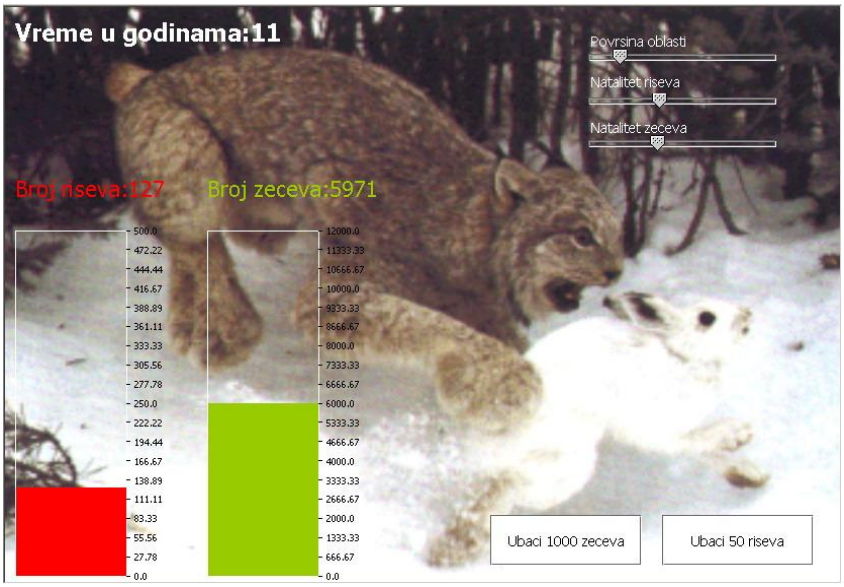
Teaching Mathematics and Statistics in Sciences, Modeling and Computer-Aided Approach

IPA HU-SRB/0901/221/088

Matematički i simulacioni modeli u programu AnyLogic

Arpad Takači, Dušan Mijatović



Departman za matematiku i informatiku,

Prirodno-matematički fakultet,

Univerzitet u Novom Sadu

2011

Sadržaj

Uvod

Nastavni material (engl. *teaching material*) za IPA Projekat HU-SRB/0901/221/088, je važan deo našeg rada na Projektu. U ovom delu nastavnog materijala, korišćen je odličan ruski program za modeliranje *AnyLogic*. Modeli su izabrani tako da pokažu razne aspekte modeliranja, te su u tom cilju predstavljeni kontinualni, diskretni i hibridni modeli.

U prilogu je dat i jedan broj apleta, koji, korišćenjem klizača, dinamički pokazuju promene modela u vremenu.

Autori obrađuju sledeće matematičke modele:

1. Lorencov model vremena
2. Kermack-McKendrick model (SIR model)
3. Matematičko klatno
4. Odskakanje lopte
5. Modeli Lotka-Voltera
6. Model katapultiranja pilota
7. Univerzalan, jednostavan model masovnog opsluživanja

1. Lorencov model vremena

Lorencov matematički model vremena napravljen 1963 godine opisuje kretanje vazduha kroz atmosferu. Konkretno, Lorenc je posmatrao je zagrevanje vazduha koji se podiže u atomosferi, hladi se i pada. Posmatrenjem vremenskih prilika Lorenc je primetio da se vreme ne ponaša uvek na predvidljiv način. U pokušaju da napravi matematički model obrazaca po kojima se vremenske prilike dešavaju, otkrio je da je to u stvari *haotično ponašanje*. Model vremena je nelinearni sistem diferencijalnih jednačina od tri promenljive.

U početku, Lorenc je proučavao sistem od 12 jednačina i primetio je da male promene početnih vrednost promenljivih izazivaju velike promene u ponašanju modela. Ova osetljivost na početne uslove u modelu danas je poznata kao *efekat leptira* (enlg. *Butterfly effect*). Zaključak koji se nameće iz ovog je da su sve prognoze vremena za više od nedelju dana generalno netačne.

Tehnički posmatrano, Lorencov oscilator je nelinearan, trodimenzionalan i deterministički. Dodajmo da se i danas ovaj sistem koristi za demonstraciju sistema sa haotičnim ponašanjem. U nastavku dajemo Lorencov sistem ODJ po nepoznatim funkcijama x, y i z, koje zavise od vremena t.

$$\frac{dx}{dt} = ay - ax$$
$$\frac{dy}{dt} = rx - y - xz \qquad (1.1)$$
$$\frac{dz}{dt} = xy + bz$$

Fizičko tumačenje nepoznatih dajemo u nastavku:

- *x* predstavlja proporcionalnu brzinu kretanja vazduha zbog konvekcije.
- *y* predstavlja vrednost temperaturne razlike između toplog vazduha koji se kreće na gore i hladnog vazduha koji pada.
- *z* je vrednost vertikalne temperaturne razlike u sistemu od dole na gore.

Parametri *a*. *b* i *r* u (1.1) predstavljaju bitan deo sistema, uzimajući u obzir njihov veliki uticaj na sistem.

Parametar *a* odgovara Prandtl-ovom broju koji je dobijen na osovnu prirode vazduha koji se posmatra i obično ima vrednost 10.

Parametar *b* predstavlja površinu koja se posmatra u modelu; Lorenz je za *b* uzimao vrednost 8/3, odnosno 2.666.

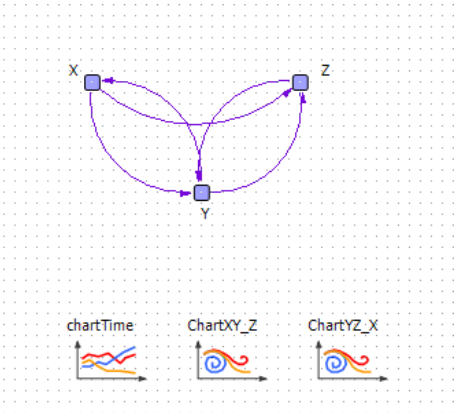
Parametar *r* je Rejlogov broj - parametar koji određuje u kom momentu će početi konvekcija vazduha u sistemu. Taj broj, koji obično ima vrednost 28, je veoma bitan za promenu sistema iz stabilnog u haotičan.

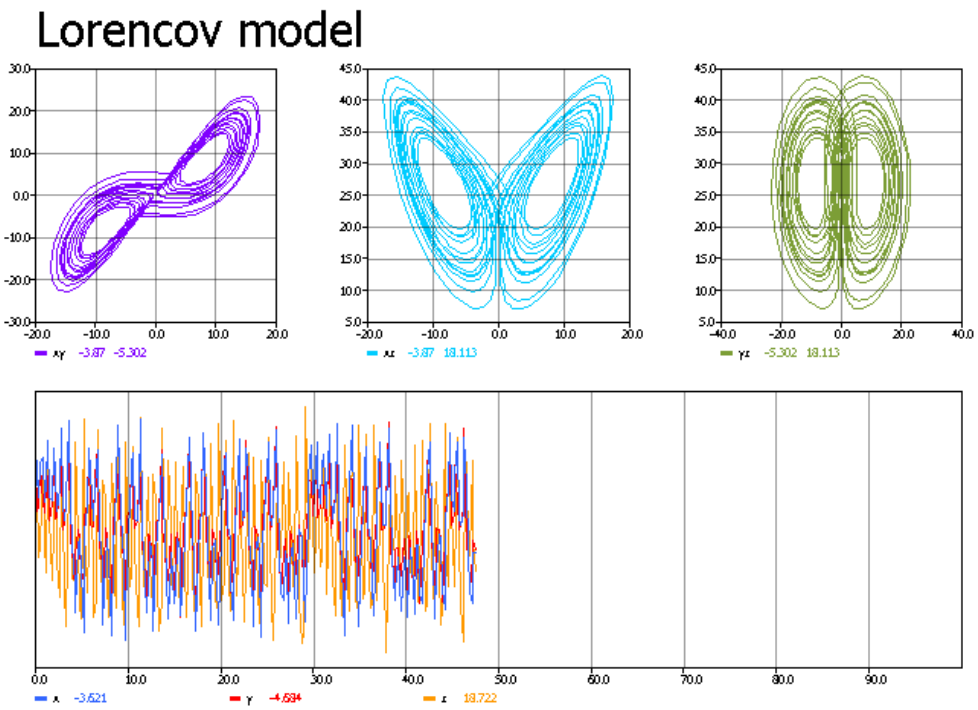
U zavisnosti od vrednosti parametra *r*, model ima sledeća ponašanja. Ako je a=10 i b=8/3, onda je za $0<r<1$ sistem stabilan, dok je za $r>1$ nestabilan. Ako je $1<r<24.74$ tada će ravnotežne tačke c_1 i c_2, date sa

$$c_1 = (\sqrt{b(r-1)}, \sqrt{b(r-1)}, r-1) \quad , \quad c_2 = (-\sqrt{b(r-1)}, -\sqrt{b(r-1)}, r-1)$$

biti stabilne, ali ako je $r>24.74$, onda tačke c_1 i c_2 postaju nestabilne a samim tim se i ceo sistem tada ponaša nestabilno.

truktura modela u programu *AnyLogic* je vrlo jednostavna, pošto se ovakvi modeli jednostavno unose direktno matematičkim izrazom za diferencijalnu jednačinu. U našem slučaju, ta strukura *AnyLogic*-u izgleda ovako:





Slika 1.1. Rezultati izvršavanja Lorencovog modela u programu *Any Logic*

Primer Lorencovog modela vremena napravljenog u programu *AnyLogic* možete pogledati u priloženom apletu.

2. Kermack-McKendrick model (SIR model)

2.1. Uvod

Model koji su početkom 20-tog veka uveli Kermack i McKendrick pre predstavlja praktičan i pouzdan sistem jednačina za praćenje širenja infektivnih bolesti. Korišćen je za ispitivanje širenja opasnih bolesti kao što je kuga. Iako jednostavan, model je kompletan i efikasan.

Model se zasniva se na tri promenljive koje predstavljaju populaciju ljudi podeljenu u tri grupe: neinficirani (*suspected*), inficirani i zaraženi (*infected*), oporavljeni/imuni/umrli (*recovered, imune, removed*). Bitno je primetiti, da je model u osnovi razvijen da prati ograničenu populaciju gde se broj osoba u toku trajanja epidemije ne menja. U principu, model se odnosi na bolesti koje nemaju fatalan ishod, ale se uz manje modifikacije može se uključiti u model tako da se menja broj osoba u populaciji.

Kermack-McKendrick model dozvoljava uključivanje neograničeno mnogo karakteristika bolesti i karakteristika epidemije. Upravo ti parametri određuju ponašanje modela, stim da vrednosti za te parametre moraju biti pažljivo birane i detaljno testirane.

2.2. Sistem jednačina sa kojim se opisuje epidemija

Sistem koristi parametre koji sadrže u sebi različite uticaje koji generalno utiču na napredovanje odnosno sprečavanje epidemije. Sa *a* ćemo označiti grupu parametara koji utiču pozitivno na epidemiju, a sa *b* grupu parametara koji smanjuju razmere epidemije. Dalje, sa *S* je označena grupa neinficiranih, sa *I* grupa inficiranih i sa *R* grupa oporavljenih jedinki. Originalan sistem jednačina *Kermack-McKendrick* modela (pogledati [10]) je dat u sistemu (2.1):

$$\begin{aligned} \frac{dS}{dt} &= -aSI \\ \frac{dI}{dt} &= aSI - bI \\ \frac{dR}{dt} &= bI \end{aligned} \tag{2.1}$$

2.3. Analiza sistema (2.1)

Iz prve jednačine može se zaključiti da je broj neinficiranih koji će se inficirati proporcionalan broju kontakata između osoba iz grupe *S* i grupe *I*, pod pretpostavkom da broj kontakata zavisi samo od broja osoba u svakoj grupi, odnosno da postoji uniformno mešanje u populaciji.

Treća jednačina se zasniva na pretpostavci da je broj oporavljenih proporcionalan broju inficiranih, što predstavlja neki prosek vremena koje osobe provedu u stanju inficiranosti dok ne pređu u stanje kada ne mogu da prenose infekciju, niti da se ponovo inficiraju.

