Implementacija sistem-dinamičkih modela u edukativnim računalnim igrama korištenjem LIBGDX okvira

Pačandi, Hugo

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

HUGO PAČANDI

IMPLEMENTACIJA SISTEM-DINAMIČKIH MODELA U EDUKATIVNIM RAČUNALNIM IGRAMA KORIŠTENJEM LIBGDX OKVIRA

Diplomski rad

Pula, rujan 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

HUGO PAČANDI

IMPLEMENTACIJA SISTEM-DINAMIČKIH MODELA U EDUKATIVNIM RAČUNALNIM IGRAMA KORIŠTENJEM LIBGDX OKVIRA

Diplomski rad

JMBAG: 0016092403, redoviti student

Studijski smjer: Diplomski sveučilišni studij Informatika

Predmet: Mobilne aplikacije

Mentor: doc.dr.sc. Siniša Sovilj

Pula, rujan 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Hugo Pačandi, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Hugo Pačandi

U Puli, 19. rujna 2018. godine

IZJAVA

o korištenju autorskog djela

Ja, Hugo Pačandi dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Implementacija sistem-dinamičkih modela u edukativnim računalnim igrama korištenjem LibGDX okvira“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 19. rujna 2018. godine

Potpis

SVEUČILIŠTE JURJA DOBRILE U PULI
ODJEL ZA INFORMACIJSKO-KOMUNIKACIJSKE TEHNOLOGIJE

Pula, 1. ožujka 2018.

DIPLOMSKI ZADATAK

Pristupnik	**Hugo Pačandi (0016092403)**
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	Implementacija sistem-dinamičkih modela u edukativnim računalnim igrama korištenjem libGDX okvira.
Naslov (eng.):	Implementation of system dynamics models in educational computer games using libGDX framework.
Opis zadatka:	Zadatak je razviti metodologiju implementacije računalnih sistem-dinamičkih (SD) modela na Android platformi. Osnovni blokovi SD modela nazivaju se „strukturne molekule" te je za više njih potrebno izraditi edukativne simulacije u obliku računalnih igara koristeći libGDX programski okvir. U sklopu razvoja istražiti prednosti i nedostatke libGDX-a u razvoju edukativnih igara i simulacija te ga usporediti s nekoliko drugih popularnih programa za razvoj računalnih igara: Unity, Unreal Engine, Cocos2D.

Zadatak uručen pristupniku: 1. ožujak 2018.
Rok za predaju rada: 1. veljače 2019.

Mentor:

Siniša Sovilj

doc.dr.sc. Siniša Sovilj

Kazalo

1. Uvod

Početak ovog diplomskog rada posvećen je sistemskoj dinamici i modelima kojima se rješavaju problemi sistemske dinamike. Sistemska dinamika je metodologija nastala sredinom 1950-ih godina koja služi za prikaz toka procesa kompleksnih sustava u simulacijskom modelu. Metodologija se oslanja na sustavno razmišljanje s povratnim vezama, kao i dijagrame kauzalnih veza kao osnovne metode za kreiranje sistem-dinamičkih modela. Sustavno razmišljanje omogućuje da se na temelju modela donesu ispravne odluke uzimajući u obzir što više varijabli, mogućih rezultata i opcija. Dijagrami kauzalnih veza su grafički prikazi ponašanja sustava na principu kauzalnih (posljedičnih) veza. Oni se sastoje od varijabli i veza kroz koje podaci kruže i mogu biti rastući i padajući.

Budući da se sistemska dinamika danas većinom odvija pomoću računala u radu sam spomenuo i nekoliko programskih alata koji služe za sustavno modeliranje. Odabrao sam najpopularnije kroz povijest sistemske dinamike i ukratko ih objasnio.

Prva dva poglavlja služe za lakše razumijevanje generičkih struktura odnosno molekula koje su fokusna točka ovog diplomskog rada. To su standardizirani dijelovi modela, koji predstavljaju neku situaciju u modelu, koja se često javlja kod sistem-dinamičkih modela. One su popisane od strane društva sistemske dinamike i postoji ih oko sedamdesetak. U ovom poglavlju fokusirao sam se na njihovo približavanje i pojašnjavanje te pojasnio njihovo spajanje i kako se iz njih formulira model. Detaljnije sam obradio nekoliko češće korištenih i jednostavnijih molekula sa općom primjenom i prikazao sam ih na primjeru u alatu Vensim. Uz to, za svaku sam naveo i matematičke formule prema kojima se mogu izračunati neke njihove varijable i tokovi.

Kako bih usporedio dva alata, Unity i LibGDX, potrebno mi je bilo steći iskustva sa svakim od njih. Sa Unity-jem sam se bavio tokom studija kroz izradu projekata i kolegije te sam o njemu stekao iskustva već prije pisanja ovog diplomskog rada. Da bih ga mogao usporediti sa LibGDX-om, morao sam istražiti i taj alat. To sam učinio razvojem edukativne dvodimenzionalne igre u tom alatu koja implementira sistem-dinamički model i njegove molekule o kojima sam pisao u prvom djelu ovog diplomskog rada. Nakon izrade aplikacije smatram da sam stekao dovoljno znanja o alatima kako

bih ih mogao usporediti, pronaći njihove prednosti i mane te izraziti svoje mišljenje o svakome od njih.

Stoga sam kraju rada napravio usporedbu navedenih alata, Unity i LibGDX, koji služe za izradu računalnih igara. Definirao sam nekoliko kriterija prema kojima sam ih usporedio te sam prema njima napravio usporedbu i donio zaključke. Iz same usporedbe prema danim kriterijima vidljivo je da je LibGDX alat za napredniju korisničku bazu, nego što je to Unity, te nudi mogućnosti za razvoj naprednijih i jačih igara za manji spektar platformi. Unity je alat koji pokušava biti dobar u svemu te u isto vrijeme zadovoljiti sve korisnike od početnika do profesionalaca, što nije lako. On također podržava i veliki broj platformi i time isto pridobiva mnogo korisnika.

U smislu edukativnih igara, sve ovisi o tipu igre te koja se kompleksnost želi postići, kakva je struktura igre i što se želi postići. U oba alata može se napraviti jednostavnija edukativna igra koja na primjer implementira sistem-dinamički model kao ona koju sam ja razvio za potrebe ovog diplomskog rada. Oni nude dovoljno funkcionalnosti i alata za razvoj te uz to i mnogo više od potrebnog za ovakvu edukativnu igru.

2. Sistem-dinamički modeli

2.1. Povijest sistem-dinamičkih modela

Sistemska dinamika, kao tehnika i metodologija javlja se prvi puta na MIT-u (Massachusetts Institute of Technology) sredinom 1950ih godina. Za njezin nastanak zaslužan je profesor Jay Forrester koji je tih godina postao predavač na institutu te je započeo sa simulacijama poslovnih procesa. U početku, njegove simulacije su se odnosile na ljudske resurse, zapošljavanje i upravljanje te ih je vršio uz pomoć papira i olovke. Specifično, simulirao je kako bi zapošljavanje ljudi utjecalo na zalihe i proizvodnju, oscilacije i nestabilnost sustava te primjenu kod donošenja odluka u organizaciji. Kako bi što jasnije predstavio svoju simulaciju, za primjer je uzeo „Beer game“ – igra u kojoj timovi nastoje što točnije simulirati potražnju piva i opskrbni lanac uz minimiziranje zaliha. Ta simulacija je predstavljala početak sistemske dinamike.

Kako su se simulacije pokazale uspješnima, te kako bi ih dalje unaprijedio, započeo je suradnju sa programerom Richardom Bennettom. Zajedno su u simulacije počeli primjenjivati računalo, te su napravili programski jezik koji je omogućavao automatsko kreiranje koda za simulacije i nazvali ga SIMPLE – Simulation of Industrial Management Problems with Lots of Equations. Kasnije je taj jezik proširen i implementiran u poznati programski jezik DYNAMO. (Systemdynamics, 2018.)

Početkom 1960ih godina Forrester je objavio knjigu Industrial Dynamics koja je predstavljala metodologiju i filozofiju primjene simulacija u industriji. Time se sistemska dinamika počela sve više primjenjivati u menadžerskim pitanjima i problemima, no ne i šire u drugim područjima. Krajem 1960ih godina John F. Collins, bivši gradonačelnik Bostona, postao je profesor na MIT-u, što je značilo suradnju sa Forresterom. Zajedno su počeli istraživati urbane probleme i na njih primjenjivati modele te je tako nastala urbana dinamika – primjena sistemske dinamike na urbana naselja. Zaključili su da je popularna politika gradnje jeftinih nastamba zapravo štetna te se do tada vjerovalo da je to pravi način za oživljavanje gradova. Simulacija je objavljena u knjizi Urban Dynamics i dokazala je da gradnja jeftinih nastamba u gradovima zauzima mjesta na

kojima se mogu stvarati poslovi te dovodi više ljudi u grad koji su siromašni i kojima su potrebni poslovi. Takvi rezultati modela poprimljeni su vrlo burno, budući da su bili kontradiktorni tadašnjim vjerovanjima, stoga se javljaju prve reakcije i problemi. Suprotne strane smatrale su da se modelom žele smanjiti prava siromašnih te da su urbani i rasni problemi zapravo usko povezani. Kroz sljedeće dvije godine ispostavilo se da je Urban Dynamics u pravu te da MIT ima jedino rješenje. Uskoro se počinju javljati novi projekti koji koriste sistemsku dinamiku te se ona širi diljem svijeta. (Jay W. Forrester, 1996.)

U 1970. godini uprava Rimskog kluba (eng. Club of Rome), neprofitne organizacije s ciljem zalaganja za probleme čovječanstva kao što su okoliš, siromaštvo, bolest i kriminal, sastaje se sa Forresterom kako bi raspravili o rješavanju takvih problema. Tada su postavljene osnove za model nazvan WORLD1 koji je predstavljao širenje sistemske dinamike na cijeli svijet, te se ubrzo širi i u Njemačku gdje se javlja i suradnja sa Volkswagenom. J. Forrester napisao je knjigu sa istim nazivom – World Dynamics koja je ubrzo postala popularna iako to nije očekivao, budući da je knjiga bila vrlo stručna, puna računica, grafova koji su nepristupačni javnosti te je također bila izdana od nepoznatog izdavača. Knjiga je izdana u lipnju 1971. te su se recenzije počele javljati u mnogim popularnim novinama i časopisima, kao što su London Observer, Singapore Times, Fortune, Wall Street Journal pa čak i Playboy-u. Ubrzo nakon toga objavio je i drugu knjigu, Limits To Growth, koja je također doživjela uspješan odaziv javnosti te je još više povećala popularnost sistemske dinamike. (ClubOfRome, 2018.)

U 1972. godini, Institute Of Electrical and Electronics Engineers (IEEE) dodjeljuje Jay W. Forresteru nagradu (Medail of Honor) za svoje doprinose na području računalstva i sistemske dinamike (ETHW, 2017.).

Kroz sljedeća dva desetljeća sistemska dinamika počinje se primjenjivati u širim i dosad nezastupljenim područjima. Najprije je nastao nacionalni model iz kojega se sistemska dinamika počela primjenjivati u ekonomskim programima. Tamo se sistemska dinamika opet susreće sa problemom ljudskog mišljenja – specifično, promjena ljudskog odlučivanja, što je zahtjevan proces te je vrlo teško promijeniti ljudsko mišljenje i navike. Mišljenje se ne mijenja logikom, već sistemska dinamika nastoji uključiti kadrove organizacije u modeliranje procesa, kako bi dobili saznanja o procesu te kako bi odmah mogli izraziti svoje mišljenje i dati povratnu informaciju. Takvim pristupom, kasnijih godina sistemska dinamika podučava sve mlađe generacije

pa se počinje javljati i u školama. Za takve napretke zaslužan je Barry Richmond, američki znanstvenik koji se bavio dinamičkim sustavima i koji je sistemsku dinamiku implementirao u nekoliko desetaka osnovnih i srednjih škola, kako bi se sistemska dinamika širila od malih nogu. On je primjenjivao STELLA software za edukaciju mladih, koji je bio pristupačan i jednostavan za korištenje, te je kasnije dobio i nagradu za vodič namijenjen tom programu. U svojem radu Beginning of System Dynamics, Jay W. Forrester navodi primjer uporabe STELLA software-a u osmom razredu biologije, gdje je njegov kolega profesor Frank Draper počeo primjenjivati software u nastavi. Najprije ga je planirao koristiti jednom ili dva puta po semestru, no ubrzo je software postao dio svakog predavanja. To je dovelo do razmišljanja da neće stići odraditi planirano gradivo biologije, budući da je trošio mnogo vremena na sistemsku dinamiku. No nakon što je prošlo dvije trećine semestra, profesor je obradio svo gradivo zbog većeg angažmana učenika u nastavi te drugačijim pogledima na gradivo zahvaljujući sistemskoj dinamici. (Mckinsey Quarterly, 1995.)

U edukaciji menadžmenta, sistemska dinamika može se usporediti sa studijama slučaja, budući da oboje počinju prikupljanjem podataka o organizaciji. Prednost sistemske dinamike je to što ona te informacije dalje koristi za modeliranje, provedbu eksperimenta i testova te utjecaje promjena na rezultat.

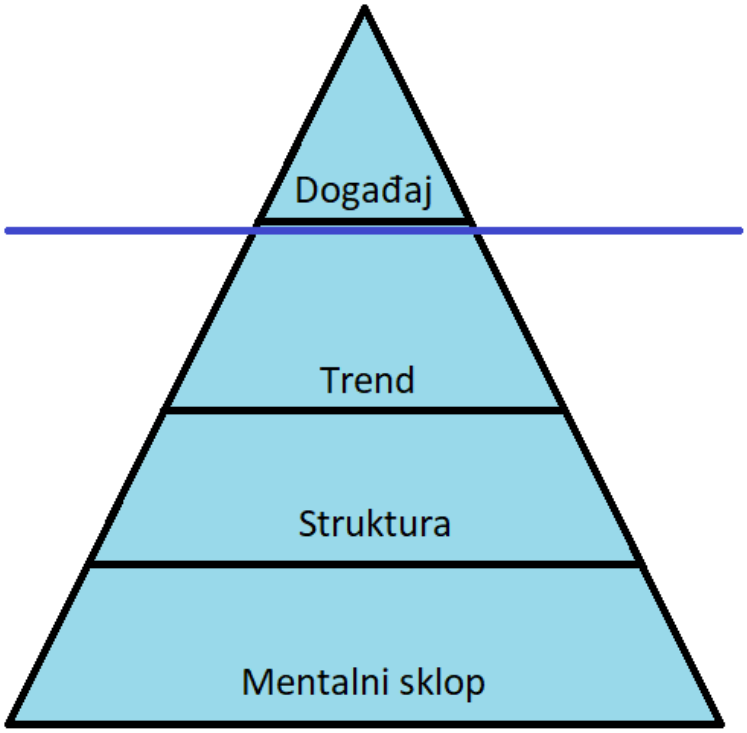
Forrester navodi kako se u budućnosti može očekivati naglasak na generičke modele, koji bi se nalazili i koristili pretežito u više modela, te bi se oni mogli dalje proširivati, kako bi sistemska dinamika omogućila što jednostavniji i učinkovitiji model za rješavanje što kompleksnijih problema. (Jay W. Forrester, 1996.)

2.2. Sustavno razmišljanje s povratnim vezama

Sistemska dinamika je kompleksna metodologija te kako bi ju se lakše pojasnilo potrebno je početi od jednostavnih dijelova sistemske dinamike – modela. No sistemska dinamika nisu samo modeli, već proces koji se sastoji od mnogo dinamičkih koraka. Stoga je potrebno otići još dublje u pojam sistemske dinamike te krenuti od samog koncepta razmišljanja, točnije sustavnog razmišljanja s povratnim vezama. Sustavno razmišljanje s povratnim vezama je osnova za pristup sistemskoj dinamici, a prikazuje se dijagramima tokova informacija kako bi se lakše predstavili, predočili i razumjeli kompleksni problemi kojima se bavi sistemska dinamika. (Craig W. Kirkwood, 1998.)

Sustavno razmišljanje je filozofija koja nudi metode i alate za proučavanje događaja i podataka, pronalazak uzoraka i promjena u ponašanju, bilo ljudi, organizacija, modela i slično. Ono omogućuje da odluke koje se donose budu informirane te, iako nije garantirano da je odluka ispravna, šanse za donošenje ispravne odluke su mnogo veće. Korištenjem takvih metoda i alata može se utjecati na broj izbora koji su ponuđeni i pronaći nova rješenja koja do tada nisu bila vidljiva. U isto vrijeme, takvo razmišljanje prikazuje kako će neka odluka utjecati na druge varijable te pomaže nam predvidjeti i lakše odabrati kompromis koji je potreban kod donošenja i odabira odluka. Sustavno razmišljanje koristi se kod važnih problema, koji se ponavljaju te im je poznata povijest i još nisu riješeni (Daniel H. Kim, 1999.).

Rješavanje problema sustavnim razmišljanjem predstavlja postepen pristup rješavanju, na način da se problem dijeli na manje sustavne cjeline. Tu se često koristi koncept sante leda koji pomoću piramidalnog prikaza koji predstavlja problem rastavljen na događaj (eng. event), trend (eng. pattern), strukturu (eng. structure) i mentalne sklopove (eng. mental models).



Slika 1. Koncept sante leda za rješavanje problema sustavnim razmišljanjem

Na slici je prikazana takva struktura sa četiri razine na kojoj rješavanje problema počinje na vrhu te se zatim spušta na niže razine. Na najvišoj razini vidi se samo vrh piramide koji predstavlja najmanji dio problema. Taj dio je vidljiv odmah i on se nalazi iznad *razine vode* koja je predstavljena plavom vodoravnom crtom. Spuštanjem kroz razine dubina problema se širi te se tako i saznaju nove informacije korisne za donašanje ispravne odluke i rješenja problema.

