prikaze jednom slikom, pa se ceo ispis pored standardnog izlaza na konzoli (*eng. Command Prompt*) ispisuje u tekstualnu datoteku, i prikazana je na kraju ovog rada u prilogu.



```

C:\Users\IkerKristian\Documents\Visual Studio 2013\Projects\testC++\Debug\t...
16000      1 0.00657895      1.06219      0.0980907
17000      5 0.0328947      1.12858      0.202793
18000      6 0.0394737      1.19497      0.307496
19000      9 0.0592105      1.26136      0.412199

HISTOGRAM - TAB28
Broj ulaza u histogram: 146
Prosečna vrednost overflowa: 247902
Frekvencija overflowa: 5
Verovatnoca overflowa: 0.0342466
Srednja vrednost histograma: 15257.6
Standardna devijacija histograma: 8758.58
Suma argumenata histograma: 2.22761e+006
int      frek      vero      umno      dev
1000      3 0.0205479      0.065541      -1.62785
2000      7 0.0479452      0.131082      -1.51367
3000      5 0.0342466      0.196623      -1.3995
4000      7 0.0479452      0.262164      -1.28533
5000      5 0.0342466      0.327705      -1.17115
6000      6 0.0410959      0.393246      -1.05698
7000      0 0.0000000      0.458787      -0.942804
8000      4 0.0273973      0.524328      -0.82863
9000      4 0.0273973      0.589869      -0.714457
10000     4 0.0273973      0.65541      -0.600283
11000     4 0.0273973      0.720951      -0.486109
12000     3 0.0205479      0.786492      -0.371936
13000     5 0.0342466      0.852033      -0.257762
14000     6 0.0410959      0.917574      -0.143588
15000    12 0.0821918      0.983115      -0.0294143
16000     5 0.0342466      1.04866      0.0847594
17000     3 0.0205479      1.1142      0.198933
18000     5 0.0342466      1.17974      0.313107
19000     4 0.0273973      1.24528      0.427281

Sadržaj memorijskog objekata:
kliz[1]: 115
kliz[2]: 117
kliz[3]: 100
kliz[4]: 184
kliz[5]: 63
kliz[6]: 123
kliz[7]: 152
kliz[8]: 146
kliz[9]: 0 <- medjunarodni saobracaj ka RS
posl: 30784.5
Press any key to continue . . .

```

Slika 7. Prikaz rezultata simulacije realizovane u C++ jeziku

## 5.1. Generatori slučajnih brojeva i raspodele modela

Koristeći prednosti objektnog orijentisanog programiranja, napravljeni su objekti klase `raspodela`, takvi da se njihovo seme međusobno razlikuje, samim tim, vodimo računa o statističkoj nezavisnosti modela, dok sa druge strane seme postavljamo na vrednosti koje i GPPS/FON verzija koristi za svojih 8 generatora slučajnih brojeva.

```

raspodela rni(1), rn1(3), rn2(5), rn3(7), rn4(11), rn5(13),
rn6(17);

```

Ovako zadati generatori slučajnih brojeva potpuno odgovaraju generatorima koji su korišćeni u diplomskom radu [2].

Objekte (kao npr. `rni` ili `rn1`) klase `raspodela` koristimo na sledeći način. U slučaju kada želimo da kreiramo slučajan broj na uniformnom intervalu  $[1,10]$  i zatim tu vrednost sačuvamo promenljivom `p` sve što je potrebno jeste da pozovemo odgovarajuću metodu klase `raspodela`.

```
double p = rni.unif(1.00,10.00); // double je tip podataka
koji koristimo za brojeve sa decimalnom tackom.
```

Slicno možemo dobiti vrednosti za eksponencijalnu ili normalnu raspodelu.

```
double p1 = rn4.expo(m); // m je matematicko ocekivanje.
double p2 = rn2.norm(a,b); // a i b su matematicko ocekivanje
i disperzija
```

Jasno je da ovakvim pristupom možemo lako i brzo kreirati veliki broj različitih raspodela sa druge strane isto je u GPSS-u veliki problem. Da bi stvari učinili još zanimljivijim u C++ imamo potpunu slobodu nad našim programskim kodom i znajući tip parametara koje funkcija uzima i vraća (u našem slučaju `double`) možemo napraviti raspodele ovakvog tipa:

```
double p3 = rni.norm( rn1.unif(1.00,3.00) , rn4.expo(9.00) );
```

U ovom slucaju prvo ce objekat `rn1` pozvati svoju metodu i vrednost koja se vrati bice uzeta kao matematičko očekivanje `rni.norm(...)` raspodele, a zatim će se sračunati standardno odstupanje kao rezultat `rn4.expo(9.00)` metode. Konačno u promenljivu `p3` bice upisana vrednost raspodele `rni.norm(...)`.

Konkretna primer u modelu jeste vreme koje se paket zadrži u bloku ADVANCED FN\$KLI i možemo ga predstaviti kao:

```
double fn_kli(){
    return rn6.unif(20.000, 40.000);
}
```

## 5.2. Diskretna funkcija izbora kliznica

U radu diskretna i kontinualna funkcija je predstavljena strukturom `funkcija` i ove dve metode su date kao statičke (*eng. static*), u tom smislu da ove dve funkcije ne zavise od članova strukture, i da se ponašaju kao globalne funkcije vidljive iz celog programa. Međutim zadržana je definicija strukture, u smislu da nije apstraktna. Klase iz kojih se ne mogu napraviti objekti nazivamo apstraktnim klasama. Drugim rečima u modelu je potrebno napraviti objekat strukture `funkcija` a zatim koristiti isti prilikom pozivanja diskretne ili kontinualne funkcije. U slučaju da postojale još neke funkcije, mogle su sve biti objedinjene unutar iste strukture. Ovime sa ponajviše povećava čitljivost i razumljivost koda i to je jako bitna stvar u programiranju [4,5,6].

U konkretnom primeru to izgleda ovako:

```
funkcija func; // Objekat func strukture funkcija
double rezultat = func.diskretna(rn5(), X, Y, n);
```

U oba slučaja funkcija kao parametre uzima, vrednost na intervalu  $[0.00, 1.00]$  koju vraća funktor `rn5()`, niz (ili polje) realnih brojeva koji predstavljaju kumulativnu verovatnoću predstavljenu promenljivom `x`, niz realnih brojeva koji predstavljaju vrednosti koje funkcija može da vrati `y`, i promenljivu `n` koja predstavlja dužinu nizova `x` i `y`. Napomena, funktor je kraći naziv za objekat funkcije (*eng. function object*). To je osobina u programiranju koja dozvoljava objektu da bude korišćen kao obična funkcija. Definisan je unutar klase `raspodela` operatorom `()` [4,5,6].

Funkcija izbora kliznice izgleda ovako:

```
double X[7] = { 0.119, 0.223, 0.504, 0.574, 0.695, 0.839, 1.000 };
double Y[7] = { 1.000, 2.000, 4.000, 5.000, 6.000, 7.000, 8.000 };
int izaberi_kliznicu() {
    return int(func.diskretna(rn5(), X, Y, 7));
}
```

int(...) će pretvoriti realan broj u celobrojni i vratiće njegovu vrednost.

### 5.3. Klasa simulacija

Klasa simulacija predstavlja suštinu simulacionog modela i kao takva bice detaljno objasnjena. Programski kod je realizovan u sledećem obliku.

*Listing 1. Klasa simulacija C++*

```
// Klasa simulacija
class simulacija {
    // Brojac entiteta
    int eid;
    // Terminacioni brojac
    int tb;
    // Vreme simulacije
    double vreme_simulacije, apsolutno_vreme_simulacije;
    // Polje entiteta u LBD listi
    deque<entitet*> lista_nastupanja_entiteta;
    // Sortiranje LBD - Sortiranje izborom
    void sortiraj_listu_nastupanja() {
        stable_sort(lista_nastupanja_entiteta.begin(),
        lista_nastupanja_entiteta.end(), simulacija::uporedi_entitete);
    }
public:
    // Konstruktor
    simulacija() :eid(1), tb(0), vreme_simulacije(0.0),
    apsolutno_vreme_simulacije(0.0) {}
    // Destruktor
    ~simulacija() {
        // Uklanjanje presotale entitete iz LBD na kraju
        // simulacije
        ocisti();
    }
    // Pravljenje entiteta
    entitet* napravi_entitet() {
        // Pravimo novi entitet
        return new entitet(eid++);
    }
    // Unistavanje entiteta
    void unisti_entitet(entitet *e, int b) {
        // Unistavamo entitet
        delete e;
        // Umanjujemo terminacioni brojac
        tb -= b;
    }
    // Rasporedjivanje dogadjaja
    void rasporedi(entitet* e, int dog, double vreme) {
        // Postavljamo dogadjaj
        e->naredni_dogadjaj = dog;
        // Postavljamo vreme nastupanja dogadjaja
        e->vreme_nastupanja_dogadjaja = vreme;
        // Smetamo entitet u listu
        lista_nastupanja_entiteta.push_back(e);
        // Sortiramo LBD
        sortiraj_listu_nastupanja();
    }
    // Izvršavanje simulacije
    void izvrsi(int b) {
        entitet *tekuci;
        // Postavljamo terminacioni brojac
```

```

        tb = b;
        // Izvršavamo simulaciju. Simulacija se završava
        ukoliko
        // nema entiteta u LBD ili ukoliko je terminacioni
        brojac 0.
        do {
            // Vadimo prvi entitet iz liste. Taj entitet
            postaje tekuci entitet.
            tekuci = lista_nastupanja_entiteta.front();
            lista_nastupanja_entiteta.pop_front();
            // Faza 1: Azuriramo vreme simulacije
            vreme_simulacije = tekuci->
            vreme_nastupanja_dogadjaja;
            // Faza 2: Izvršavamo dogadjaj
            izvršavanje_dogadjaja(tekuci->naredni_dogadjaj,
            tekuci);
        } while (!lista_nastupanja_entiteta.empty() && tb>0);
    }
    // Uklanjammo entitete iz LBD
    void ocisti() {
        // Uvecavamo apsolutno vreme simulacije
        apsolutno_vreme_simulacije += vreme_simulacije;
        // Resetujemo vreme simulacije
        vreme_simulacije = 0.0;
        for (deque<entitet*>::iterator it =
        lista_nastupanja_entiteta.begin();
            it != lista_nastupanja_entiteta.end();
            it++) delete *it;
        // Cistimo listu
        lista_nastupanja_entiteta.clear();
    }
    // Izvršava dogadjaj
    void izvršavanje_dogadjaja(int dog, entitet* e);
    // Vraca vreme simulacije
    double vrati_vreme_simulacije() { return vreme_simulacije; }
    double vrati_apsolutno_vreme_simulacije() { return
    apsolutno_vreme_simulacije + vreme_simulacije; }
    // Komparacija vremena nastupanja entiteta
    static bool uporedi_entitete(entitet* ea, entitet* eb) {
        return ea->vreme_nastupanja_dogadjaja < eb-
        >vreme_nastupanja_dogadjaja;
    }
};

```

Privatni članovi klase su brojač entiteta, terminacioni brojač, vremena simulacije, polje entiteta u listi nastupanja događaja kao i jedna metoda sortiranja liste nastupanja događaja.

Lista javnih metoda klase je znatno veća, i neke od podrazumevanih su konstruktor, destruktor, metode koje vraćaju vrednosti privatnih parametara kao i metode za izvršavanje i sortiranje simulacije.

Lista nastupanja događaja je kreirana pomoću šablona `deque` (dek – dvostrano povezana lista).

```
deque<entitet*> lista_nastupanja_entiteta;
```

Dek je vrsta kontejnera, koja se nalazi u standardnoj biblioteci šablona (*eng. STL - Standard Template Library*). U naš programski kod je uključena naredbom `#include <deque>`. Kontejneri (*eng. containers*) su specijalni tip klasa u C++ jeziku, tako definisanih da je njihovo ponašanje potpuno isto bez obzira na tip podataka koji sadrže, ovim pristupom se jasno vidi osobina generičkog programiranja. U našem slučaju tip podataka koji dek sadrži jeste pokazivač na entitet (*entitet\**). U slučaju da je napisano `deque<int>` tip bi bio celobrojni, slično `deque<float>`, realni tip. Postoje više vrsta

i podela kontejnera, svaka sa svojim prednostima i manama. Na primer neki su znatno sporiji prilikom pretraživanja podataka (`list` u odnosu na `vector`), sa druge strane neki su sporiji prilikom unosa podataka u sredinu (`vektor` u odnosu na `list`) itd. Suština je da su kontejneri tako organizovani da bilo koji algoritam iz STL može biti izvršen nad istim. Sa jedne strane imamo kontejnere, sa druge algoritme. Ovakvom vrstom kodiranja broj mogućih operacija i mogućnosti raste brzo. Svaki kontejner se povezuje sa odgovarajućim algoritmom pomoću specijalnih objekata nazvanih iteratorima (eng. *iterator*). Uloga iteratora je jasno prikazana metodom `ocisti` koja uklanja preostale entitete iz modela [4,5,6].

```
void ocisti(){
    ...

    for (deque<entitet*>::iterator it =
        lista_nastupanja_entiteta.begin();
        it != lista_nastupanja_entiteta.end();
        it++) delete *it;

    ...
}
```

Kod možda izgleda pomalo zastrašujuće na prvi pogled, ali je u suštini vrlo jednostavan, i objasnićemo ga na sledećem primeru:

```
int pocetak_liste = 10;
int kraj_liste = 50;
for (int i = pocetak_liste ; i != kraj_liste; i++ ) cout << i
<< endl;
```

Ovako napisan kod radi sledeće, prvo se kreira pomoćna promenljiva `i` celobrojnog tipa i inicijalizira se vrednošću promenljive `pocetak_liste`, drugim rečima `i` sada ima vrednost 10, zatim se proverava vrednost uslova, da li je trenutna vrednost `i` jednaka vrednosti promenljive `kraj_liste`, ako nije izvršavaju se naredbe *for* petlje, u ovom slučaju je to samo ispis vrednosti `i` na ekran, zatim se `i` povećava za 1, i ponovo se vrši provera uslova, sve do onog trenutka dok se isti ne ispuni.

Analogno celobrojnom iteratoru `i`, prvo se kreira objekat iteratora `it` tipa koji je u ovom slučaju pokazivač na entitet. Zatim se vrši njegova inicijalizacija na vrednost koju vraća metoda `.begin()`, drugim rečima iterator se postavlja na prvu poziciju u deku. Ispituje se uslov da li ova vrednost odgovara vrednosti koju vraća metoda `.end()`, ako je uslov neispunjen izvršavaju se naredbe *for* petlje. U ovom slučaju briše se objekat, na koji iterator `it` pokazuje, iz memorije.

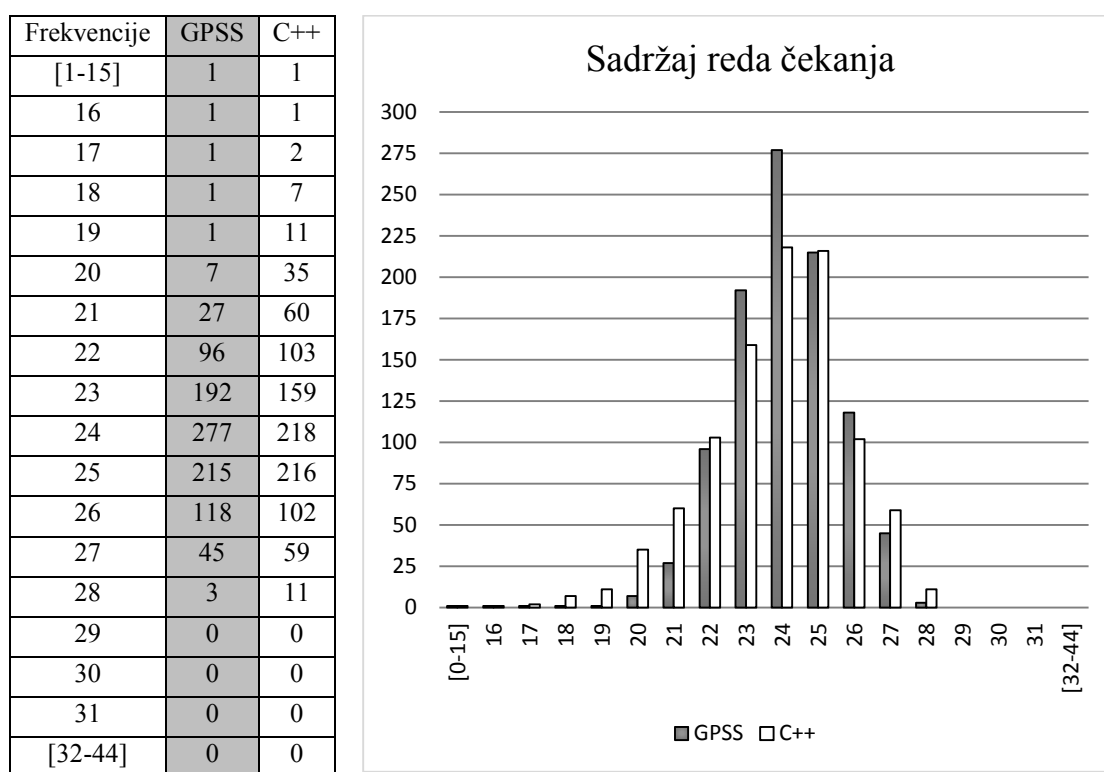
Metoda raspoređivanja događaja obavlja sledeće poslove. Entitetu koji se prosledi kao parametar metode, zadaje se vrsta događaja i vreme nastupanja tok događaja preko preostala dva parametra koji se prosleđuju metodi. Nakon toga entitet se dodaje u listu nastupanja događaja i vrši se sortiranje liste.