U skladu sa pretpostavkom o konstantnom broju osoba u populaciji važi da je

$$S(t) + I(t) + R(t) = N,$$

gde je *N* ukupan broj osoba u populaciji. Primetimo i da je stopa umnožavanja bolesti data sa $R_0 = a/b$. U početku važi jednakost

$$S(0) + I(0) = N.$$

Kako mora biti bar nekoliko osoba koje su inficirane mora važiti $I(0) > 0$. Zanimaju nas jedino rešenja gde su *S*, *I*, *R* nenegativne veličine, iz čega sledi da je $\frac{dS}{dt} < 0$ kada postoje osobe koje su neinficirane i osobe koje su inficirane. Budući da se *S* smanjuje kako vreme teče:

$$S(t) < S(t_1) < S(0),$$

to, za $t>t_1>0$, konstantno smanjivanje vrednosti *S* i činjenica da *S* mora biti nenegativno, znači da kad $t \to \infty$, *S* mora imati graničnu vrednost (koja može biti i 0), tj. postoji

$$S(\infty) = \lim_{t\to\infty} S(t).$$

Iz druge jednačine sledi da je $\frac{dI}{dt} < 0$, ako je $aS<b$. Kako se S smanjuje kroz vreme, sledi da ako je $aS(0)<b$ i $\frac{dI}{dt} < 0$ za svako t, epidemija će nestati. Postoji kritična vrednost b/a koju početna populacija neinficiranih mora dostići da bi epidemija počela. Ova vrednost je mala ako je $b<<a$, što znači da je potvrđen imunitet i bolest se ne može širiti.

Iz treće jednačine R monotono raste, pri čemu važi $R(t) \le N$, pa granična vrednost $R(\infty) = \lim_{t\to\infty} R(t)$ postoji. Ali, kako postoji i granična vrednost $I(\infty) = \lim_{t\to\infty} I(t)$, veličina $\frac{\{I(\infty) - R(\infty)\}}{N}$ predstavlja „snagu“ kojom je epidemija prošla kroz populaciju. Sada je potrebno odrediti te granične vrednosti. Iz prve i treće jednačine modela sledi

$$\frac{dS}{dR} = -aS/b,$$

tako da je

$$S = S(0)\exp(-aR/b).$$

Kako je $R \le N$, sledi da je

$$S \ge S(0)\exp(-aN/b) \text{ i } S(\infty) > 0.$$

To znači da će uvek ostati jedan broj neinficiranih. Naime, jedan broj osoba u populaciji se neće razboleti iako će se epidemija završiti.

$$\frac{\frac{dI}{dt}}{d\left(\frac{dS}{dt}\right)} = -1 + \frac{b}{as}.$$

Uz $S(0) + I(0) = N$ važi

$$I = N - S + \left(\frac{b}{a}\right)\ln\left\{\frac{S}{S(0)}\right\}$$

Očigledno je $S(t) - S(\infty) > 0$ i $I(t) \to 0, t \to \infty$. Posledica ove analize, uz $S = S(0)e^{-\frac{aR}{b}}$ i jednakosti

$$S(t) + I(t) + R(t) = N,$$

je da važi

$$S(\infty) = S(0)\exp(-a\{N - S(\infty)\}/b),$$

što je transcendentna jednačina koja određuje $S(\infty)$. Jednačina je zadovoljena samo za jednu pozitivnu vrednost od $S(\infty)$ manju od $\frac{b}{a}$. Kada se odredi $S(\infty)$, $R(\infty)$ se može odrediti iz $R(\infty) = N - S(\infty)$ i širenje epidemije koje se određuje sa $\frac{R(\infty)}{N}$.

2.4. Model vakcinacije

Do sada smo radili sa pretpostavkom da su osobe koje imaju prirodan imunitet u malom broju, pa mogu biti zanemarene. Držaćemo se i dalje ove pretpostavke, ali u razmatranje ćemo uključiti slučaj kada se epidemija toliko brzo širi da je potrebno zaustaviti širenje vakcinacijom.

Da bi se pojednostavio model, pretpostavimo da vakcinacija uklanja osobu iz grupe *S* ili *R* odmah i da postoji način da se spreči vakcinisanje onih koji su inficirani (grupa *I*). Sada se pravi nova grupa *V* koju čine vakcinisane osobe i imamo

$$\frac{dS}{dt} = -aSI - \alpha(t)$$
$$\frac{dV}{dt} = \alpha(t)$$

Prva jednačina u ovom sistemu menja prvu jednačinu početnog sistema SIR modela, videti jednačinu(2.1). Funkcija $\alpha(t)$ predstavlja stopu vakcinacije.

Izbor funkcije koja će predstavljati stopu vakcinacije nije lak. Svaki program vakcinacije uključuje troškove koje čine angažovanje osoblja koje će sprovesti vakcinaciju, opremu i sredstva. Pored ovoga mora se uzeti u obzir i šteta koju trpi društvo zbog epidemije. Poželjno je da program vakcinacije ograniči ukupan broj inficiranih u nekom vremenskom periodu ili da drži broj inficiranih ispod nekog željenog nivoa. Da bi osigurali da ne više od N_1 jedinki od populacije dobije bolest za $0 \le t \le T$, mora biti

$$R(T) + I(T) \le N_1.$$

Da bi držali broj inficiranih ispod neke vrednosti N_2 u istom vremenskom intervalu, potrebno je da važi

$$\max_{0 \le t \le T} I(t) \le N_2.$$

Kontrolisati epidemiju na ovaj način podrazumeva da $\alpha(t)$ treba da bude izabrano tako da zadovolji ograničenja zadata sa prethodne dve nejednačine i pri tome drži troškove na minimumu. Primetimo da smo dobili problem dinamičkog programiranja.

2.5. Parametri modela

Prvi korak je da se odrede formule koje će se koristiti za faktor infekcije *a* i faktor povlačenja bolesti *b*. Radi jednostavnosti, koriste se samo četiri parametra koji imaju značajan uticaj na sistem odnosno ponašanje epidemije. Za te parametre su izabrani

- *d*: trajanje bolesti
- *c*: stopa kontakata
- *p*: verovatnoća zaraze
- *n*: ukupna populacija

Za *a* ćemo koristiti sledeću formulu

$$a = \frac{cp}{n},$$

iz koje se vidi da će pozitivan uticaj na širenje epidemije imati stopa kontakata koji se dešavaju između osoba u populaciji i verovatnoća da se bolest prenese u neposrednom kontaktu između osoba. Broj $1/n$ pomnožen sa brojem neinficiranih S daje deo populacije koji je neinficiran.

Za *b* važi sledeća formula:

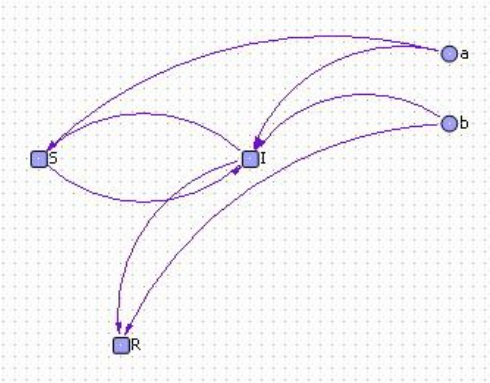
$$b = \frac{1}{d},$$

što znači da na smirivanje epidemije dužina trajanja bolesti utiče obrnuto proporcionalno. To se može protumačiti na sledeći način: što bolest kraće traje epidemija će se pre završiti. Primer za ovo je *Ebola virus*. Razlog zašto se ova izuzetno infektivna i smrtonosna bolest nije proširila po svetu je to što brzo ubija svoje žrtve, epidemija jako kratko traje, te je mala verovatnoća da se bolest raširi na većoj teritoriji.

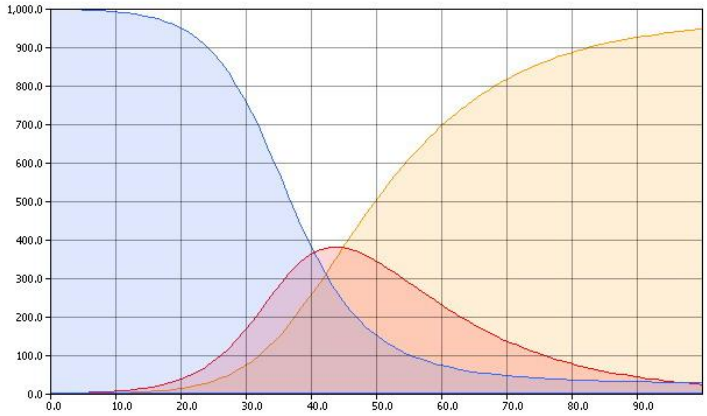
Testiranjem parametara može se utvrditi koje vrednosti parametara su opravdane za potrebno ponašanje modela. Za sada ćemo koristiti sledeće vrednosti za parametre: c=5, p=0.05, n=1000, d=15. Unošenjem jednačina u *AnyLogic*, može se napraviti simulacija modela, sistem (2.2).

$$\begin{aligned} \frac{dS}{dt} &= -aSI \\ \frac{dI}{dt} &= aSI - bI \\ \frac{dR}{dt} &= bI \\ a &= \frac{cp}{n} \\ b &= \frac{1}{d} \end{aligned} \tag{2.2}$$

Struktura modela u programu *AnyLogic* izgleda ovako:



Unošenjem početnih vrednosti za S=999, I=1 i R=0 i pokretanjem simulacije dobija se sledeći grafik (plavom bojom je prikazan broj neinficiranih, crvenom bojom broj inficiranih i žutom bojom broj oporavljenih).

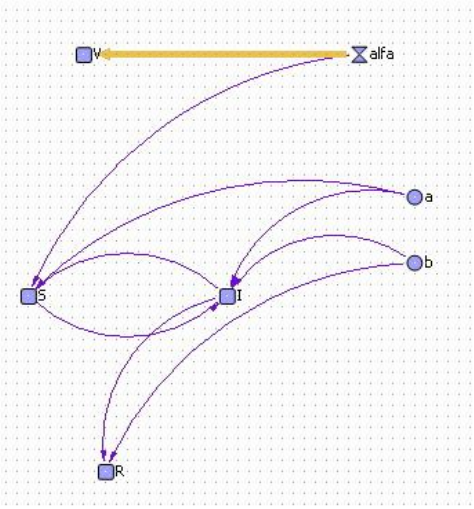


Slika 2.1. Rezultat simulacije SIR modela.

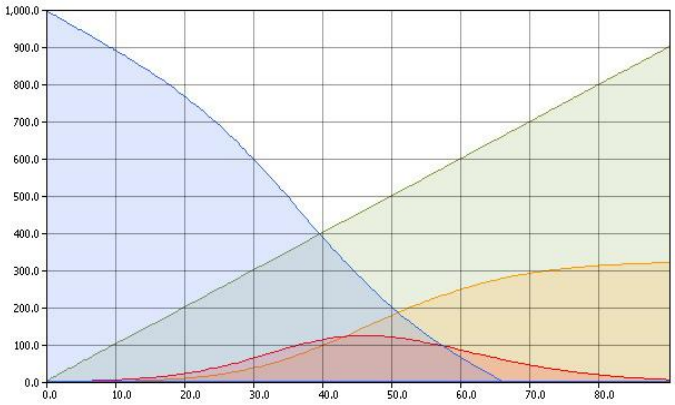
Ako se u model uključi vakcinacija, trebalo bi značajno da se smanji broj inficiranih. Promenom sistema jednačina možemo uključiti vakcinaciju, videti sistem (2.3).

$$\begin{aligned}
\frac{dS}{dt} &= -aSI - \alpha \\
\frac{dI}{dt} &= aSI - bI \\
\frac{dR}{dt} &= bI \\
\frac{dV}{dt} &= \alpha \\
a &= \frac{cp}{n} \\
b &= \frac{1}{d}
\end{aligned} \tag{2.3}$$

Struktura modela u *AnyLogic*-u je sada drugačija



Pokretanjem simulacije se dobija sledeći grafik:

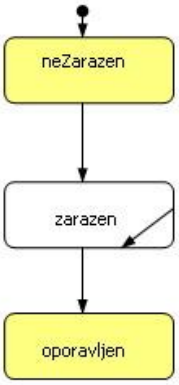


Slika 2.2. Rezultat simulacije SIR modela sa vakcinacijom.

Kako su parametri u oba primera bili isti mogu se porediti rezultati simulacija. Najinterensantinije je posmatrati broj inficiranih. U prvom primeru broj inficiranih je došao do skoro 400. U modelu sa vakcinacijom broj inficiranih je malo prešao preko 100.