Događaji se nalaze na najvišoj razini sante leda te predstavljaju dio koji je vidljiv svima (dio sante leda koji viri iz vode). To je dio problema koji je jasan i vidljiv golim okom, no problem je zapravo mnogo dublji. Kao primjer događaja izvor navodi: „Žena kasni na posao nakon što je ostavila dijete u vrtiću. Njezin nadređeni ovaj puta razumije i opravdava kašnjenje.“ Iz navedenog primjera može se primijetiti da je kašnjenje događaj koji se dogodio jednom te bez dodatnih informacija zaključujemo da problem nema dubinu.

No promatranjem problema dolazi se spoznaje da problem ima veću dubinu nego je ona vidljiva na prvi pogled. To je grafički prikazano dijelom sante leda koja je ispod vode. Ta dubina problema je zapravo trend, koji predstavlja neko ponavljanje istih

događaja – „Žena opet kasni na posao tjedan dana kasnije“. Kako bi se potvrdilo da je ovaj događaj trend, a ne novi zasebni događaj, najprije ga je potrebno usporediti sa prethodnim te utvrditi jesu li oni povezani. Ako jesu povezani, znači da je problem postao dublji, nego što je najprije pretpostavljeno, te predstavlja trend u ponašanju.

Na sljedećoj razini pokušava se objasniti problem trenda – zašto je do njega došlo te na koji način ga se može otkloniti. Ta razina predstavlja strukturu, razlog zbog kojeg dolazi do ponašanja na višim razinama sante leda. Najčešće je predstavljena sa pravilima, politikom i procedurama u organizaciji koji utječu na ponašanje jedinaca. Na danom primjeru žene koja kasni, najprije je potrebno pitati se zašto ona kasni svaki tjedan te kako riješiti taj problem. U ovom slučaju, stroga organizacijska politika zahtijeva od zaposlenika da budu prisutni na poslu za vrijeme strogo definiranog radnog vremena, bez uzimanja u obzir kada se otvaraju dječji vrtići. Žena nije u mogućnosti ostaviti dijete u vrtiću prije odlaska na posao jer je on zatvoren, nema drugu opciju te je prisiljena na kašnjenje na posao. Kada je uzrok problema ustanovljen postavlja se drugi problem – „Kako ga riješiti?“. Potrebno je pronaći rješenje problema pa se pronalazi nekoliko mogućih rješenja s obje strane. Na primjer, žena može zaposliti dadilju ili se za problem može pobrinuti i organizacija omogućavajući ženi fleksibilno radno vrijeme ili rad od kuće. Ovisno o obliku problema postoji jedno i više rješenja te svako od njih ima svoje prednosti i mane, stoga je zadatak pronaći optimalno rješenje.

Četvrta razina predstavlja mentalne sklopove i još dublji pogled na razine iznad. Ova razina direktno je povezana s razinom iznad te nudi detaljniji pogled na donašanje rješenja problema. Problem se gleda s aspekta zadovoljstva te se uzima u obzir i politika organizacije. Optimalno rješenje pronalazi se na način da se pokušava uskladiti postojeća politika kompanije zadovoljstvom i interesima zaposlenika. Računa se isplativost svake odluke te se donosi najbolje rješenje. Od navedenih mogućih rješenja na razini iznad analizira se zadovoljstvo obje strane kod svakog rješenja problema. Na primjer rješenje problema zapošljavanjem dadilje smanjuje zadovoljstvo žene, budući da mora sama financirati dadilju, dok je pogodno za organizaciju jer nema potrebe za mijenjanjem politike i pravila. No tu se javlja nova varijabla – nezadovoljstvo žene koje utječe na organizaciju indirektno, na način da je nezadovoljan radnik manje produktivan. Stoga je potrebno analizirati sva rješenja što je dublje moguće te pronaći optimalno.

Santa leda može biti i dublja od četiri razine ovisno o kompleksnosti i dubini problema, te svaka razina omogućava da se problem sagleda dublje i sa šireg aspekta. Bez obzira na razine, bitno je da se obuhvati cijeli sustav kako bi se što lakše donijele ispravne i optimalne odluke, te kako bi se proširilo znanje o sustavu koji se promatra i unaprijedilo ga se. (Ed Cunliff, 2018)

2.3. Dijagrami kauzalnih veza

Dijagrami kauzalnih veza zasnivaju se na kauzalnim modelima koji su prikazani dijagramima. Kauzalni modeli su mehanizmi sustava koji pomažu donesti zaključke na temelju varijabli i odnosa između njih. Prikazuju se grafičkim dijagramima pomoću kojih se vizualiziraju uzročne veze između varijabli u modelu. Dijagram se sastoji od varijabli koje su definirane u nekom rasponu modela. Varijable se povezuju strelicama i time se prikazuju njihove veze te utjecaj jedne varijable na drugu. Veze mogu biti pozitivne i negativne. Pozitivne veze označavaju da ako se jedna varijabla povećava, druga će se isto povećavati, dok za negativne veze vrijedi suprotno – jedna raste, a druga se smanjuje ili obrnuto. Dijagrami kauzalnih veza imaju dvije osnovne petlje - jačajuću (eng reinforcing) i balansirajuću (eng balancing). (Daniel H. Kim, 1992.).

Jačajuća petlja odnosi se na dijagram u kojem se prolaskom kroz petlju ukupna vrijednost povećava ili smanjuje konstanto. Kod sljedećeg ulaska u petlju, početna varijabla opet je uvećana (ili umanjena) za određenu vrijednost. Kao jednostavan primjer predstavit ću štedni račun u banci gdje postoji početna svota novca i kamatna stopa. Početna svota novca stavlja se na štednju te se ona povećava za kamatnu stopu te je, nakon prvog kruga štednje, svota uvećana za neki postotak. Tada se na štednju može staviti veća svota novca nego što je bila u prvom krugu štednje, što zapravo predstavlja jačajuću petlju. (Gene Bellinger, 2004.)

Balansirajuća petlja predstavlja dijagram sa varijablama koje međusobno negativno utječu jedna na drugu. Dok jedna varijabla raste, druga se smanjuje te obrnuto. Koristi se kada se na temelju nekog trenutnog stanja želi predvidjeti buduće ili željeno stanje kroz neku akciju. Primjer takve balansirajuće petlje može se prikazati pomoću pada populacije ljudi. Varijabla smrtnost je u suprotnom odnosu od varijable

populacije. Kada se smrtnost povećava, populacija će padati, te kada smrtnost pada, populacija raste. (Gene Bellinger, 2004.)

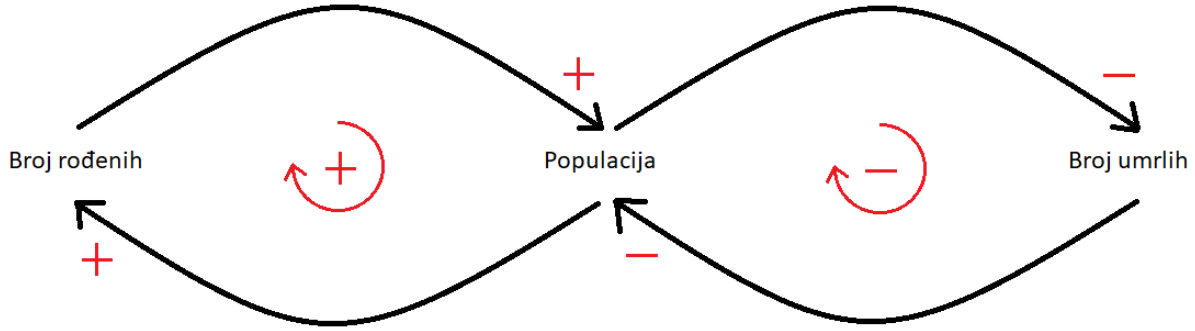
Kauzalni dijagrami pojavili su se davno prije sistemske dinamike, početkom 20. stoljeća, no u obliku grafova koji nisu uključivali petlje, već samo utjecaj varijabli u grafu. Takvi grafovi nazivaju se usmjereni aciklički grafovi. Kasnije su iz grafova nastale petlje te su daljnje proširene upotrebnom računala.

Proces kreiranja kauzalnog dijagrama veza nije striktno definiran, no postoje smjernice kojih se valja držati kako bi se što lakše dobio točan dijagram. Najprije je potrebno odabrati temu koju će dijagram prikazivati u smislu procesa i problema koji dijagram prikazuje. Bez teme nema smisla izrađivati dijagram jer takav dijagram ne prikazuje smislene podatke, ne pomaže kod rješavanja problema te samim time ne služi kod planiranja procesa. Uz temu, potrebno je definirati i vrijeme trajanja ili najmanje neki interval promatranja problema, kako bi se rješavanje problema moglo predvidjeti u tom vremenskom periodu. Na primjer, to može biti dulji vremenski period od nekoliko godina za neke velike procese, kao na primjer restrukturiranje organizacije ili kraći u smislu razvoja novog proizvoda kroz nekoliko tjedana ili mjeseci. Vrijeme utječe na mnoge varijable pa ih je potrebno ispravno definirati jer se one mijenjaju kroz vrijeme. To se naziva ponašanje varijabli kroz vrijeme te je vrlo bitan korak kod kreiranja kauzalnog dijagrama toka. Najprije je potrebno definirati varijable koje utječu na proces te zatim utvrditi njihovo ponašanje kroz tok, odnosno vrijeme. Ključne varijable moraju biti usklađene sa procesom te predstavljati njegovo trenutno stanje kako bi se što lakše predvidjelo buduće. Buduće stanje je nemoguće točno predvidjeti, no cilj je ostvariti što točnije stanje pa je zato bitno ispravno definirati varijable i njihovu promjenu kroz vrijeme. Sljedeća točka kreiranja dijagrama odnosi se na granice modela koje je potrebno postaviti. Koliko varijabli dodati u dijagram? Koliko ih je dovoljno za prikazati problem? Kako ih povezati? Što modelirati? To su samo neka pitanja koja se koriste kod postavljanja granica modela i ona pomažu definirati što se sve obuhvaća modelom. Ovisno o veličini problema potrebno je napraviti i dijagram veličine koja zadovoljava da se problem rješava efektivno. Bitno je uzeti u obzir i vrijeme koje je definirano u prethodnim koracima te varijable postaviti tako da ne prikazuju događaje, već neko ponašanje u problemu. Na kraju se pregledavaju veze između varijabli kako bi se pronašla usporavanja, greške i kašnjenja koja narušavaju dijagram toka. (Daniel H. Kim, 1992.)

Sa samog tehničkog aspekta bitno je ispravno definiranje varijabli kod crtanja dijagrama. To se odnosi na imenovanje varijabli za koje se preferira korištenje imenica, kako bi se izbjegla konfuzija s vezama za koje se koriste glagoli. Kombinacije se isto tako ne preporučaju, budući da, na primjer, varijabla „*rast plaća*“ predstavlja plaće koje mogu rasti i padati. Promjena „*pad rasta plaća*“ zbunjujuća je zbog loše definirane varijable i optimalno bi bilo varijablu nazvati samo plaće. Druga smjernica za odabir varijable odnosi se na mjerivost varijable. Preferira se da varijabla predstavlja nešto mjerivo, kao na primjer novac. Varijable bi također trebale biti imenovane u pozitivnom smislu – ako varijablu nazovemo „*rast*“ što predstavlja pozitivan smisao, pad „*rasta*“ je mnogo jasniji od rasta „*pada*“ koji bi dobili da smo varijablu nazvali „*pad*“.

Kod konstruiranja petlje, prilikom dodavanja veza između varijabli, potrebno je razmišljati o posljedicama koje se događaju kod promjene varijable. Na primjer, kod intenzivne proizvodnje raste broj proizvoda i stres zaposlenika te se smanjuje kvaliteta proizvoda. Bitno je u obzir uzeti varijable i kako one međusobno utječu jedna na drugu. Potrebno je definirati ciljeve koje želimo postići u dijagramu u smislu postavljenih granica koje se mogu postići nakon simulacije dijagrama. Neke varijable je potrebno dodatno definirati – točnije bitno je razumjeti kada koristiti dvije varijable umjesto jedne. Najbolji primjer za to je varijabla „*kvaliteta*“, koju se dijeli na dvije slične varijable: „*željena kvaliteta*“ i „*stvarna kvaliteta*“. Te varijable imaju mjerljivu razliku te doprinose točnosti modela i prikazu stvarnog stanja. Isto tako vrijedi i suprotno, ako varijabla ima više istih rezultata, može ih se grupirati u jednu varijablu kako bi se izbjegla nezgrapnost i dobilo na preglednosti i jednostavnosti dijagrama. Na kraju kod definiranja veza bitno ih je naznačiti ovisno o duljini trajanja. Koriste se dulje strelice za dugotrajne akcije te kraće strelice za kratkotrajne akcije.

Na kraju je potrebno provjeriti varijable te ako nisu jasno definirane, podijeliti ih na više ili dodati nove varijable kako bi dijagram bio što očitiji. Također kako bi se što lakše utvrdilo da li je petlja balansirajuća ili jačajuća, prebroje se negativne veze označene sa „*o*“ ili „-“ te ako ih je neparan broj tada je petlja balansirajuća, a ako je paran onda je jačajuća. (Colleen Lannon, 2018.)



Slika 2. Dijagram kauzalnih veza za razvoj populacije

Na dijagramu iznad prikazan je razvoj populacije pomoću kauzalnih veza. Definirane su tri varijable: *populacija, broj rođenih* i *broj umrlih*, te veze između njih. Vidljivo je da, ako *broj rođenih* raste, raste i *populacija,* te vrijedi i obrnuto da, ako raste *populacija*, raste i *broj rođenih*. Takva veza predstavlja jačajuću petlju zato što sadrži paran broj istih polariteta – pluseva. Isto vrijedi i za desnu stranu dijagrama koja prikazuje odnos populacije i broja umrlih, samo što je ona negativnog polariteta. Zajedno ove dvije petlje čine dijagram koji je suprotnog naboja, odnosno balansirajući dijagram kauzalnih veza. Unutar obje petlje na slici prikazan je polaritet petlje te je ukupan broj suprotnih polariteta neparan što predstavlja balansirajući dijagram. Populacija oscilira ovisno o *broju rođenih* i *broju umrlih* te se mijenja svakim prolaskom kroz petlju. Rast ili pad populacije nije garantiran, već se kroz svaku petlju populacija balansira ovisno o vrijednostima drugih varijabli.

2.4. Sistemska dinamika

Sistemska dinamika je metodologija koja pomoću matematičkog modeliranja nastoji pronaći optimalno rješenje za neki postavljeni problem. Problemi i sustavi kojima se bavi izuzetno su kompleksni i veliki te se oni najčešće javljaju unutar velikih organizacija ili na globalnoj razini. Za rješavanje takvih problema sistemska dinamika koristi se prethodno spomenutim metodama – sustavnim razmišljanjem s povratnim vezama te dijagramima kauzalnih veza. One su međusobno povezane te prilikom

kreiranja dijagrama, koji služi za grafičku analizu i predviđanje, koristi se sustavno razmišljanje s povratnim vezama, kako bi se proučila dubina problema i pronašlo optimalno rješenje.

Sistemska dinamika je i pogled na teoriju sustava (znanost o sustavima) koja pruža metode za razumijevanje kompleksnih sustava. Kao takva nastoji prepoznati strukturu sustava, veze između komponenata, njihovo ponašanje, tok podataka i ponašanje vremena u sustavu. Budući da je sustav teško promatrati kao cjelinu zbog kompleksnosti i veličine, on se dijeli na elemente čije se ponašanje zatim proučava zasebno i na temelju tog proučavanja donose se zaključci.

Od njezinog nastanka sredinom 20. stoljeća, sistemska dinamika puno je napredovala i znatno je modernizirana te je najveći pomak uzrokovan primjenom računala u sistemskoj dinamici. Danas se bez računala sistemska dinamika ne može ni zamisliti, budući da su problemi, koji se simuliraju, toliko kompleksni da uvjetuju uporabu računala. Kod simuliranja problema, koriste se tokovi i akumulativne varijable koje kroz određeno vrijeme daju neki rezultat iz kojeg se donosi zaključak te se na temelju njega poduzimaju akcije. Vrijeme, tokovi, akumulacije, varijable i zaključci glavini su dijelovi svakog sistem-dinamičkog modela. Akumulacije i tokovi osnovni su elementi strukture sistemske dinamike bez kojih ne postoji model. Akumulacije se još nazivaju i varijable stanja i predstavljaju stanje elemenata sustava te omogućavaju dinamičnost modela koji je uzrokovan stalnim dotokom informacija kroz tokove (veze) iz drugih varijabli. Kreiranje takvih modela nije jednostavno te zahtijeva mnogo vježbe i iskustva te kompanije često zapošljavaju specijalizirane tvrtke i stručnjake za modeliranje sustava kod rješavanja organizacijskih problema.

Cilj sistemske dinamike je poboljšati i olakšati razumijevanje organizacijske strukture i operativnih politika prema kupcima, dobavljačima, konkurenciji i zaposlenicima, kao i definiranju takvih politika te poboljšanje učinkovitosti. Pružanje informacija pomaže pri donošenju informiranih i ispravnih odluka za organizaciju. Informacije se odnose na predviđanje stanja kroz vrijeme, te predviđanje dugoročnih posljedica svake odluke koja se može donijeti, ubrzano učenje o organizaciji i razumijevanje kompliciranih sustava i strategija organizacije. (Systemdinamics, 2018).

Primjena sistemske dinamike započela je od jednostavnijih menadžerskih odluka kod samog nastanka discipline te se danas primjenjuje u mnogo različitih slučajeva diljem svijeta. U početku se koristila samo u organizacijama kod donošenja

menadžerskih odluka, no ubrzo se proširila i na druge dijelove organizacija. Danas postoje mnoge primjene sistemske dinamike te se primjenjuje u gotovo svim sektorima. Na globalnoj razini postoji svjetska dinamika koja se bavi globalnim problemima ljudske evolucije, populacije, prirodnih resursa, zagađenja i kvalitete života. Ekonomska dinamika bavi se ekonomskim pitanjima, dok je u industrijskoj dinamici fokus na menadžerske probleme u industriji kao što su kontroliranje, tok informacija, politike, struktura industrije, zalihe, rad, kapital i slično. Urbanu dinamiku Forrester je predstavio kao disciplinu koja se bavi urbanim naseljima i njihovim problemima. Ona uključuje i dijelove globalne dinamike, no fokusira se specifično na probleme koji se javljaju u gradovima i drugim visoko naseljenim mjestima. Postoji i mnogo drugih primjena sistemske dinamike u gotovo svim znanstvenim, uslužnim, proizvodnim i drugim granama, te se i one dijele na pod grane koje predstavljaju specifičniju i detaljniju primjenu sistemske dinamike na tom području. (Jay W. Forrester, 2011.)