Metoda izvršavanja je nešto drugačija, parametar koje se prosleđuje je samo jedan i predstavlja vrednost terminacionog brojača, u našem slučaju je to 1000, jer je cilj analizirati model za prvih 1000 paketa. Svaki put kada se paket sortira na bilo kojoj kliznici, ova vrednost se umanjuje za 1. Stoga vrednost terminacionog brojača uzimamo kao dodatni uslov koji se ispituje u simulacionoj petlji. Kada kažemo simulaciona petlja

mislimo na *do while* petlju, koja se izvršava u samoj metodi sve do onog trenutka kada se lista nastupanja narednih događaja ne isprazni ili terminacioni brojač ne postane jednak nuli. Sama petlja je jako prosta, izvlači se entitet iz liste koji je prvi na redu, ažurira se vreme simulacije na vreme nastupanja događaja čija je vrednost zapisana u parametru izvučenog entiteta, zatim se izvršava sam događaj. Sama realizacija funkcije koja izvršava događaje jeste ta da se potpis funkcije nalazi u klasi simulacija, međutim telo se definiše u samom modelu, koji se nalazi u *main.cpp* datoteci. U našem simulacionom modelu ova provera izvršena je korišćenjem *switch* naredbe, međutim ista je mogla biti realizovana i na primer velikim brojem *if the else* naredbi.

## 5.4. Poređenje rezultata dobijenih realizacijama u GPSS i C++

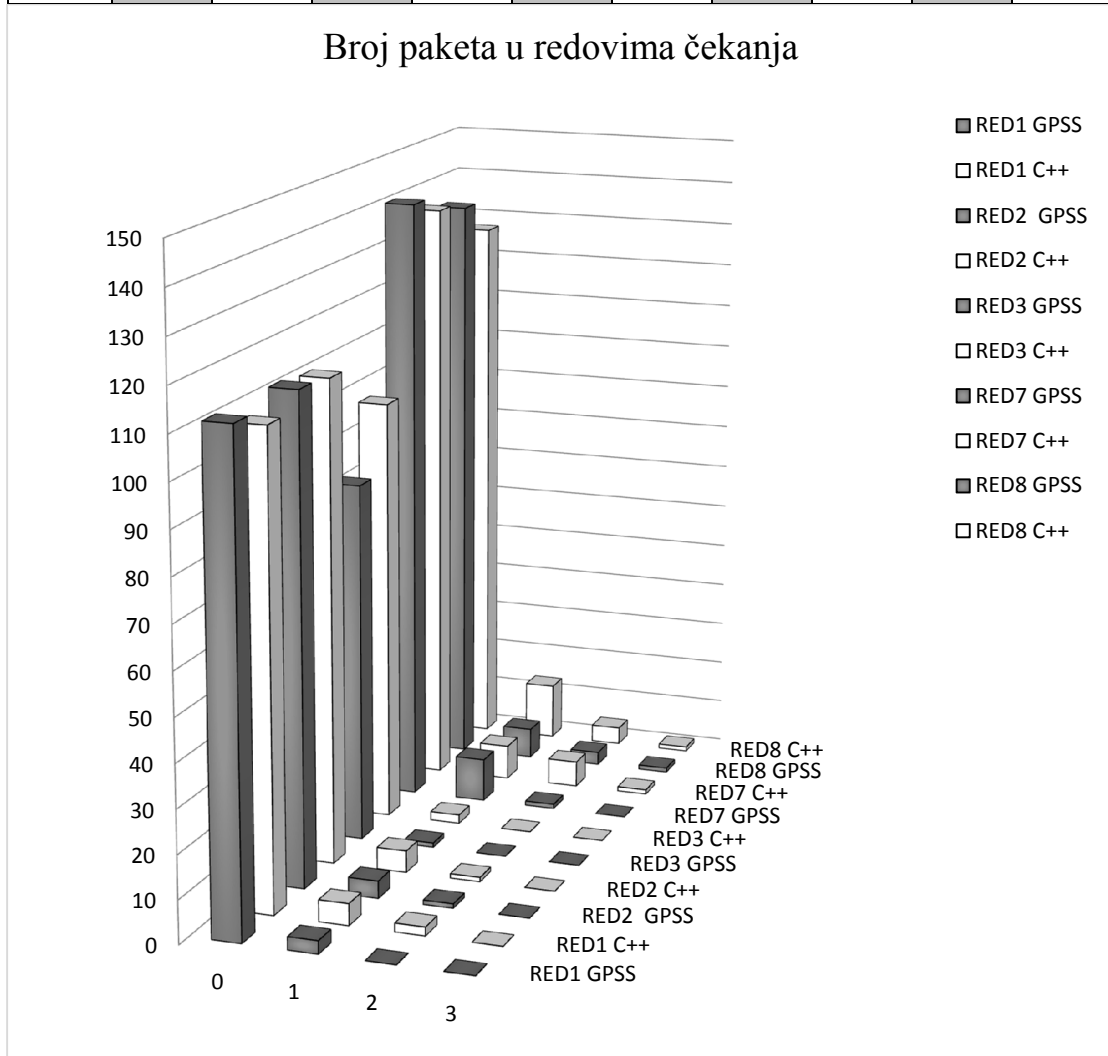
Na slici 8 je prikazan histograma broja paketa u reda čekanja RED, ova merenja su izvršena prilikom prelaska paketa preko trakastih transportera 8 i 7. Broj svih paketa (suma svih frekvencija) jednaka je broju 1000, potvrđujući model koji je simulirao rad sistema za prvih 1000 paketa.



Slika 8. Histogram broja paketa u redu čekanja red

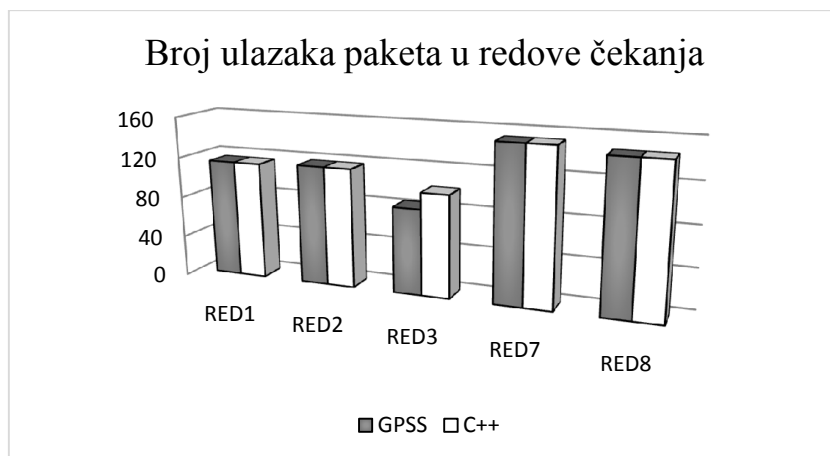
U sledećoj tabeli (slika 9) su date vrednosti redova čekanja, redom RED1, RED2, RED3, RED7 i RED8. Ove vrednosti označavaju broj paketa u redovima čekanja na manuelnom sortiranju, pri tome se zauzimaju radnici JEDAN (kliznica 1), DVA (kliznica 2), TRI (kliznica 3), SEDAM (kliznica 7), OSAM (kliznica 8) respektivno. Na slici 9 je dat grafički prikaz ovih vrednosti. Jasno se vidi da su redovi čekanja veći deo vremena prazni.

Broj Paketa	RED1 GPSS	RED1 C++	RED2 GPSS	RED2 C++	RED3 GPSS	RED3 C++	RED7 GPSS	RED7 C++	RED8 GPSS	RED8 C++
0	112	108	112	111	83	98	141	137	135	127
1	3	5	4	5	1	2	10	8	7	13
2	0	2	1	1	0	0	1	6	3	4
3	0	0	0	0	0	0	0	1	1	1



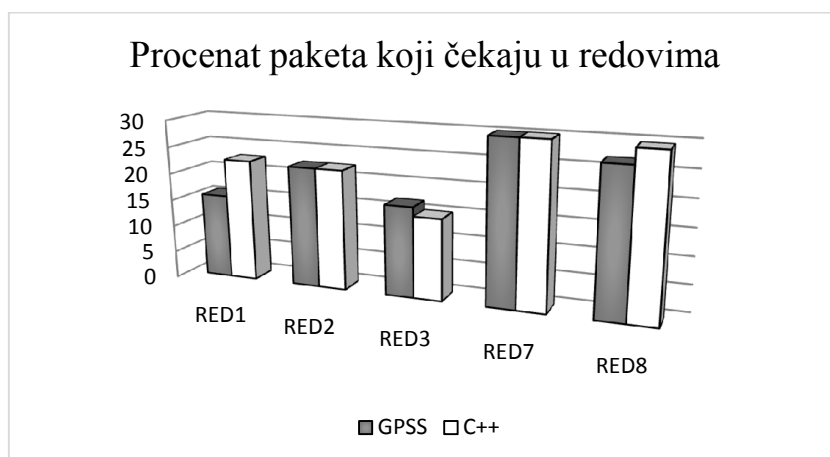
*Slika 9. Broj paketa u redovima čekanja na manuelnom sortiranju*

Na slici 10 su date vrednosti broja ulazaka paketa u redove čekanja za manuelno sortiranje paketa.



*Slika 10. Broj ulazaka paketa u redove čekanja*

Sa slike 11 se vide vrednosti broja paketa koji se zadržavaju u redovima čekanja. Zaključak u oba slučaja jeste da svaki peti paket stane u red u slučaju redova RED1, RED2 i RED3, dok je ova vrednost malo veća u slučaju redova RED7 i RED8 i možemo reći da svaki treći/četvrti paket staje u red.

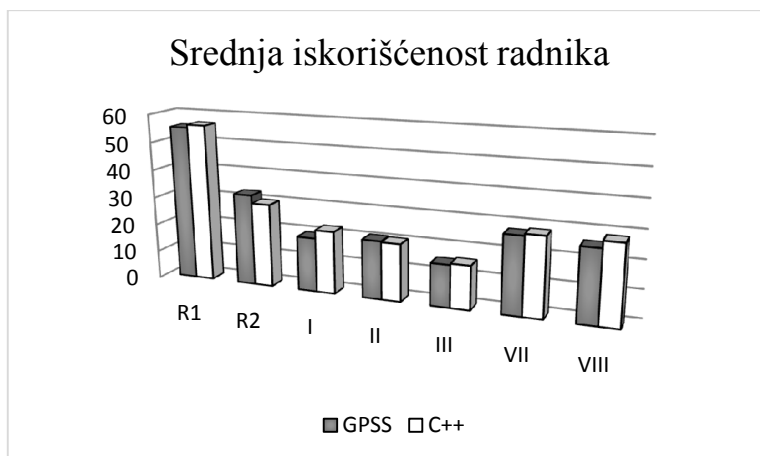


*Slika 11. Procenat paketa koji čekaju u redovima*

Na slici 12 su prikazane vrednosti iskorišćenosti radnika. Sa grafikona se vidi da je radnik na kodnom mestu R1 mnogo više opterećeniji od svih ostalih radnika, skoro duplo više posla se obavi na kodnom mestu R1 u odnosu na kodno mesto R2.



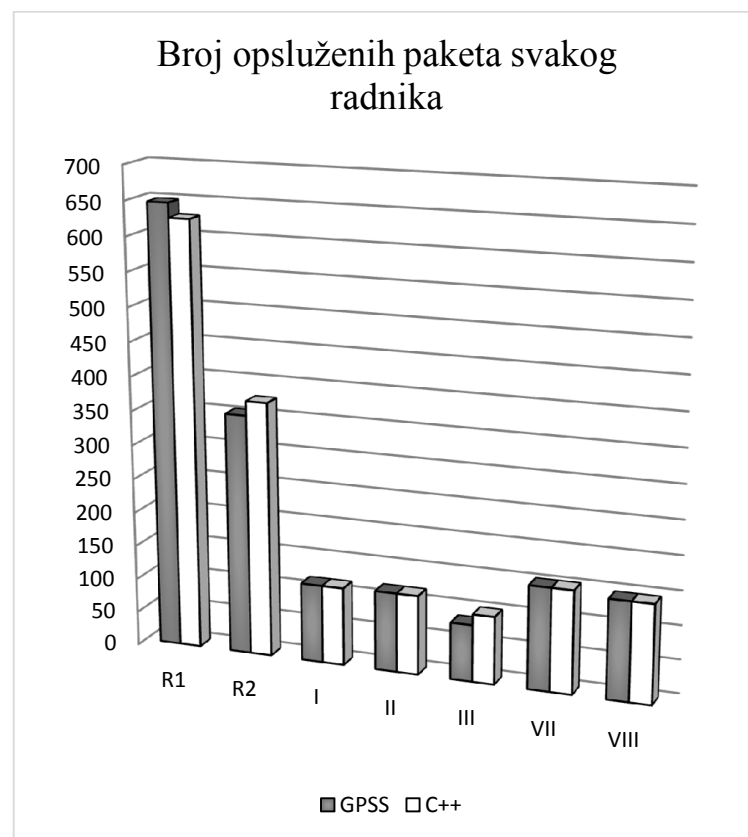
Radnik	GPSS	C++
R1	55.20	56.30
R2	32.60	29.76
I	19.30	22.29
II	20.70	20.41
III	15.00	15.52
VII	27.40	28.07
VIII	25.80	28.23



*Slika 12. Srednja iskorišćenost radnika na kodnim mestima i na ručnom sortiranju*

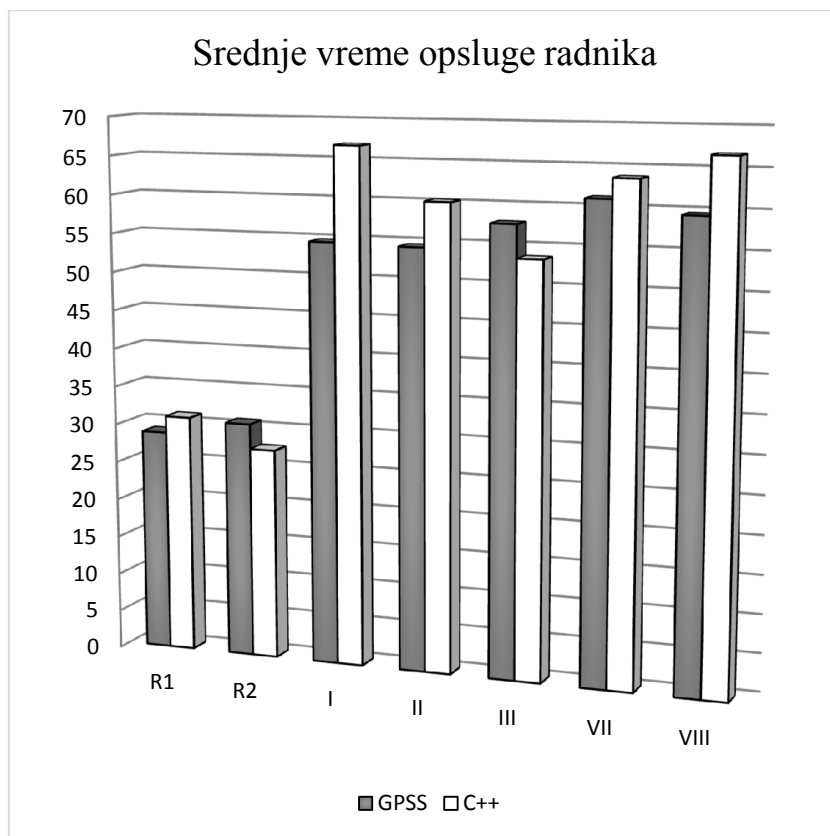
Na slici 13 su prikazane vrednosti broja paketa koje svaki od radnika opsluži. Sa grafikona se jasno vidi da je radnik na kodnom mestu R1 mnogo više opterećeniji od svih ostalih radnika, što se poklapa sa srednjom iskorišćenošću radnika.