2.6. Agentni SIR model u programu *AnyLogic*

Kao što je u ranijim poglavljima rečeno, svaki model napravljen u sistemskoj dinamici možemo prevesti u model sa agentima. Prvo se napravi novi model i nova klasa aktivnog objekta sa imenom *Osoba*. U klasu *Osoba* treba ubaciti dijagram stanja. Dalje je potrebno napraviti dijagram stanja koji će opisivati ponašanje agenta. Očigledno je da dijagram stanja mora imati tri stanja: nezaražen, zaražen i oporavljen. Na slici 2.3 je prikazan dijagram stanja napravljen u *AnyLogic*-u.



Slika 2.3. Dijagram stanja klase *Osoba*.

Kada je dijagram nacrtan, treba zadati uslove za prelaze između stanja. prelaz između stanja neZarazen i zarazen napravićemo preko signalnog događaja. Ako agent u stanju neZarazen dobije signal „kontakt“ što znači da je ostvaren kontakt sa zaraženim agentom, on prelazi u stanje zaražen. Unutrašnji prelaz u stanju zaražen predstavlja mogućnost zaaraženog agenta da šalje signal kontakt. Stavićemo da sa nekom stopom stopa kontakta se izvrši sledeća akcija:

```
if (randomTrue(verovatnocaZaraze)) {

        Osoba neko=main.ljudi.random();

        neko.statechart.fireEvent("kontakt");

}
```

Prelaz iz stanja *zarazen* u stanje *oporavljen* se takođe dešava sa nekom stopom. Za stopu prelaska ćemo uzeti vrednost *1/trajanjeBolesti*.

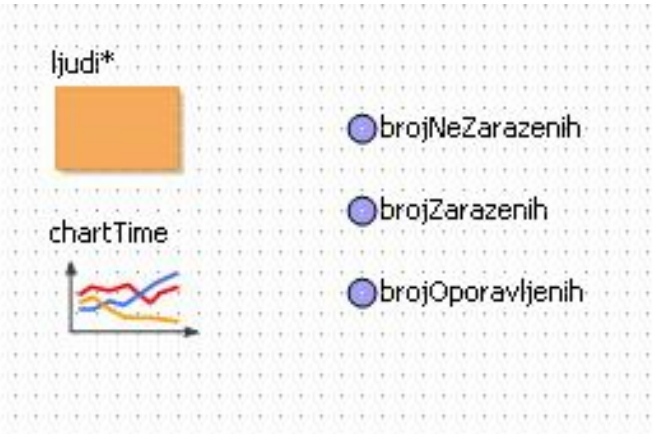
Takođe se trebaju uneti ulazne i izlazne akcije za svako stanje. Promenljive *brojNeZarazenih, brojZarazenih* i *brojOporavljenih* prate broj agenata koji se nalaze u ovim stanjima tako da za ulaznu akciju u stanje treba uneti da se poveća za jedan, a izlaznu akciju da se smanji broj agenata u tom stanju. Na slici 2.4 je prikazano kako je to urađeno za stanje *zarazen*. Na isti način se može uraditi i za stanja *neZarazen* i *oporavljen*.

Slika 2.4. Uazne i izlazne akcije za stanje *zarazen*.

Vratimo se na dijagram strukture klase *Osoba*. Tu je potrebno definisati promenljivu main koja je tipa *Main* i ima početnu vrednost `(Main)getOwner()`.

Klasa O*soba* treba da sadrži i tri parametra koji su osobine agenta. Potrebno je definisati parametre *trajanjeBolesti* koji je tipa *double* i ima početnu vrednost 15, *verovatnocaZaraze* koji je tipa *double* i ima početnu vrednost 0.05 i parametar *stopaKontakata* tipa *double* sa početnom vrednošću 5. Ovim je klasa *Osoba* definisana.

U dijagramu strukture klase *Main* treba uvući klasu *Osoba* prevlačenjem mišem. Na dijagramu strukture će se pojaviti narandžasti pravougaonik koji predstavlja agenta definisanog preko klase osoba. Još je potrebno u svojstva ove klase uneti ime za klasu ljudi i u kartici *Replication* 1000. To znače da će u našem modelu biti napravljeno 1000 agenta. Pored ovoga na dijagramu strukture treba da se nađu i promenljive *brojNeZarazenih*, *brojZarazenih* i *brojOporavljenih* koje su tipa *Integer*, slika 2.5.



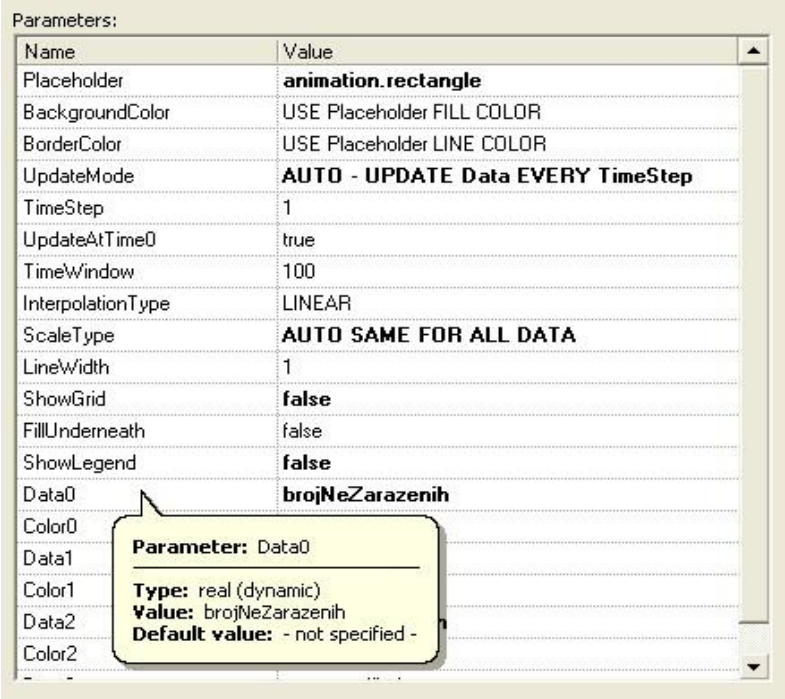
Slika 2.5. Dijagram strukture klase *Main*.

Model je sada skoro gotov, ali ako se pokrene neće raditi. Podsetimo se dijagrama stanja za agenta svaki agent će se po kreiranju nalaziti u stanju *neZarazen* zato što na to stanje pokazuje početni pokazivač. Kako nema nijedne zaražene osobe ništa se neće dogoditi. Zato je potrebno da se za jednog agenta ručno unese kod koji će početi epidemiju. Potrebno je otvoriti *Code* polje za klasu *Main* i u odeljak *Startup Code* uneti

```
ljudi.item(0).statechart.fireEvent("kontakt");
```

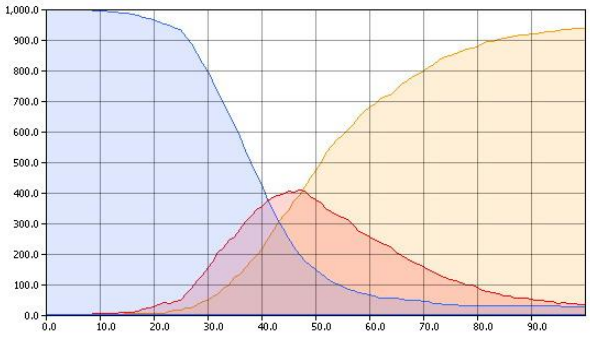
čime smo nultom agentu dali naredbu da pusti signalni događaj kontakt.

Animaciju možemo držati jednostavnom za sada pa ćemo u animaciji nacrtati jedan pravougaonik koji će služiti za grafički prikaz broja zaraženih, nezaraženi i oporavljenih. Grafički prikaz je omogućen korišćenjem *Business Graphic Library*, odakle je potrebno uvući objekat *ChartTime* na dijagram strukture klase *Main*. Podešavanja svojstava *chartTime* treba uraditi kao na slici 2.6.



Slika 2.6. Podešavanje svojstava *chartTime.*

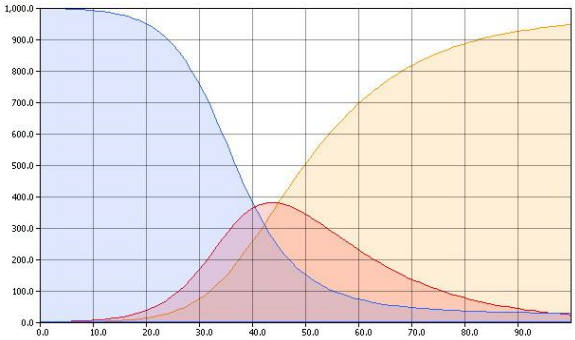
Ako se pokrene simulacija dobija se sledeći grafik, slika 2.7:



Slika 2.7. Grafik modela posle pokretanja simulacije.

Poređenjem sa grafikom dobijenim u modelu sistemske dinamike koje je napravljen sa istim parametrima mogu se uočiti razlike,

uporediti sliku 2.7 sa slikom 2.8



Slika 2.8. Grafik modela u sistemskoj dinamici.

Na grafiku dobijenom u modelu sa agentima vidi se stohastička priroda modela, ali to je i dalje jako slično rezultatu dobijenim u modelu sistemske dinamike. Sada se prirodno nameće pitanje: Zašto je onda potreban agentni model i koje su njegove prednosti?

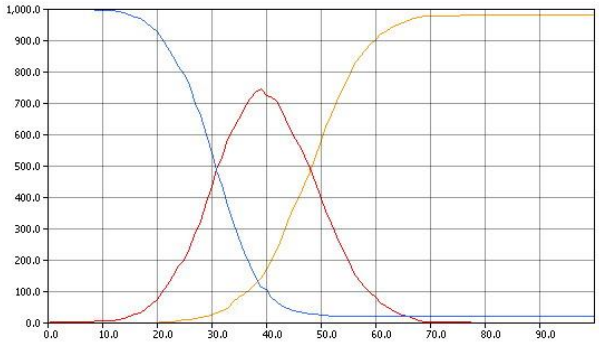
U modelu sistemske dinamike po prirodi modela dužina trajanja bolesti je prikazana preko eksponencijalne raspodele. To nije baš realna situacija za neke bolesti. Bilo bi bolje da je dužina trajanja bolesti prikazana korišćenjem trougaone raspodele.

Prelaz između stanja *zarezen* i stanja *oporavljen* treba modifikovati usmesto zadate stope prelaska treba izabrati *Fire After timeout* i u polje *Timeout* treba uneti sledeće:

```
Triangular

      {trajanjeBolesti/2,trajanjeBolesti,trajanjeBolesti*1.5}
```

Sada se simulacijom dobija sledeći grafik:



Slika 2.9. Grafik agentnog modela.

Ovakva promena je moguća i u modelu sistemske dinamike. Promenom jednačina se može pokušati da se reprodukuje ovakvo ponašanje, ali se jednačine značajno komplikuju. Čak i ako se postigne da se dobije ovakvo ponašanje u modelu sistemske dinamike, to je samo pokušaj da se reprodukuje poznat rezultat dok u agentnom modelu predstavlja način da se modelira stvarna karakteristika neke bolesti.

Ako se podsetimo modela u sistemskoj dinamici koji su ranije opisani, jedan model je sadržao u sebi i vakcinaciju. Kako bi izgledalo ubaciti proces vakcinacije u agentni model? Zbog mogućnosti kombinovanja paradigmi u modeliranju u *AnyLogic*-u možemo modelirati vakcinaciju preko sistemske dinamike.