Područje sistemske dinamike konstantno raste, kao i njezina upotreba u organizacijama, vladama i drugim organizacijama. Sistemska dinamika sve je zastupljenija u školama i fakultetima, s ciljem educiranja ljudi i širenja znanja. Simulacija sistemskom dinamikom koristi se i kod računalnih igara, pa je na temelju modela kreiranog sistemskom dinamikom, 2006. godine nastala i igra „SimCity“ koja je bazirana na urbanoj dinamici. Ona, kao što i samo ime govori, simulira kompleksne gradove i predviđa kako će se oni razvijati uz određene utjecaje raznih varijabli. No sistemska dinamika ne koristi se samo za računalnu simulaciju, već se koristeći nju rješavaju i matematički modeli, znanstvena pitanja te se kreiraju politike organizacija.

3. Programski jezici za sistemsku dinamiku

3.1. DYNAMO

Naziv DYNAMO potječe iz kombinacije engleskih riječi „dynamic“ i „models“, kombinirajući prva četiri i prva dva slova. DYNAMO je predstavljao programski jezik baziran na simulaciji, što znači da predstavlja stvaran model te nastoji predvidjeti njegove rezultate uz neke ulazne varijable koje utječu na model. Proširivao se isto kao i sama sistemska dinamika, počevši od industrijske dinamike te je zatim proširen na urbanu dinamiku i svjetsku dinamiku.

Sve je započelo 1958. godine kada je J. W. Forrester zatražio Richarda Bennetta, programera s MIT-a, da mu uz pomoć računala izračuna neke jednadžbe za simulacije sistemske dinamike. Richard Bennett je tada izradio program SIMPLE koji je kao input uzimao jednadžbe i računao je rješenja simulacija. SIMPLE je predstavljao prethodnika DYNAMO jeziku te se može reći kako je SIMPLE koncept za DYNAMO programski jezik. SIMPLE je kasnije proširen i dodane su nove funkcionalnosti, kako bi se smanjila potreba za programerom koji bi mijenjao tvrdo kodirane inpute te je korisniku omogućena sloboda inputa. Tako je proširena verzija zapravo objavljena kao novi alat pod novi imenom – DYNAMO.

Programski jezik DYNAMO nastojao je biti jednostavan za korištenje za modele industrijske dinamike, budući da je bio namijenjen za takve skupine. Napisani model mogao se odmah izvršiti unutar programa bez dodatnih datoteka te je output bio prikazan grafički putem linijskog printera i tadašnje vektorske grafike. U to doba bio je iznad standarda, podržavao je jasan prikaz greški te je bio nadograđivan kroz verzije. Početne verzije bile su napisane za velika IBM-ova računala (IBM 704, IBM 709 i IBM 7090) u asembleru koji je pretvaran u strojni jezik. DYNAMO II verzija implementirala je algoritamski jezik ALGOL 60 dok su kasnije verzije od DYNAMO III na dalje bile u FORTRAN-u. Iako je bio jednostavan i jasan za korištenje te je imao implementirane mnoge statističke funkcije, nedostatak mu je bio korištenje samo Eulerovog algoritma, bez alternativa. DYNAMO se koristio do početka 80ih godina kada su se počeli javljati novi, moderniji i jači alati te se DYNAMO postepeno zamjenjuje njima. (Jadranka Božikov, 2006.)

3.2. STELLA

Jedan od programskih jezika koji je kasnije zamijenio DYNAMO bio je STELLA, programski jezik sa grafičkim sučeljem čije ime je kratica za Systems Thinking, Experimental Learning Laboratory with Animation. Jezik je napravio i predstavio Barry Richmond u 1985. godini te je on također namijenjen za modeliranje sistemske dinamike. Za razliku od DYNAMO-a, STELLA je imala grafičko sučelje za rad, stoga je bila pristupačnija za rad i upoznavanje sa sistemskom dinamikom. U tome sučelju kreirao se model koji je sadržavao jednake tipove inputa i outputa - tokova (eng. flows) i akumulacija (eng. stocks) koji su grafički predočeni pomoću geometrijskih tijela i veza između njih. Uz njih dodani su još pretvarači (eng. converters) koji pretvaraju varijable prema parametrima i konektori (eng. connectors) koji povezuju sve elemente. Također dodan je i vremenski period kao varijabla za simulaciju, koja omogućava, kod pokretanja simulacije, da se ona prikaže u vremenskim koracima te kako bi izgledala simulacija u određenim vremenskim periodima. Alat računa jednadžbe kako bi došao do nekog konačnog rješenja te za to, uz Eulerov algoritam, nudi i druge metode (npr. Runge-Kutta). Rješenje simulacije ispisuje se u grafičkom ili tabličnom obliku, a modeli se spremaju sa ekstenzijama .ste, .stm, .stmx, .itm i .itmx.

Budući da je alat bio puno pristupačniji ljudima, koji nisu stručnjaci u području sistemske dinamike, te je imao grafičko sučelje čime nije zahtijevao dublje znanje o radu na računalima, on je bio idealan za korištenje na fakultetima i akademijama za podučavanje i približavanje sistemske dinamike studentima i učenicima. Ubrzo nakon nastanka jezika objavljen je i vodič za edukativno korištenje jezika zajedno sa primjerima i vježbama kako bi se dalje inicirala i potaknula edukacija o alatu pa tako i sistemskoj dinamici.

Alat se održao i do danas te je i danas aktivan i ažuriran, no s manjim obujmom korisnika. Postoje i mnoge knjige, priručnici a od 2001. godine je otvorenog koda i besplatan. (Wikipedia, 2018.)

3.3. Vensim

Kompanija Ventana Systems počela je s radom davne 1985. godine razvijajući simulacijske modele za rješavanje težih menadžmentskih problema i odluka te je za razvoj koristila postojeće programske jezike. Za takav razvoj potrebno je bilo mnogo vremena te su se pojavljivala ograničenja u mogućnostima postojećih programskih jezika, stoga se pojavila potreba za razvojem vlastitog programskog jezika. Time je započeo razvoj programskog jezika i alata nazvanog Vensim.

Iako je nudila mnogo više mogućnosti od proteklih alata razvijenih od strane Ventana Systems, prva verzija Vensima nije bila samostalan alat, već oblikovana kao ekstenzija za programski jezik Pascal. Model razvijen u Vensimu bio je preveden i pretvaran u oblik za izvršavanje u Pascalu. Baza podataka modela bila je razvijena u Lisp-u, dok je alat radio na OpenVMS (Virtual Memory System) operativnom sustavu. Alat je bio korišten samo interno od strane kompanije te nije bio dostupan svima. Sljedeća velika verzija Vensim-a, 1988. godine, značila je prebacivanje na programski jezik C sa grafičkim sučeljem i na prve verzije Windows operativnih sustava. Popularizacijom Windows operativnog sustava javlja se prilika za širenje Vensima pa alat postaje javno dostupan (uz plaćenu licencu). Zbog svojeg stručnog konteksta, alat je bio namijenjen specijaliziranim stručnjacima koji su se bavili simulacijskim modelima. U daljnjim verzijama dodana je dokumentacija kako bi se daljnje olakšalo korištenje i ostvarila pristupačnost. Pojavljuju se i konfiguracije – verzije sa različitim obujmom funkcionalnosti kreirane za određene ciljane skupine ljudi: Vensim Standard, Vensim Professional i Vensim DSS (eng. decision support system). Verzija 1.62 otvorila je alat i korisnicima Macintosh računala koji su do sada bili zapostavljeni, te je dodano mnogo novih funkcionalnosti zahtijevanih od strane korisnika kao što su npr. Monte-Carlo simulacija, mogućnost pregledavanja modela bez instaliranog alata i Vensim PLE (Personal Learning Edition) verzija koja je služila kao besplatna distribucija alata u edukaciji i nekomercijalnoj uporabi. (Vensim, 2018.)

U 1997. godini izdana je nova inačica Vensima 3.0 u kojoj je napravljen kompletan redizajn korisničkog sučelja, ponovo napisanu dokumentaciju te je izdana za sve Windows i Macintosh platforme u svim konfiguracijama. To je bila prva velika verzija koja je modernizirala, unaprijedila i poboljšala Vensim. Napravljen je i DLL, koji

je bilo moguće ubaciti u druge programske alate, kako bi se i u njima mogle koristit Vensim funkcionalnosti.

Vensim 4, izdan u 1999. godini donio je Venapp editor koji je omogućio kreiranje modela grafičkim putem i uz to nove objekte te funkcije.

Vensim 5, izdan u 2002. godini, unaprijedio je generiranje modela korištenjem SyntheSim tehnologije koja je ubrzala generiranje modela i prikaz rezultata. Druge funkcionalnosti uključivale su nove načine ažuriranja, podržavanje ODBC baza podataka koji su kroz manje verzije dalje nadopunjavane. Poboljšana je i lokalizacija uvrštavanjem Unicode standarda.

Posljednja verzija koja je objavljena do danas je Vensim 7.2, a konfiguracije koje uključuje su:

- Professional – za profesionalnu uporabu, velike i komplicirane modele namijenjena stručnjacima sistemske dinamike po trenutnoj cijeni od $1195 po korisniku
- DSS (Decision Support System) – dodatno proširuje funkcionalnosti Professional konfiguracije dodajući još sučelje za simulacije letova, pisanje vlastitih funkcija i makroa i drugo po trenutnoj cijeni od $1995
- PLE (Perosnal Learning Edition) – prvotno besplatna verzija za nekomercijalnu uporabu, no licenca se može koristiti i komercijalno uz naknadu od $50
- PLE Plus – dograđena verzija Vensima PLE sa više funkcionalnosti te približena konfiguraciji Professional dostupna za komercijalnu uporabu uz naknadu od $169

Uz konfiguracije nude se i alati koji dopunjavaju Vensim kao što su:

- Model Reader – služi za objavu i distribuciju modela napravljenih u Vensimu, bez obzira da li druga strana posjeduje Vensim ili ne
- Molecules – predstavljaju dijelove struktura sistemske dinamike te su dodane u Vensim suradnjom više autora pretežito članova System Dynamics Socienty gdje se molekule također mogu pronaći. Molekule su vrlo bitne za sistemsku dinamiku jer predstavljaju osnovne elemente sistem-dinamičkih modela te ću ih detaljnije obraditi kasnije u zasebnom poglavlju ovog diplomskog rada.

(Vensim, 2018.)

4. Molekule

Analizom i proučavanjem kreiranih sistem-dinamičkih modela, stručnjaci su došli do spoznaje da se dijelovi modela ponekad ponavljaju u različitim modelima. Kako bi daljnje unaprijedili razvoj sistemske dinamike, vidjeli su priliku da se redundantni modeli mogu definirati kao posebne jedinice koje bi uvijek izgledale jednako i bile standardizirane. Tako se pojavila ideja o generičkim (općim) elementima koji bi bili jasno definirani i upotrebljivi u više modela. Kao problem, takve strukture pojavile su se prvi puta u 1985. godini u znanstvenom članku Marka Paicha objavljenom u časopisu System Dynamics Review gdje se spominju kao tema budućih istraživanja. On navodi njihovo značenje, vrijednost, korisnost i pristup proučavanju, pronalasku i ovjeravanju kao glavne točke istraživanja takvih elemenata. Tek deset godina kasnije, u 1995. godini, Lane i Smart napravili su prvi koncept generičkih elemenata, ali pojavio se problem raznih interpretacija koncepta i drugačijih rezultata, ovisno o primjeni u područjima sistemske dinamike. Zbog toga stručnjaci su podijelili istraživanja na dvije grane:

- istraživanja koja nastoje pronaći generičke strukture u teoriji i povezati ih s ponašanjem
- istraživanja koja nastoje unaprijediti modeliranje.

Zbog takvog pristupa, rad oko struktura i njihovog istraživanja vrlo je rascijepan i nema jasnu predodžbu u kojem smjeru bi se one trebale istraživati. Literatura je također razbacana i to je otežavalo istraživanje, pa čak i učenje o postignućima i rezultatima istraživanja i kako ih efektivno primijeniti u praksi. Stoga su stručnjaci, članovi društva sistemske dinamike (System Dynamics Society), odlučili pokušati ujediniti istraživanja, smanjiti razlike između dva smjera istraživanja i kreirati jedinstvene i jasne elemente. Kako bi što jasnije prikazali elemente, pristupili su istraživanju od početka prikupljajući protekla istraživanja, objašnjavajući ih i zatim su ih nastavili pokušavajući ih organizirati i ujediniti u jedno.

Početak je bio Lane-ov i Smart-ov znanstveni članak pod nazivom „Reinterpreting 'generic structure': Evolution, application and limitations of a concept“ objavljen u ranije

spomenutom časopisu System Dynamics Review u 1996. godini. U tome članku su se javili prvi pokušaji definiranja generičkih struktura. Oni navode tri ključna smjera definiranja takvih struktura. Najprije se proučava povijest razvoja i korištenja elementa i zatim se on svrstava u jednu od tri kategorija:

- situacijski modeli (eng. situational models)
- apstraktne mikro-strukture (eng. abstracted micro-structures)
- neintuitivni tipovi (eng. counter-intuitive archetypes)

Situacijski modeli predstavljeni su kao opći modeli koji sadrže osnovne strukture za prikaz ponašanja modela u sistemskoj dinamici. Takvi modeli su numerički, što znači da su prikazani u obliku formule i mogu se prilagoditi području proučavanja. Kako bi se takvi modeli raspoznali od ostalih, Lane i Smart navode empirijske testove (npr. analiza osjetljivosti), kao metodu za razlikovanje. Za primjer takvog situacijskog modela navode Forresterov model razvoja urbanih naselja.

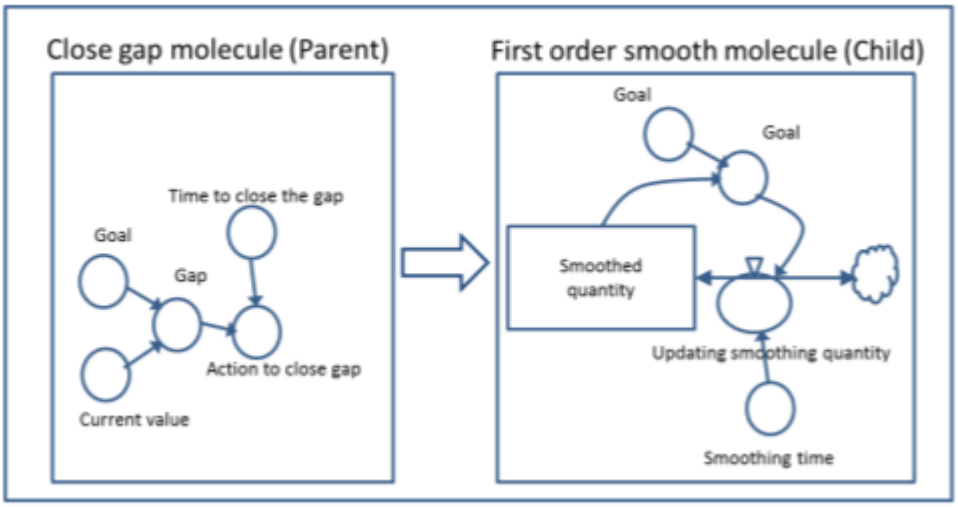
Apstraktne strukture predstavljaju osnovne strukture svakog sistem-dinamičkog modela, pa čak i situacijskih modela. Neke od ovih struktura često su implementirane kao gotove funkcije u softverskim programima. One objašnjavaju ponašanje elemenata na najnižoj razini te su zaslužne za kreiranje ponašanje cjelokupnog modela kroz interakciju sa modelom. Kao primjer navode se negativna petlja prvog reda (eng. first order negative loop) i pozitivna petlja prvog reda (eng. first order positive loop).

Neintuitivni tipovi ne mogu biti simulirani i ne predstavljaju ponašanje sustava, već su oni vezani za povratne informacije. Služe kao pomoć kod komunikacije i učenja i daju informacije o kvaliteti – što bi se moglo dogoditi, a ne što će se dogoditi. Oni se mogu opisati kao tipovi koji pomažu kod pravog interpretiranja povratnih informacija, te Wolstenholme (2003) kao primjer navodi četiri takva tipa: neuspješnost, relativna postignuća, relativna kontrola i izvan kontrole. (Sondoss ElSwah, 2013.)

4.1. Naziv molekule

Generičke strukture modela, kao što su ranije navedene, prvi puta su nazvali molekulama Eberlein i Hines u 1996. godini na međunarodnoj konferenciji sistemske dinamike. Naziv molekule inspiriran je molekulama iz biologije koje su sačinjene od grupa atoma povezanih vezama. Takva inspiracija javila se iz razloga što ranije spomenute generičke strukture u sistem-dinamičkim i računalnim modelima podsjećaju na molekule. No sistem-dinamičke molekule ne predstavljaju naziv za prethodno navedene generičke modele, već su one nova kreacija koja omogućuje kreiranje zbirke prikazane kao veza roditelja i djece za elemente sistem-dinamičkih modela. U računalnom smislu one predstavljaju objektno orijentirano programiranje i razmišljanje gdje su prikazane kao objekti koji imaju svoje atribute i mogu se povezivati. Neke molekule ne sadrže varijable, veze i povratne informacije, nego mogu biti kombinacije bilo koja od tri ranije navedenih oblika elementa. Na primjer, molekula smanjenja razmaka (eng. close gap) sačinjena je od nekolicine pomoćnih varijabli koja koristi druge molekule za smanjivanje razlike između neke varijable i željene vrijednosti. Molekule ne definiraju vezu između struktura i njihovo ponašanje, već nastoje definirati i organizirati predloške različitih struktura u sustav – omogućiti međusobno vezanje i povezanost molekula u sustavu. Njihova struktura varira te se koriste i kombiniraju međusobno da se dođe do željenog rezultata. Prvo popisivanje zbirke molekula rezultiralo je brojem od otprilike 50-ak sistematski definiranih molekula, sa opisanom strukturom, funkcijom, problemom kojeg rješavaju i tehničkim naputcima i upozorenjima. Definirane su i formalne smjernice za kreiranje takvih molekula i prve molekule kreirane su uz pomoć alata Vensim.