Radnik	GPSS	C++
R1	647	626
R2	353	374
I	115	115
II	117	117
III	84	100
VII	152	152
VIII	146	146



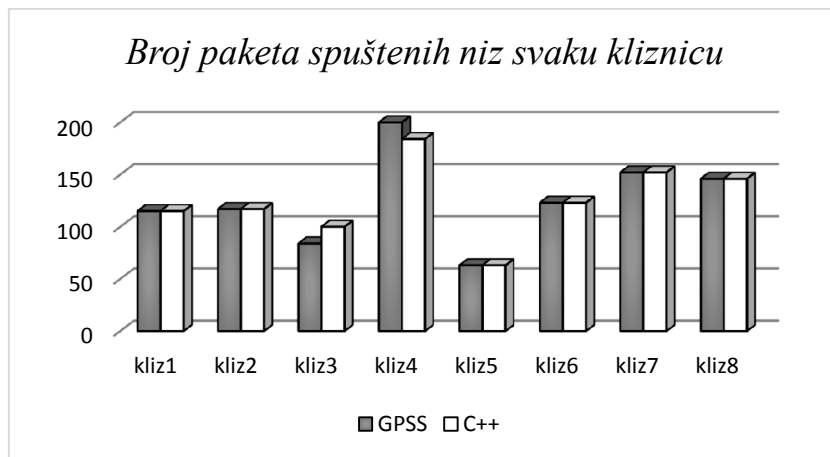
*Slika 13. Broj opsluženih paketa svakog radnika*

Sa druge strane vreme koje svako od radnika provede za razvrstavanje paketa drastično se povećava na mestima manualnog sortiranja. Prikazano na slici 14, jasno se vidi da je vreme koje radnik utroši prilikom manualnog sortiranja paketa duplo veće u odnosu na vreme potrebno radnicima na kodnim mestima R1 i R2.



*Slika 14. Srednje vreme opsluge radnika*

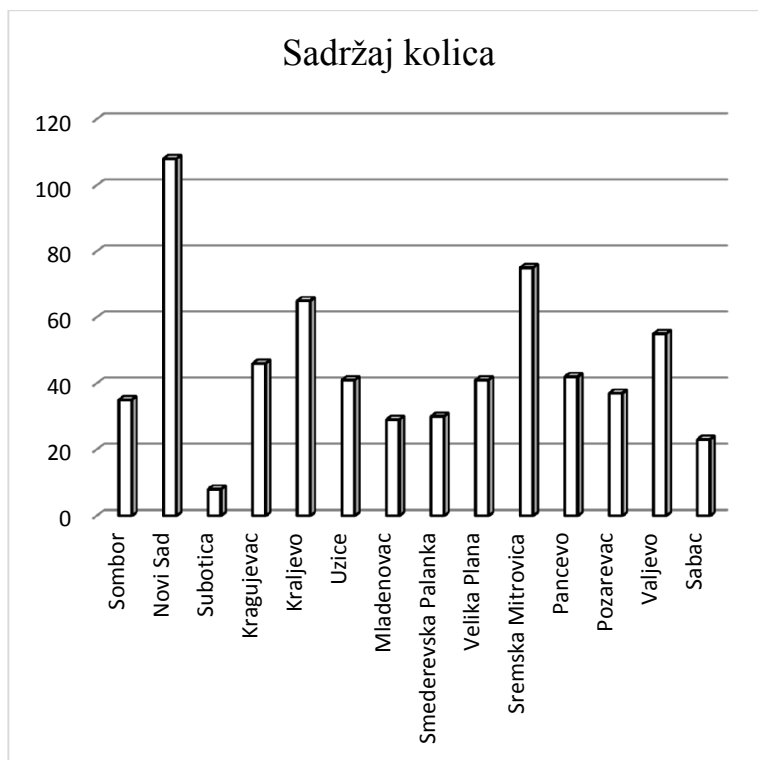
Sadržaj promenjive koju smo nazvali  $mem$  memoriše broj paketa koji prođu određenom kliznicom. Svih 1000 paketa se raspoređuje na 8 kliznica, i simulacije u oba modela (GPSS i C++) su skoro potpuno slične u ovom slučaju. Na slici 15 su prikazane vrednosti.



*Slika 15. Broj paketa spuštenih niz svaku kliznicu*

Na slici 16 je dat broj paketa u kolicima na kraju razvrstavanja, gde se tačno vidi količina paketa razvrstana po određišnim lokacijama. Ove vrednosti nisu merene u GPSS modelu.

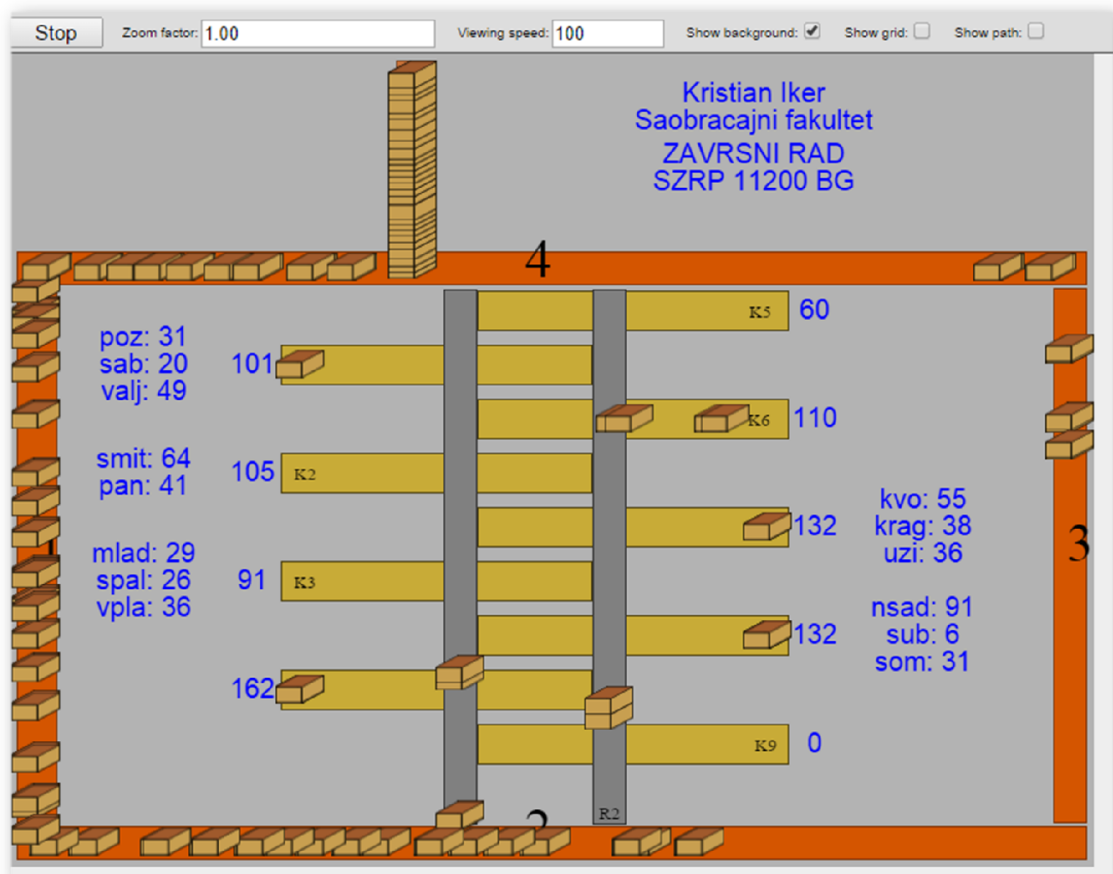
Kolica	C+ +
Sombor	35
Novi Sad	108
Subotica	8
Kragujevac	46
Kraljevo	65
Uzice	41
Mladenovac	29
Smederevska Palanka	30
Velika Plana	41
Sremska Mitrovica	75
Pancevo	42
Pozarevac	37
Valjevo	55
Sabac	23



*Slika 16. Sadržaj kolica*

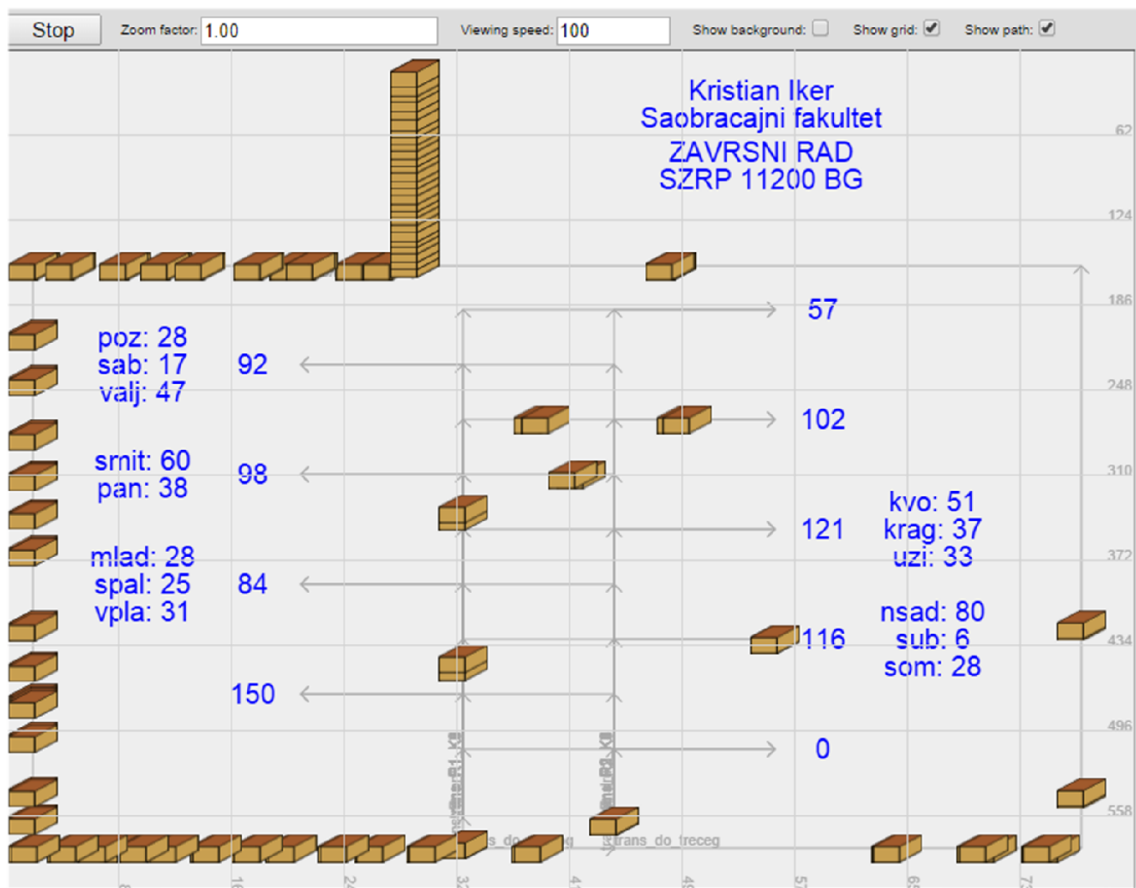
## 5.5. Realizacija animacije

Na slici 17 je prikazana slika izvršenja animacije simulacionog modela. Na slici se pored canvas elementa u kome se prikazuje animacija vide i kontrole koje služe za upravljanje animacijom. Pored Start/Stop/Reload dugmeta, koje služi za započinjanje i zaustavljanje animacije tu se nalaze i dugmići za prikaz pozadine, putanja objekata i mreže [7].



Slika 17: Prikaz animacije

Na slici 18 je prikazana slika izvršenja animacije bez pozadine sa vidljivom mrežom i putanjama. Tekstualnim kontrolama se vrši unos faktora zumiranja, kao i brzine izvršenja animacije[7].



*Slika 18: Prikaz animacije bez pozadine, sa putanjama i mrežom*

## 6. ZAKLJUČAK

Cilj ovog završnog rada jeste primena objektno orijentisane simulacije na rešavanje problema sistema masovnog opsluživanja, kakav je i sistem za razvrstavanje paketa pošte 11200 Beograd. Objektno orijentisana simulacija je u ovom radu izvršena primenom programskog jezika opšte namene C++. Implementiranim modelom dobijeni su rezultati koji se poklapaju sa rezultatima dobijenim prilikom realizacije simulacije nad istim modelom u simulacionom jeziku GPSS/FON [2]. Ovim je pokazano da je model izrađen u C++ prograskom jeziku verifikovan i validan. U radu je animiran sam proces razvrstavanja paketa, i celokupna simulacija je grafički prikazana korišćenjem internet tehnologija HTML5 Canvas i JavaScript.

Na osnovu dobijenih rezultata simulacije za postojeći sistem izvršena je uporedna analiza rezultata sa rezultatima modela prezentovanim u diplomskom radu [2] i utvrđeno je da su rezultati približno jednaki. Vršena je analiza parametara redova čekanja, analizirani su uređaji (resursi, radnici), takođe prikazane su i statistike histograma i kolica na samom kraju razvrstavanja.

Realizovana animacija jasno prikazuje kretanje svakog pojedinačnog paketa kroz sistem razvrstavanja, od samog ulaza u sistem preko trakastih transportera 7 i 8 do konačnog sortiranja (što spuštanjem niz kliznice, što tokom manuelnog razvrstavanja). Indikatori na ekranu vrše ispis vrednosti kolica i broja paketa spuštenih niz određene kliznice tako da se jasno vidi stanje u sistemu. Ovakav način praćenja realizovane simulacije pruža nam mnogo više informacija od same analize rezultata prikazane tabelom ili grafikom.

# LITERATURA

- [1] B. Radenković, M. Stanojević, A. Marković, *Računarska Simulacija*, Saobraćajni fakultet, FON, Beograd 2007.
- [2] B. Pekez, *Simulacija sistema za razvrstavanje paketa u pošti 11200*, Diplomski rad, Saobraćajni fakultet, Univerzitet u Beogradu, 1999.
- [3] M. Đogatović, *Simulacione strategije*, Seminarski rad, Saobraćajni fakultet, Univerzitet u Beogradu, 2010.
- [4] B. Stroustrup, *The C++ Programming Language*, 3rd Edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1997.
- [5] S. Lippman, J. Lajoie, B. Moo, *C++ Primer*, 5th Edition, Addison-Wesley Publishing Company, Westford, Massachusetts, 2013.
- [6] J. Liberty, *Teach Yourself C++ in 21 Days*, 2nd Edition, SAMS Publishing, Indianapolis, 1997.
- [7] M. Đogatović, M. Stanojević, *Animacija stohastičke simulacije primenom HTML5 kanvasa*, neobjavljeni materijal, Jul 2014.