Na dijagram strukture *Main* klase ćemo uneti potrebne promenljive za vakcinaciju. Jedna promenljiva treba da ima ime *razvojVakcine* sa početnom vrednošću 10 dok će druga promenljiva biti data sa diferencijalnom jednačinom

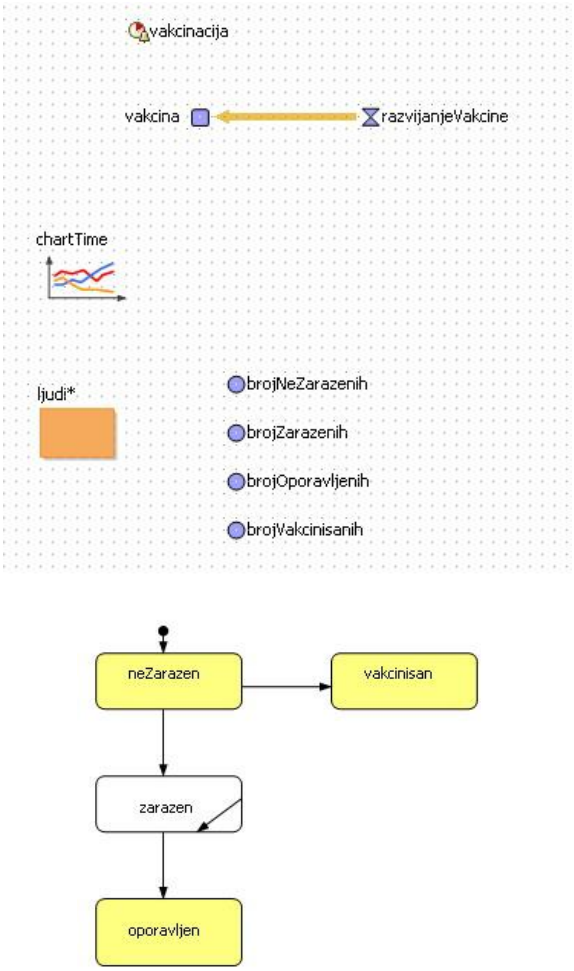
$$\frac{dVakcina}{dt} = razvojVakcine\ .$$

Dodatno, na dijagram treba uneti statički tajmer kojem treba dodeliri ime *vakcinacija*. Taj tajmer će biti cikličan sa timeout 1 i akcija koja će se izvršavati je zadata sledećim kodom

```
while (vakcina>1) {
```

```
        ljudi.random().statechart.fireEvent("vakcinacija");
        vakcina--;
    }
```

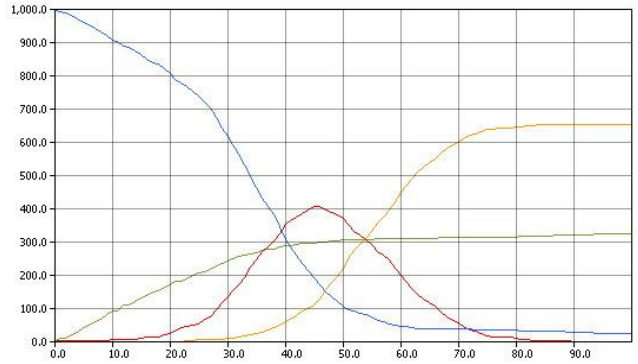
Ovaj kod kaže da će se svaki put kada se tajmer izvrši, obavljati vakcinacija dogod ima vakcine na zalihama. Primaoci vakcine će biti birani na slučajan način, ali samo nezaraženi agenti mogu preći u stanje *vakcinisan*.



Slika 2.10. Agentni model i dijagram stanja sa vakcinacijom.

U dijagramu stanja za agenta treba dodati još jedno stanje - *vakcinisan*. Ulazna akcija za to stanje će biti *main.brojVakcinisanih*++. Prelaz će se napraviti iz stanja *neZarazen* u stanje *vakcinisan* i taj prelaz čeka signalni događaj „vakcinisan“. Na ovaj način agent može preći u stanje *vakcinisan* samo iz stanja *neZarazen*.

Pre pokretanja animacije, potrebno je samo u svojstva *chartTime* uneti da se prikazuje i promenljiva *brojVakcinisanih*. Grafik koji se dobije jasno prikazuje efikasnost vakcinacije. Broj zaraženih je drastično opao u poređenju sa rezultatom dobijenim u prethodnoj simulaciji.



Slika 2.11. Grafici promenljivih posle vakcinacije.

Jedan od bitnih uticaja na širenje epidemije je prostorno okruženje u kojem se epidemija širi. Uvesti uticaj okruženja u model sistemske dinamike zanči da će se raditi sa parcijalnim diferencijalnim jednačinama.

Takvi modeli mogu biti jako komplikovani. Uvođenje uticaja okruženja u agentni model nije teško. Zahvaljujući biblioteci za agentno modeliranje može se lako napraviti agent sa dosta različitih osobina. Na dijagram strukture treba uvući *agentBase* objekat iz biblioteke *AgentBase*. Dalje treba zadati sledeća svojstva tih agenta.

Okruženje u kojem će se agent nalaziti će biti dimenzija 300 sa 300. Podrazumevana mreža agenata (*default network*) će biti *all in range* i *contact range* treba podesiti na 30. Još je potrebno promenuti kod za unutrašnju tranziciju u stanju zarazen:

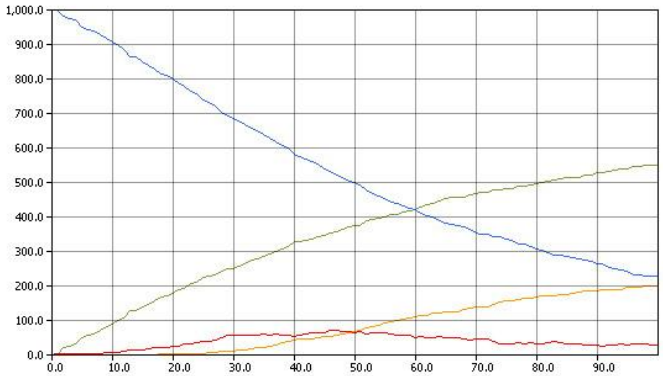
```
if (randomTrue(verovatnocaZaraze))

agentBase.sendToRandomContact("kontakt");
```

i dodati u svojstvo *onRecive* agentBase objekta sledeći izraz:

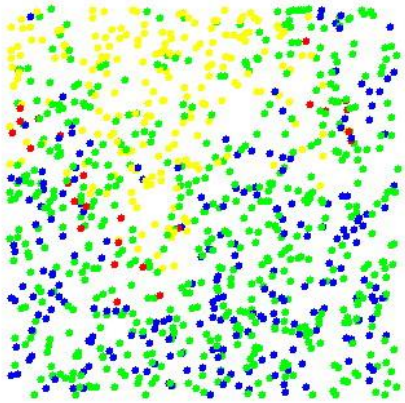
```
statechart.fireEvent(message)
```

Rezultat ove simulacije pokazuje još veće smanjenje broja zarazenih, zato što sada okruženje u kojem se agenti nalaze utiče na stopu kontakata koji oni ostvaruju pa se epidemija sporije širi kroz populaciju, a proces vakcinacije postaje efikasniji.



Slika 2.12. Grafici promenljivih posle uvodjenja vakcinacije.

Radi bolje prezentacije uticaja okruženja na širenje epidemije, može se u model dodati animacija za svakog agenta. Kako su agentu umnoženi objekti dovoljno je napraviti animaciju za jednog agenta, a na konačnoj animaciji će se kroz replikaciju pojaviti svih 1000 agenta koliko ih ima u modelu. Dakle, u klasu osoba treba dodati animaciju i nacrtati kružić koji će prikazivati agenta u svojstva kruga u polje *Fill color* treba uneti boja.



Slika 2.13. Model sa uvodjenjem agenata.

Promenljivu boja treba definisati na dijagramu klase Osoba i dodeliti joj tip Color. U dijagramu stanja ulazne akcije za svako stanje treba promeniti dodavanjem sledećih kodova. Za stanje *neZarazen* treba dodati boja= Color.blue, za stanje *zarazen* boja= Color.red, za stanje *oporavljen* boja= Color.yellow i za stanje *vakcinisan* boja=Color.green.

Animacija je sada mnogo efektnija pošto se može posmatrati kako se kroz populaciju u prostoru širila epidemija.

3. Matematičko klatno

Sa matematičkog gledišta, rlatno je oscilatorni sistem koji se sastoji iz neistegljive niti zanemarljive mase na koju je obešena kuglica zanemarljivo malih dimenzija u odnosu na dužinu niti i znatno veće mase od mase niti i koji može da osciluje pod uticajem Zemljine teže. Problem se sastoji u posmatranju kretanja klatna pod uticajem gravitacije.

Radi jednostavnosti, za sada možemo uzeti da je dužina klatna l=1 i da je masa klatna m=1. Prva predpostavka je da gravitacija ima konstantan uticaj sile na klatno silom delovanja g vertikalno na dole. Sada treba posmatrati kretanje klatna na jednu stranu u pozitivnom smeru odnosno na drugu stranu u negativnom smeru.

Pri tome merimo ugao α u odnosu na stanje klatna u mirovanju i ugaonu brzinu klatna ω. Jednačina kretanja klatna je data u (3.1):

$$\alpha'' = -g \sin \alpha$$

gde je

$$\frac{d^2\alpha}{dt^2} = \alpha'' \tag{3.1}$$

Ako još modelu dodamo ugaonu brzinu ω u prethodni izraz, dobijamo sistem (3.2):

$$\begin{aligned} \alpha' &= \omega \\ \omega' &= -g\sin\alpha \end{aligned} \tag{3.2}$$

Analizom ponašanja klatna možemo da zaključimo da se idelizovan model matematičkog klatna koji posmatramo može naći u četiri stanja;

1. klatno se nalazi u mirovanju vertikalno na dole;
2. klatno se kreće između dve tačke u kojima trenutno miruje sa jednakim otklonom na obe strane od ravnotežnog položaja;
3. klatno konstantno rotira u istom smeru i nikad ne dolazi u ravnotežno stanje;
4. klatno se nalazi vertikalno uspravno u mirovanju (nestabilno stanje).

Ovakav model klatna je prilično jednostavno modelirati u *AnyLogic*-u. Dovoljno je uneti gore navedene jednačine i posmatrati ponašanje klatna. U modelu ćemo iskoristi promenljive *x* i *y* koje će predstavljati koordinate klatna prilikom animacije, promenljive α i ω za ugao i ugaonu brzinu. Za početak uvodimo i dva parametra: početni ugao α_0 i dužinu klatna *l*.

3.1. Parametri u matematičkom modelu klatna.

Uvođenjem još jednog parametra u *AnyLogic*-u lako je dodati i otpor sredine *r* na klatno. U stvari, dovoljno je izmeniti jednačinu za ugaonu brzinu, videti (3.3):

$$\omega' = -g\sin\alpha - r\omega \tag{3.3}$$

Umesto detaljnijeg opisa kako je ovaj model napravljen u programu *AnyLogic*, iskoritićemo izveštaj koji je moguće generisati za svaki *AnyLogic* model. Izveštaj je dobar način da se nekome predstavi detaljna stuktura modela.

Active Object: *Main*

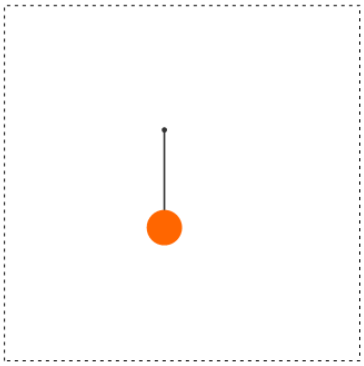
Parameters

Name	alfa0
Type	real
Default value	3.14
Name	l
Type	real
Default value	100
Name	g
Type	real
Default value	9.81

Icon	
Picture	

Structure	
Picture	
Variable	Omega
Variable type	Real
Equation type	Integral or Stock
Equation	d(omega)/dt = (-g*sin(alfa))/l
Initial value	0
Variable	Alfa
Variable type	Real

Equation type	Integral or Stock
Equation	d(alfa)/dt = omega
Initial value	alfa0
Variable	y
Variable type	real
Equation type	Formula
Equation	y = l*cos(alfa)
Variable	x
Variable type	real
Equation type	Formula
Equation	x = l*sin(alfa)

Animation	
Name	animation
Picture	
Oval	Oval1
X	x
Y	y
Line	Line2
End point X	x
End point Y	y

Slika 3.2. Model klatna.