Pojavio se i koncept kreiranja modela uz pomoć molekula koji se sastoji od tri ključna djela. Najprije postoji hijerarhijska zbirka molekula (taksonomija) koja sadrži matematički definirane molekule od kojih su najjednostavnije varijable, tokovi i pravila. Zatim su molekule povezane u hijerarhiju gdje imaju definirane roditelje i djecu te se mogu mijenjati sa drugim molekulama koje odgovaraju matematički. Na slici niže prikazana je veza između dvije molekule gdje je jedna roditelj, a druga dijete te kako se one mogu međusobno povezati – na mjesto djeteta može se postaviti i druga molekula koja za roditelja podržava molekulu smanjenja razmaka.



Slika 3. Veza roditelj-dijete između dvije molekule

Izvor: Sondoss ElSwah, 2013.

Na molekule se može gledati i s objektno orijentiranog aspekta gdje se razvijaju blokovi za razvoj modela i učenje sistemske dinamike. Vavik i Myrtveit navode da objekti potiču učenje uspoređujući model sa elementima stvarnog sustava i nude mogućnost uređivanja modela od parametara do drugih razina složenosti. Blokovi su definirani klasama koje sadrže različita svojstva i zajedno su dodane u biblioteku. Takav objektno orijentiran pogled na molekule omogućio je lakšu implementaciju modela u računalstvu, te je kasnije proširen, te uključuje mnogo dijagrama koji vizualiziraju modele i pripomažu kod učenja i upoznavanja sa istima.

Uz objektno orijentiran pogled na modeliranje molekula pojavilo se i meta-modeliranje. Ono nastoji pojačati napredak kod implementacije molekula u računalstvu s pogleda da modeli budu razvijeni tako da su jasni i ljudima koji nisu stručnjaci za sistemsku dinamiku. Za to je potreban poseban oblik implementacije kod razvoja alata za molekule gdje se klasama u modelu, koje predstavljaju molekule, dodaju entiteti, svojstva i veze. Zatim se kreiraju objekti za specifične slučajeve i na kraju se prevađa u model. Krajnji korisnik ima mogućnost unašanja parametra, kako bi program kreirao model prema željama i potrebama, a korisnik se ne susreće sa tokovima, varijablama, strukturama i matematičkim dijelovima modela. U usporedbi sa objektno orijentiranim modeliranjem, ovaj pristup ima svoje prednosti u obliku količine znanja potrebnog za rad sa alatom te jednostavnost i brzo upoznavanje i savladavanje jednostavnijih modela. Negativne strane su zahtjevnija i kompliciranija implementacija od strane

programera te ograničenost kod kreiranja modela zbog nemogućnosti unašanja vlastitih varijabli, tokova, veza i matematičkih formula.

4.2. Svrha korištenja molekula

Razlog poticanja istraživanja nad temom generičkih struktura i molekula bilo je širenje znanja o sistemskoj dinamici. Znanstvenici su smatrali da su molekule korisne za širenje znanja zato što bi jasno definirale elemente modela i ujedno i pružile pristupačniji i organiziraniji pristup modelima sistemske dinamike. Isto tako, primjenom molekula ostvaruje se standardizirani pristup sistemskoj dinamici gdje postoje pravila za izradu modela. Generičke strukture pomogle su sistemsku dinamiku proširiti na druge grane modeliranja, kao što su na primjer modeliranje na temelju agenata (eng. agent-based modeling) i diskretne simulacije (eng. discrete-event simulation). Prednost kreiranja takvog općenitog modela je što se može vrlo lako integrirati sa drugim ili u druge oblike modela. No i dalje postoji dosta nepoznanica kod pitanja kako generičke strukture ponovo upotrebljavati u praksi i pojavilo se pitanje koliko su zapravo one korisne. Na to pitanje stručnjaci su dugo vremena pokušavali pronaći odgovor te izmjeriti korisnost molekula. Primjenjivanjem studija slučaja pronašlo se rješenje za spoznaju koliko zapravo generičke strukture i molekule doprinose modeliranju i učenju o sistemskoj dinamici i kako ih pravilno upotrijebiti. Uz pomoć empirijskih istraživanja uspjeli su procijeniti i kontrolirati korisnost različitih molekula i generičkih struktura. Studiji slučaja moraju se fokusirati na tri bitne grupe pitanja: (Sondoss ElSwah, 2013.)

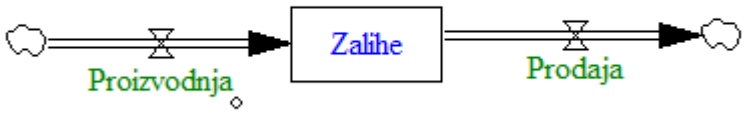
- Pitanja koja utječu na odabir pravog pristupa za iskorištavanje molekula i generičkih struktura u različitim modelima. Na primjer, kako korištenje molekula utječe na učenje o sistemskoj dinamici i modeliranju, budući da ranije spomenuto meta-modeliranje smanjuje mogućnosti učenja zbog povećane jednostavnosti i ograničenih informacija o modelima. Takav pristup povećava brzinu učenja modeliranja, no ograničava mogućnosti napretka kod savladavanja modela jer su modeli jednostavni.

- Pitanja koja utječu razumijevanje u razlikama u očekivanjima kod učenja među pojedincima i skupinama. Na primjer, koji su učinci korištenja modularnog pristupa na razini pojedinca i grupe? Koje su razlike između početnika i stručnjaka kod korištenja generičkih struktura?
- Pitanja koja utječu na dizajn cjelokupnog modela u smislu alata koji su korišteni uz generičke strukture i molekule kako bi model bio potpun (npr. dokumentacija).

4.3. Molekule kao generičke strukture

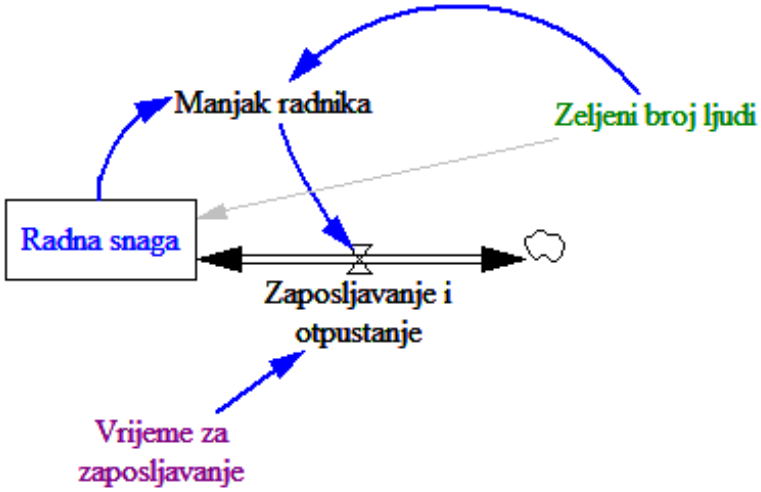
U ovom dijelu diplomskog rada koristit ću alat Vensim Personal Learning Edition sa besplatnom licencom za osobnu i edukativnu upotrebu, kako bih prikazao i pojasnio neke od najbitnijih molekula. Alat ima službeno implementirane molekule kao zasebnu funkcionalnost unutar sebe te su one već integrirane kao zasebni modeli koji se jednostavno otvaraju. Svaka molekula sastoji se od varijabli, kauzalne strukture koja je grafički prikazana i matematičkih jednadžbi na kojima se temelji svaka varijabla. One se mogu mijenjati, preimenovati, brisati i dopunjavati, kako bi se molekula primijenila na vlastitom primjeru i problemu. Mogu im se mijenjati i matematičke jednadžbe na čijem principu funkcioniraju te je moguće dodati i spojiti nekoliko molekula zajedno.

Na primjer, otvaranjem molekule pod nazivom kada (eng. bathtub) kreira se model molekule sa već definiranim varijablama i tokovima. Prema potrebi promijenit ću imena varijabli da predstavljaju tok proizvodnje u bilo kojem proizvodnom poduzeću. Postojeće ime varijable „*level*“ (hrv. razina) zamijenio sam sa „*zalihe*“, dok se tokovi umjesto „*inflow*“ i „*outflow*“ sada zovu „*proizvodnja*“ i „*prodaja*“. Sada ovaj model predstavlja molekulu kade, ali u drugačijem kontekstu od prijašnjeg – model predstavlja proizvodnju proizvoda i njegovu prodaju. Količina proizvoda na zalihi ovisi o proizvodnji te ako je proizvodnja veća od prodaje tada se zalihe povećavaju. (Jim Hines, 2015.)



Slika 4. Model molekule kada prikazan u alatu Vensim PLE

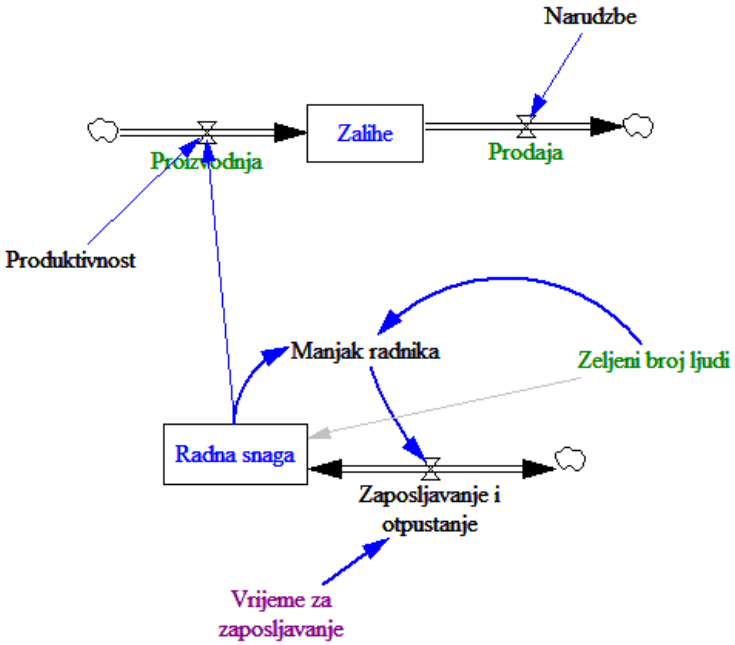
Ako pretpostavimo da ulazni tok proizvodnje ovisi o nekoj drugoj varijabli, otvara se mogućnost za proširenje modela dodavanjem nove molekule. Ovdje odlično odgovara molekula nazvana „*radna snaga*“ (eng. workforce) koja simulira radnu snagu i može se direktno povezati sa molekulom kade. Ona se sastoji od varijable radna snaga, toka zapošljavanje i otpuštanje radnika i nekoliko sporednih varijabli kao što su manjak zaposlenika, željeni broj radnika i vrijeme zapošljavanja.



Slika 5. Model molekule radna snaga prikazan u alatu Vensim PLE

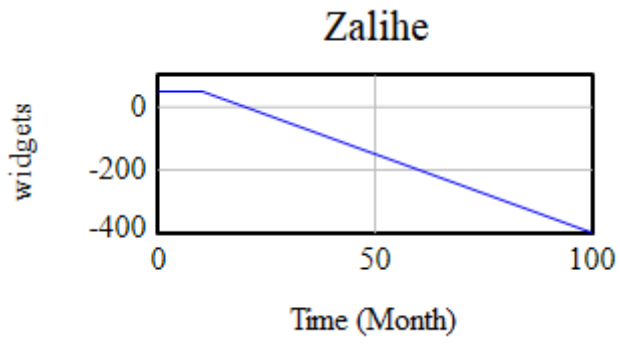
Povezivanjem dvije molekule na način da se spoje varijable „radna snaga“ sa „tokom proizvodnje“ dobiva se gotov i cjelokupni model. No samo spajanje nije dovoljno za ispravnu funkcionalnost modela, već je potrebno napraviti i manje korekcije. Najprije se pregledava strukture i treba li dodati neke dodatne pomoćne varijable. U ovom primjeru može se dodati varijabla prosječna produktivnost koja se odnosi na radnike i umanjuje idealnu produktivnost radne snage na realnu produktivnost. Kao primjer iz stvarnog života može uzeti da radnik nije produktivan svih osam sati kojih radi u jednom danu, budući da ima pravo na pauzu, odmor, odlazak na

wc i slično. Druga korekcija modela odnosi se na vrijednosti, odnosno mjerne jedinice u kojima su izražene varijable. Njih je potrebno uskladiti kako bi bile kompatibilne jedna s drugom te da ih je moguće uspoređivati, pretvarati i računati jednadžbe. Ako je potrebno, matematičke jednadžbe mogu se također korigirati prema potrebama. Nakon svih ispravaka uspostavljen je odnos između dvije molekule i radna snaga utječe direktno na zalihe i prodanu robu te je model gotovo spreman za simulaciju.



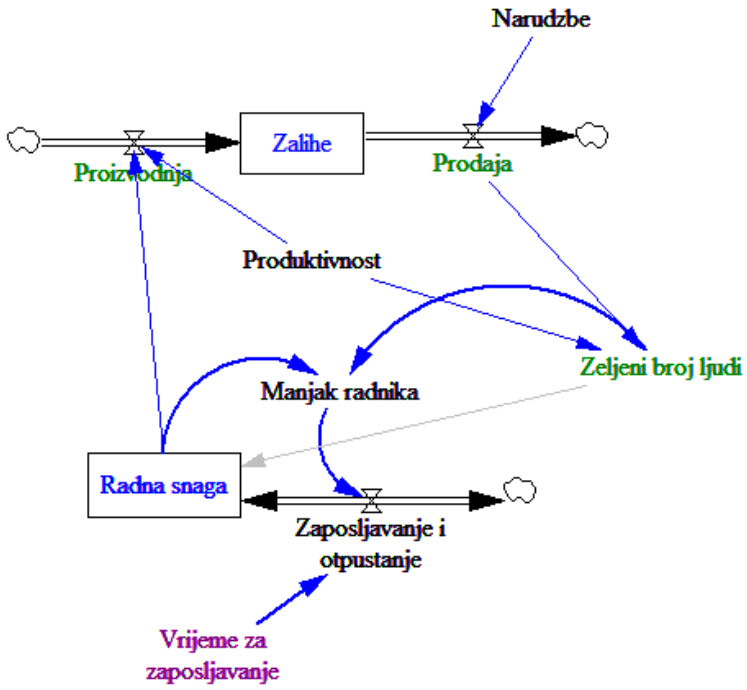
Slika 6. Dvije molekule spojene u jedan model

Nakon simulacije dolazi se do zaključka da model funkcionira ispravno za prvih nekoliko mjeseci, no kasnije se dobivaju nelogične vrijednosti. To možemo vidjeti na grafu s rezultatima zaliha gdje vrijednost pada naglo nakon 10 mjeseci.



Slika 7. Graf sa naglim padom zaliha nakon 10 mjeseci

Problem se rješava dodavanjem povratnih veza kako bi se ispravilo otpadanje vrijednosti u negativne i nelogične vrijednost nakon određenog perioda. Na temelju tih varijabli dolazi se do informacije kada je potrebno povećati proizvodnju, na način da se poveća radna snaga. Radna snaga usko je povezana sa željenim brojem koji mora biti dinamičan, tj. njegova vrijednost se sada korigira s obzirom na broj narudžba i prodaje.

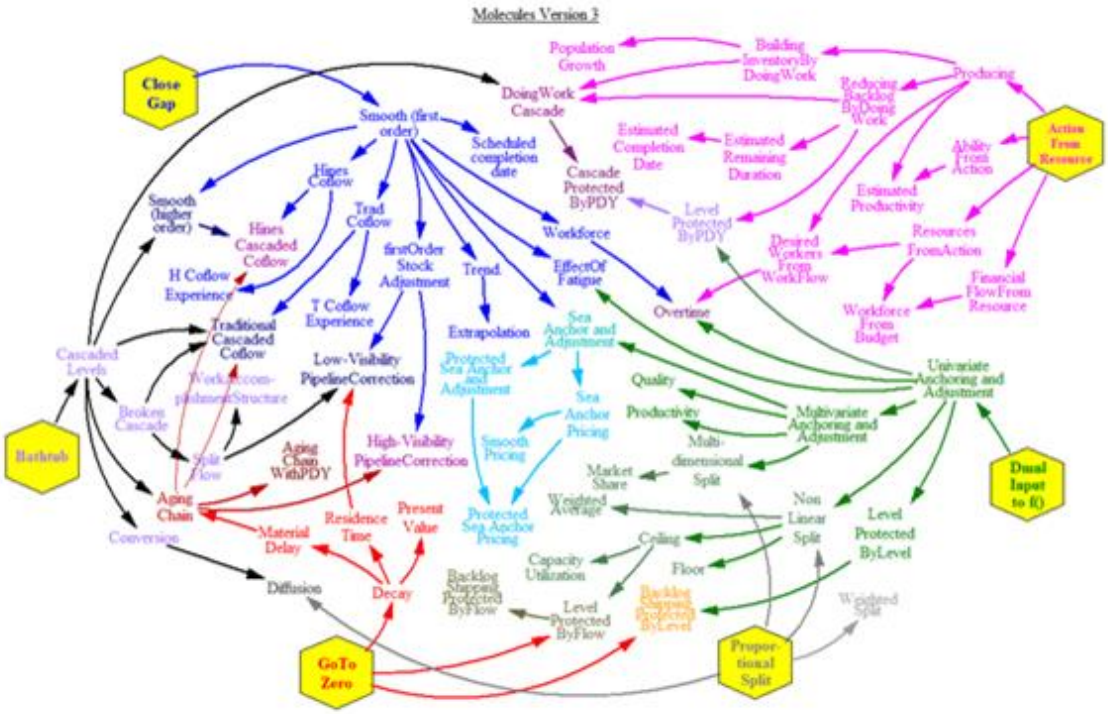


Slika 8. Konačan model spajanja dviju molekula

Time dolazimo do konačnog modela koji se funkcionalan i predstavlja jedan proces i tok unutar neke industrijske tvrtke. Kod simulacije radna snaga se blago povećava usporedno s povećanjem potražnje, što je cilj ovog modela, te je on ostvaren.

https://vensim.com/modeling-with-molecules-2-02/

Nakon što sam pojasnio proces rada s molekulama u alatu, sada ću se detaljnije osvrnuti na molekule i njihove specifičnosti, kao i strukture nekih od njih te pokušati pronaći njihovu primjenu u praksi. Za prikaz ću i dalje koristiti alat Vensim i njegovu implementaciju molekula.