## **PRILOG**



## Datoteka main.cpp

```
// Datoteka simul.h
#include "simul.h"
// Datoteka anim.h
#include "anim.h"

#include <iostream>
#include <string>
#include <iomanip>

// Makro naredbe
#define M1 (e)->vrati_parametar(4)
#define VREME simu.vrati_vreme_simulacije()
#define VRTR(vreme) simu.vrati_vreme_simulacije()+(vreme)
#define TERMINATE(vrednost) simu.unisti_entitet(e, vrednost)
#define LOGIC_S(uredjaj) (uredjaj) = true;
#define LOGIC_R(uredjaj) (uredjaj) = false;
#define TEST_E_P1(vrednost) e->vrati_parametar(1) ==
(vrednost)

#define CREATE_TEXTTAG(tt,value,x,y) \
        strcpy_s(obj, (tt)); \
        ANIM_TIME(VREME); \
        ANIM_CREATE("Poruka", obj); \
        ANIM_MESSAGE(obj, (value)); \
        ANIM_PLACE_AT(obj, (x), (y))

#define UPDATE_TEXTTAG(tt,value) \
        char msg[100]; \
        ANIM_TIME(VREME); \
        ANIM_MESSAGE((tt), value)

#define DESTROY_ANIM_OBJECT() \
        char obj[100]; \
        ANIM_TIME(VREME); \
        sprintf_s(obj, "%d", e-
>vrati_id()); \
        ANIM_DESTROY(obj)

// Uključujemo prostore imena simul i std
using namespace simul;
using namespace std;

// Deklarisanje memoriskih lokacija
struct Memorija{
    int inn, kliz[10];
    double posl;

    // Inicijalizacija vrednosti memorije.
    Memorija():inn(0),posl(0){ for(int i=0; i<10; i++)
kliz[i]=0;}

    // Ispis memoriskog sadržaja
    void print(){
        cout << "Sadržaj memorijskog objekata: " << endl;
        for (int i = 1; i < 10; ++i)
            if (i == 9) cout << "kliz[" << i << "]: " <<
kliz[i] << " <- međunarodni saobraćaj ka RS" << endl;
            else cout << "kliz[" << i << "]: " << kliz[i]
<< endl;
    }
};
```

```

        cout << "posl: " << posl << endl;
    }
}mem;

// Objekat simulacije
simulacija simu;
// Raspodele
raspodela rni(1), rn1(3), rn2(5), rn3(7), rn4(11), rn5(13),
rn6(17);
// Redovi cekanja
red_cekanja red(&simu), red1(&simu), red2(&simu),
red3(&simu), red7(&simu), red8(&simu);
// Kanali
resurs prvi(&simu, 1), drugi(&simu, 1), jedan(&simu, 1),
dva(&simu, 1), tri(&simu, 1), sedam(&simu, 1), osam(&simu,
1);
// Funkcije
funkcija func;

// Kreiranje tabela/histograma
// TRED TABLE Q$RED,0,1,45
histogram TRED(0, 1, 45);
// TAB1 TABLE M1,1000,1000,20
histogram TAB1(1000, 1000, 20);
// TAB2 TABLE M1,1000,1000,20
histogram TAB2(1000, 1000, 20);
// TRED1 TABLE Q$RED1,0,1,5
histogram TRED1(0, 1, 5);
// TRED2 TABLE Q$RED2,0,1,5
histogram TRED2(0, 1, 5);
// TRED3 TABLE Q$RED3,0,1,5
histogram TRED3(0, 1, 5);
// TRED7 TABLE Q$RED7,0,1,5
histogram TRED7(0, 1, 5);
// TRED8 TABLE Q$RED8,0,1,5
histogram TRED8(0, 1, 5);
// TAB21 TABLE M1,1000,1000,20
histogram TAB21(1000, 1000, 20);
// TAB22 TABLE M1,1000,1000,20
histogram TAB22(1000, 1000, 20);
// TAB23 TABLE M1,1000,1000,20
histogram TAB23(1000, 1000, 20);
// TAB27 TABLE M1,1000,1000,20
histogram TAB27(1000, 1000, 20);
// TAB28 TABLE M1,1000,1000,20
histogram TAB28(1000, 1000, 20);

// Inicijalizacija animacije.
ANIM_INIT;

// Kumulativna verovatnoca izbora kliznice
double X[7] = { 0.119, 0.223, 0.504, 0.574, 0.695, 0.839,
1.000 };
// Vrednosti izbora kliznice
double Y[7] = { 1.000, 2.000, 4.000, 5.000, 6.000, 7.000,
8.000 };
// Funkcija vraca slucajan ceo broj izabrane kliznice na
intervalu [1,8]
int izaberi_kliznicu(){
    return int(func.diskretna(rn5(), X, Y, 7));
}

```

```

// Funkcija vraca vrednost FN$KLI funkcije
double fn_kli(){
    return rn6.unif(20.000, 40.000);
}

// Inicijalizacija vrednosti skretnica
bool prav1 = false; bool skr1 = false;
bool prav2 = false; bool skr2 = false;
bool prav3 = false; bool skr3 = false;
bool prav4 = false; bool skr4 = false;
bool prav5 = false; bool skr5 = false;
bool prav6 = false; bool skr6 = false;
bool prav7 = false; bool skr7 = false;
bool prav8 = false; bool skr8 = false;
bool prav9 = false; bool skr9 = false;

// Definisanje vrednosti dogadjaja simulacionog modela
const int BLOK_GENERATE = 1;
const int BLOK_ADVANCE_728 = 2;
const int BLOK_ADVANCE_1022 = 3;

const int BLOK_ADVANCE_30 = 4;
const int BLOK_ADVANCE_30_10 = 5;

const int BLOK_ADVANCE_30_10_160 = 6;
const int BLOK_ADVANCE_30_10_160_14 = 7;
const int BLOK_ADVANCE_30_10_160_14_60 = 8;

const int BLOK_ADVANCE_30_10_120 = 9;
const int BLOK_ADVANCE_30_10_120_14 = 10;
const int BLOK_ADVANCE_30_10_120_14_60 = 11;

const int BLOK_ADVANCE_30_10_80 = 12;
const int BLOK_ADVANCE_30_10_80_14 = 13;
const int BLOK_ADVANCE_30_10_80_14_60 = 14;

const int BLOK_ADVANCE_30_10_40 = 15;
const int BLOK_ADVANCE_30_10_40_14 = 16;
const int BLOK_ADVANCE_30_10_40_14_FN = 17;

const int BLOK_ADVANCE_30_10_181 = 18;
const int BLOK_ADVANCE_30_10_181_14 = 19;
const int BLOK_ADVANCE_30_10_181_14_FN = 20;

const int BLOK_ADVANCE_30_10_141 = 21;
const int BLOK_ADVANCE_30_10_141_14 = 22;
const int BLOK_ADVANCE_30_10_141_14_FN = 23;

const int BLOK_ADVANCE_30_10_101 = 24;
const int BLOK_ADVANCE_30_10_101_14 = 25;
const int BLOK_ADVANCE_30_10_101_14_60 = 26;

const int BLOK_ADVANCE_30_10_61 = 27;
const int BLOK_ADVANCE_30_10_61_14 = 28;
const int BLOK_ADVANCE_30_10_61_14_60 = 29;

const int BLOK_ADVANCE_30_10_21 = 30;
const int BLOK_ADVANCE_30_10_21_14 = 31;
const int BLOK_ADVANCE_30_10_21_14_FN = 32;

```

```

const int BLOK_ADVANCE_90 = 33;
const int BLOK_ADVANCE_90_30 = 34;
const int BLOK_ADVANCE_90_30_10 = 35;
const int BLOK_ADVANCE_90_1432 = 36;

const int BLOK_ADVANCE_90_30_10_171 = 37;
const int BLOK_ADVANCE_90_30_10_171_14 = 38;
const int BLOK_ADVANCE_90_30_10_171_14_FN = 39;

const int BLOK_ADVANCE_90_30_10_130 = 40;
const int BLOK_ADVANCE_90_30_10_130_14 = 41;
const int BLOK_ADVANCE_90_30_10_130_14_FN = 42;

const int BLOK_ADVANCE_90_30_10_89 = 43;
const int BLOK_ADVANCE_90_30_10_89_14 = 44;
const int BLOK_ADVANCE_90_30_10_89_14_FN = 45;

const int BLOK_ADVANCE_90_30_10_48 = 46;
const int BLOK_ADVANCE_90_30_10_48_14 = 47;
const int BLOK_ADVANCE_90_30_10_48_14_FN = 48;

const int BLOK_ADVANCE_90_30_10_193 = 49;
const int BLOK_ADVANCE_90_30_10_193_14 = 50;
const int BLOK_ADVANCE_90_30_10_193_14_FN = 51;

const int BLOK_ADVANCE_90_30_10_153 = 52;
const int BLOK_ADVANCE_90_30_10_153_14 = 53;
const int BLOK_ADVANCE_90_30_10_153_14_FN = 54;

const int BLOK_ADVANCE_90_30_10_112 = 55;
const int BLOK_ADVANCE_90_30_10_112_14 = 56;
const int BLOK_ADVANCE_90_30_10_112_14_FN = 57;

const int BLOK_ADVANCE_90_30_10_72 = 58;
const int BLOK_ADVANCE_90_30_10_72_14 = 59;
const int BLOK_ADVANCE_90_30_10_72_14_FN = 60;

const int BLOK_ADVANCE_90_30_10_31 = 61;
const int BLOK_ADVANCE_90_30_10_31_14 = 62;
const int BLOK_ADVANCE_90_30_10_31_14_FN = 63;

// Deklaracije funkcija koje se izvrsavaju kada se
// odredjeni dogodjaj u simulacionom modelu dogodi
void blok_generate(entitet *e);
void blok_advance_728(entitet *e);
void blok_advance_1022(entitet *e);

void blok_advance_30(entitet *e);
void blok_advance_30_10(entitet *e);

void blok_advance_30_10_160(entitet *e);
void blok_advance_30_10_160_14(entitet *e);
void blok_advance_30_10_160_14_60(entitet *e);

void blok_advance_30_10_120(entitet *e);
void blok_advance_30_10_120_14(entitet *e);
void blok_advance_30_10_120_14_60(entitet *e);

void blok_advance_30_10_80(entitet *e);
void blok_advance_30_10_80_14(entitet *e);
void blok_advance_30_10_80_14_60(entitet *e);

```

```

void blok_advance_30_10_40(entitet *e);
void blok_advance_30_10_40_14(entitet *e);
void blok_advance_30_10_40_14_FN(entitet *e);

void blok_advance_30_10_181(entitet *e);
void blok_advance_30_10_181_14(entitet *e);
void blok_advance_30_10_181_14_FN(entitet *e);

void blok_advance_30_10_141(entitet *e);
void blok_advance_30_10_141_14(entitet *e);
void blok_advance_30_10_141_14_FN(entitet *e);

void blok_advance_30_10_101(entitet *e);
void blok_advance_30_10_101_14(entitet *e);
void blok_advance_30_10_101_14_60(entitet *e);

void blok_advance_30_10_61(entitet *e);
void blok_advance_30_10_61_14(entitet *e);
void blok_advance_30_10_61_14_60(entitet *e);

void blok_advance_30_10_21(entitet *e);
void blok_advance_30_10_21_14(entitet *e);
void blok_advance_30_10_21_14_FN(entitet *e);

void blok_advance_90(entitet *e);
void blok_advance_90_30(entitet *e);
void blok_advance_90_30_10(entitet *e);
void blok_advance_90_1432(entitet *e);

void blok_advance_90_30_10_171(entitet *e);
void blok_advance_90_30_10_171_14(entitet *e);
void blok_advance_90_30_10_171_14_FN(entitet *e);

void blok_advance_90_30_10_130(entitet *e);
void blok_advance_90_30_10_130_14(entitet *e);
void blok_advance_90_30_10_130_14_FN(entitet *e);

void blok_advance_90_30_10_89(entitet *e);
void blok_advance_90_30_10_89_14(entitet *e);
void blok_advance_90_30_10_89_14_FN(entitet *e);

void blok_advance_90_30_10_48(entitet *e);
void blok_advance_90_30_10_48_14(entitet *e);
void blok_advance_90_30_10_48_14_FN(entitet *e);

void blok_advance_90_30_10_193(entitet *e);
void blok_advance_90_30_10_193_14(entitet *e);
void blok_advance_90_30_10_193_14_FN(entitet *e);

void blok_advance_90_30_10_153(entitet *e);
void blok_advance_90_30_10_153_14(entitet *e);
void blok_advance_90_30_10_153_14_FN(entitet *e);

void blok_advance_90_30_10_112(entitet *e);
void blok_advance_90_30_10_112_14(entitet *e);
void blok_advance_90_30_10_112_14_FN(entitet *e);

void blok_advance_90_30_10_72(entitet *e);
void blok_advance_90_30_10_72_14(entitet *e);
void blok_advance_90_30_10_72_14_FN(entitet *e);

```

```

void blok_advance_90_30_10_31(entitet *e);
void blok_advance_90_30_10_31_14(entitet *e);
void blok_advance_90_30_10_31_14_FN(entitet *e);

// Izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
    switch (dog) {
        case BLOK_GENERATE:
            blok_generate(e);
            break;
        case BLOK_ADVANCE_728:
            blok_advance_728(e);
            break;
        case BLOK_ADVANCE_1022:
            blok_advance_1022(e);
            break;
        case BLOK_ADVANCE_30:
            blok_advance_30(e);
            break;
        case BLOK_ADVANCE_30_10:
            blok_advance_30_10(e);
            break;
        case BLOK_ADVANCE_30_10_160:
            blok_advance_30_10_160(e);
            break;
        case BLOK_ADVANCE_30_10_160_14:
            blok_advance_30_10_160_14(e);
            break;
        case BLOK_ADVANCE_30_10_160_14_60:
            blok_advance_30_10_160_14_60(e);
            break;
        case BLOK_ADVANCE_30_10_120:
            blok_advance_30_10_120(e);
            break;
        case BLOK_ADVANCE_30_10_120_14:
            blok_advance_30_10_120_14(e);
            break;
        case BLOK_ADVANCE_30_10_120_14_60:
            blok_advance_30_10_120_14_60(e);
            break;
        case BLOK_ADVANCE_30_10_80:
            blok_advance_30_10_80(e);
            break;
        case BLOK_ADVANCE_30_10_80_14:
            blok_advance_30_10_80_14(e);
            break;
        case BLOK_ADVANCE_30_10_80_14_60:
            blok_advance_30_10_80_14_60(e);
            break;
        case BLOK_ADVANCE_30_10_40:
            blok_advance_30_10_40(e);
            break;
        case BLOK_ADVANCE_30_10_40_14:
            blok_advance_30_10_40_14(e);
            break;
        case BLOK_ADVANCE_30_10_40_14_FN:
            blok_advance_30_10_40_14_FN(e);
            break;
        case BLOK_ADVANCE_30_10_181:
            blok_advance_30_10_181(e);
    }
}

```

```

        break;
case BLOK_ADVANCE_30_10_181_14:
    blok_advance_30_10_181_14(e);
    break;
case BLOK_ADVANCE_30_10_181_14_FN:
    blok_advance_30_10_181_14_FN(e);
    break;
case BLOK_ADVANCE_30_10_141:
    blok_advance_30_10_141(e);
    break;
case BLOK_ADVANCE_30_10_141_14:
    blok_advance_30_10_141_14(e);
    break;
case BLOK_ADVANCE_30_10_141_14_FN:
    blok_advance_30_10_141_14_FN(e);
    break;
case BLOK_ADVANCE_30_10_101:
    blok_advance_30_10_101(e);
    break;
case BLOK_ADVANCE_30_10_101_14:
    blok_advance_30_10_101_14(e);
    break;
case BLOK_ADVANCE_30_10_101_14_60:
    blok_advance_30_10_101_14_60(e);
    break;
case BLOK_ADVANCE_30_10_61:
    blok_advance_30_10_61(e);
    break;
case BLOK_ADVANCE_30_10_61_14:
    blok_advance_30_10_61_14(e);
    break;
case BLOK_ADVANCE_30_10_61_14_60:
    blok_advance_30_10_61_14_60(e);
    break;
case BLOK_ADVANCE_30_10_21:
    blok_advance_30_10_21(e);
    break;
case BLOK_ADVANCE_30_10_21_14:
    blok_advance_30_10_21_14(e);
    break;
case BLOK_ADVANCE_30_10_21_14_FN:
    blok_advance_30_10_21_14_FN(e);
    break;
case BLOK_ADVANCE_90:
    blok_advance_90(e);
    break;
case BLOK_ADVANCE_90_30:
    blok_advance_90_30(e);
    break;
case BLOK_ADVANCE_90_30_10:
    blok_advance_90_30_10(e);
    break;
case BLOK_ADVANCE_90_1432:
    blok_advance_90_1432(e);
    break;
case BLOK_ADVANCE_90_30_10_171:
    blok_advance_90_30_10_171(e);
    break;
case BLOK_ADVANCE_90_30_10_171_14:
    blok_advance_90_30_10_171_14(e);
    break;