U apletu možete pogledati model klatna sa i bez otpora sredine

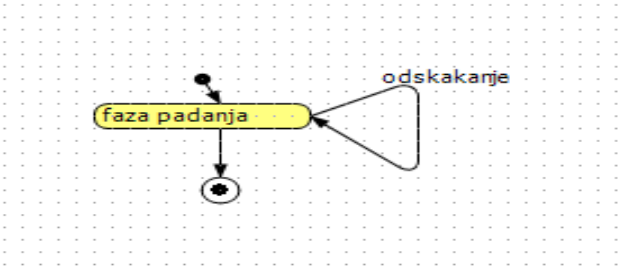
4. Odskakanje lopte

Ovaj model je jedan od najjednostavnih hibridnih modela i često se koristi kao primer hibridnog modeliranja. Lopta se pušta sa neke određene visine ne čvrstu podlogu, udara o podlogu i kreće se na gore pa ponovo pada. Pri svakom udarcu o podlogu lopta gubi određen deo energije, tako da vremenom prestaje da odskače. Samo kretanje lopte na dole se jednostavno opsuje sistemom diferecijalnih jednačina (4.1).

$$\frac{dy}{dt} = V_y$$
$$\frac{dV_y}{dt} = -g, \tag{4.1}$$

gde je V_y brzina kretanja lopte po y-osi, a g je gravitacija,

Momenat kada lopta udara o podlogu predstavlja deo zbog kojeg je ovaj model hibridan. U *AnyLogic*-u ovo se jednostavno modelira pomoću dijagrama stanja.

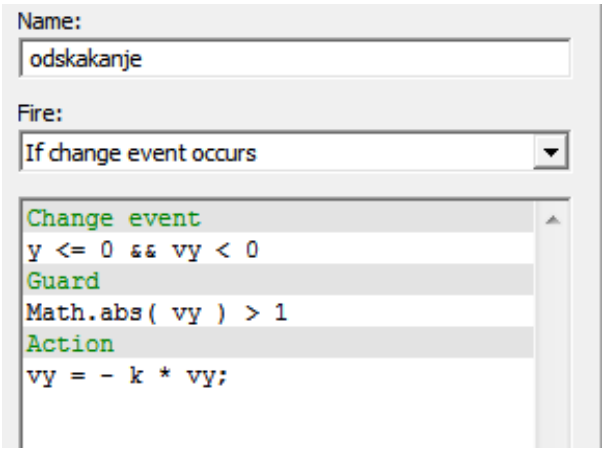


Slika 4.1. Dijagram stanja hibridnog modela odskakanja lopte.

U fazi padanja loptica se ponaša po gore navedem jednačinama. Prelaz „odskakanje“ čeka na uslov $y \le 1$ i $V_y < 0$, stim da mora biti zadovoljeno $|V_y| > 1$. Ako se ti uslovi ispune, izvršava se akcija data jednačinom:

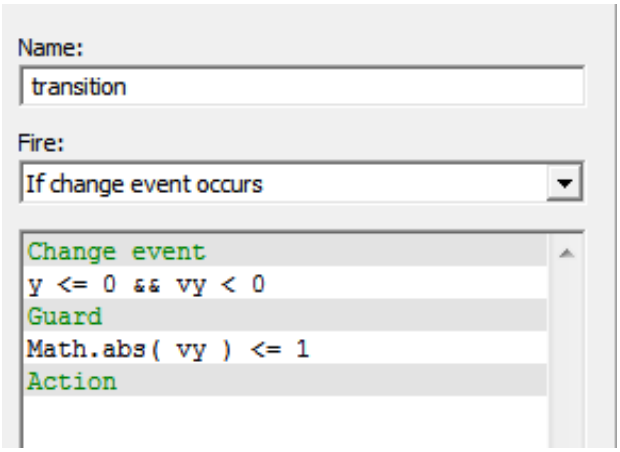
$$V_y' = -kV_y \tag{4.2}$$

gde je *k* koeficijent koji predstavlja gubitak energije loptice. Posle ovog loptica opet ide u „fazu padanja“.



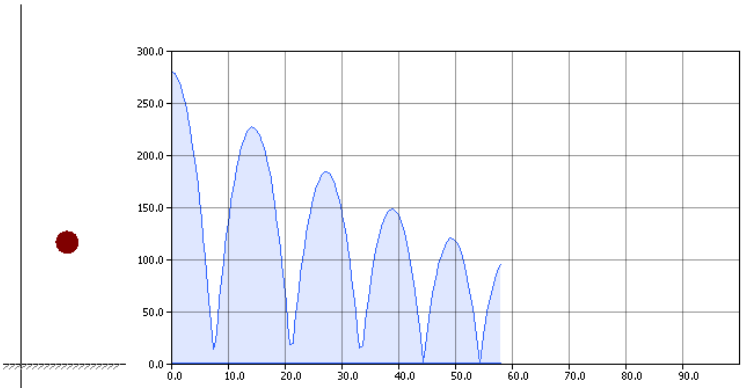
Slika 4.2. Odskakanje loptice.

Kada se ispune uslovi $y \leq 0$ i $V_y < 0$, stim da je zadovoljen uslov $|V_y| \leq 1$, dijagram stanja prelazi u završno stanje, što znači da loptica nema više dovoljno energije da odskoči.



Slika 4.3. Odskakanje loptice pred završno stanje.

Ovaj primer pokazuje kako i naizgled jednostavni modeli mogu imati hibridno ponašanje i da je neophodno imati alat koji omogućava jednostavno kreiranje hibridnih modela.



Slika 4.4. Grafik kretanja loptice pred završno stanje.

5. Model Lotka-Volterra

Model Lotka Voltera je sastavljen od para diferencijalnih jednačina koje opisuju odnos između lovaca i žrtava u najjednostavnijem slučaju. Model je oko 1925 godine razvijen nezavisno od strane dva autora: biologa Alfreda Lotke i čuvenog italijanskog matematičara Vita Volterre. Volterra je razvio model da bi pojasnio zetu ponašanje populacije dve vrste riba u Jadranskom moru, pri čemu se jedna vrsta (lovci, ali često u literaturi zvana „ajkulom“) hrani sa drugom (žrtve).

Prethodni model predstavlja prirodno proširenje tzv. logističkog modela, uvedenog oko 1845 od strane belgijskog naučnika Verhulsta. U modelu populacije lovaca i žrtava osciluju tako da se najveća vrednost za lovce nalazi malo iza najveće vrednosti za žrtve. Taj model ima sledeće pretpostavke:

- populacija žrtve će rasti eksponencijalno ako nema predatora;
- populacija lovaca će izgladnjivati (nestajati) u odsustvu žrtvi;
- lovac može da pojede neograničeno mnogo žrtvi;
- obe populacije se kreću na homogenom, ograničenom prostoru na slučajan način.

Uvedimo sledeće promenljive:

- P: broj lovaca
- N: broj žrtava
- t: vreme
- r: stopa rasta populacije žrtve
- a: stopa napada lovaca na žrtve
- q: stopa mortaliteta lovaca
- c: efikasnost u prevođenju hrane u potomke (efikasnost konverzije)

U nekim modelima uvodi se još jedna pretpostavka. Naime, pored parametra *a, tj.* stope napada lovaca na žrtve, važan je i parametar *b, tj.* stopa mortaliteta žrtve. (Uglavnom se pretpostavi u modelu da žrtva ne izumire prirodno, nego bude pojedena). Dodajmo da je u nekim modelima proizvod parametara *c* i *a* označen kao stopa rasta populacije lovaca (tzv. produktivnost lovaca).

Posmatrajmo šta se događa sa populacijom lovaca kada populacija žrtava ne postoji. Bez hrane očekuje se da broj lovaca opada eksponencijalno, videti (5.1):

$$\frac{dP}{dt} = -qP \tag{5.1}$$

U toj jednačini imamo proizvod broja lovaca i stope mortaliteta lovaca sa negativnim predznakom koji označava pad broja lovaca. Ako imamo populaciju žrtve, tada proizvod *caPN* nadjačava pad populacije lovca. Sam proizvod opisuje stopu napada na žrtvu

pomnoženu sa brojem lovaca i brojem žrtvi. Sve to je još pomnoženo sa stepenom konverzije, jednačina (5.2).

$$\frac{dP}{dt} = caPN - qP \tag{5.2}$$

Ako se posmatra populacija žrtve očekuje se da u odsustvu lovaca broj žrtava raste eksponencijalno, jednačina (5.3):

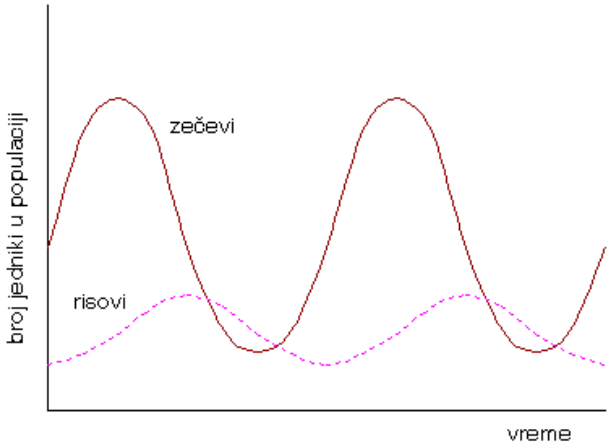
$$\frac{dN}{dt} = rN \tag{5.3}$$

Prisustvo lovaca sprečava eksponencijalni rast žrtve. Uvodi se proizvod *aPN* kojim je opisan mortalitet žrtve, jednačina (5.3):

$$\frac{dN}{dt} = rN - aPN \tag{5.4}$$

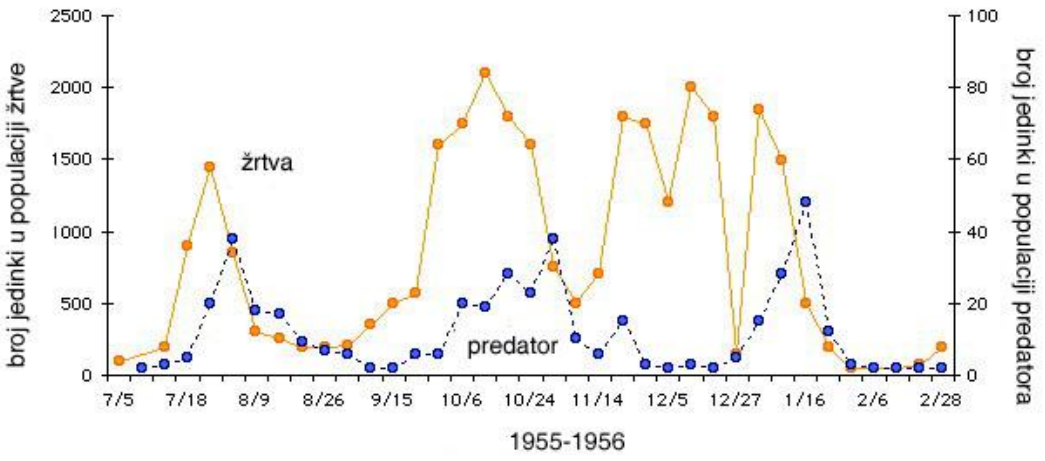
Jednačine koje su dobijene opisuju model l*ovac-žrtva*, koji predviđa cikličnu zavisnost. Kako raste broj lovaca *P,* tako raste i proizvod *aPN* koji dodatno pojačava rast broja lovaca. U isto vreme proizvod *aPN* smanjuje broj žrtava *N*, koji posredno utiče na smanjenje broja lovaca *P*. Kako se smanjuje vrednost za *aPN* populacija žrtve se oporavlja i *N* počinje da raste. Tada se i *P* povećava i ciklus se ponavlja. Idealna kriva je data na grafiku.

Model je testiran i u eksperimentalnim uslovima praćenjem populacija. Dva eksperimenta se mogu razmatrati kao validna. Prvi eksperiment je izveo Huffaker 1958 godine. Posmatrao je populacije grinja. Jedna vrsta su lovci, a druga vrsta je uzeta za žrtve. U okruženju gde je posmatrao populacije postavio je nekoliko narandži (hrana za žrtve). Narandže je delimično pokrio voskom da bi kontrolisao količinu hrane.



Slika 5.1. Grafici promenljivih u sistemu jednačina tipa Lotka-Volterra.

U okruženju je postavio i gumene lopte. Na grafiku je dat rezultat dobijen za jedan raspored narandži i lopti.



Slika 5.2. Grafik dobijen praćenjem dve populacije u laboratorijskim uslovima

Na grafiku se vidi da su populacije pokazale ciklično ponašanje i da se maksimalne vrednosti za lovce nalaze iza maksimalnih vrednosti za žrtve. Da bi zaključili da ovaj eksperiment oslikava pretpostavke modela Lotka-Volterra moramo razmatrati jednu činjenicu.