Slika 9. Prikaz mreže molekula i veze između njih

Na slici iznad prikazana je mreža svih molekula iz alata Vensim koja je objavljena i na stranicama društva sistemske dinamike. Za popisivanje molekula i ovu mrežu molekula zaslužno je mnogo ljudi članova društva sistemske dinamike koji su ih popisali na jednom mjestu. Podrijetlo svake molekule nije poznato i neke su izumljene po više puta, no popisane su na temelju objavljenih radova i publikacija. Mreža ne predstavlja sistem-dinamički model, već predstavlja hijerarhiju između molekula, njihovu strukturu i odnose između njih. Na vanjskim rubovima, obojane u žuto, osnovne su molekule koje nemaju roditelje i nalaze se na najvišoj (vanjskoj) razini. One su jednostavne i često predstavljaju specijalizirane slučajeve kombinacija nekih od osnovnih tipova modela - varijable, tokova i slično. Iz njih proizlaze druge molekule koje predstavljaju njihovu djecu, iz kojih se tada ponovo granaju nove molekule. Grane koje kreću iz rubnih molekula na kraju se međusobno spajaju sa drugim granama te tako nastaju nove molekule. Takvo grananje na slici je vrlo dobro prikazano bojama - grana je obojana u jednu od boja te zatim kako se spaja s drugom granom boja se mijenja u boju kombinacije ili nijanse tih grana. Kretanjem dublje po granama dolazi se do sve kompliciranijih i specifičnijih molekula. Osim roditelja i djece, svaka molekula

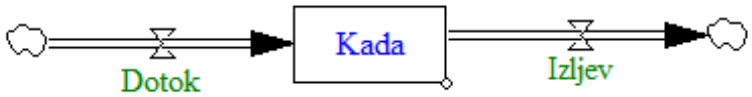
ima funkciju koju obavlja u sistem-dinamičkom modelu, te se primjenjuju po potrebi ovisno o slučaju i funkciji koju obavljaju i koju model zahtijeva. Funkcija svake od molekula može se prikazati grafički na modelu, gdje se prikazuju inputi, outputi, parametri i ključne vrijednosti koji sačinjavaju model, te tokovi između njih koji prikazuju tok informacija u modelu. Molekule se mogu prikazati i matematičkom formulom koje se kreću od jednostavnijih pa do sve kompliciranijih, kao i same molekule. Molekula ima mnogo, oko 70-ak i nije ih sve moguće primijeniti u jednom modelu. Tako ni ja neću prolaziti kroz sve molekule, već ću u nastavku detaljnije obraditi neke od najkorištenijih i najbitnijih za kreiranje općih i čestih modela sistemske dinamike. (Vensim, 2018.)

4.3.1. Kada (eng. bathtub)

Kada je prva od nekoliko osnovnih molekula sistemske dinamike i kao takva obavlja i jednostavnu funkciju. Ona se sastoji od varijable koja se na primjeru kade naziva "*razina vode*" (eng. level) te dva toka od kojih je jedan rastući, a drugi padajući. Rastući tok u molekuli predstavlja dotok vode u kadu, kao npr. tuš, dok je odvod predstavljen padajućim tokom te se zato ova molekula naziva kadom. Razina vode u kadi povećava se dotokom vode i smanjuje se izljevom - tako se i varijabla u molekuli povećava ovisno o dotoku i izljevu vode. U slučaju kad je dotok vode veći od izljeva, kada se puni, a kad je dotok manji od izljeva, kada se prazni. Kod ovakvog modela, razina vode ne može poprimiti negativnu vrijednost u smislu da varijabla razina vode postane negativna - vode može izaći iz kade samo onoliko koliko je ima. Kada predstavlja akumulaciju vrijednosti, te da se prevenira negativna vrijednost, izljev mora biti direktno iz kade bez ikakvih tokova i varijabli između. Takva struktura naziva se tok povratnih informacija prvog reda, odnosno povratna veza koja ima samo jednu razinu. Osim kade, povratnost prvog reda koriste i molekule glađenja (eng. smooth), raspadanja (eng. decay) i razine zaštićene tokom i akumulacijom (eng. level protected by flow/level).

Za molekulu kade postoji dosta primjera u čijim modelima se ona može primijeniti i to ju čini dosta popularnom i korištenom molekulom. Ranije napomenuti primjer kade u modelu industrijskog procesa proizvodnje, prodaje i zaliha samo je jedan od mnogo primjera koji djelomično ili kompletno implementiraju molekulu kade. Primjena kade

može se prikazati i na modelu financija gdje profit poprima mjesto varijable kade, dok su na mjestu ulaznog toka prihodi, a na mjestu izlaznog toka rashodi. Rast trave te njezina košnja može se uzeti kao generalan primjer – trava je definirana kao varijabla gdje je njezin rast ulazni tok, a košnja je izlazni tok. (Jim Hines, 2015.)



Slika 10. Molekula kade

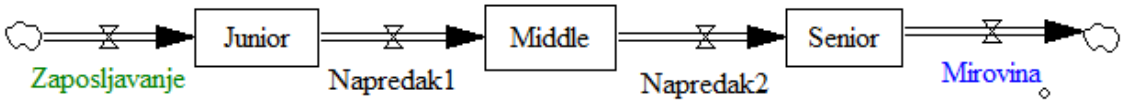
Na slici je prikazana model u alatu Vensim koji predstavlja molekulu kade sa ulaznim i izlaznim tokom od kojih je jedan rastući, a drugi padajući. Razina je prikazana varijablom oblika kvadrata i njezina vrijednost raste ili pada s obzirom na tokove. Razina se može prikazati matematičkom formulom gdje se razina računa razlikom ulaznog i izlaznog toka.

$$Razina\ (nivo) = dotok - izljev$$

4.3.2. Lanac (eng. chain, cascaded levels)

Lanac je druga molekula koju ću spomenuti te predstavlja proširenje molekule kade na način da se vrijednost akumulira na više mjesta. Možemo ju zamisliti kao nekoliko kada spojenih uzastopno ili kao model kade sa više varijabli razine vode. Ona je direktno dijete molekule kade unutar mreže molekula. Ova molekula također sadrži ulazni i izlazni tok, te dodaje i tok između svake varijable tako da su neki tokovi u isto vrijeme ulazni i izlazni tokovi. Broj varijabli u lancu mora biti veći od jedan, no to je jedini uvjet za ograničenje varijabla te ih može biti vrlo mnogo. I kod ove molekule vrijednosti varijabli ne mogu poprimiti negativne vrijednosti te se samo mogu isprazniti i doći do vrijednosti 0. Molekula lanca zapravo dijeli jednu razinu kod kade na više manjih akumulacija i tome ostvaruje specifičniju strukturu koja zahtijeva i posebne slučajeve za primjenu takve molekule. Iako je teže pronaći primjenu ove molekule, i dalje postoji dosta primjena u okolini. Tako na primjer za prikaz kvalifikacija radne

snage možemo koristiti ovu molekulu sa vrijednostima junior, srednji i senior. Svaki od zaposlenika kroz svoju karijeru prolazi kroz ove razine kvalifikacije gdje se početkom radnog odnosa (ulazni tok) dolazi do pozicije junior da bi se kasnije sa godinama iskustva došlo do drugih razina – napredovalo. Osim kvalifikacija zaposlenika, molekula lanca može se prikazati na proizvodnji gdje je slučaj da varijable predstavljaju početak proizvodnje, robu koja je u procesu proizvodnje i gotovu robu. (Jim Hines, 2015.)



Slika 11. Molekula lanca

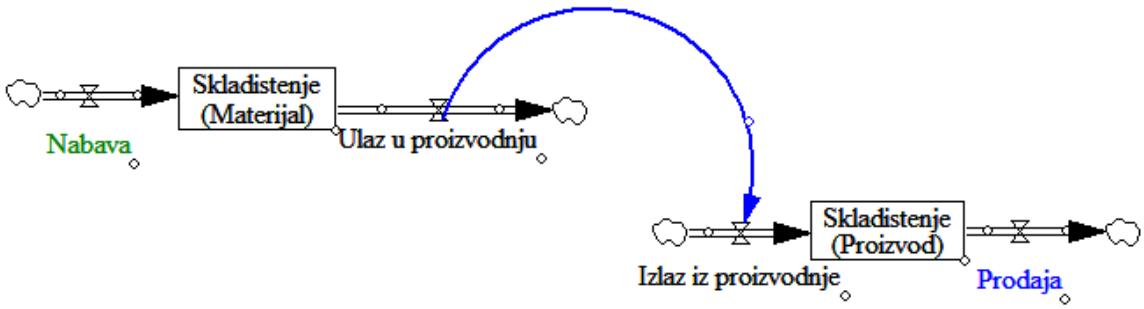
Slika predstavlja molekulu lanca iz alata Vensim sa tri razine – varijable. Molekula je u alatu implementirana u općenitom stanju i treba je prilagoditi kako bi se primijenila u nekom modelu na način da se pronađe specifična funkcija molekule i zadaća koju obavlja te da se ona preimenuje, proširi, izmjene se vrijednosti i mjerne jedinice i slično. Matematičke jednadžbe koje podupiru ovu molekulu izgledaju ovako:

$$Razina1\ (nivo) = dotok - tok\ razine\ 2$$
$$Razina2\ (nivo) = tok\ razine\ 3 - tok\ razine\ 3$$
$$Razina3(nivo) = tok\ razine\ 3 - izljev$$

4.3.3. Prekinuti lanac (eng. broken chain, broken cascade)

Molekula prekinutog lanca modifikacija je molekule klasičnog lanca čije razine nisu lančano povezane, već se između njih javlja lom. Time je ona direktan potomak molekule lanca i ima molekulu kade kao zajedničkog krajnjeg roditelja s molekulom lanca. Funkcija tog loma je najčešće promjena mjernih jedinica, no to ne mora vrijediti uvijek. Kod loma se može javiti i gubitak vrijednosti te, ako ne postoji ni konverzija

mjernih jedinica ni gubitak vrijednosti, tada je ova molekula prekinutog lanca zapravo jednaka onoj klasičnog lanca. Kao i klasičan lanac, prekinuti lanac ima ulazni tok, izlazni tok i minimalno dvije varijable – akumulacije. Izlazni tok prve razine jednak je ulaznom toku druge razine umanjen za neku promjenu i u ovom modelu oni su prikazani zasebnim tokovima - za razliku od jednog toka kod klasičnog lanca. Te dvije razine povezuje konverzija koja pretvara mjerne jedinice iz jedne u drugu ili ih mijenja za neku vrijednost. Prekinuti lanac može se prikazati na primjeru proizvodnog procesa gdje je ulazni tok nabava materijala te se materijal skladišti i nakon toga šalje na proizvodnju. Nakon obavljene proizvodnje materijal se pretvara u gotovi proizvod i se tu dešava konverzija iz materijala u proizvod te se u isto vrijeme događa i gubitak vrijednosti u obliku materijala koji predstavlja škart prilikom proizvodnje. Na drugoj razini ove molekule ulazni tok je gotovi proizvod koji se zatim odvodi na skladište te zatim do prodaje kao izlazni tok iz modela. (Jim Hines, 2015.)



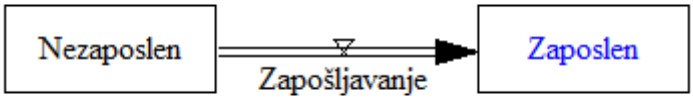
Slika 12. Molekula prekinutog lanca

Prema slici modela prekinutog lanca iz Vensim-a ovu molekulu može se predočiti kao dvije zasebne molekule povezane u jednu. Plavom strelicom prikazana je veza između ta dva djela modela koja predstavlja lom lanca i u njoj se odvijaju spomenute promjene. Tok ovog modela se ponaša kao i kod klasičnog lanca te ima svoje matematičke jednadžbe na čijem principu funkcionira. Svaka varijabla može se izračunati prema funkcijama:

$$Razina1\ (nivo) = \ dotok - izljev\ razine\ 1$$
$$Razina2\ (nivo) = \ dotok\ razine\ 2 - izljev$$

4.3.4. Pretvorba ili konverzija (eng. conversion)

Molekula konverzije vrlo je jednostavna i generalna te nema mnogo kompliciranе logike u pozadini. Sastoji se od dvije varijable i toka koji ih povezuje te je njezina funkcija pretvaranje jedne vrijednosti u drugu. Koristi se kod raznih konverzija u svim mogućim oblicima iz jedne kategorije u drugu. Na primjer, to može biti promjena stanja iz nezaposlenog u zaposleno ili aktivnog u neaktivno i slično. U Vensim-u ova molekula prikazana je sa dvije varijable i jedim tokom i direktno je dijete prethodne molekule lanaca dok joj je krajnji roditelj molekula kade. (Jim Hines, 2015.)



Slika 13. Molekula konverzije

4.3.5. Smanjenje razlike (eng. close gap)

Molekula smanjenje razlike jedna je od osnovnih rubnih molekula koje se nalaze na vanjskim rubovima mreže molekula. Ona nema direktnih indirektnih roditelja u stablu molekula i time pruža osnovu za građu drugih molekula na temelju sličnosti i modifikacija, kao na primjer molekula izglađivanja (eng. smooth) koja je jedna od njezine djece. Ova molekula rješava problem smanjenja razlike između količine i željene vrijednosti kroz neko vrijeme, odnosno približava nas nekom cilju, koristeći tokove ili akciju. Akcija, u slučaju da je konstanta, umanjuje razliku kroz određeno vrijeme koja se mijenja postepeno ili jednom. Molekula je načinjena samo od tokova i jedna je od varijabli za generiranje tokova. Kroz ovu molekulu vrijednost se može smanjiti ili povećati ovisno o modelu i primjeni. U slučaju da se vrijednost postavlja na nulu, ova molekula ista je kao i molekula „raspada“ (eng. decay), kojoj je cilj input dovesti do nule. Generička struktura kao kod ove molekule može se prikazati na modelu bankarskog kredita gdje je kroz neko vrijeme cilj do kraja otplatiti kredit,

odnosno svesti svotu duga na nulu. Drugi primjer ove molekule može se pronaći kod zapošljavanja ljudi gdje je cilj neke kompanije postići potreban broj zaposlenih za normalno funkcioniranje organizacije. Akcijom zapošljavanja smanjuje se razlika između *potrebnog broja zaposlenih* (cilja) i *trenutnog broja zaposlenih* (trenutnog stanja). Za pronalazak pravih kandidata, intervjue i druge akcije kod procesa zapošljavanja potrebno je *vrijeme* kroz koje se, zapošljavanjem novih ljudi, smanjuje razlika između cilja i trenutnog stanja sve dok trenutno stanje ne postigne vrijednost cilja. Time se dolazi do željenog broja zaposlenika i molekula završava s radom. (Jim Hines, 2015.)

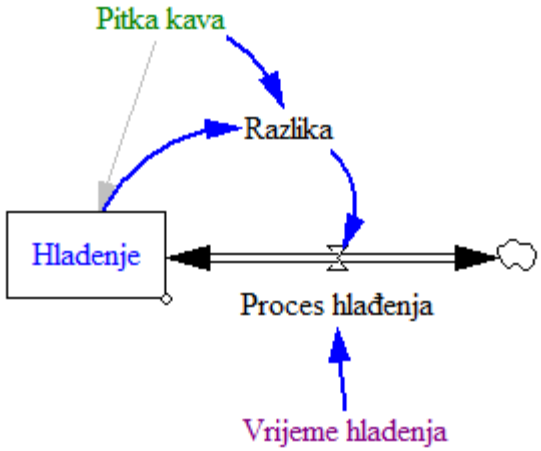
Slika 14. Molekula smanjenja razlike

Ranije naveden primjer prikazan je na slici gdje su sve pomoćne varijable povezane tokovima koji vode do akcije za smanjivanje razlike odnosno stvarnog cilja ove molekule. Molekula je vrlo slične implementacije kao i molekule postavljanja na nulu (eng. go to zero) i cijeloj grani stabla koja potječe iz nje. Jednadžbe koje izračunavaju model molekule smanjenja razlike mogu se prikazati kao:

$$Zapošljavanje = razlika - potrebno\ vrijeme$$
$$Razlika = ciljani\ broj - trenutni\ broj$$

4.3.6. Izglađivanje (eng. smooth)

Molekula izglađivanja direktan je potomak odnosno dijete, prethodno spomenute molekule smanjivanja razlike. Ona je vrlo popularna molekula i koristi se kod mnogo drugih molekula kao što su reguliranje zaliha prvog reda, Hinesov protok (eng. Hines co-flow), tradicionalni protok, trend, efekt umora, radna snaga, planirani rok završetka i još nekoliko drugih. Neke od ovih molekula detaljno ću pojasniti, dok ću druge samo spomenuti. Ova molekula riješava sličan problem kao i molekula smanjivanja razlike, koji je umanjivanje ili povećavanje vrijednosti neke varijable prema cilju. Razlika između tih dviju molekula je u tome što izglađivanje umanjuje razliku postepeno i izglađeno kroz neko vrijeme. Molekula služi izglađivanju toka informacija, za postepen prikaz količine i određenih očekivanih vrijednosti. Izglađivanje je varijabla sa specifičnim ulaznim i izlaznim tokom u kojoj se ulazni tok računa prema neto stopi. To znači da negativne vrijednosti smanjuju vrijednost varijable za iznos neto stope dok ju pozitivne povećavaju. Kao i u molekuli smanjivanja razlike, cilj ove molekule je smanjiti razliku između ciljne vrijednosti i trenutnog stanja, ali eksponencijalnim dovođenjem vrijednost razlike na nulu. Razlika se umanjuje prema konstanti nazvanoj *vrijeme izglađivanja*. Ovisno o smanjivanju razlike, ulazni tok se mijenja te se time postiže eksponencijalnost. Može se javiti i slučaj da ciljana vrijednost nije konstanta te se u tom slučaju javlja usporavanje izglađivanja. Ova molekula koristi se gotovo u svim sistem-dinamičkim modelima te ima mnogo primjena u stvarnom životu. Na primjer, uzmimo šalicu kave koja se hladi. Kava je određene temperature i postoji ciljna temperatura koju želimo postići (temperatura kada je kava pitka). Kava se hladi i temperatura se smanjuje prema željenoj, a vrijeme hlađenja ovisno je o atributima kao što su veličina šalice, materijal i površina. Pretpostavimo da se osoba ipak odlučila napraviti ledenu kavu, kavu je potrebno ohladiti još više i time se ciljana temperatura promijenila. Ova molekula izglađivanja odnosi se na izglađivanje prvog reda (eng. first-order) te postoji još slična molekula nazvana izglađivanje višeg reda (eng. higher-order). Ona lančano povezuje više molekula prvog reda kako bi izgladila neke vrijednosti različitim brzinama. Izglađivanje počinje sporo i zatim ubrzava da bi na kraju opet usporilo. (Jim Hines, 2015.)