```

```

case BLOK_ADVANCE_90_30_10_171_14_FN:
    blok_advance_90_30_10_171_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_130:
    blok_advance_90_30_10_130(e);
    break;
case BLOK_ADVANCE_90_30_10_130_14:
    blok_advance_90_30_10_130_14(e);
    break;
case BLOK_ADVANCE_90_30_10_130_14_FN:
    blok_advance_90_30_10_130_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_89:
    blok_advance_90_30_10_89(e);
    break;
case BLOK_ADVANCE_90_30_10_89_14:
    blok_advance_90_30_10_89_14(e);
    break;
case BLOK_ADVANCE_90_30_10_89_14_FN:
    blok_advance_90_30_10_89_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_48:
    blok_advance_90_30_10_48(e);
    break;
case BLOK_ADVANCE_90_30_10_48_14:
    blok_advance_90_30_10_48_14(e);
    break;
case BLOK_ADVANCE_90_30_10_48_14_FN:
    blok_advance_90_30_10_48_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_193:
    blok_advance_90_30_10_193(e);
    break;
case BLOK_ADVANCE_90_30_10_193_14:
    blok_advance_90_30_10_193_14(e);
    break;
case BLOK_ADVANCE_90_30_10_193_14_FN:
    blok_advance_90_30_10_193_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_153:
    blok_advance_90_30_10_153(e);
    break;
case BLOK_ADVANCE_90_30_10_153_14:
    blok_advance_90_30_10_153_14(e);
    break;
case BLOK_ADVANCE_90_30_10_153_14_FN:
    blok_advance_90_30_10_153_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_112:
    blok_advance_90_30_10_112(e);
    break;
case BLOK_ADVANCE_90_30_10_112_14:
    blok_advance_90_30_10_112_14(e);
    break;
case BLOK_ADVANCE_90_30_10_112_14_FN:
    blok_advance_90_30_10_112_14_FN(e);
    break;
case BLOK_ADVANCE_90_30_10_72:
    blok_advance_90_30_10_72(e);
    break;
case BLOK_ADVANCE_90_30_10_72_14:

```



```

        blok_advance_90_30_10_72_14(e);
        break;
    case BLOK_ADVANCE_90_30_10_72_14_FN:
        blok_advance_90_30_10_72_14_FN(e);
        break;
    case BLOK_ADVANCE_90_30_10_31:
        blok_advance_90_30_10_31(e);
        break;
    case BLOK_ADVANCE_90_30_10_31_14:
        blok_advance_90_30_10_31_14(e);
        break;
    case BLOK_ADVANCE_90_30_10_31_14_FN:
        blok_advance_90_30_10_31_14_FN(e);
        break;
}
}

int counter = 0;
void blok_generate(entitet *e) {
    entitet *novi;

    // ASSIGN 1,FN$PRAVAC
    e->postavi_parametar(1, izaberi_kliznicu());
    // Pamtimo u parametru 4 trenutak pristizanja entiteta
    e->postavi_parametar(4, VREME);
    // QUEUE RED
    // Postavi klijenta u red cekanja
    red.ured(e);
    // ADVANCE 728
    simu.rasporedi(e, BLOK_ADVANCE_728, VRTR(728));

    char obj[100];
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_TIME(VREME);
    ANIM_CREATE("Paket", obj);
    ANIM_DURATION(obj, 728.00);
    ANIM_PLACE_ON(obj, "trans_dolazak");

    // GENERATE V1,,1,1000
    // Postavljamo vreme narednog dolaska
    if (++counter<1000) {
        // Stvaramo naredni paket
        novi = simu.napravi_entitet();
        // Postavljamo vreme narednog dolaska
        simu.rasporedi(novi, BLOK_GENERATE, VRTR(20 +
rn1.expo(10)));
    }
}

void blok_advance_728(entitet *e) {
    entitet *tekuci;
    // DEPART RED
    tekuci = red.izred();
    // TABULATE TRED
    TRED(red.velicina());
    // SAVEVALUE INN+,1
    mem.inn++;
    // TEST E X$INN,1000,BLOK1
    if (mem.inn == 1000){
        // ASSIGN 3,C1
        e->postavi_parametar(3, VREME);
    }
}

```

```

        // SAVEVALUE POSL,P3
        mem.posl = VREME;
    }
    // BLOK1 ADVANCE 1022
    simu.rasporedi(tekuci, BLOK_ADVANCE_1022, VRTR(1022));

    char obj[100];
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_TIME(VREME);
    ANIM_DURATION(obj, 1022.0);
    ANIM_PLACE_ON(obj, "trans_do_prvog");
}

void blok_advance_1022(entitet *e) {
    // GATE NU PRVI,BLOK2
    if (prvi.raspoloziv()){
        // SEIZE PRVI
        prvi.zauzmi(e);
        // ADVANCE 30,FN$EXPO2
        double v = rn3.expo(30);
        simu.rasporedi(e, BLOK_ADVANCE_30, VRTR(v));

        char obj[100];
        sprintf_s(obj, "%d", e->vrati_id());
        ANIM_TIME(VREME);
        ANIM_DURATION(obj, v);
        ANIM_PLACE_ON(obj, "trans_R_prvi");
    }else{
        // BLOK2 ADVANCE 90
        simu.rasporedi(e, BLOK_ADVANCE_90, VRTR(90));

        char obj[100];
        sprintf_s(obj, "%d", e->vrati_id());
        ANIM_TIME(VREME);
        ANIM_DURATION(obj, 90);
        ANIM_PLACE_ON(obj, "trans_do_drugog");
    }
}

void blok_advance_30(entitet *e) {
    // RELEASE PRVI
    prvi.oslobodi(e);
    // ADVANCE 10
    simu.rasporedi(e, BLOK_ADVANCE_30_10, VRTR(10));
}

void blok_advance_30_10(entitet *e) {
    char obj[100];
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_TIME(VREME);

    // TEST E P1,1,RED1
    if (TEST_E_P1(1)){
        // ADVANCE 160
        simu.rasporedi(e, BLOK_ADVANCE_30_10_160,
VRTR(160));

        ANIM_DURATION(obj, 160.00);
        ANIM_PLACE_ON(obj, "trans_R1_K1");
    }
    // TEST E P1,2,RED2

```

```

else if (TEST_E_P1(2)){
    // ADVANCE 120
    simu.rasporedi(e, BLOK_ADVANCE_30_10_120,
VRTR(120));

    ANIM_DURATION(obj, 120.00);
    ANIM_PLACE_ON(obj, "trans_R1_K2");
}
// TEST E P1,3,RED3
else if (TEST_E_P1(3)){
    // ADVANCE 80
    simu.rasporedi(e, BLOK_ADVANCE_30_10_80,
VRTR(80));

    ANIM_DURATION(obj, 80.00);
    ANIM_PLACE_ON(obj, "trans_R1_K3");
}
// TEST E P1,4,RED4
else if (TEST_E_P1(4)){
    // TRANSFER .30,,TRECI
    if (rni() < 0.30)
    {
        // ADVANCE 80
        simu.rasporedi(e, BLOK_ADVANCE_30_10_80,
VRTR(80));

        ANIM_DURATION(obj, 80.00);
        ANIM_PLACE_ON(obj, "trans_R1_K3");
    }
    else
    {
        // ADVANCE 40
        simu.rasporedi(e, BLOK_ADVANCE_30_10_40,
VRTR(40));

        ANIM_DURATION(obj, 40.00);
        ANIM_PLACE_ON(obj, "trans_R1_K4");
    }
}
// TEST E P1,5,RED5
else if (TEST_E_P1(5)){
    // ADVANCE 181
    simu.rasporedi(e, BLOK_ADVANCE_30_10_181,
VRTR(181));

    ANIM_DURATION(obj, 181.00);
    ANIM_PLACE_ON(obj, "trans_R1_K5");
}
// TEST E P1,6,RED6
else if (TEST_E_P1(6)){
    // ADVANCE 141
    simu.rasporedi(e, BLOK_ADVANCE_30_10_141,
VRTR(141));

    ANIM_DURATION(obj, 141.00);
    ANIM_PLACE_ON(obj, "trans_R1_K6");
}
// TEST E P1,7,RED7
else if (TEST_E_P1(7)){
    // ADVANCE 101

```

```

        simu.rasporedi(e, BLOK_ADVANCE_30_10_101,
VRTR(101));

        ANIM_DURATION(obj, 101.00);
        ANIM_PLACE_ON(obj, "trans_R1_K7");
    }
    // TEST E P1,8,RED8
    else if (TEST_E_P1(8)){
        // ADVANCE 61
        simu.rasporedi(e, BLOK_ADVANCE_30_10_61,
VRTR(61));

        ANIM_DURATION(obj, 61.00);
        ANIM_PLACE_ON(obj, "trans_R1_K8");
    }
    // TEST E P1,9,RED9
    else if (TEST_E_P1(9)){
        // ADVANCE 21
        simu.rasporedi(e, BLOK_ADVANCE_30_10_21,
VRTR(21));

        ANIM_DURATION(obj, 21.00);
        ANIM_PLACE_ON(obj, "trans_R1_K9");
    }
}

int poz = 0;
int valj = 0;
int sab = 0;
void blok_advance_30_10_160(entitet *e) {
    // LOGIC S PRAV8
    LOGIC_S(prav8);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_160_14, VRTR(14));
}

void blok_advance_30_10_160_14(entitet *e) {
    // LOGIC R PRAV8
    LOGIC_R(prav8);
    // SAVEVALUE KLIZ1+,1
    mem.kliz[1]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[1]);
    ANIM_MESSAGE("Por1", msg);

    // TRANSFER ,LAB1
    // LAB1 QUEUE RED1
    red1.ured(e);

    if (jedan.raspoloziv())
    {
        // DEPART RED1
        entitet* tekuci = red1.izred();
        // TABULATE TRED1
        TRED1(red1.velicina());
        // SEIZE JEDAN
        jedan.zauzmi(tekuci);
        // ADVANCE 60,FN$EXPO3

```

```

        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_160_14_60, VRTR(rn4.expo(60)));
    }
}

void blok_advance_30_10_160_14_60(entitet *e) {
    // RELEASE JEDAN
    jedan.oslobodi(e);
    if (red1.velicina() > 0)
    {
        entitet* novi = red1.izred();
        // TABULATE TRED1
        TRED1(red1.velicina());
        // SEIZE JEDAN
        jedan.zauzmi(novi);
        // ADVANCE 60, FN$EXPO3
        simu.rasporedi(novi, BLOK_ADVANCE_30_10_160_14_60,
VRTR(rn4.expo(60)));
    }
    // TABULATE TAB21
    TAB21(M1);
    // TRANSFER .37,,POZ
    // TRANSFER .32,,SAB
    // VALJ TERMINATE 1
    // POZ TERMINATE 1
    // SAB TERMINATE 1
    if (rni() < 0.37){
        poz++;

        UPDATE_TEXTTAG("jedan_1", "poz: " +
to_string(poz));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
        return;
    }

    if (rni() < 0.32){
        sab++;

        UPDATE_TEXTTAG("jedan_2", "sab: " +
to_string(sab));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
        return;
    }
    valj++;

    UPDATE_TEXTTAG("jedan_3", "valj: " + to_string(valj));
    DESTROY_ANIM_OBJECT();

    TERMINATE(1);
}

int smit = 0;
int pan = 0;
void blok_advance_30_10_120(entitet *e) {
    // LOGIC S PRAV6
    LOGIC_S(prav6);
    // ADVANCE 14

```

```

        simu.rasporedi(e, BLOK_ADVANCE_30_10_120_14, VRTR(14));
    }

void blok_advance_30_10_120_14(entitet *e) {
    // LOGIC R PRAV6
    LOGIC_R(prav6);
    // SAVEVALUE KLIZ2+,1
    mem.kliz[2]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[2]);
    ANIM_MESSAGE("Por2", msg);

    // TRANSFER ,LAB2
    // LAB2 QUEUE RED2
    red2.ured(e);

    if (dva.raspoloziv())
    {
        // DEPART RED2
        entitet* tekuci = red2.izred();
        // TABULATE TRED2
        TRED2(red2.velicina());
        // SEIZE DVA
        dva.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_120_14_60, VRTR(rn4.expo(60)));
    }
}

void blok_advance_30_10_120_14_60(entitet *e) {
    // RELEASE DVA
    dva.oslobodi(e);
    if (red2.velicina() > 0)
    {
        entitet* novi = red2.izred();
        // TABULATE TRED2
        TRED2(red2.velicina());
        // SEIZE DVA
        dva.zauzmi(novi);
        // ADVANCE 60, FN$EXPO3
        simu.rasporedi(novi, BLOK_ADVANCE_30_10_120_14_60,
VRTR(rn4.expo(60)));
    }
    // TABULATE TAB22
    TAB22(M1);
    // TRANSFER .65,,SMIT
    // PAN TERMINATE 1
    // SMIT TERMINATE 1
    if (rni() < 0.65){
        smit++;

        UPDATE_TEXTTAG("dva_1", "smit: " +
to_string(smit));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
        return;
    }
}

```

```

        pan++;

        UPDATE_TEXTTAG("dva_2", "pan: " + to_string(pan));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
    }

    int mlad = 0;
    int spal = 0;
    int vpla = 0;
    void blok_advance_30_10_80(entitet *e) {
        // LOGIC S PRAV4
        LOGIC_S(prav4);
        // ADVANCE 14
        simu.rasporedi(e, BLOK_ADVANCE_30_10_80_14, VRTR(14));
    }

    void blok_advance_30_10_80_14(entitet *e) {
        // LOGIC R PRAV4
        LOGIC_R(prav4);
        // SAVEVALUE KLIZ3+,1
        mem.kliz[3]++;

        char msg[10];
        ANIM_TIME(VREME);
        sprintf_s(msg, "%d", mem.kliz[3]);
        ANIM_MESSAGE("Por3", msg);

        // TRANSFER ,LAB3
        // LAB3 QUEUE RED3
        red3.ured(e);

        if (tri.raspoloziv())
        {
            // DEPART RED3
            entitet* tekuci = red3.izred();
            // TABULATE TRED3
            TRED3(red3.velicina());
            // SEIZE TRI
            tri.zauzmi(tekuci);
            // ADVANCE 60,FN$EXPO3
            simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_80_14_60, VRTR(rn4.expo(60)));
        }
    }

    void blok_advance_30_10_80_14_60(entitet *e) {
        // RELEASE TRI
        tri.oslobodi(e);
        if (red3.velicina() > 0)
        {
            entitet* novi = red3.izred();
            // TABULATE TRED3
            TRED3(red3.velicina());
            // SEIZE TRI
            tri.zauzmi(novi);
            // ADVANCE 60,FN$EXPO3
            simu.rasporedi(novi, BLOK_ADVANCE_30_10_80_14_60,
VRTR(rn4.expo(60)));
        }
    }

```

```

        // TABULATE TAB23
        TAB23(M1);
        // TRANSFER .33,,MLAD
        // TRANSFER .33,,SPAL
        // VPLA TERMINATE 1
        // MLAD TERMINATE 1
        // SPAL TERMINATE 1
        if (rni() < 0.33){
            mlad++;

            UPDATE_TEXTTAG("tri_1", "mlad: " +
to_string(mlad));
            DESTROY_ANIM_OBJECT();

            TERMINATE(1);
            return;
        }

        if (rni() < 0.33){
            spal++;

            UPDATE_TEXTTAG("tri_2", "spal: " +
to_string(spal));
            DESTROY_ANIM_OBJECT();

            TERMINATE(1);
            return;
        }
        vpla++;

        UPDATE_TEXTTAG("tri_3", "vpla: " + to_string(vpla));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
    }

void blok_advance_30_10_40(entitet *e) {
    // LOGIC S PRAV2
    LOGIC_S(prav2);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_40_14, VRTR(14));
}

void blok_advance_30_10_40_14(entitet *e) {
    // LOGIC R PRAV2
    LOGIC_R(prav2);
    // SAVEVALUE KLIZ4+,1
    mem.kliz[4]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[4]);
    ANIM_MESSAGE("Por4", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_30_10_40_14_FN,
VRTR(fn_kli()));
}

void blok_advance_30_10_40_14_FN(entitet *e) {
    // TABULATE TAB1