Primetimo da u ovom eksperimentu postoji dosta složeno okruženje u kojem se nalaze populacije, što je u suprotnosti sa pretpostavkom napravljenom u modelu. Samim tim je izgubljena i pretpostavka da su susreti lovca i žrtve nasumični.

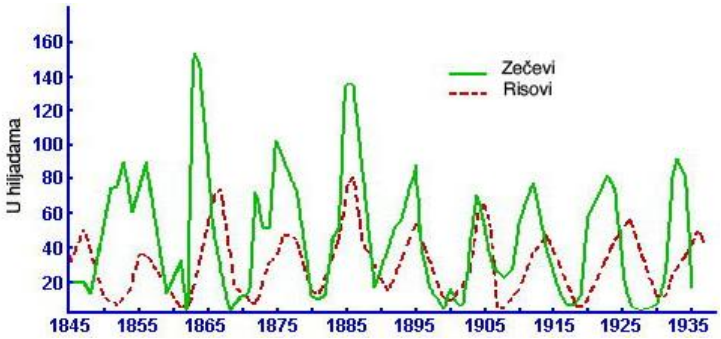
Drugi eksperiment je zasnovan na praćenju broja životinja u prirodi. Problem u posmatranju populacije u prirodi i modeliranju populacije je u tome što se većina predatora oslanja na više različitih žrtava. Na sreću ima nekoliko vrsta koje su se strogo specijalizovale u toku evolucije i hrane se samo jednom vrstom. U kanadskim šumama postoji idealan primer i to su populacija risova i snežnih zečeva.

Na sreću ima nekoliko vrsta koje su se strogo specijalizovale u toku evolucije i hrane se samo jednom vrstom. U kanadskim šumama postoji idealan primer i to su populacija risova i snežnih zečeva. Kompanija Hudson Bay je pažljivo pratila broj svih risova i zečeva između 1800 i 1900 godine. Pretpostavlja se da su te populacije dobre za razmatranje zbog čestih napada risova na zečeve i zbog načina lova.

Podaci koji su skupljeni pokazuju jasne oscilacije u populaciji na otprilike svakih 12 godina. Na grafiku su dati dobijeni podaci.

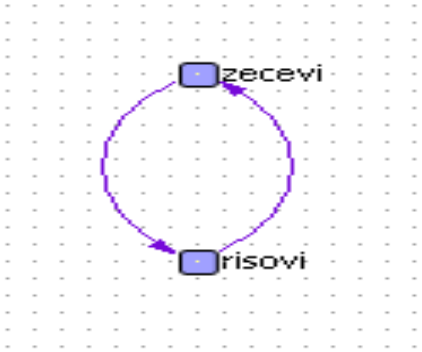
Nedostatak modela Lotka-Volterra je oslanjanje na nerealne pretpostavke. Populacija žrtve ima ograničenu količinu hrane u prirodi i to utiče na njihov broj osim predatora koji sa druge strane ne mogu pojesti neograničenu količinu hrane.

Ako se posmatra sa matematičke strane, ciklično ponašanje koje dobijamo sistemom diferencijalnih jednačina u modelu Lotka-Volterra teži da se ponavlja beskonačno i može se pokazati da će se slično ponašanje dobiti za svaki skup vrednosti za četiri parametra modela.



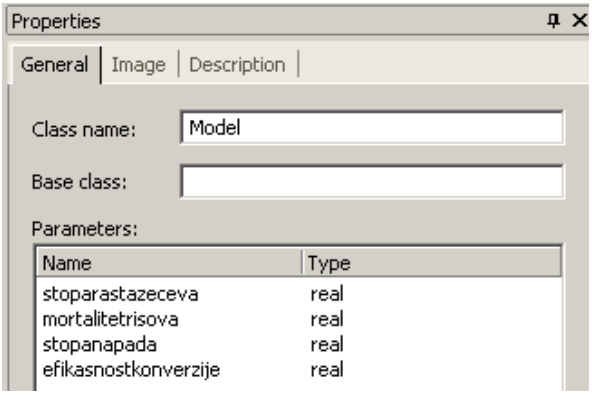
Slika 5.3. Grafik dobijen posmatranjem polupacija zečeva i risova u prirodi

Zaključak je da model Lotka-Volterra nije dovoljan da bi se prikazalo ponašanje većine populacija u prirodi. Moraju se posmatrati dodatne informacije specifične za taj sistem.



Slika 5.4. Početni model Lotka-Volterra u *AnyLogic*-u.

Predstavljanje modela u *AnyLogic*-u počinjemo sa najjednostavnijim modelom. Zbog dobrog mehanizma za rešavanje diferencijalnih jednačina najlakše je jednostavno definisati dve promenljive koje će predstavljati populacije zečeva i risova. Tim promenljivim se dodeljuju diferencijalne jednačine za model Lotka-Volterra opisane u prethodnom poglavlju. Promenljive su smeštene u klasu aktivnog objekata model što je i jedna klasa u modelu. Na nivou klase su definisani parametri *natalitetrsova, stoparastazeceva,stopanapada, efikasnostkonverzije* sa početnim vrednostima. Animacija prikazuje broj risova i broj zečeva grafički i brojno.



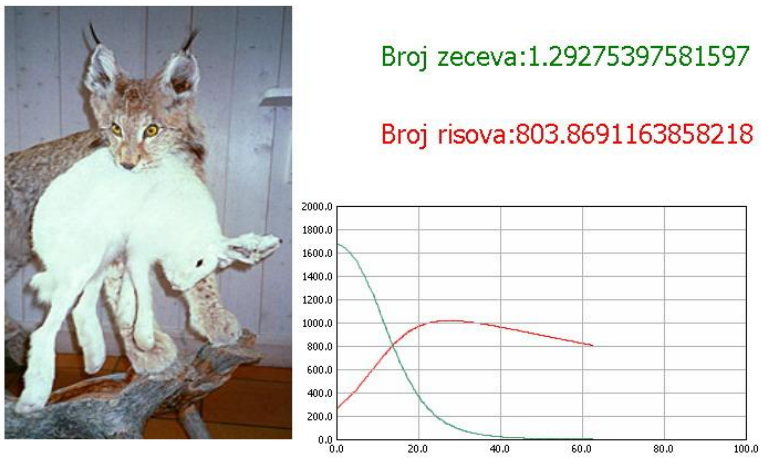
```
Import

Implements interfaces

Startup code

Equations
d(risovi)/dt = risevi*(-mortalitetrisova+stopanapada*efikasnostkonverzije*zecevi)
d(zecevi)/dt = zecevi*(stoparastazeceva-stopanapada*risovi)

Additional class code
```

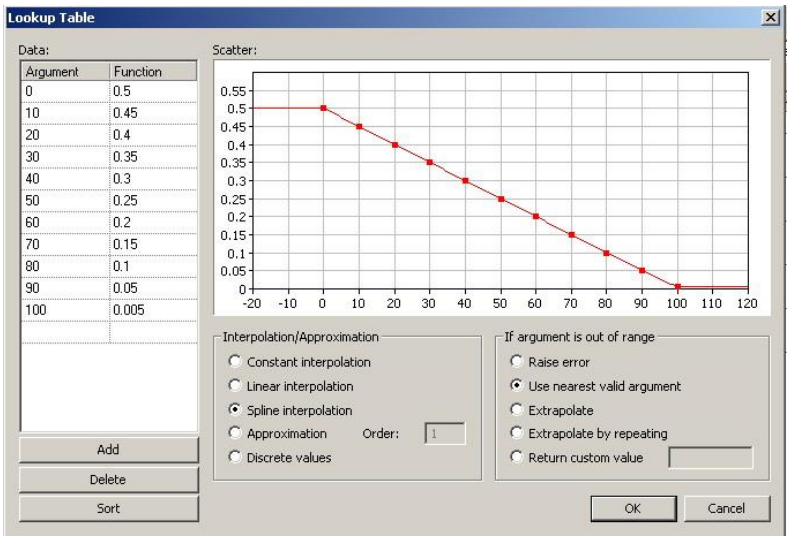


Slika 5.5. Složeniji model tipa Lotka-Volterra u *AnyLogic*-u

Sledeći model predstavlja pomak u smislu malo finije strukture modela i boljem iskorišćenju mogućnosti *AnyLogic* programa, u kojem imamo klasičan primer sistemske dinamike. Diferencijalne jednačine su zadate malo drugačije, ali su i dalje u skladu da modelom Lotka-Volterra, sistem (5.5).

$$\begin{aligned}
&\frac{d(Zecevi)}{dt} = RodjeniZecevi - UginuliZecevi \\
&\frac{d(Risevi)}{dt} = RodjeniRisevi - UginuliRisevi \\
&RodjeniZecevi = Zecevi \cdot NatalitetZeceva \\
&UginuliZecevi = GustinaZeceva \cdot Risevi \\
&RodjeniRisevi = Risevi \cdot NatalitetRiseva \\
&gustinazeceva = \frac{Zecevi}{PovrsinaOblasti}
\end{aligned} \tag{5.5}$$

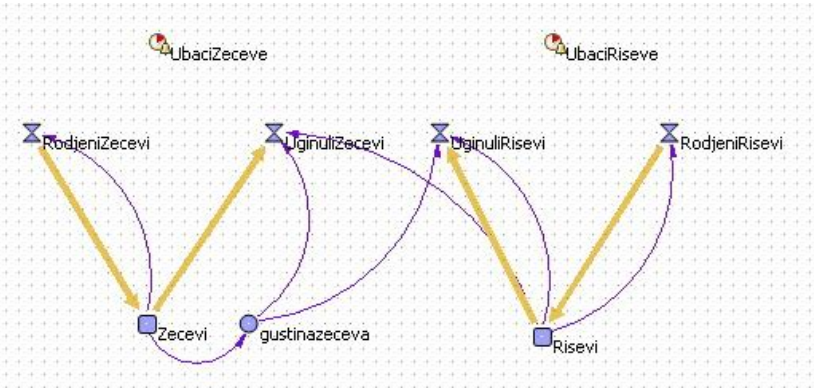
Mortalitet risova je zadat kao tabela pretrage i na osnovu tih podataka je napravljena funkcija pomoću splajn interpolacije.

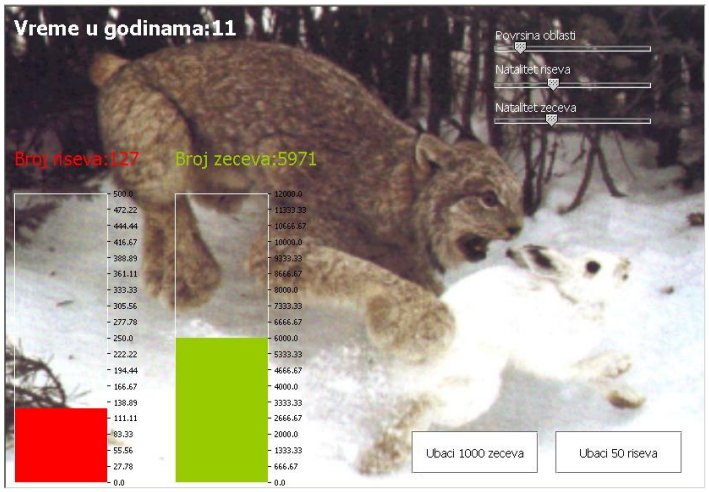


Slika 5.6.Mortalitet risova.

U strukturu su ubačena i dva tajmera koja služe za ubacivanje risova i zečeva u sistem u toku izvršavanja modela. Za oba tajmera je podešeno da rade u manuelnom modu. Prilikom izvršavanja tajmera *UbaciZeceve* izvršava se Java naredba Zecevi+=1000. Za tajmer *UbaciRiseve* izvršava se Risevi+=50.

U aktivnoj klasi objekta nalaze se tri realna parametra *povrsinaoblasti*, *NatalitetZeceva* i *NatalitetRiseva* sa početnim vrednostima.