Slika 15. Molekula izglađivanja

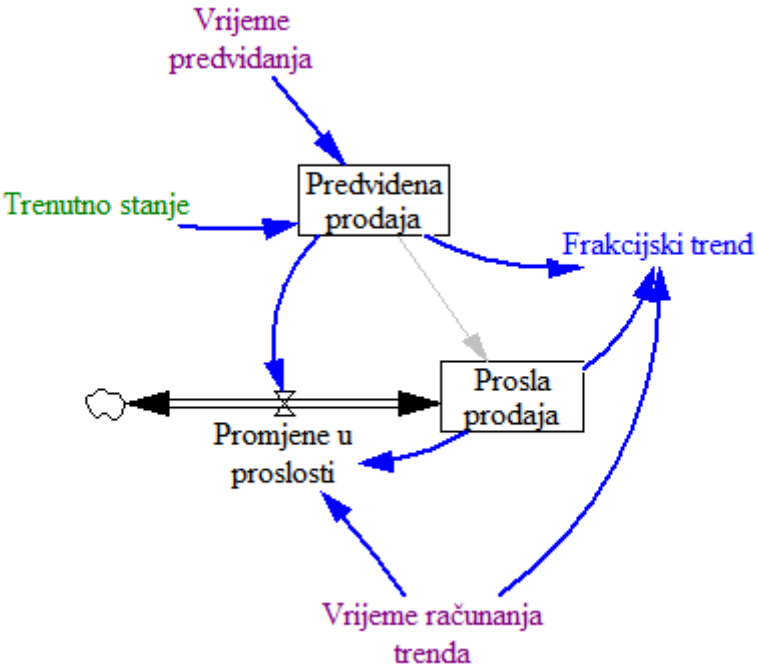
Proces hlađenja kave izglađivanjem prikazao sam na primjeru molekule u Vensimu. Na modelu je vidljiv cirkularni proces - razlika -> proces hlađenja -> hlađenje koji se odvija kroz vrijeme hlađenja te tako dugo dok razlika ne postane nula. Takav model možemo prikazati i matematičkim jednadžbama za koje vrijedi da je vrijednost hlađenja omjer razlike i vremena hlađenja:

$$Hladenje = \frac{razlika}{vrijeme\ hlađenja}$$

4.3.7. Trend

Molekula trenda pozicionirana je u mreži molekula kao direktno dijete molekule izglađivanja te joj je krajnji roditelj molekula smanjivanja razlike. Trend je pristup modeliranju koje je zastupljen i u drugim disciplinama kao što je na primjer statistika te se ponekad javlja pod imenom linearne regresije. Tako se i ova molekula bazira na istom smislu riječi trenda gdje se na temelju promatranja vrijednosti iz proteklih razdoblja i trenutnih procjena nastoji utvrditi pad ili rast u budućnosti. Pomoću trenda može se izračunati jedinični rast ili pad u nekom vremenskom periodu i na temelju njega pokušati predvidjeti buduće stanje i kretanja. Molekula trenda koristi se

molekulom izglađivanja te je to vidljivo unutar same molekule, a predviđena vrijednost dobiva se uglađivanjem trenutnog stanja i vremena do točke predviđanja. Primjer ovakvog modela može se pronaći kod raznih statističkih procjena i trendova. Za njega su potrebni podaci o prošlim događanjima na temelju koji se radi trend. Na primjer, trend se radi za procjenu prodaje u budućim razdobljima na temelju podataka iz prethodnog poslovanja. Frakcijski trend je krajnji rezultat molekule i predstavlja omjer razlike između predviđene i prošle prodaje sa produktom trajanja trenda i prošle prodaje. Ova molekula je kompliciranija od proteklih, budući da zahtijeva podatke iz prethodnih razdoblja na temelju kojih radi procijene. (Jim Hines, 2015.)



Slika 16. Molekula trenda

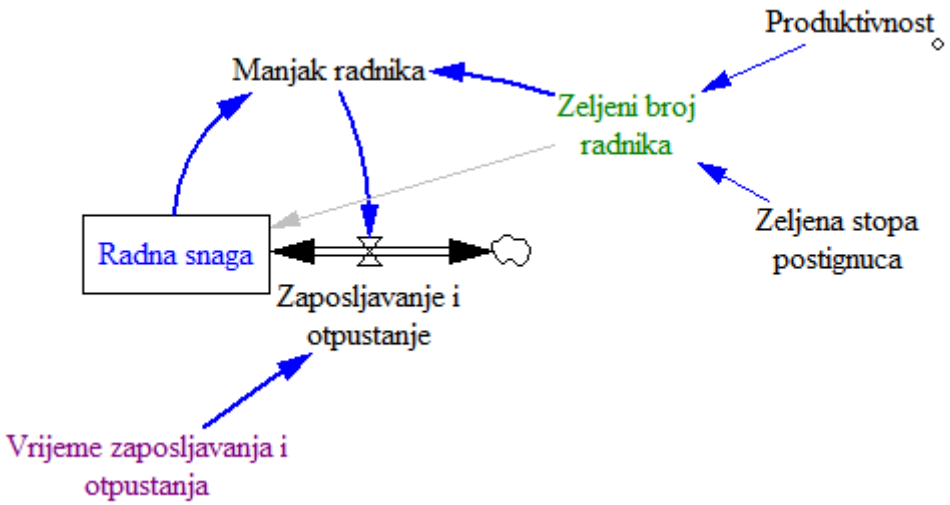
Model zahtijeva i više računanja i matematičkih jednadžbi nego druge spomenute molekule pa sam ih odlučio pokušati prikazati što jasnije na općem primjeru:

$$Frakcijski\ trend = \frac{predviđena\ količina - prošla\ količina}{prošla\ količina * vrijeme\ trenda}$$

$$Predviđena\ količina = SMOOTH(trenutna\ količina, vrijeme\ predvidanja)$$

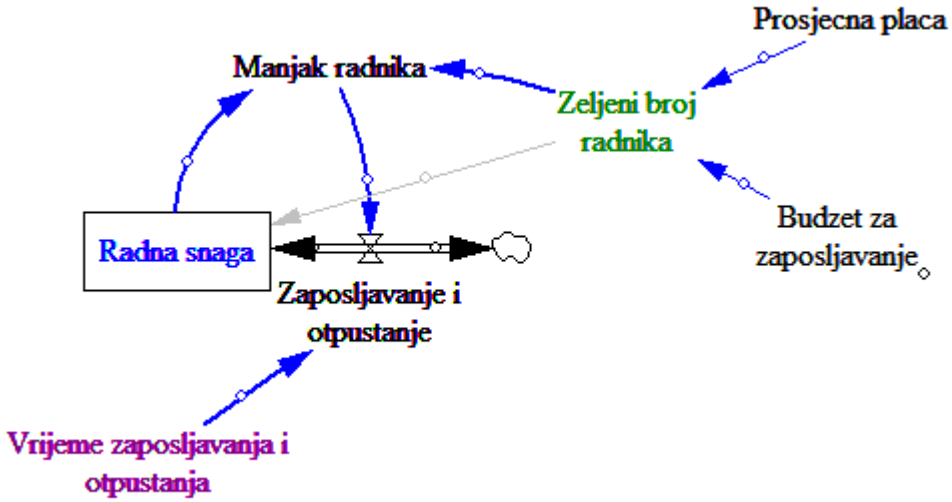
4.3.8. Radna snaga (eng. workforce)

Druga molekula koja implementira i primjenjuje molekulu izglađivanja je molekula radne snage. Ona je direktno dijete te molekule i krajnji roditelj joj je molekula smanjenja razlike. Ova molekula ima specifičnu upotrebu odnosno ona primjenjuje izglađivanje kod procesa zapošljavanja radne snage. Molekula rješava problem prikaza brojeva zaposlenika koji rade u nekoj organizaciji ili na nekom projektu. Molekula funkcionira tako da se definira željeni broj zaposlenika te su oni zatim zapošljavani i otpuštani te se njihov broj postepeno kreće prema željenom broju. Varijabla *vrijeme zapošljavanja i otpuštanja* postoji za reguliranje brzine zapošljavanja odnosno omogućuje izglađivanje vrijednosti. Ona uključuje vrijeme koje je potrebno za prepoznavanje problema nedostatka zaposlenih, odobravanje zapošljavanja novih radnika, pronalazak ljudi, intervjuiranje i njihov dolazak i integraciju. Pomoćna varijabla *željena radna snaga* može se implementirati u ovu molekuli kao samo varijabla, ali prema potrebi postoji i molekula pod imenom „željena radna snaga prema radu“ koja se može upotrijebiti na tom mjestu. Ta molekula nudi dublji pogled u varijablu *željenog broja zaposlenih* koju zamjenjuje i nudi mogućnost određivana njezine vrijednosti prema produktivnosti i postignućima. (Jim Hines, 2015.)



Slika 17. Molekula radna snaga spojena s molekulom željena radna snaga prema radu

Budući da sam molekulu radna snaga spomenuo već ranije u uvodnom djelu ovog poglavlja, model molekule iz Vensima proširio sam sa spomenutom molekulom željene radne snage prema radu. Sada se varijabla *željeni broj radnika* ne definira direktno, nego ovisi o produktivnosti i željenoj stopi postignuća. Osim te molekule, postoji i molekula radne snage prema budžetu koja također proširuje odnosno zamjenjuje varijablu *željenog broja radnika*. Ona ju proširuje varijablama *prosječne plaće i budžeta za zapošljavanje* i prema tim varijablama traži poželjnu ili povoljnu radnu snagu.



Slika 18. Molekula radna snaga spojena s molekulom željena radna snaga prema budžetu

Vrijednosti varijabla molekule radne snage može se izračunati matematičkim funkcijama prema integralu zapošljavanja i željenog broja zaposlenika:

$$Radna\ snaga = INTEG(zapošljavanje\ i\ otpuštanje, željeni\ broj\ ljudi)$$

$$Zapošljavanje\ i\ otpuštanje = \frac{manjak\ radnika}{vrijeme\ zapošljavanja\ i\ otpuštanja}$$

$$Manjak\ radnika = \ željeni\ broj\ radnika - radna\ snaga$$

$$Željeni\ broj\ radnika = \frac{budžet\ za\ zapošljavanje}{prosječna\ plaća}\ \ ili\ \ \frac{željena\ stopa\ postignuća}{produktivnost}$$

4.3.9. Postavljanje na nulu (eng. go to zero)

Kao što i njeno ime govori, ova molekula postavlja vrijednost neke akcije s trenutne vrijednosti na nulu kroz neko vrijeme. Ona je treća od osnovnih rubnih molekula iz mreže molekula koju ću obraditi. Njezina funkcionalnost je vrlo slična molekuli smanjivanja razlike te ona je praktički specifična implementacija te molekule koja vrijednost postavlja na nulu umjesto na specifičnu vrijednost. Kod ove molekule bitno je spomenuti da se varijabla smanjuje prema konstantnoj vrijednosti za razliku od molekule raspada koju ću kasnije spomenuti. Ovo je vrlo jednostavna molekula i najčešće se ne koristi zasebno, već kao dio drugih molekula kao što su raspad, razina zaštićena tokom i zaostala dostava zaštićena tokom. (Jim Hines, 2015.)

Slika 19. Molekula postavljanja na nulu

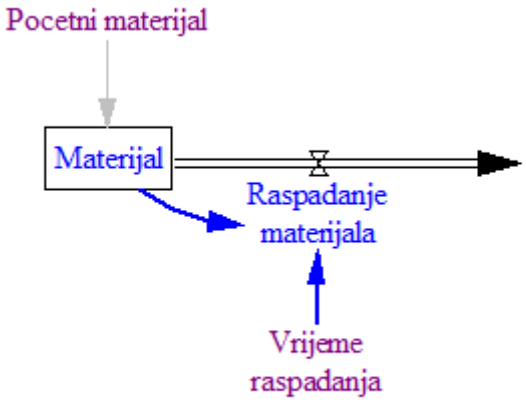
Molekula se može prikazati i sljedećom matematičkom jednadžbom:

$$Akcija\ postavljanja\ na\ nulu = \frac{trenutna\ vrijednost}{vrijeme\ postavljanja\ na\ nulu}$$

4.3.10. Raspad (eng. decay)

Molekula raspada ima vrlo sličnu funkciju kao i molekula postavljanja na nulu koja je dovođenje zadane vrijednosti na nulu. Ova molekula koristi varijablu akumulacije koju postavlja na nulu kroz neko dano vrijeme. Kod raspada, vrijednost se prazni postepeno te se iz toga dolazi do zaključka da je ona zapravo molekula glađenja sa

ciljem postavljanja na nulu. To znači da je za smanjivanje vrijednosti varijable na pola potrebno oko otprilike 70% vremena raspadanja i ta vrijednost dobiva se prirodnim logaritmom od X gdje je X=0.5 čime dobivamo otprilike 0.7 što čini 70%. Iz toga zaključujemo da se raspad događa (smanjuje) eksponencijalno prema nuli. Kao primjer ove molekule može se navesti raspadanje radioaktivnih elemenata od kojih svaki ima definirano vrijeme raspadanje te se raspadaju eksponencijalno s većim vremenom raspadanja na početku te sporijim pred kraj raspadanja. (Jim Hines, 2015.)



Slika 20. Molekula raspada

Ranije spomenuti izračun polovine raspada vrijednosti materijala kroz vrijeme kao i raspadanje cijelog materijala mogu se prikazati formulama:

$$Raspadanje\ materijala = \frac{materijal}{vrijema\ raspadanja}$$

$$\frac{Materijal}{2} = \ln(0.5)$$

4.4. Prednosti i mane molekula

Proučavanjem generičkih struktura molekula mogu zaključiti da je njihova funkcionalnost u sistemskoj dinamici vrlo bitna zbog standardiziranog pristupa koji nude. Time se ostvaruje primjena istih komponenata u svim modelima i dobiva se na kvaliteti modela. Također se postiže i lakše učenje i širenje znanja o molekulama i sistem-dinamičkim modelima, budući da su elementi modela jasno definirani i postoje smjernice prema kojima se on kreira i koristi. Molekule predstavljaju norme koje se koriste kod izrade modela u obliku komponenta sa definiranima varijablama, tokovima, vezama te jasno definiranim situacijama i slučajevima u kojima se određena molekula koristi. Mreža molekula je velika i postoji mnogo molekula koje se vrlo jednostavno implementiraju u bilo koji model prema potrebi. One se vrlo lako povezuju te imaju jasno definirane veze. Molekule su implementirane i u alatu Vensim i time je znatno olakšano upoznavanje sa modeliranjem, kreiranjem i spajanjem molekula kao i učenje o njima. U dokumentaciji objavljene su i matematičke funkcije koje podupiru model i računaju vrijednosti varijabla i tokova. Time je olakšano shvaćanje logike koja se nalazi iza modela i princip na kojem molekule funkcioniraju.

Nedostataka kod molekula također postoji, no to ne znači da su molekule loše za upotrebu, već nedostaci nude smjer u kojem se mogu dalje poboljšati i unaprijediti. Jedan od nedostataka je to što su kreirane od mnogo različitih autora i nisu pravljene prema definiranom procesu i time su se neke od njih javljale više puta. U uvodnom dijelu objavljenog izvora o molekulama od strane društva sistemske dinamike stoji da je mnogo različitih autora molekula i nekima je čak nepoznat izvor, što potkrepljuje i dokazuje tu tvrdnju. Oni su ujedinili molekule u jedan dokument sa definiranom mrežom molekula i time ih sistematizirali kao na principu roditelj-dijete. Do tada, literatura o molekulama bila je prilično raspršena i nesistematizirana te je bilo mnogo teže doći do željenih informacija. Što se tiče literature na hrvatskom jeziku, ona je vrlo ograničena i gotovo nepostojeća, posebice na temu molekula.

U nastavku ću molekule primijeniti u alatima za izradu igara kao što su Unity i LibGDX, usporediti ih te pokušati pronaći prednosti i mane svakog alata kod izrade sistem-dinamičkog modela u mobilnoj igri na Android platformi.