```



```

    TAB1(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_30_10_181(entitet *e) {
    // LOGIC S PRAV9
    LOGIC_S(prav9);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_181_14, VRTR(14));
}

void blok_advance_30_10_181_14(entitet *e) {
    // LOGIC R PRAV9
    LOGIC_R(prav9);
    // SAVEVALUE KLIZ5+,1
    mem.kliz[5]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[5]);
    ANIM_MESSAGE("Por5", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_30_10_181_14_FN,
VRTR(fn_kli()));
}

void blok_advance_30_10_181_14_FN(entitet *e) {
    // TABULATE TAB1
    TAB1(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_30_10_141(entitet *e) {
    // LOGIC S PRAV7
    LOGIC_S(prav7);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_141_14, VRTR(14));
}

void blok_advance_30_10_141_14(entitet *e) {
    // LOGIC R PRAV7
    LOGIC_R(prav7);
    // SAVEVALUE KLIZ6+,1
    mem.kliz[6]++;

    char msg[10];

```

```

    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[6]);
    ANIM_MESSAGE("Por6", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_30_10_141_14_FN,
VRTR(fn_kli()));
}

void blok_advance_30_10_141_14_FN(entitet *e) {
    // TABULATE TAB1
    TAB1(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

int kvo = 0;
int krag = 0;
int uzi = 0;
void blok_advance_30_10_101(entitet *e) {
    // LOGIC S PRAV5
    LOGIC_S(prav5);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_101_14, VRTR(14));
}
void blok_advance_30_10_101_14(entitet *e) {
    // LOGIC R PRAV5
    LOGIC_R(prav5);
    // SAVEVALUE KLIZ7+,1
    mem.kliz[7]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[7]);
    ANIM_MESSAGE("Por7", msg);

    // TRANSFER ,LAB7
    // LAB7 QUEUE RED7
    red7.ured(e);

    if (sedam.raspoloziv())
    {
        // DEPART RED7
        entitet* tekuci = red7.izred();
        // TABULATE TRED7
        TRED7(red7.velicina());
        // SEIZE SEDAM
        sedam.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_101_14_60, VRTR(rn4.expo(60)));
    }
}

void blok_advance_30_10_101_14_60(entitet *e) {

```

```

// RELEASE SEDAM
sedam.oslobodi(e);
if (red7.velicina() > 0)
{
    entitet* novi = red7.izred();
    // TABULATE TRED7
    TRED7(red7.velicina());
    // SEIZE SEDAM
    sedam.zauzmi(novi);
    // ADVANCE 60, FN$EXPO3
    simu.rasporedi(novi, BLOK_ADVANCE_30_10_101_14_60,
VRTR(rn4.expo(60)));
}
// TABULATE TAB27
TAB27(M1);
// TRANSFER .42,,KVO
// TRANSFER .42,,KRAG
// UZI TERMINATE 1
// KVO TERMINATE 1
// KRAG TERMINATE 1
if (rni() < 0.42){
    kvo++;

    UPDATE_TEXTTAG("sedam_1", "kvo: " +
to_string(kvo));
    DESTROY_ANIM_OBJECT();

    TERMINATE(1);
    return;
}

if (rni() < 0.42){
    krag++;

    UPDATE_TEXTTAG("sedam_2", "krag: " +
to_string(krag));
    DESTROY_ANIM_OBJECT();

    TERMINATE(1);
    return;
}
uzi++;

UPDATE_TEXTTAG("sedam_3", "uzi: " + to_string(uzi));
DESTROY_ANIM_OBJECT();

TERMINATE(1);
}

int nsad = 0;
int sub = 0;
int som = 0;
void blok_advance_30_10_61(entitet *e) {
    // LOGIC S PRAV3
    LOGIC_S(prav3);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_61_14, VRTR(14));
}

void blok_advance_30_10_61_14(entitet *e) {
    // LOGIC R PRAV3

```

```

LOGIC_R(prav3);
// SAVEVALUE KLIZ8+,1
mem.kliz[8]++;

char msg[10];
ANIM_TIME(VREME);
sprintf_s(msg, "%d", mem.kliz[8]);
ANIM_MESSAGE("Por8", msg);

// TRANSFER ,LAB8
// LAB8 QUEUE RED8
red8.ured(e);

if (osam.raspoloziv())
{
    // DEPART RED8
    entitet* tekuci = red8.izred();
    // TABULATE TRED8
    TRED8(red8.velicina());
    // SEIZE OSAM
    osam.zauzmi(tekuci);
    // ADVANCE 60, FN$EXPO3
    simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_61_14_60, VRTR(rn4.expo(60)));
}
}

void blok_advance_30_10_61_14_60(entitet *e) {
    // RELEASE OSAM
    osam.oslobodi(e);
    if (red8.velicina() > 0)
    {
        entitet* novi = red8.izred();
        // TABULATE TRED8
        TRED8(red8.velicina());
        // SEIZE OSAM
        osam.zauzmi(novi);
        // ADVANCE 60, FN$EXPO3
        simu.rasporedi(novi, BLOK_ADVANCE_30_10_61_14_60,
VRTR(rn4.expo(60)));
    }
    // TABULATE TAB28
    TAB28(M1);
    // TRANSFER .71,,NASD
    // TRANSFER .16,,SUB
    // SOM TERMINATE 1
    // NASD TERMINATE 1
    // SUB TERMINATE 1
    if (rni() < 0.71){
        nsad++;

        UPDATE_TEXTTAG("osam_1", "nsad: " +
to_string(nsad));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
        return;
    }

    if (rni() < 0.16){
        sub++;
    }
}

```

```

        UPDATE_TEXTTAG("osam_2", "sub: " +
to_string(sub));
        DESTROY_ANIM_OBJECT();

        TERMINATE(1);
        return;
    }
    som++;

    UPDATE_TEXTTAG("osam_3", "som: " + to_string(som));
    DESTROY_ANIM_OBJECT();

    TERMINATE(1);
}

void blok_advance_30_10_21(entitet *e) {
    // LOGIC S PRAV1
    LOGIC_S(prav1);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_30_10_21_14, VRTR(14));
}

void blok_advance_30_10_21_14(entitet *e) {
    // LOGIC R PRAV1
    LOGIC_R(prav1);
    // SAVEVALUE KLIZ9+,1
    mem.kliz[9]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[9]);
    ANIM_MESSAGE("Por9", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_30_10_21_14_FN,
VRTR(fn_kli()));
}

void blok_advance_30_10_21_14_FN(entitet *e) {
    // TABULATE TAB1
    TAB1(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_90(entitet *e) {
    // GATE NU DRUGI,BLOK3
    if (drugi.raspoloziv())
    {
        // SEIZE DRUGI
        drugi.zauzmi(e);
        double v = rn3.expo(30);
        // ADVANCE 30,FN$EXPO2
        simu.rasporedi(e, BLOK_ADVANCE_90_30, VRTR(v));
    }
}

```

```

        char obj[100];
        sprintf_s(obj, "%d", e->vrati_id());
        ANIM_TIME(VREME);
        ANIM_DURATION(obj, v);
        ANIM_PLACE_ON(obj, "trans_R_drugi");
    }else{
        // BLOK3 ADVANCE 1432
        simu.rasporedi(e, BLOK_ADVANCE_90_1432,
VRTR(1432));

        char obj[100];
        sprintf_s(obj, "%d", e->vrati_id());
        ANIM_TIME(VREME);
        ANIM_DURATION(obj, 1432.00);
        ANIM_PLACE_ON(obj, "trans_do_treceg");
    }
}

void blok_advance_90_1432(entitet *e) {
    // TRANSFER ,BLOK1
    // BLOK1 ADVANCE 1022
    simu.rasporedi(e, BLOK_ADVANCE_1022, VRTR(1022));

    char obj[100];
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_TIME(VREME);
    ANIM_DURATION(obj, 1022);
    ANIM_PLACE_ON(obj, "trans_do_prvog");
}

void blok_advance_90_30(entitet *e) {
    // RELEASE DRUGI
    drugi.oslobodi(e);
    // ADVANCE 10
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10, VRTR(10));
}

void blok_advance_90_30_10(entitet *e) {
    char obj[100];
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_TIME(VREME);

    // TEST E P1,1,RED9
    if (TEST_E_P1(1)){
        // ADVANCE 171
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_171,
VRTR(171));

        ANIM_DURATION(obj, 171.00);
        ANIM_PLACE_ON(obj, "trans_R2_K1");
    }
    // TEST E P1,2,RED10
    else if (TEST_E_P1(2)){
        // ADVANCE 130
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_130,
VRTR(130));

        ANIM_DURATION(obj, 130.00);
        ANIM_PLACE_ON(obj, "trans_R2_K2");
    }
    // TEST E P1,3,RED11

```

```

else if (TEST_E_P1(3)){
    // ADVANCE 89
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_89,
VRTR(89));

    ANIM_DURATION(obj, 89.00);
    ANIM_PLACE_ON(obj, "trans_R2_K3");
}
// TEST E P1,4,RED12
else if (TEST_E_P1(4)){
    // TRANSFER .30,,TRECA
    if (rni() < 0.30)
    {
        // TRECA ADVANCE 89
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_89,
VRTR(89));

        ANIM_DURATION(obj, 89.00);
        ANIM_PLACE_ON(obj, "trans_R2_K3");
    }
    else
    {
        // ADVANCE 48
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_48,
VRTR(48));

        ANIM_DURATION(obj, 48.00);
        ANIM_PLACE_ON(obj, "trans_R2_K4");
    }
}
// TEST E P1,5,RED13
else if (TEST_E_P1(5)){
    // ADVANCE 193
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_193,
VRTR(193));

    ANIM_DURATION(obj, 193.00);
    ANIM_PLACE_ON(obj, "trans_R2_K5");
}
// TEST E P1,6,RED14
else if (TEST_E_P1(6)){
    // ADVANCE 153
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_153,
VRTR(153));

    ANIM_DURATION(obj, 153.00);
    ANIM_PLACE_ON(obj, "trans_R2_K6");
}
// TEST E P1,7,RED15
else if (TEST_E_P1(7)){
    // ADVANCE 112
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_112,
VRTR(112));

    ANIM_DURATION(obj, 112.00);
    ANIM_PLACE_ON(obj, "trans_R2_K7");
}
// TEST E P1,8,RED16
else if (TEST_E_P1(8)){
    // ADVANCE 72

```

```

        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_72,
VRTR(72));

        ANIM_DURATION(obj, 72.00);
        ANIM_PLACE_ON(obj, "trans_R2_K8");
    }
    // TEST E P1,9
    else if (TEST_E_P1(9)){
        // ADVANCE 31
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_31,
VRTR(31));

        ANIM_DURATION(obj, 31.00);
        ANIM_PLACE_ON(obj, "trans_R2_K9");
    }
}

void blok_advance_90_30_10_171(entitet *e){
    // LOGIC S SKR8
    LOGIC_S(skr8);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_171_14,
VRTR(14));
}

void blok_advance_90_30_10_171_14(entitet *e){
    // LOGIC R SKR8
    LOGIC_R(skr8);
    // SAVEVALUE KLIZ1+,1
    mem.kliz[1]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[1]);
    ANIM_MESSAGE("Por1", msg);

    // ADVANCE FN$KLIZ
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_171_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_171_14_FN(entitet *e){
    // QUEUE RED1
    red1.ured(e);
    // SEIZE JEDAN
    if (jedan.raspoloziv())
    {
        // DEPART RED1
        entitet* tekuci = red1.izred();
        // TABULATE TRED1
        TRED1(red1.velicina());
        // SEIZE JEDAN
        jedan.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        // -----
        // Funkcija se ponavlja, isti dogadjaj!
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_160_14_60, VRTR(rn4.expo(60)));
        // -----
    }
}

```



```

    }
}

void blok_advance_90_30_10_130(entitet *e){
    // LOGIC S SKR6
    LOGIC_S(skr6);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_130_14,
VRTR(14));
}

void blok_advance_90_30_10_130_14(entitet *e){
    // LOGIC R SKR6
    LOGIC_R(skr6);
    // SAVEVALUE KLIZ2+,1
    mem.kliz[2]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[2]);
    ANIM_MESSAGE("Por2", msg);

    // ADVANCE FN$KLIZ
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_130_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_130_14_FN(entitet *e){
    // QUEUE RED2
    red2.ured(e);
    // SEIZE DVA
    if (dva.raspoloziv())
    {
        // DEPART RED2
        entitet* tekuci = red2.izred();
        // TABULATE TRED2
        TRED2(red2.velicina());
        // SEIZE DVA
        dva.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        // -----
        // Funkcija se ponavlja, isti dogadjaj!
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_120_14_60, VRTR(rn4.expo(60)));
        // -----
    }
}

void blok_advance_90_30_10_89(entitet *e){
    // LOGIC S SKR4
    LOGIC_S(skr4);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_89_14,
VRTR(14));
}

void blok_advance_90_30_10_89_14(entitet *e){
    // LOGIC R SKR4

```

```

    LOGIC_R(skr4);
    // SAVEVALUE KLIZ3+,1
    mem.kliz[3]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[3]);
    ANIM_MESSAGE("Por3", msg);

    // ADVANCE FN$KLIZ
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_89_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_89_14_FN(entitet *e){
    // QUEUE RED3
    red3.ured(e);
    // SEIZE TRI
    if (tri.raspoloziv())
    {
        // DEPART RED3
        entitet* tekuci = red3.izred();
        // TABULATE TRED3
        TRED3(red3.velicina());
        // SEIZE TRI
        tri.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        // -----
        // Funkcija se ponavlja, isti dogadjaj!
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_80_14_60, VRTR(rn4.expo(60)));
        // -----
    }
}

void blok_advance_90_30_10_48(entitet *e){
    // LOGIC S SKR2
    LOGIC_S(skr2);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_48_14,
VRTR(14));
}

void blok_advance_90_30_10_48_14(entitet *e){
    // LOGIC R SKR2
    LOGIC_R(skr2);
    // SAVEVALUE KLIZ4+,1
    mem.kliz[4]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[4]);
    ANIM_MESSAGE("Por4", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_48_14_FN,
VRTR(fn_kli()));
}

```

```

void blok_advance_90_30_10_48_14_FN(entitet *e){
    // TABULATE TAB2
    TAB2(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_90_30_10_193(entitet *e){
    // LOGIC S SKR9
    LOGIC_S(skr9);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_193_14,
VRTR(14));
}

void blok_advance_90_30_10_193_14(entitet *e){
    // LOGIC R SKR9
    LOGIC_R(skr9);
    // SAVEVALUE KLIZ5+,1
    mem.kliz[5]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[5]);
    ANIM_MESSAGE("Por5", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_193_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_193_14_FN(entitet *e){
    // TABULATE TAB2
    TAB2(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_90_30_10_153(entitet *e){
    // LOGIC S SKR7
    LOGIC_S(skr7);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_153_14,
VRTR(14));
}

void blok_advance_90_30_10_153_14(entitet *e){
    // LOGIC R SKR7

```

```

    LOGIC_R(skr7);
    // SAVEVALUE KLIZ6+,1
    mem.kliz[6]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[6]);
    ANIM_MESSAGE("Por6", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_153_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_153_14_FN(entitet *e){
    // TABULATE TAB2
    TAB2(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void blok_advance_90_30_10_112(entitet *e){
    // LOGIC S SKR5
    LOGIC_S(skr5);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_112_14,
VRTR(14));
}

void blok_advance_90_30_10_112_14(entitet *e){
    // LOGIC R SKR5
    LOGIC_R(skr5);
    // SAVEVALUE KLIZ7+,1
    mem.kliz[7]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[7]);
    ANIM_MESSAGE("Por7", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_112_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_112_14_FN(entitet *e){
    // QUEUE RED7
    red7.ured(e);
    // SEIZE SEDAM
    if (sedam.raspoloziv())
    {
        // DEPART RED7
        entitet* tekuci = red7.izred();
        // TABULATE TRED7
        TRED7(red7.velicina());
    }
}