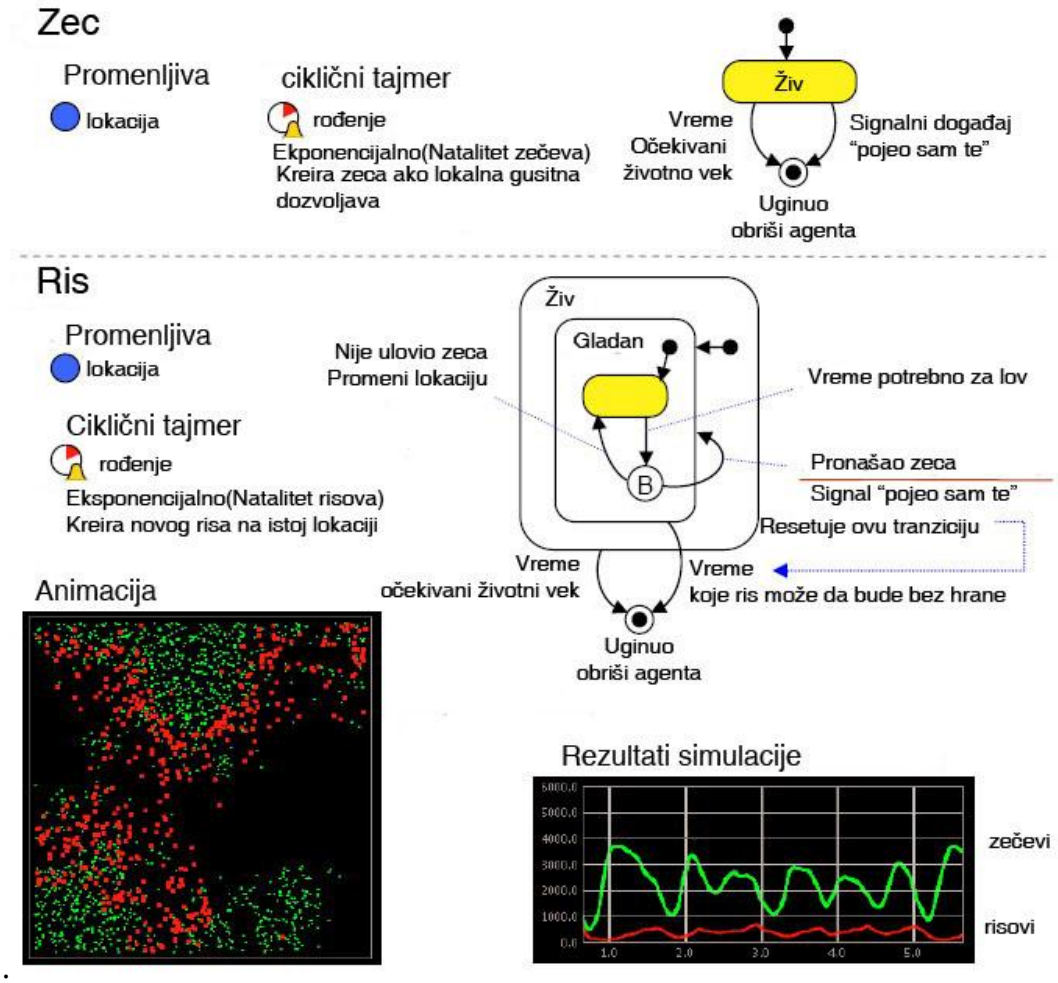


Slika 5.7.Natalitet risova i zečeva.

Animacija je u ovom modelu nešto složenija. Osim prikaza broja jedinki za zečeve i risove brojčano i pomoću dijagrama, ubačeni su i interaktivni oblici. Dva komandna dugmeta su vezana za tajmere u dijagramu strukture koji povećavaju broj jedinki. Tri klizača su vezana za parametre *povrsinaoblasti*, *NatalitetZeceva* i *NatlitetRisova*. Pomoću klizača u toku izvršavanja možemo menjati vrednosti za ove parametre.

Model zasnovan na agentima predstavlja prave mogućnosti *AnyLogic* programa, jer je model pomoću agenata je malo realističniji. Uvedeno je šest pretpostavki, koje su date u nastavku.

- Zečevi i risovi imaju očekivani životni vek. Uginuća su izazvana zbog starosti, izgladnjavanja i napadima risova.
- Zečevi i risovi su svesni dvodimenzionalnog prostora u kome se nalaze.
- Gustina populacije zečeva je ograničena, te se razmnožavaju samo ako ima dovoljno prostora oko njih.
- Risevi se love samo u određenoj oblasti oko njih i to sa određenom učestalošću.
- Ako ris ne ulovi zeca u toku lova menja svoju poziciju u prostoru.
- Ako u određenom vremenskom periodu ris ne pojede zeca.



Slika 5.8. Model lovac-žrtva kreiran pomoću agenata

Svakom agentu je dodeljena promenljiva lokacija koja sadrži podatak o trenutnoj poziciji agenta u prostoru. Početna pozicija svakom agentu se dodeljuje na slučajan način. Vrednost promenljive lokacija se ažurira kada se agent kreće u prostoru i to utiče na njegovo ponašanje.

Tajmeri koji se nalaze u modelu su definisani za rođenja risova i zečeva. Ponašanje tajmera je ciklično i kreira novog agenta po eksponencijalnoj raspodeli.

Rođenja zečeva zavise još i od jnihove lokalne gustine. Za zečeve dijagram stanja je jednostavan i njime je opisano ponašanje zečeva. Stanja u dijagramu su živ i konačno stanje uginuo.

Definisane su dve tranzicije: jedna za signal (poruka) od risova «pojeo sam te!». Druga tranzicija je određenja vremenski i predstavlja očekivani životni vek zeca. Dijagram stanja za risa je mnogo složeniji. Risevi love određeno vreme period lova. Da li će ris naći zeca zavisi od gustine zečeva i broja risova u okolini. Ako pronađe i pojede zeca šalje signal (poruku) zecu «pojeo sam te!» napušta stanje gladan i odmah ponovo ulazi u stanje gladan. Ako nema sreće i ne ulovi zeca pomera se u prostoru ali ostaje u stanju gladan.

Kada ris ulovi zeca, odmah se resetuje vreme koje označava koliko ris može da bude bez hrane pre nego što ugine. U tom vremenu koje ris može da provede bez hrane dovoljan mu je jedan zec da preživi. U modelu su definisane algoritamske funkcije za određivanje da li je oblast oko zeca prenaseljena, funkcija za određivanje nove oblasti za risa na slučajan način.

Rezultat koji se dobija u toku simulacije daje grafik koji jako liči na grafike koji su dobijeni posmatranjem populacija u prirodi. Najveće vrednosti za risove su malo iza najvećih vrednosti za zečeve.

Za razliku od prethodnih modela zbog parametra koji se zadaju, moguće ja da se dogodi totalno izumiranje jedne populacije. U simulaciju je dodata i animirana 2D slika prostora u kojoj se nalaze risovi i zečevi.

Model je napravljen na osnovu modela kompanije *XJtek*.

6. Model katapultiranja pilota

Posmatrajmo avion sa mogućnošću katapultiranja pilota. Ako avion leti brzinom V_p i ako pilot povuče ručicu za katapultiranje, onda započinje proces katapultiranja. Prvo se pali raketni motor ispod sedišta pilota, koji izbacuje pilota iz aviona. Pretpostavimo da se pre napuštanja aviona sedište kreće konstantnom brzinom V_e duž šina koje su pod uglom θ_e u odnosu na avion. Kada pilot sa sedištem napusti avion, na njega utiču dve sile koje mu određuju putanju: otpor vazduha i gravitacija.

Na slici je prikazan otpor vazduha D koje je projektovan na x i y osu koordinatnog sistema koji je vezan za avion. Posle katapultiranja, ima dve mogućnosti. Naime, ili je

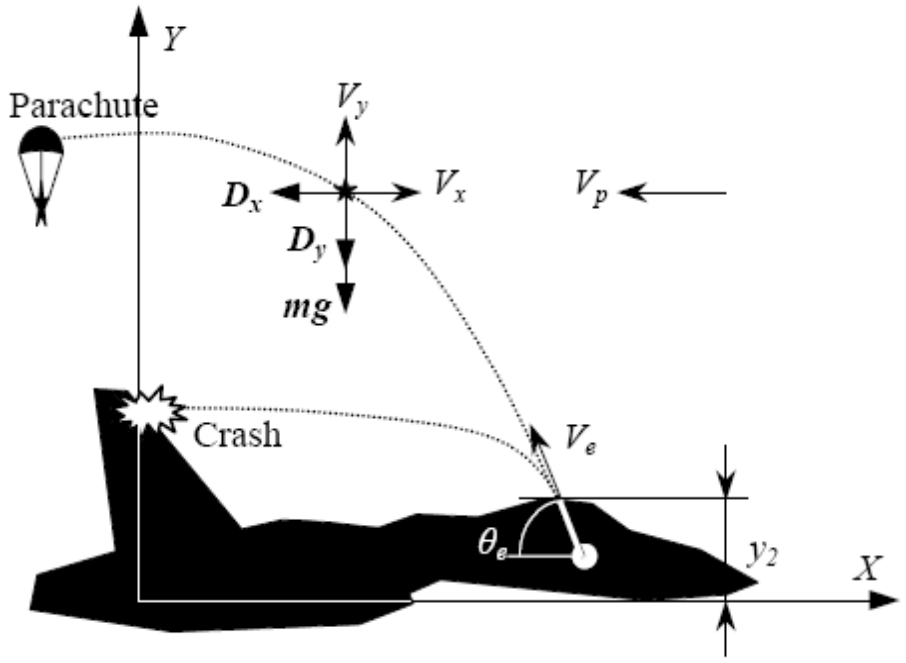
pilot uspešno otvorio padobran i onda se bezbedno spustio na zemlju, ili udara u rep aviona. Da bi saznali kako na katapultiranje modela utiču parametri kao što su brzina aviona, težina pilota i sedišta, ugao pod kojim sedište napušta avion i sam oblik aviona napravimo model ovog sistema.

Očigledno je da postoje četiri diskretna događaja u sistemu:

- povlačenje ručice za katapurltianje;
- sedište napušta avion;
- pilot udara u rep aviona;
- i pilot otvara padobran.

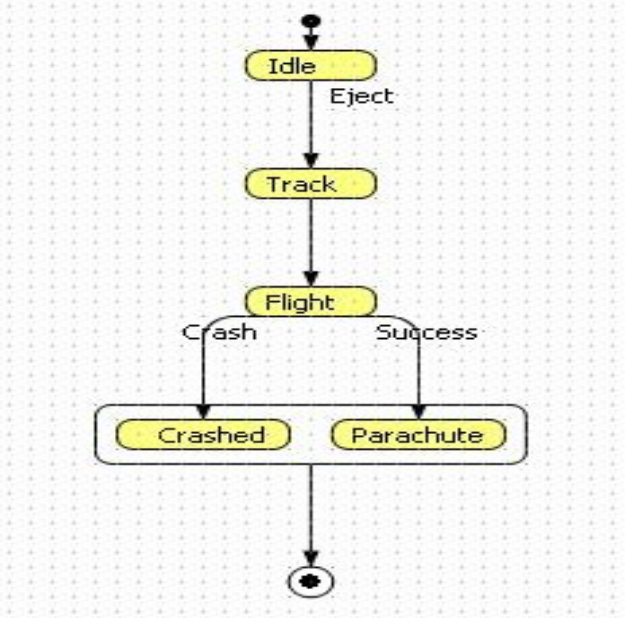
Povlačenje ručice za katapultiranje se modelira komandom, pilot će napustiti avion kada je y koordinata veća od y_2. Logička funkcija proverava da li je sedište udarilo u rep aviona na osnovu koordinata sedišta i geometrije aviona. Padobran se otvara posle nekog vremena t_p od trenutka kada je sedište napustilo avion.

Između ovih diskretnih događaja dinamika pilota se opisuje sa dva sistema jednačina. Jedan je sistem za vreme kada je pilot u avionu, a drugi kada je pilot već napustio avion.



Slika 6.1. Simulacija katapultiranja pilota.

Cilj je da se odredi kojom brzinom je potrebno da se pilot kreće van aviona da bi se izbegao udar u repno krilo aviona. Ovaj model je hibridan pošto se kontiunalno kretanje prekida sa diskretnim događajem.



Slika 6.2. Dijagram stanja katapultiranja pilota.