5. Aplikacija kamatnog računa uz implementaciju sistem-dinamičkog modela u libGDX-u

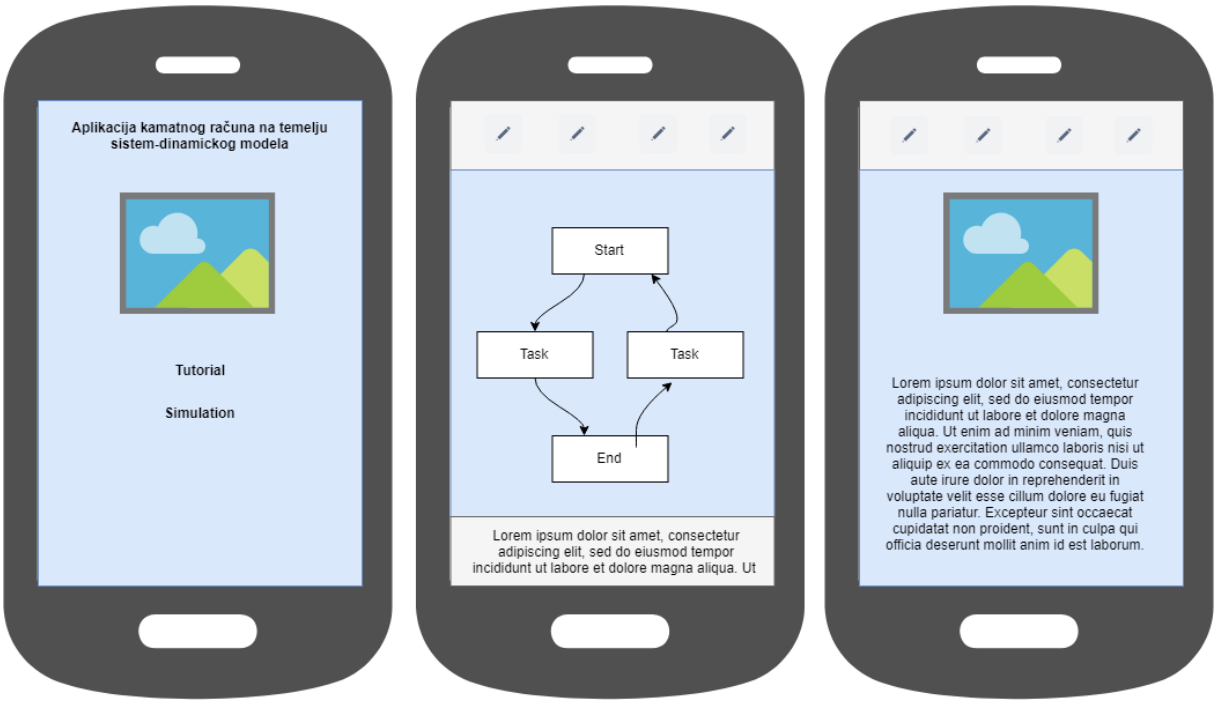
Kako bih mogao usporediti dva alata, libGDX i Unity, najprije sam se morao upoznati s tim alatima, pokušati ih koristiti u praksi i naučiti ponešto o njima. Sa programskim alatom Unity susreo sam se tokom studija i o njemu sam dobio dosta saznanja i naučio sam se koristiti njime kroz kolegij Dizajn i programiranje računalnih igara. Na tom kolegiju bio sam vođa tima od četiri člana koji je kroz jedan semestar nastave razvijao računalnu igru u Unity-ju kao temu projekta za polaganje kolegija. Tada sam se prvi puta susreo i imao priliku upoznati sa razvijanjem računalnih igara kao i alatom Unity. Stekao sam dosta znanja i iskustva, susreo se sa problemima i rješenjima i naučio o procesu razvoja igre, od smišljanja koncepta do konačnog sastavljanja potrebne dokumentacije. Na kolegiju smo izradili 3D igru iz ptičje perspektive kojoj je premisa bila prolaženje različitih nivoa preživljavanjem napada protivnika. Osim same igrivosti, igra je sadržavala i izbornik te priču prikazanu kroz stripove prije i nakon svakog nivoa. Nakon završetka projekta dobio sam dojam da je Unity dosta moćan alat koji nudi mnogo različitih mogućnosti za razvoj gotovo bilo kakve igre koju je moguće smislit. Kroz kolegij sam se susreo i s razvojem 2D igara u Unityju na primjeru igre Flappy Bird i još nekoliko drugih. Razvoj takvih igara pomalo je drugačiji, budući da su svi asseti zapravo slike i animacije - u usporedbi s 3D gdje su oni gotovi 3D modeli. Iako razvoj 2D igre izgleda jednostavnije, to zapravo ovisi o tipu igre i njezinoj veličini i kompleksnosti.

No kako bih usporedio alat Unity sa drugim alatom, libGDX-om, potrebno je bilo upoznati se i sa tim alatom, budući se do tada nikad nisam susreo s njime. Budući da od ranije imam iskustva sa Javom i izradom Andorid aplikacija, te sam se i za vrijeme studija bavio istim takvim kolegijima, smatrao sam da ne bih trebao imati problema sa razvojem jedne igre uz pomoć libGDX okvira. Odlučio sam napraviti edukativnu dvodimenzionalnu igru za mobilne platforme i u nju implementirati sistem-dinamički model odnosno neke od molekula koje sam spomenuo ranije u ovom diplomskom radu. Nakon dugog razmišljanja i raznih ideja na koju temu bi model bio najkorisniji i kako bi ga najbolje bilo prikazati, odlučio sam se za kamatni račun. Odlučio sam izraditi aplikaciju koja na temelju sistem-dinamičkog modela prikazuje kamatni račun, izračun

kredita, vrijednosti potrebnih za izračun te što jasniji prikaz konačnih rezultata i vrijednosti. Model bi se bazirao na molekuli raspada gdje bi se dug umanjivao kroz određeno vrijeme dok ne bi bio otplaćen. Već na samom početku pojavilo se dosta nejasnosti i pitanja tipa: Kako najbolje prikazati model? Kako izračunati sve vrijednosti? Kako i koje vrijednosti prikazati? I mnogo drugih tehničkih pitanja oko samog razvoja aplikacije. Uz sve to, prvi puta sam se susreo sa programskim okvirom libGDX pa mi je i to otežavalo posao.

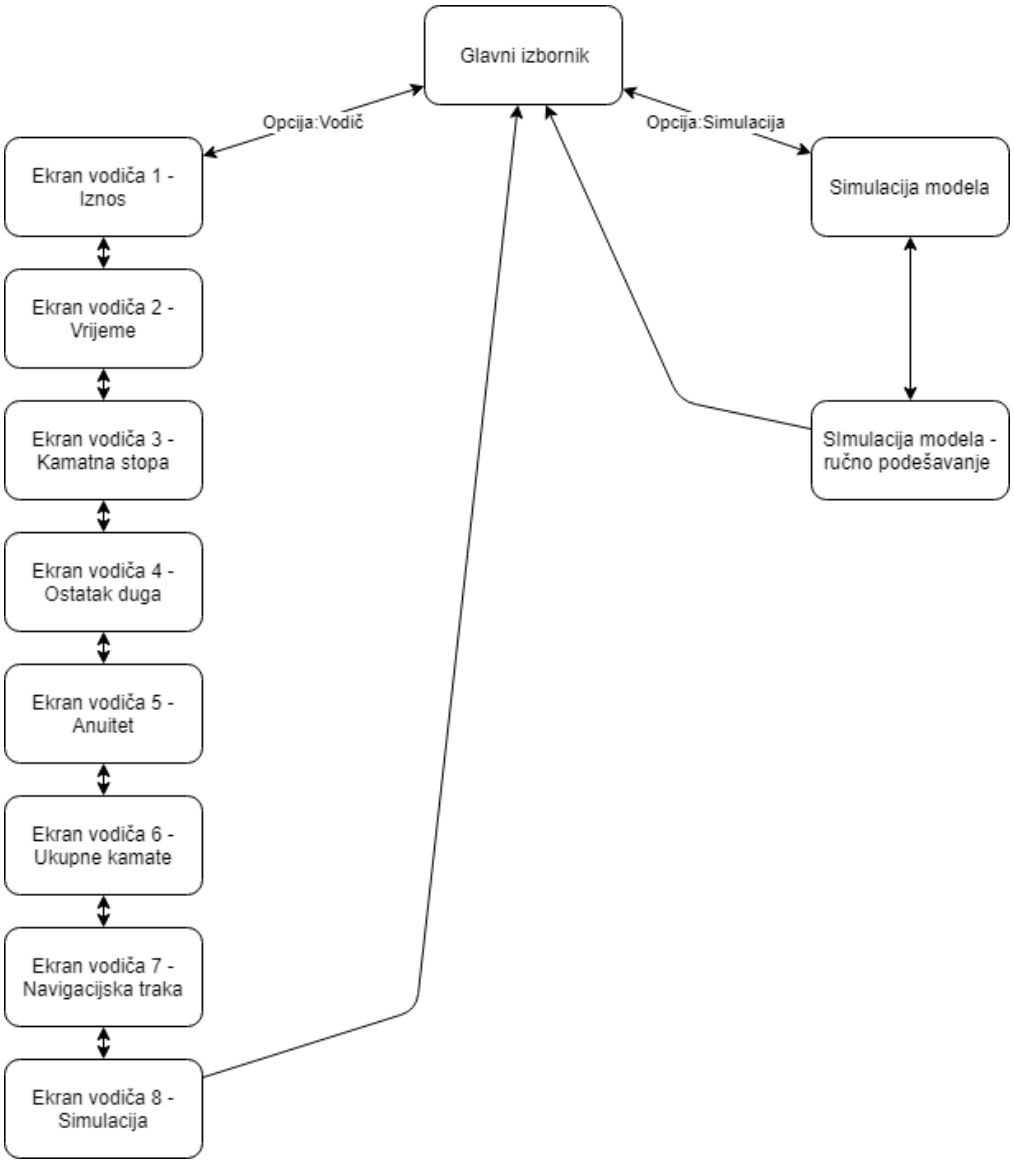
5.1. Planiranje

Najprije sam započeo sa planiranjem aplikacije i smišljanjem njezinog konačnog izgleda. Budući da aplikacija predstavlja igru, krenuo sam od glavnog izbornika koji bi se otvarao kod pokretanja igre. On predstavlja ekran na koji se korisnik uvijek može vratiti i sadrži glavnu navigaciju aplikacije. Aplikacija bi se granala u dva smjera: opcija izbornika ili priče koja bi opisivala aplikaciju i upoznavala korisnika s njom, i opcija simulacije modela koja bi bila glavna funkcionalnost aplikacije. Obje opcije bi se dalje granale kroz nekoliko ekrana. Napravio sam prototip sučelja prema kojemu sam kasnije oblikovao stvarno sučelje aplikacije. Prototip je uključivao ekrane glavnog izbornika, simulacije modela i vodiča te je okvirno prikazivao elemente koje bi svaki ekran trebao sadržavati kao i njihov međusobni raspored. Prototip ekrana glavnog izbornika sadržavao bi sliku i opcije izbornika te naslov aplikacije i bio bi vrlo jednostavan te bi prvenstveno služio za navigaciju po aplikaciji. Drugi ekran prototipa predstavlja simulaciju modela koji ću izraditi. Na vrhu ekrana postojale bi opcije navigacije i funkcionalnosti, a na dnu opisi varijabli slični onima iz vodiča. U sredini će se nalaziti grafički prikaz modela koji će predstavljati simulaciju kamatnog računa. Zadnji prototip je ekran vodiča koji bi se ponovo koristio na više mjesta kroz vodič aplikacije sa različitim sadržajima. Na slici niže prikazao sam prototip ekrane koje sam izradio u alatu Draw.io, a inspiraciju sam uzeo iz sličnih edukativnih aplikacija.



Slika 21. Prototip sučelja aplikacije

Nakon što sam isplanirao izgled i raspored ekrana, potrebno je bilo smisliti navigaciju između njih. Ekrani vodiča sadržavali bi navigaciju na prethodni i sljedeći ekran, dok bi se na zadnjem ekranu vraćalo na glavni izbornik. Osim toga, oni su zamišljeni kao informativni ekrani te nemaju nikakvu drugu funkciju osim upoznavanja korisnika sa dijelovima modela i kako se njime koristiti. Druga opcija simulacije modela imala bi nešto manje ekrana, no znatno opširnije funkcionalnosti. Ona bi sadržavala samu simulaciju modela, unos vrijednosti, informacije o varijablama i ručno podešavanje (simuliranje) modela na zasebnom ekranu. Kako bi što jasnije prikazao ovu arhitekturu navigacije kroz aplikaciju, odlučio sam izraditi dijagram toka.



Slika 22. Dijagram toka aplikacije

Na temelju dijagrama, kasnije sam razvio tok stvarne aplikacije koji je u konačnici vrlo sličan planiranome. Aplikacija se sastoji od ukupno jedanaest ekrana od kojih se većina odnosi na vodič. Svi ekrani su međusobno povezani, no nije moguće pristupiti bilo kojem ekranu sa nekog željenog ekrana. Oni su smisleno povezani te vodič čini jednu cjelinu, a simulacija drugu i međusobno su povezani samo izbornikom. Navigacija kroz ekrane prikazana je jednosmjernim i dvosmjernim strelicama i vidljivo je da neki ekrani vode samo u jednom smjeru kako bi tok kroz aplikaciju bio logičan.

5.2. Izrada

Nakon planiranja krenuo sam na izradu aplikacije, točnije postavljanjem Android Studio okruženja i dodavanjem samog libGDX okvira. Iako je libGDX okvir za razvoj u Javi on se ne dodaje u Android Studio projekt kao svi ostali, već ima vlastiti generator projekta koji kreira željene module i pod-projekte. Modularnost omogućava razvoj jedne aplikacije koja se na kraju može generirati i koristiti na različitim uređajima kao što su desktop računalo, Android, iOS pa čak i Html.

Slika 23. Generiranje libGDX projekta

Nakon što sam uspješno započeo libGDX projekt, krenuo sam s upoznavanjem alata i razvojem same aplikacije. Najprije sam odlučio ispravno postaviti programsku logiku za izračunavanje kreditnog računa te sam pri tome koristio programski okvir namijenjen za programiranje sistem-dinamičkih modela kako bi odmah podržao i izradu modela. U svoj projekt dodao sam okvir nazvan *SystemDynamics-Framework* koji sam preuzeo sa otvorenog repozitorija (https://github.com/matthiasstein/SystemDynamics-Framework) koji pruža podršku za kreiranje sistem-dinamičkih modela u Javi. On se sastoji od klasa i metoda te logike

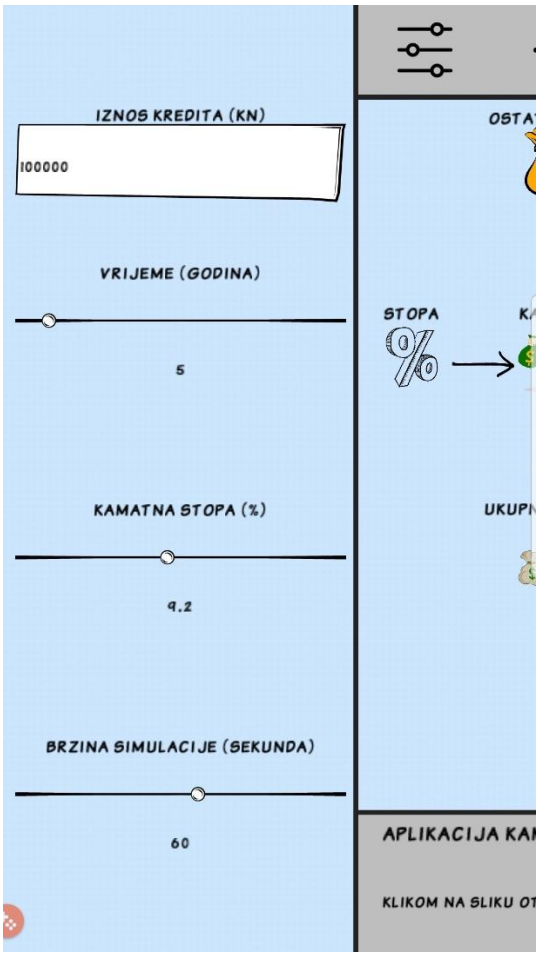
za potrebe simulacija sistem-dinamičkih modela. Neke od klasa koje sadrži su na primjer `Stock`, `Flow`, `Variable`, `Model` i `Simulation` čija sama imena jasno daju predodžbu njihove funkcije. Svaka od klasa ima i vlastite metode koje služe za upravljanje i postavljanje njihovih vrijednosti kao i međusobno povezivanje. U praksi definiranje varijabli, tokova i stogova koristeći ovaj okvir vrlo je jednostavno te ću na sljedećem isječku koda prikazati definiranje i postavljanje vrijednosti.

```
Variable amountVar = (Variable)
model.createModelEntity(ModelEntityType.VARIABLE, KEY_VARIABLE_AMOUNT);
amountVar.setInitialValue(amount);

Stock ostatakDugaStock = (Stock)
model.createModelEntity(ModelEntityType.STOCK, KEY_STOCK_OSTATAK_DUGA);
ostatakDugaStock.setInitialValue(amountVar.getInitialValue());
```

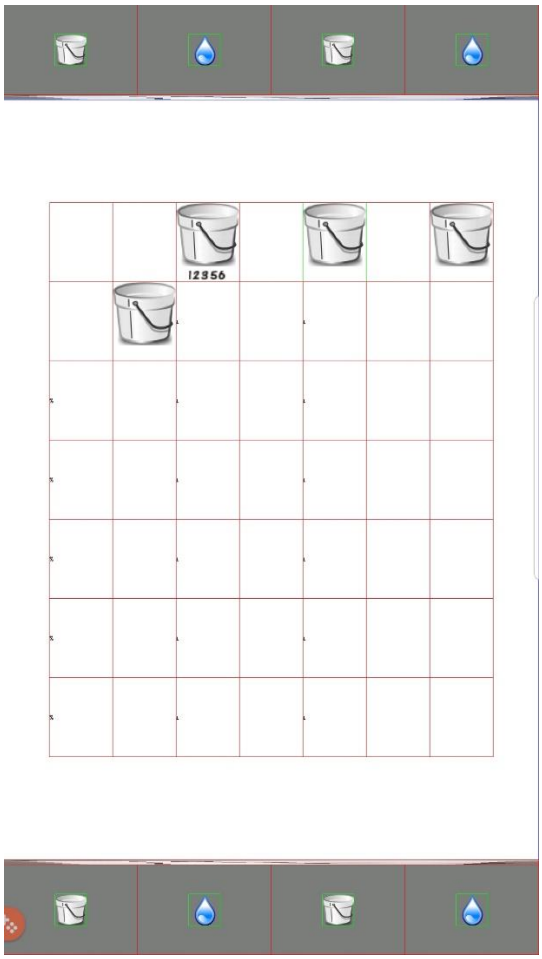
Ovaj programski okvir dosta mi je olakšao sistem-dinamički aspekt ove aplikacije te je, nakon što sam završio sa izradom programskog dijela modela, uslijedila izrada grafičkog sučelja aplikacije. Tu sam se zapravo potpuno susreo sa programiranjem u libGDX-u i njegovim elementima izrade aplikacije. Alat je prvenstveno namijenjen za programiranje 2D igara pa se samim time bazira na koordinatnom sustavu odnosno korištenjem tablica za smještanje sadržaja na ekran. Uz to postoji i komponenta *Scene2d* koja je namijenjena izradi 2D korisničkog sučelja no o njoj ću pisati više kasnije u ovom diplomskom radu kod usporedbe dvaju alata.