```

```

        // SEIZE SEDAM
        sedam.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        // -----
        // Funkcija se ponavlja, isti dogadjaj!
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_101_14_60, VRTR(rn4.expo(60)));
        // -----
    }
}

void blok_advance_90_30_10_72(entitet *e){
    // LOGIC S SKR3
    LOGIC_S(skr3);
    // ADVANCE 14
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_72_14,
VRTR(14));
}

void blok_advance_90_30_10_72_14(entitet *e){
    // LOGIC R SKR3
    LOGIC_R(skr3);
    // SAVEVALUE KLIZ8+,1
    mem.kliz[8]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[8]);
    ANIM_MESSAGE("Por8", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_72_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_72_14_FN(entitet *e){
    // QUEUE RED8
    red8.ured(e);
    // SEIZE OSAM
    if (osam.raspoloziv())
    {
        // DEPART RED8
        entitet* tekuci = red8.izred();
        // TABULATE TRED8
        TRED8(red2.velicina());
        // SEIZE OSAM
        osam.zauzmi(tekuci);
        // ADVANCE 60, FN$EXPO3
        // -----
        // Funkcija se ponavlja, isti dogadjaj!
        simu.rasporedi(tekuci,
BLOK_ADVANCE_30_10_61_14_60, VRTR(rn4.expo(60)));
        // -----
    }
}

void blok_advance_90_30_10_31(entitet *e){

```

```

        // LOGIC S SKR1
        LOGIC_S(skr1);
        // ADVANCE 14
        simu.rasporedi(e, BLOK_ADVANCE_90_30_10_31_14,
VRTR(14));
    }

void blok_advance_90_30_10_31_14(entitet *e){
    // LOGIC R SKR1
    LOGIC_R(skr1);
    // SAVEVALUE KLIZ9+,1
    mem.kliz[9]++;

    char msg[10];
    ANIM_TIME(VREME);
    sprintf_s(msg, "%d", mem.kliz[9]);
    ANIM_MESSAGE("Por9", msg);

    // ADVANCE FN$KLI
    simu.rasporedi(e, BLOK_ADVANCE_90_30_10_31_14_FN,
VRTR(fn_kli()));
}

void blok_advance_90_30_10_31_14_FN(entitet *e){
    // TABULATE TAB2
    TAB2(M1);

    char obj[100];
    ANIM_TIME(VREME);
    sprintf_s(obj, "%d", e->vrati_id());
    ANIM_DESTROY(obj);

    // TERMINATE 1
    TERMINATE(1);
}

void print_statistike_reda_cekanja(const red_cekanja& red,
const char* opis) {
    statistike stat = red.vrati_statistike();
    // Statistike reda cekanja
    cout << "STATISTIKE REDA CEKANJA - " << opis << endl;
    cout << "Broj entiteta koji je usao u red cekanja: " <<
stat.vrati_broj() << endl;
    cout << "Preostali broj entiteta u redu cekanja: " <<
stat.vrati_tekuci_broj() << endl;
    cout << "Maksimalni broj entiteta u redu: " <<
stat.vrati_maksimalni_broj() << endl;
    cout << "Srednje vreme cekanja u redu: " <<
stat.vrati_srednje_vreme() << endl;
    cout << "Srednji broj entiteta u redu: " <<
stat.vrati_srednji_broj(simu.vrati_vreme_simulacije()) <<
endl;
    cout << "Broj entiteta koji je prosao kroz red bez
zadrzavanja: " << stat.vrati_broj_bez_cekanja() << endl;
    cout << "Procenat ulaza bez zadrzavanja: " <<
stat.vrati_ucesce_bez_cekanja() << endl;
    cout << "Srednje vreme cekanja u redu (iskljuc. ent.
koji se nisu zadrzali): " <<
stat.vrati_srednje_vreme_bez_cekanja() << endl;
    cout << endl;
}

```

```

void print_statistike_resursa(const resurs& salt, const char*
opis) {
    statistike stat = salt.vrati_statistike();
    // Statistike resursa
    cout << "STATISTIKE RESURSA - " << opis << endl;
    cout << "Broj entiteta koji je usao u resurs: " <<
stat.vrati_broj() << endl;
    cout << "Preostali broj entiteta u resursu: " <<
stat.vrati_tekuci_broj() << endl;
    cout << "Maksimalni broj zauzetih kanala opsluge: " <<
stat.vrati_maksimalni_broj() << endl;
    cout << "Srednje vreme opsluge: " <<
stat.vrati_srednje_vreme() << endl;
    cout << "Srednji broj zauzetih kanala: " <<
stat.vrati_srednji_broj(simu.vrati_vreme_simulacije()) <<
endl;
    cout << "Iskoriscenost: " <<
stat.vrati_iskoriscenost(simu.vrati_vreme_simulacije()),
salt.vrati_broj_mesta() << endl;
    cout << endl;
}

void print_histogram(histogram& histo, const char* opis){
    // Statistike histograma
    cout << "HISTOGRAM - " << opis << endl;
    cout << "Broj ulaza u histogram: " <<
histo.vrati_broj_ulaza_u_histogram() << endl;
    if
(histo.vrati_frekvenciju_intervala(histo.vrati_broj_intervala
() - 1) > 0){
        cout << "Prosecna vrednost overflowa: " <<
histo.vrati_prosecnu_vrednost_overflowa() << endl;
        cout << "Frekvencija overflowa: " <<
histo.vrati_frekvenciju_intervala(histo.vrati_broj_intervala(
) - 1) << endl;
        cout << "Verovatnoca overflowa: " <<
histo.vrati_verovatnocu_intervala(histo.vrati_broj_intervala(
) - 1) << endl;
    }
    cout << "Srednja vrednost histograma: " <<
histo.vrati_srednju_vrednost_histograma() << endl;
    cout << "Standardna devijacija histograma: " <<
histo.vrati_standardnu_devijaciju_histograma() << endl;
    cout << "Suma argumenata histograma: " <<
histo.vrati_sumu_argumenata_histograma() << endl;
    cout << setw(10) << "int    " << setw(10) << "frek    " <<
setw(10) << "vero    " << setw(10) << "umno    " << setw(10) <<
"dev    " << endl;
    for (int i = 0; i < histo.vrati_broj_intervala() - 1;
i++){
        cout << setw(10) <<
histo.vrati_gornju_granicu_intervala(i) << " "
        << setw(10) <<
histo.vrati_frekvenciju_intervala(i) << " "
        << setw(10) <<
histo.vrati_verovatnocu_intervala(i) << " "
        << setw(10) <<
histo.vrati_umnozak_od_srednje_vrednosti_intervala(i) << " "

```

```

        << setw(10) <<
histo.vrati_devijaciju_od_srednje_vrednosti_intervala(i) <<
endl;
    }
    cout << endl;
}

void print_statistike_kolica(){
    // Statistike kolica
    cout << "STATISTIKE KOLICA" << endl;

    cout << "Sombor: " << som << endl;
    cout << "Novi Sad: " << nsad << endl;
    cout << "Subotica: " << sub << endl;

    cout << "Kragujevac: " << krag << endl;
    cout << "Kraljevo: " << kvo << endl;
    cout << "Uzice: " << uzi << endl;

    cout << "Mladenovac: " << mlad << endl;
    cout << "Smederevska Palanka: " << spal << endl;
    cout << "Velika Plana: " << vpla << endl;

    cout << "Sremska Mitrovica: " << smit << endl;
    cout << "Pancevo: " << pan << endl;

    cout << "Pozarevac: " << poz << endl;
    cout << "Valjevo: " << valj << endl;
    cout << "Sabac: " << sab << endl;

    cout << endl;
}

int main() {
    // Zapocinjemo animaciju
    ANIM_START("Anim/", 820, 620);
    // ANIM_TIME(0);

    // Inicijalizacija brojaca paketa
    char obj[100];

    // -----
    strcpy_s(obj, "header1");
    ANIM_TIME(VREME);
    ANIM_CREATE("Poruka", obj);
    ANIM_MESSAGE(obj, "Kristian Iker");
    ANIM_PLACE_AT(obj, 550, 30);
    strcpy_s(obj, "header2");
    ANIM_TIME(VREME);
    ANIM_CREATE("Poruka", obj);
    ANIM_MESSAGE(obj, "Saobracajni fakultet");
    ANIM_PLACE_AT(obj, 550, 50);
    strcpy_s(obj, "header3");
    ANIM_TIME(VREME);
    ANIM_CREATE("Poruka", obj);
    ANIM_MESSAGE(obj, "ZAVRSNI RAD");
    ANIM_PLACE_AT(obj, 550, 75);
    strcpy_s(obj, "header4");
    ANIM_TIME(VREME);
    ANIM_CREATE("Poruka", obj);
    ANIM_MESSAGE(obj, "SZRP 11200 BG");
}

```



```

ANIM_PLACE_AT(obj, 550, 95);
// -----

CREATE_TEXTTAG("Por5", mem.kliz[5], 595, 190);
CREATE_TEXTTAG("Por1", mem.kliz[1], 180, 230);
CREATE_TEXTTAG("Por6", mem.kliz[7], 595, 270);
CREATE_TEXTTAG("Por2", mem.kliz[2], 180, 310);
CREATE_TEXTTAG("Por7", mem.kliz[7], 595, 350);
CREATE_TEXTTAG("Por3", mem.kliz[3], 180, 390);
CREATE_TEXTTAG("Por8", mem.kliz[8], 595, 430);
CREATE_TEXTTAG("Por4", mem.kliz[4], 180, 470);
CREATE_TEXTTAG("Por9", mem.kliz[9], 595, 510);

CREATE_TEXTTAG("osam_1", "nsad: " + to_string(nsad),
675, 430 - 20);
CREATE_TEXTTAG("osam_2", "sub: " + to_string(sub), 675,
430);
CREATE_TEXTTAG("osam_3", "som: " + to_string(som), 675,
430 + 20);

CREATE_TEXTTAG("sedam_1", "kvo: " + to_string(kvo), 675,
350 - 20);
CREATE_TEXTTAG("sedam_2", "krag: " + to_string(krag),
675, 350);
CREATE_TEXTTAG("sedam_3", "uzi: " + to_string(uzi), 675,
350 + 20);

CREATE_TEXTTAG("tri_1", "mlad: " + to_string(mlad), 100,
390 - 20);
CREATE_TEXTTAG("tri_2", "spal: " + to_string(spal), 100,
390);
CREATE_TEXTTAG("tri_3", "vpla: " + to_string(vpla), 100,
390 + 20);

CREATE_TEXTTAG("dva_1", "smit: " + to_string(smit), 100,
310 - 10);
CREATE_TEXTTAG("dva_2", "pan: " + to_string(pan), 100,
310 + 10);

CREATE_TEXTTAG("jedan_1", "poz: " + to_string(poz), 100,
230 - 20);
CREATE_TEXTTAG("jedan_2", "sab: " + to_string(sab), 100,
230);
CREATE_TEXTTAG("jedan_3", "valj: " + to_string(valj),
100, 230 + 20);

// Rasporedjivanje prvog dogadjaja
simu.rasporedi(simu.napravi_entitet(), BLOK_GENERATE,
VRTR(20 + rn1.expo(10)));
// Izvrsavamo simulaciju za prvih 1000 paketa u sistemu
simu.izvrsi(1000);

// Završavamo animaciju
ANIM_TIME(VREME);
ANIM_END;

// Finisiramo statistike
red.finisiraj();

red1.finisiraj();
red2.finisiraj();

```

```

red3.finisiraj();
red7.finisiraj();
red8.finisiraj();

prvi.finisiraj();
drugi.finisiraj();

jedan.finisiraj();
dva.finisiraj();
tri.finisiraj();
sedam.finisiraj();
osam.finisiraj();

// Stampanje statistika
print_statistike_reda_cekanja(red, "red");

print_statistike_reda_cekanja(red1, "red1");
print_statistike_reda_cekanja(red2, "red2");
print_statistike_reda_cekanja(red3, "red3");
print_statistike_reda_cekanja(red7, "red7");
print_statistike_reda_cekanja(red8, "red8");

print_statistike_resursa(prvi, "prvi");
print_statistike_resursa(drugi, "drugi");

print_statistike_resursa(jedan, "jedan");
print_statistike_resursa(dva, "dva");
print_statistike_resursa(tri, "tri");
print_statistike_resursa(sedam, "sedam");
print_statistike_resursa(osam, "osam");

print_statistike_kolica();

print_histogram(TRED, "TRED");
print_histogram(TAB1, "TAB1");
print_histogram(TAB2, "TAB2");
print_histogram(TRED1, "TRED1");
print_histogram(TRED2, "TRED2");
print_histogram(TRED3, "TRED3");
print_histogram(TRED7, "TRED7");
print_histogram(TRED8, "TRED8");
print_histogram(TAB21, "TAB21");
print_histogram(TAB22, "TAB22");
print_histogram(TAB23, "TAB23");
print_histogram(TAB27, "TAB27");
print_histogram(TAB28, "TAB28");

mem.print();

system("pause");
}

```

## Statistike izvršene simulacije u C++ jeziku

### STATISTIKE REDA CEKANJA - red

Broj entiteta koji je usao u red cekanja: 1000  
Preostali broj entiteta u redu cekanja: 0  
Maksimalni broj entiteta u redu: 29  
Srednje vreme cekanja u redu: 728  
Srednji broj entiteta u redu: 21.1136  
Broj entiteta koji je prosao kroz red bez zadrzavanja: 0  
Procenat ulaza bez zadrzavanja: 0  
Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 728

### STATISTIKE REDA CEKANJA - red1

Broj entiteta koji je usao u red cekanja: 115  
Preostali broj entiteta u redu cekanja: 0  
Maksimalni broj entiteta u redu: 3  
Srednje vreme cekanja u redu: 18.9114  
Srednji broj entiteta u redu: 0.0630742  
Broj entiteta koji je prosao kroz red bez zadrzavanja: 89  
Procenat ulaza bez zadrzavanja: 77.3913  
Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 83.6464

### STATISTIKE REDA CEKANJA - red2

Broj entiteta koji je usao u red cekanja: 117  
Preostali broj entiteta u redu cekanja: 0  
Maksimalni broj entiteta u redu: 3  
Srednje vreme cekanja u redu: 16.032  
Srednji broj entiteta u redu: 0.0544008  
Broj entiteta koji je prosao kroz red bez zadrzavanja: 91  
Procenat ulaza bez zadrzavanja: 77.7778  
Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 72.144

### STATISTIKE REDA CEKANJA - red3

Broj entiteta koji je usao u red cekanja: 100  
Preostali broj entiteta u redu cekanja: 0  
Maksimalni broj entiteta u redu: 2  
Srednje vreme cekanja u redu: 8.25705  
Srednji broj entiteta u redu: 0.0239473  
Broj entiteta koji je prosao kroz red bez zadrzavanja: 85  
Procenat ulaza bez zadrzavanja: 85  
Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 55.047

### STATISTIKE REDA CEKANJA - red7

Broj entiteta koji je usao u red cekanja: 152  
Preostali broj entiteta u redu cekanja: 0  
Maksimalni broj entiteta u redu: 4  
Srednje vreme cekanja u redu: 36.1025  
Srednji broj entiteta u redu: 0.159152  
Broj entiteta koji je prosao kroz red bez zadrzavanja: 105  
Procenat ulaza bez zadrzavanja: 69.0789

Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadržali): 116.757

STATISTIKE REDA CEKANJA - red8

Broj entiteta koji je usao u red cekanja: 146

Preostali broj entiteta u redu cekanja: 0

Maksimalni broj entiteta u redu: 5

Srednje vreme cekanja u redu: 35.345

Srednji broj entiteta u redu: 0.149662

Broj entiteta koji je prosao kroz red bez zadržavanja: 103

Procenat ulaza bez zadržavanja: 70.5479

Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadržali): 120.008

STATISTIKE RESURSA - prvi

Broj entiteta koji je usao u resurs: 626

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 31.0086

Srednji broj zauzetih kanala: 0.562974

Iskoriscenost: 0.562974

STATISTIKE RESURSA - drugi

Broj entiteta koji je usao u resurs: 374

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 27.4391

Srednji broj zauzetih kanala: 0.297628

Iskoriscenost: 0.297628

STATISTIKE RESURSA - jedan

Broj entiteta koji je usao u resurs: 115

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 66.8359

Srednji broj zauzetih kanala: 0.222915

Iskoriscenost: 0.222915

STATISTIKE RESURSA - dva

Broj entiteta koji je usao u resurs: 117

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 60.1513

Srednji broj zauzetih kanala: 0.204109

Iskoriscenost: 0.204109

STATISTIKE RESURSA - tri

Broj entiteta koji je usao u resurs: 100

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 53.5096

Srednji broj zauzetih kanala: 0.15519

Iskoriscenost: 0.15519

#### STATISTIKE RESURSA - sedam

Broj entiteta koji je usao u resurs: 152

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 63.6658

Srednji broj zauzetih kanala: 0.280661

Iskoriscenost: 0.280661

#### STATISTIKE RESURSA - osam

Broj entiteta koji je usao u resurs: 146

Preostali broj entiteta u resursu: 0

Maksimalni broj zauzetih kanala opsluge: 1

Srednje vreme opsluge: 66.6693

Srednji broj zauzetih kanala: 0.2823

Iskoriscenost: 0.2823

#### STATISTIKE KOLICA

Sombor: 103

Novi Sad: 8

Subotica: 35

Kragujevac: 65

Kraljevo: 46

Uzice: 41

Mladenovac: 29

Smederevska Palanka: 30

Velika Plana: 41

Sremska Mitrovica: 75

Pancevo: 42

Pozarevac: 37

Valjevo: 55

Sabac: 23

#### HISTOGRAM - TRED

Broj ulaza u histogram: 1000

Srednja vrednost histograma: 23.573

Standardna devijacija histograma: 2.86581

Suma argumenata histograma: 23573

int	frek	vero	umno	dev
0	1	0.001	0	-8.22559
1	1	0.001	0.0424214	-7.87665
2	1	0.001	0.0848428	-7.52771
3	1	0.001	0.127264	-7.17877
4	1	0.001	0.169686	-6.82982
5	1	0.001	0.212107	-6.48088
6	1	0.001	0.254528	-6.13194
7	1	0.001	0.29695	-5.783
8	1	0.001	0.339371	-5.43406
9	1	0.001	0.381793	-5.08512

10	1	0.001	0.424214	-4.73618
11	1	0.001	0.466636	-4.38724
12	1	0.001	0.509057	-4.0383
13	1	0.001	0.551478	-3.68935
14	1	0.001	0.5939	-3.34041
15	1	0.001	0.636321	-2.99147
16	1	0.001	0.678743	-2.64253
17	2	0.002	0.721164	-2.29359
18	7	0.007	0.763585	-1.94465
19	11	0.011	0.806007	-1.59571
20	35	0.035	0.848428	-1.24677
21	60	0.06	0.89085	-0.897826
22	103	0.103	0.933271	-0.548884
23	159	0.159	0.975693	-0.199943
24	218	0.218	1.01811	0.148998
25	216	0.216	1.06054	0.497939
26	102	0.102	1.10296	0.84688
27	59	0.059	1.14538	1.19582
28	11	0.011	1.1878	1.54476
29	0	0	1.23022	1.8937
30	0	0	1.27264	2.24264
31	0	0	1.31506	2.59159
32	0	0	1.35749	2.94053
33	0	0	1.39991	3.28947
34	0	0	1.44233	3.63841
35	0	0	1.48475	3.98735
36	0	0	1.52717	4.33629
37	0	0	1.56959	4.68523
38	0	0	1.61201	5.03417
39	0	0	1.65444	5.38311
40	0	0	1.69686	5.73206
41	0	0	1.73928	6.081
42	0	0	1.7817	6.42994
43	0	0	1.82412	6.77888
44	0	0	1.86654	7.12782

#### HISTOGRAM - TAB1

Broj ulaza u histogram: 231

Prosečna vrednost overflowa: -0.0528824

Frekvencija overflowa: 71

Verovatnoca overflowa: 0.307359

Srednja vrednost histograma: 15138

Standardna devijacija histograma: 8301.01

Suma argumenata histograma: 3.49687e+006

int	frek	vero	umno	dev
1000	3	0.012987	0.066059	-1.70316
2000	9	0.038961	0.132118	-1.5827
3000	7	0.030303	0.198177	-1.46223
4000	9	0.038961	0.264236	-1.34176
5000	5	0.021645	0.330295	-1.22129

6000	8	0.034632	0.396354	-1.10083
7000	7	0.030303	0.462413	-0.98036
8000	7	0.030303	0.528472	-0.859892
9000	6	0.025974	0.594531	-0.739425
10000	9	0.038961	0.66059	-0.618958
11000	12	0.0519481	0.726649	-0.498491
12000	9	0.038961	0.792708	-0.378024
13000	8	0.034632	0.858767	-0.257556
14000	4	0.017316	0.924826	-0.137089
15000	8	0.034632	0.990885	-0.0166218
16000	13	0.0562771	1.05694	0.103845
17000	11	0.047619	1.123	0.224313
18000	8	0.034632	1.18906	0.34478
19000	8	0.034632	1.25512	0.465247
20000	9	0.038961	1.32118	0.585714

#### HISTOGRAM - TAB2

Broj ulaza u histogram: 139

Prosečna vrednost overflowa: -0.0405271

Frekvencija overflowa: 57

Verovatnoca overflowa: 0.410072

Srednja vrednost histograma: 15917.9

Standardna devijacija histograma: 8142.42

Suma argumenata histograma: 2.21259e+006

int	frek	vero	umno	dev
1000	4	0.028777	0.0628223	-1.83212
2000	3	0.0215827	0.125645	-1.70931
3000	4	0.028777	0.188467	-1.58649
4000	1	0.00719424	0.251289	-1.46368
5000	6	0.0431655	0.314112	-1.34087
6000	4	0.028777	0.376934	-1.21805
7000	2	0.0143885	0.439756	-1.09524
8000	4	0.028777	0.502579	-0.972426
9000	5	0.0359712	0.565401	-0.849613
10000	6	0.0431655	0.628223	-0.726799
11000	4	0.028777	0.691046	-0.603986
12000	7	0.0503597	0.753868	-0.481172
13000	3	0.0215827	0.81669	-0.358358
14000	4	0.028777	0.879513	-0.235545
15000	6	0.0431655	0.942335	-0.112731
16000	4	0.028777	1.00516	0.0100822
17000	3	0.0215827	1.06798	0.132896
18000	3	0.0215827	1.1308	0.255709
19000	6	0.0431655	1.19362	0.378523
20000	3	0.0215827	1.25645	0.501336

#### HISTOGRAM - TRED1

Broj ulaza u histogram: 115

Srednja vrednost histograma: 0.0782609

Standardna devijacija histograma: 0.328415

Suma argumenata histograma: 9

int	frek	vero	umno	dev
0	108	0.93913	0	-0.238299
1	5	0.0434783	12.7778	2.80663
2	2	0.0173913	25.5556	5.85155
3	0	0	38.3333	8.89648
4	0	0	51.1111	11.9414

#### HISTOGRAM - TRED2

Broj ulaza u histogram: 117

Srednja vrednost histograma: 0.0598291

Standardna devijacija histograma: 0.271985

Suma argumenata histograma: 7

int	frek	vero	umno	dev
0	111	0.948718	0	-0.219972
1	5	0.042735	16.7143	3.4567
2	1	0.00854701	33.4286	7.13338
3	0	0	50.1429	10.81
4	0	0	66.8571	14.4867

#### HISTOGRAM - TRED3

Broj ulaza u histogram: 100

Srednja vrednost histograma: 0.02

Standardna devijacija histograma: 0.140705

Suma argumenata histograma: 2

int	frek	vero	umno	dev
0	98	0.98	0	-0.142141
1	2	0.02	50	6.96491
2	0	0	100	14.072
3	0	0	150	21.179
4	0	0	200	28.2861

#### HISTOGRAM - TRED7

Broj ulaza u histogram: 152

Srednja vrednost histograma: 0.151316

Standardna devijacija histograma: 0.498473

Suma argumenata histograma: 23

int	frek	vero	umno	dev
0	137	0.901316	0	-0.303559
1	8	0.0526316	6.6087	1.70257
2	6	0.0394737	13.2174	3.7087
3	1	0.00657895	19.8261	5.71482
4	0	0	26.4348	7.72095

#### HISTOGRAM - TRED8

Broj ulaza u histogram: 146

Srednja vrednost histograma: 0.191781

Standardna devijacija histograma: 0.57912

Suma argumenata histograma: 28

int	frek	vero	umno	dev
-----	------	------	------	-----



0	127	0.869863	0	-0.331159
1	13	0.0890411	5.21429	1.3956
2	4	0.0273973	10.4286	3.12236
3	1	0.00684932	15.6429	4.84911
4	1	0.00684932	20.8571	6.57587

#### HISTOGRAM - TAB21

Broj ulaza u histogram: 115

Prosečna vrednost overflowa: -0.0254306

Frekvencija overflowa: 33

Verovatnoca overflowa: 0.286957

Srednja vrednost histograma: 14025.1

Standardna devijacija histograma: 9039.71

Suma argumenata histograma: 1.61288e+006

int	frek	vero	umno	dev
1000	5	0.0434783	0.0713009	-1.44087
2000	3	0.026087	0.142602	-1.33025
3000	4	0.0347826	0.213903	-1.21963
4000	5	0.0434783	0.285204	-1.109
5000	5	0.0434783	0.356504	-0.99838
6000	5	0.0434783	0.427805	-0.887757
7000	5	0.0434783	0.499106	-0.777134
8000	4	0.0347826	0.570407	-0.666511
9000	7	0.0608696	0.641708	-0.555888
10000	4	0.0347826	0.713009	-0.445265
11000	3	0.026087	0.78431	-0.334642
12000	3	0.026087	0.855611	-0.224019
13000	5	0.0434783	0.926912	-0.113396
14000	5	0.0434783	0.998213	-0.00277322
15000	4	0.0347826	1.06951	0.10785
16000	5	0.0434783	1.14081	0.218473
17000	2	0.0173913	1.21212	0.329096
18000	2	0.0173913	1.28342	0.439719
19000	2	0.0173913	1.35472	0.550342
20000	4	0.0347826	1.42602	0.660965

#### HISTOGRAM - TAB22

Broj ulaza u histogram: 117

Prosečna vrednost overflowa: -0.027149

Frekvencija overflowa: 37

Verovatnoca overflowa: 0.316239

Srednja vrednost histograma: 13947.9

Standardna devijacija histograma: 8745.09

Suma argumenata histograma: 1.63191e+006

int	frek	vero	umno	dev
1000	5	0.042735	0.0716952	-1.4806
2000	1	0.00854701	0.14339	-1.36625
3000	5	0.042735	0.215086	-1.2519
4000	5	0.042735	0.286781	-1.13755
5000	8	0.0683761	0.358476	-1.0232

6000	2	0.017094	0.430171	-0.908846
7000	7	0.0598291	0.501866	-0.794496
8000	4	0.034188	0.573562	-0.680146
9000	7	0.0598291	0.645257	-0.565796
10000	5	0.042735	0.716952	-0.451446
11000	4	0.034188	0.788647	-0.337096
12000	4	0.034188	0.860342	-0.222747
13000	3	0.025641	0.932037	-0.108397
14000	2	0.017094	1.00373	0.00595335
15000	2	0.017094	1.07543	0.120303
16000	5	0.042735	1.14712	0.234653
17000	3	0.025641	1.21882	0.349003
18000	3	0.025641	1.29051	0.463353
19000	2	0.017094	1.36221	0.577703
20000	3	0.025641	1.4339	0.692053

#### HISTOGRAM - TAB23

Broj ulaza u histogram: 100

Prosečna vrednost overflowa: -0.0218544

Frekvencija overflowa: 29

Verovatnoca overflowa: 0.29

Srednja vrednost histograma: 14902.3

Standardna devijacija histograma: 8546.7

Suma argumenata histograma: 1.49023e+006

int	frek	vero	umno	dev
1000	3	0.03	0.0671036	-1.62663
2000	3	0.03	0.134207	-1.50963
3000	2	0.02	0.201311	-1.39262
4000	4	0.04	0.268414	-1.27562
5000	3	0.03	0.335518	-1.15861
6000	5	0.05	0.402622	-1.04161
7000	6	0.06	0.469725	-0.924606
8000	2	0.02	0.536829	-0.807602
9000	2	0.02	0.603933	-0.690597
10000	1	0.01	0.671036	-0.573593
11000	4	0.04	0.73814	-0.456589
12000	3	0.03	0.805243	-0.339584
13000	3	0.03	0.872347	-0.22258
14000	6	0.06	0.939451	-0.105576
15000	3	0.03	1.00655	0.0114283
16000	1	0.01	1.07366	0.128433
17000	7	0.07	1.14076	0.245437
18000	4	0.04	1.20787	0.362441
19000	4	0.04	1.27497	0.479445
20000	5	0.05	1.34207	0.59645

#### HISTOGRAM - TAB27

Broj ulaza u histogram: 152

Prosečna vrednost overflowa: -0.0422993

Frekvencija overflowa: 56

Verovatnoca overflowa: 0.368421  
 Srednja vrednost histograma: 15063.2  
 Standardna devijacija histograma: 9550.85  
 Suma argumenata histograma: 2.2896e+006

int	frek	vero	umno	dev
1000	8	0.0526316	0.0663872	-1.47245
2000	8	0.0526316	0.132774	-1.36775
3000	8	0.0526316	0.199162	-1.26304
4000	4	0.0263158	0.265549	-1.15834
5000	3	0.0197368	0.331936	-1.05364
6000	3	0.0197368	0.398323	-0.948936
7000	7	0.0460526	0.46471	-0.844234
8000	6	0.0394737	0.531097	-0.739531
9000	6	0.0394737	0.597485	-0.634828
10000	8	0.0526316	0.663872	-0.530126
11000	2	0.0131579	0.730259	-0.425423
12000	5	0.0328947	0.796646	-0.32072
13000	1	0.00657895	0.863033	-0.216017
14000	1	0.00657895	0.92942	-0.111315
15000	1	0.00657895	0.995808	-0.00661201
16000	1	0.00657895	1.06219	0.0980907
17000	5	0.0328947	1.12858	0.202793
18000	6	0.0394737	1.19497	0.307496
19000	9	0.0592105	1.26136	0.412199
20000	4	0.0263158	1.32774	0.516902

#### HISTOGRAM - TAB28

Broj ulaza u histogram: 146  
 Prosečna vrednost overflowa: -0.036796  
 Frekvencija overflowa: 49  
 Verovatnoca overflowa: 0.335616  
 Srednja vrednost histograma: 15257.6  
 Standardna devijacija histograma: 8758.58  
 Suma argumenata histograma: 2.22761e+006

int	frek	vero	umno	dev
1000	3	0.0205479	0.065541	-1.62785
2000	7	0.0479452	0.131082	-1.51367
3000	5	0.0342466	0.196623	-1.3995
4000	7	0.0479452	0.262164	-1.28533
5000	5	0.0342466	0.327705	-1.17115
6000	6	0.0410959	0.393246	-1.05698
7000	0	0	0.458787	-0.942804
8000	4	0.0273973	0.524328	-0.82863
9000	4	0.0273973	0.589869	-0.714457
10000	4	0.0273973	0.65541	-0.600283
11000	4	0.0273973	0.720951	-0.486109
12000	3	0.0205479	0.786492	-0.371936
13000	5	0.0342466	0.852033	-0.257762
14000	6	0.0410959	0.917574	-0.143588
15000	12	0.0821918	0.983115	-0.0294143

16000	5	0.0342466	1.04866	0.0847594
17000	3	0.0205479	1.1142	0.198933
18000	5	0.0342466	1.17974	0.313107
19000	4	0.0273973	1.24528	0.427281
20000	5	0.0342466	1.31082	0.541454

Sadržaj memorijskog objekata:

kliz[1]: 115

kliz[2]: 117

kliz[3]: 100

kliz[4]: 184

kliz[5]: 63

kliz[6]: 123

kliz[7]: 152

kliz[8]: 146

kliz[9]: 0 <- medjunarodni saobracaj ka RS

posl: 30784.5

Press any key to continue . . .