Ponašanje modela je opisano dijagramom stanja. Početni pokazivač pokazuje na stanje *Idle*. To znači da se pri pokretanju modela pilot nalazi u kokpitu aviona. Prelaz iz stanja *idle* u stanje *Track* dešava se odmah po pokretanju modela, tako što se izabere opcija *fire immediatly* i za tu tranziciju je vezana akcija

```
vx = - ve * Math.sin( tetaE );

vy =   ve * Math.cos( tetaE );
```

U stanju *Track* izračunava se putanja po kojoj se pilot kreće pre napuštanja aviona. Sledeće jednačine opisuju tu putanju:

```
d( y )/dt = vy

d( x )/dt = vx
```

Prelaz iz stanja *Track* u stanje *Flight* čeka da se dogodi diskretan događaj (*if event occurs*) y>y3 definisan u polju *trigger*. U stanju *Flight* sledeće jednačine opisuju putanju i brzinu pilota

$$d(y)/dt = vy$$

$$d(x)/dt = vx$$

$$d(vx)/dt = -rho*cd*sd*(vp+vx)*(vp+vx)/(2*m)$$

$$d(vy)/dt = -g-rho*cd*sd*vy*abs(vy)/(2*m)$$

Iz stanja *flight* postoje dva prelaza. Jedan prelaz je ka stanju *crashed* koje označava da katapultiranje nije bilo uspešno. *Trigger* za tranziciju predstavlja logičku funkcija u Javi koja na osnovu dva parametra računa da li će pilot udariti u repno krilo.

```
double k = ( y3 - y2 ) / ( x3 - x2 );

double b = k * x2 + y2;

if( x >= x2 && x <= x3 && y < k * x + b ) {
```

```
  return true;
}
else
if( x >= x3 && x <= x4 && y < y3 ) {
  return true;
}
else {
  return false;
}
```

Uspešno katapultiranje aktivira prelaz ka stanju *parachute* koje u prelazu ima *trigger* `x <= xSafe`.

Stanja *crashed* i *parachute* su podstanja jednog stanja koje ima prelaz ka završnom stanju koje se događa nakon 0.2 sekunde.

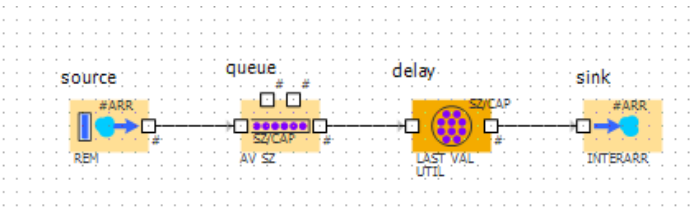
Model koji je ovde opisan predstavlja deo bogate bitbiloteke modela koji se dobijaju uz *AnyLogic*.

7. Univerzalan, jednostavan model masovnog opsluživanja

Model koji će biti opisan može biti primenjen na različite slučajeve masovnog opluživanja. Često se svi susrećemo sa čekanjem u redu da bi „dobili“ neku uslugu. Nebitno je da li je to banka, pošta, red ispred bankomata, kupovina karata za bioskop čekanje u lekarskoj ordinaciji.

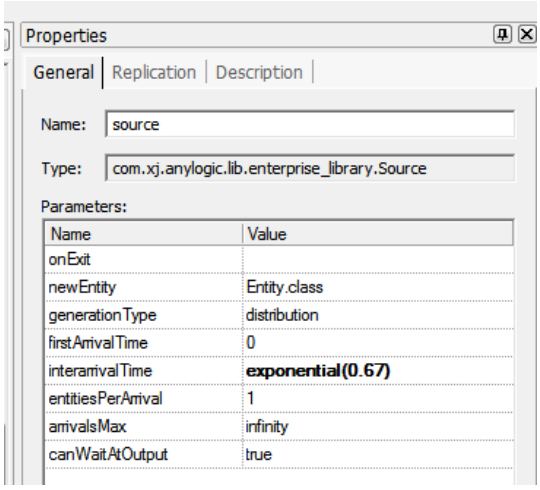
Svi ti modeli imaju zajedničko da klijent dolazi, staje u red, čeka, prima uslugu koja mu je potrebna i napušta sistem. Ovako nešto je vrlo jednostavno modelirati i u stvari veći problem u pravljenju ovakvog modela predstavlja pronalaženje odgovarajućih slučajnih raspodela koje će modelu omogućiti da se ponaša kao i realan sistem koji se posmatra.

Korišćenjem *Enterprise Library* koju *AnyLogic* sadrži, moguće je sastaviti model iz pojedinačnih elemenata. Podešavanjem parametara svakog od elemenata koji se koristi dobija se željeno ponašanje modela. Za sada ćemo koristiti samo četiri elementa i napraviti jednostavan model masovnog opsluživanja.



Slika 7.1. Osnovni model sistema masovnog opsluživanja (MO)

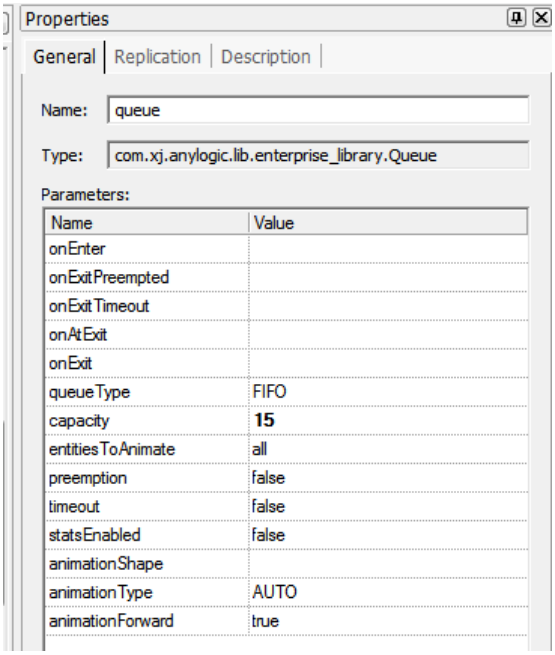
Element *source* se koristiti za generisanje entiteta (klijenata, delova za proizvodnju itd.) koji dolaze u sistem. Parametri za *source* su jednostavni i u pricipu svode se na upisivanje slučajne raspodele po kojoj se generišu entiteti.



Slika 7.2. Parametri sistema MO

U našem primeru dolazak entiteta se dešava po eksponencijalnoj raspodeli sa parametrom 0.67 i u svakom generisanju novog entiteta u sistem se ubacuje tačno jedan entitet zato što je u polje *entetiesPerArriva* upisano *1*. *arrivalMax* definiše maksimalan broj gernerisanih entiteta i upisivanjem *infinity* dobijamo neograničeno mnogo generisanih entiteta, odnosno eniteti će se ubacivati u sistem svo vreme trajanja simulacije modela.

Sledeći element u modelu je *queue* koji omogućava da generisani entiteti čekaju u redu. Kod ovog elementa bitno je definisanje maksimalnog kapaciteta reda, što je u našem slučaju 15.

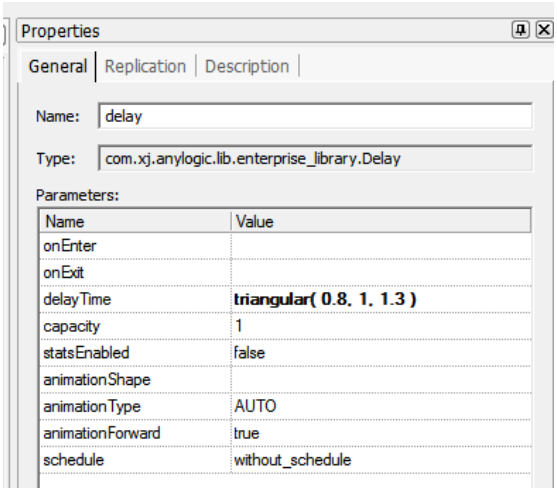


Slika 7.3. Sistem MO u *AnyLogic*-u..

Takođe može se izabrati i tip reda od ponuđenih opcija. U našem slučaju red je tipa FIFO (*first in first out*). Polja *preemtatio* i *timeout* su takođe veoma bitna pošto nam omogućavaju da na jednosavan način modeliramo tipično ponašanje klijenata u ovakvim sistemima. Ako je red velik, klijent neće ni hteti da stane i čeka nego će napustiti sistem. Ako se u polje *preemtation* upiše vrednost *true,* ova opcija se aktivira i u našem modelu.

Šta će se dogoditi dalje sa tim entitetom, moguće je definisati tako što se na izlaz sa elementa queue koji odgovara opciji preemptation poveže nov element iz *Enterprise Library,* na primer *sink* koji uništava entitet. Po istom prinicpu koristi se i opcija *timeout* za predstavljanje onih klijenata koji su stali u red ali im je vreme čekanja predugo pa žele da odustanu.

Element *delay* predstavlja „pružanje usluge“ odnosno na primer vreme koje je klijent potrošio dok je čekao da mu blagajnik na šaleru banke isplati novac.



Slika 7.4. Nešto složeniji sistem MO u *AnyLogic*-u.

U polju *delay time* se upisuje slučajna raspodela koja odgovara vremenu trajanja pružanja usluge (u našem slučaju to je trougaona raspodela sa parametrima 0.8, 1, 1.3 što znači da klijent se zadržava na primer na šaleteru banke minimalno 0.8 vremenskih jedinica, maksimalno 1.3 sa prosečnim vremenom od 1 vremenske jedinice). Polje *capacity* određuje koliko klijenata može biti opsluženo istovremeno.

Element *sink* nema ništa za podešavanje od značaja za ponašanje modela pošto on jednostavno uništava generisane entitete.

Predstavljeni opšti model MO (masovnog opsluživanja) predstavlja polaznu tačku za učenje i razumevanje takvih modela. Proširivanjem ovog modela sa drugim elementima iz *Enterprise Library* se mogu dobiti vrlo složeni modeli kao što su modeli aerodroma, hitne službe, proizvodnje, itd. Dodajmo da se u ovakvim modelima ponašanje modela vezuje za elemente u modelu, a ne za entitete koji se nalaze u modelu, što je slučaj sa agnetnim modelima gde je ponašanje definisano za agenta (entitet).

Bitno je još napomenuti da ovaj model ne uključuje korišćenje resursa, što takođe predstavlja korak bliže ka „realnijem“ modelu, na primer ako se modelira doktorska ordinacija: lekar, medicinska sestra i recimo EKG aparat su resursi koje će pacijent morati

da koristi u toku primanja usluge. Deljenje resursa u stvari predstavlja i najveći problem koji je potrebno rešiti matematčkim modelom. Na pitanje da li će se lekarskoj ordinaciji isplatiti kupovina još jednog EKG uređaja da bi se smanjilo vreme čekanja pacijenata i obavilo više pregleda što znači i veći prihod ordinaciji, ali i zadovoljnije pacijente, može se jednostavno proveriti modelom masovnog opsluživanja.

Literatura

[1] J. Caldwell and J. Douglas K. S. Ng, Mathematical Modeling, Kluwer Academic Publishers, 2004.

[2] H. Caswell, Matrix Population Models: construction, analysis, and interpretation,

[3] Sinauer Associates, Inc. Publishers, 2001.

[4] V. Čerić, Simulacijsko modeliranje, Školska knjiga, Zagreb, 1993.

[5] N. D. Fawkes, J. J. Mahony, An Introduction to Mathematical Modelling, John Willey and Sons, Second Edition, 1996.

[6] R. Haberman, Mathematical Models, Fourth Edition, Prentice-Hall 1977. S. Ross, Simulation, Third Edition, Academic Press, 2002.

[7] M. Haddin, Modelling and Quantitative Methods in Fisheries, Chapman & Hall/CRC, 2001.

[8] J. A. Hamilton, Jr., D. A. Nash, U. W. Pooch. Distributed Simulation, CRC Press, 1997.

[9] Mark M. Meerschaert, Mathematical Modeling, Second Edition, Academic Press, 1998.

[10] D. Mooney, R. Swift, A Course in Mathematical Modeling, Mathematoical Association of America, 1999.

[11] A. Takači (with L. Juhas and D. Mijatović), Skripta za matematičko modeliranje (in Serbian), WUS, Belgrade, and Faculty of Sciences, Novi Sad, 2006.

[12] B. P. Zeigler, H. Praehofer, T. G. Kim, Theory of Modelling and Simulation, Second Edition, Academic Press, 2000.

[13] AnyLogic User Manual, XJ Technologies, 2005.