Kreirao sam potrebne klase `Game` i `Screen` koja predstavlja jedan ekran igre te sam napravio apstraktnu klasu koju ću koristiti kao baznu i proširivati ju u drugim klasama. Nakon toga dodao sam gornju i donju traku na kojoj su prikazani gumbi i opisi, a zatim i `NavigationDrawer` u koji sam dodao kontrole za unos početnih vrijednosti. `NavigationDrawer` je kontrola koja proširuje ekran u jednu stranu iako je prošireni dio sakriven odnosno izvan vidokruga kamere. Tako sam uspio smjestiti dodatne kontrole na ekran bez problema prenapučenosti i dobio na modernom izgledu aplikacije. U kontrolu sam pomoću tablice s negativnom pozicijom na osi x smjestio druge kontrole koje služe za input podataka od strane korisnika u aplikaciju.



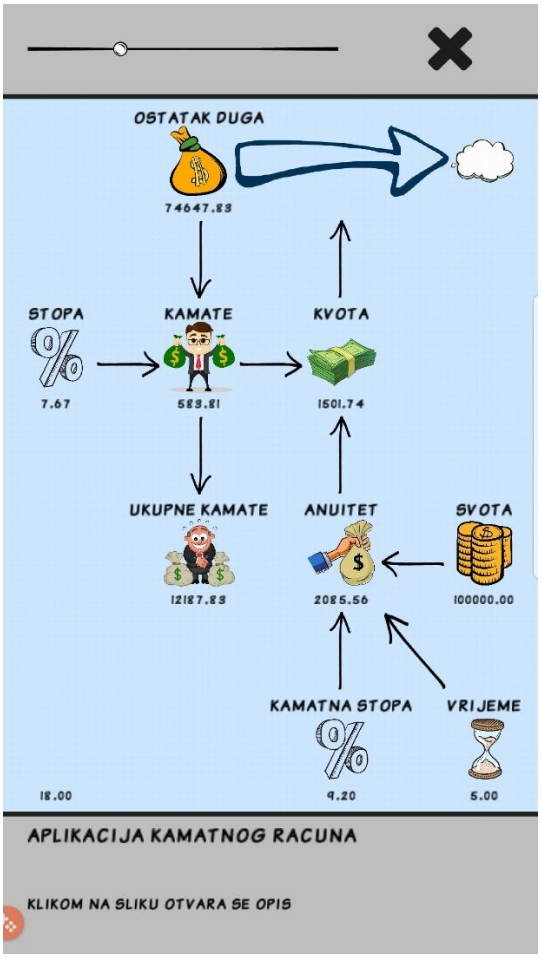
Slika 24. NavigationDrawer i druge kontrole za unos

Uz pomoć tablica započeo sam kreirati mrežu za grafički prikaz sistem-dinamičkog modela. Slike za prikaz varijabli u modela preuzeo sam s interneta, a prilagodio sam ih koristeći Adobe Photoshop CS3 kako bi odgovarale veličini i potrebama moje igre. Na slici niže prikazana je ranija verzija razvoja aplikacije gdje se može vidjeti mreža koju libGDX prikazuje, ako je uključen *debug* za bilo koji element, kako bi se lakše uočile njegove granice, budući da libGDX ne posjeduje grafičko sučelje kojim bi se na ekran lako dodale, raspodijelile i posložile kontrole i drugi elementi. Slike koje sam koristio su privremene i tek sam kasnije dodao prave slike za svaku varijablu. Svaka kontrola i element u libGDXu zahtijevaju objekt klase `Skin` kao parametar koji je zapravo skup .json, .atlas i različitih font datoteka koje predstavljaju dizajn i izgled. Ja sam preuzeo i koristio Comic dizajn preuzet sa stranice https://github.com/czyzby/gdx-skins koji je aplikaciji dao dodatan ugođaj sličniji igri kako bi edukativnu simulaciju prikazao na što pristupačniji način.



Slika 25. Slika modela u razvoju

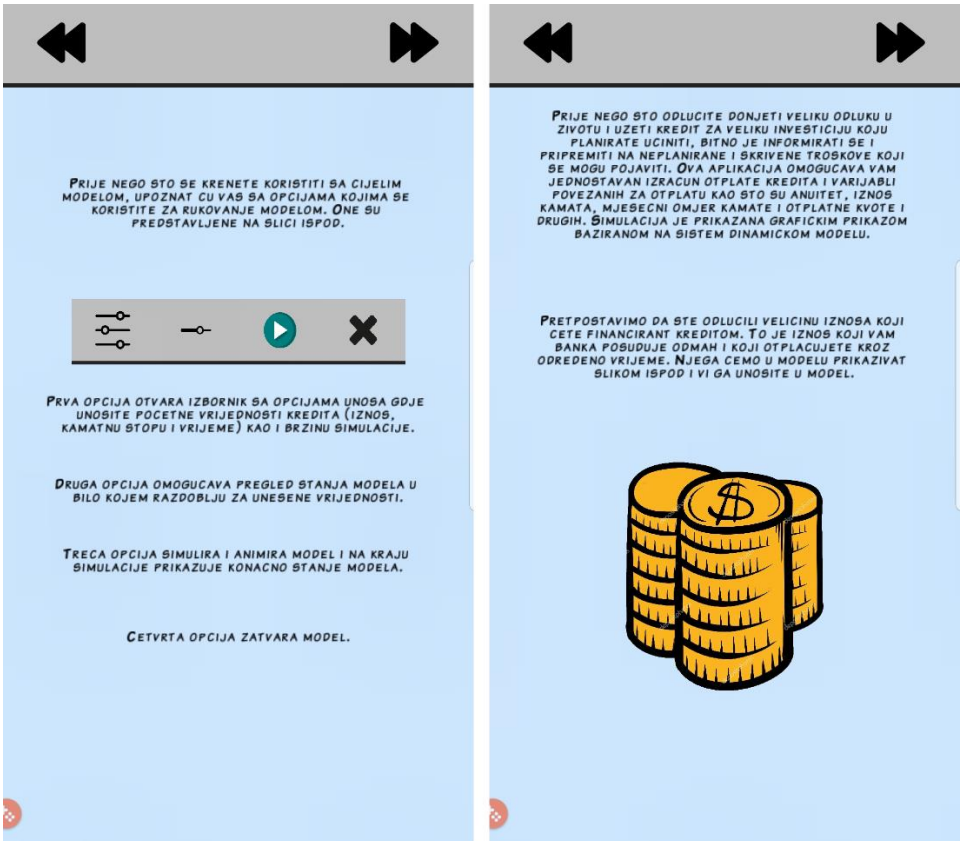
Kada sam završio s modelom dodao sam opise za svaku od varijabli koji se prikazuje u donjem dijelu alatne trake kako bi se što lakše prepoznale varijable sistem-dinamičkog modela i shvatila njihova funkcija. Uz svaku sliku dodao sam ime varijable iznad nje i vrijednost ispod nje. Kod pokretanja simulacije, pritiskom na tipku predviđenu za to, simulacija se pokreće i vrijednosti bi se trebale pomicati kako model radi izračun kredita. Tu se pojavio problem što je u kratkom vremenu potrebno ažurirati stanje tekstualnog polja mnogo puta pa sam to morao odraditi u asinkronom pozivu za svako polje. Dodao sam i brzinu simulacije kao vrijednost koja se unosi kako bi se model mogao generirati brzo ako korisnika zanima samo konačni rezultat ili sporije ako želi vidjeti postupak i tok podataka. Vidio sam priliku za dodavanjem nove funkcionalnosti za koju sam se i odlučio te sam omogućio korisniku da može sam detaljno vidjeti stanje modela u bilo kojem trenutku. On pomoću kliznog gumba može postaviti vrijeme modela na određenu točku i ispod svake varijable pokazuje se njezina vrijednost u tom trenutku. Tako na primjer korisnik može vidjeti koliko je kredita otplatio do sada ili koliko će otplatiti nakon nekog vremena za koje ga vrijednost zanima.



Slika 26. Ručno simuliranje modela, stanje u 18. mjesecu

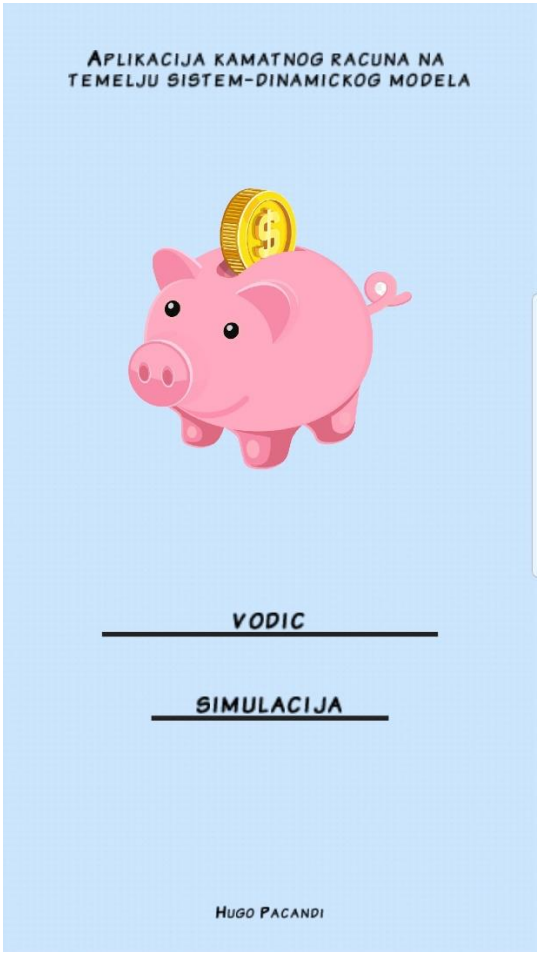
Time sam završio s kreiranjem glavne funkcionalnosti ove aplikacije – izračunom kamatnog računa uz pomoć sistem-dinamičkog modela. Taj je dio zahtijevao najviše znanja i vremena za izradu i predstavlja fokus ove aplikacije, no samo model ne čini kompletnu igru. Kako bi se korisnik što lakše snašao u igri te kako bi razumio samu poantu, smatrao sam da bi korisno bilo izraditi neki vodič kroz koji bi se korisnik upoznao sa igrom i modelom. To sam učinio na način da sam odlučio kreirati vodič koji kroz nekoliko ekrana upoznaje korisnika sa varijablama u modelu, što one predstavljaju i kako se njima koristi. Ekrani su dosta jednostavni kako ne bi zbunjivali korisnika i sastoje se od slike varijable i teksta koji ju opisuje. Na vrhu sam dodao navigaciju kroz vodič, kako bi se moglo kretati kroz njegove ekrane unaprijed, kao i unatrag prema potrebi. Za svaki sljedeći ekran vodiča samo sam proširio prvi te sam nadjačao (eng. override) nekoliko metoda kako bi pojednostavio izradu i smanjio ponavljanje programskog koda. Time sam vrlo lako napravio više ekrana dodavanjem samo drugačijih slika i tekstova u sredinu ekrana dok sam navigaciju jednostavno

preuzeo s početnog proširenog ekrana. Prvih nekoliko ekrana odnosi se na upoznavanje sa elementima modela od varijabli i tokova do vrijednosti i informacija o varijabli u smislu kamatnog računa kao i izračunu same vrijednosti. Slike su jednake kao i one u modelu da se korisnik što lakše snađe kod pokretanja modela. Drugi dio vodiča odnosi se na funkcionalnost korisničkog sučelja gdje objašnjavam uporabu svaku od kontrola koje sa nalaze na ekranu od gumbi za pokretanje simulacije do same navigacije po aplikaciji.



Slika 27. Ekrani vodiča

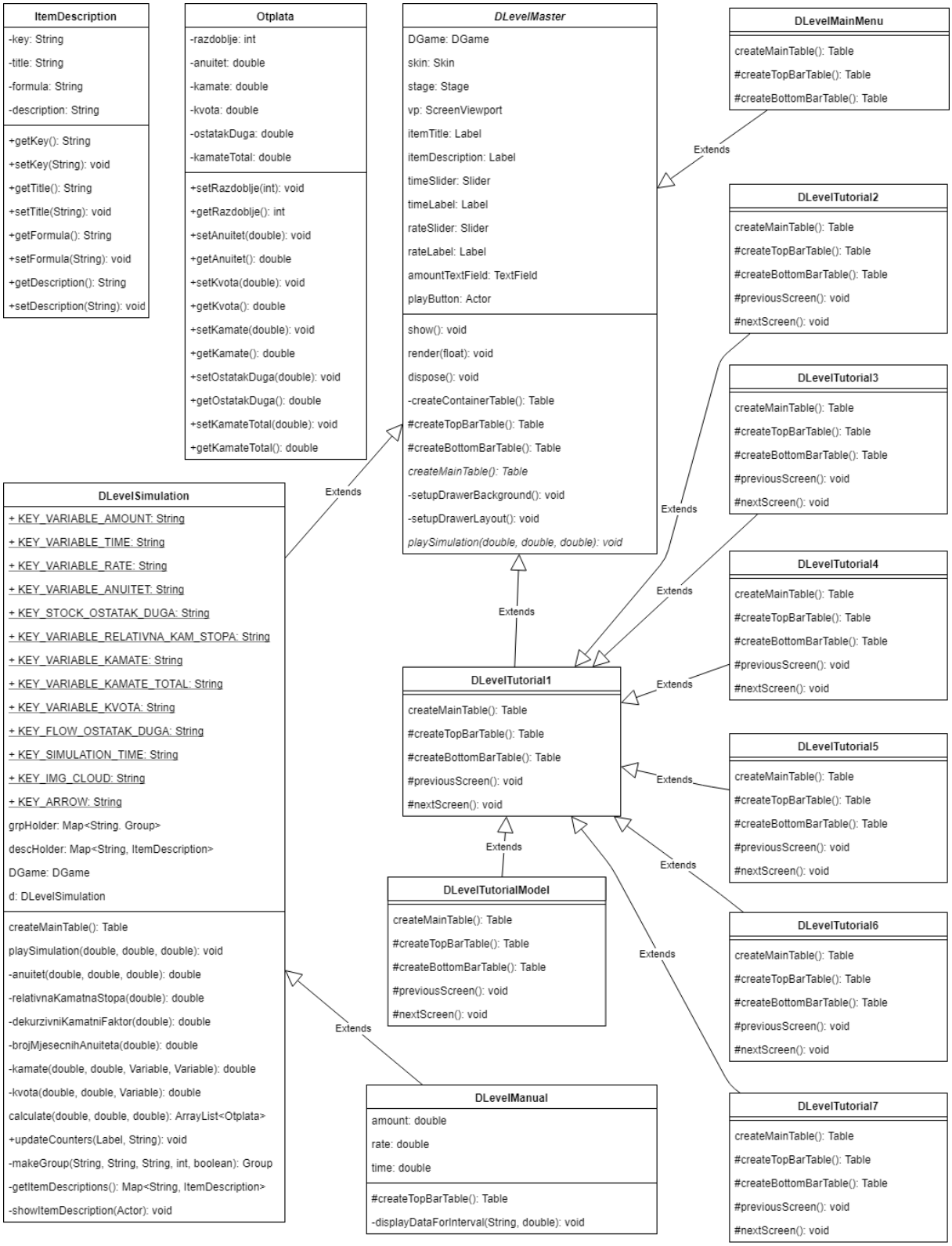
Na kraju sam morao povezati sve kreirane ekrane te kreirati jedinstveni ekran za pristup svim drugima – glavni izbornik. Odlučio sam se za dvije opcije, vodič i simulacija, iako sam razmatrao opciju da se nakon vodiča direktno odlazi na ekran modela. Time se modelu može pristupiti direktno i moguće je ignorirati vodič prema potrebi. Izbornik je vrlo jednostavan te sadrži naslov aplikacije, autora, sliku i dvije navedene opcije. On se otvara prilikom pokretanja aplikacije i služi kao početna stranica aplikacije.



Slika 28. Početni ekran aplikacije – glavni izbornik

Osim različitih ekrana kreirao sam i nekoliko klasa za prenošenje podataka, spremanje podataka i jednostavnije rukovanje, kako bi kasnije što lakše prikazao simulaciju, manualno rukovanje modelom kao i tekstualne opise prilikom klika na varijablu. Klase sam nazvao `ItemDescription` i `Otplata` te sam ih prikazao pomoću dijagrama klasa. Klasa `ItemDescription` služi za spremanje opisa svake varijable. Ona sadrži vrijednosti ključ, naslov, formulu i opis kao i funkcije za postavljanje i čitanje tih podataka. Na kraju sam koristio samo naslov i opis, dok sam formulu izbacio zbog nepreglednosti, nedostatka prostora te sam smatrao da bi formulu trebalo dodatno objasniti korisniku, što dodatno komplicira stvari. Klasa `Otplata` služi za pohranu vrijednosti simulacije u svakom vremenskom intervalu zato što klase *SystemDynamics-Framework-a* posjeduju samo početne i konačne vrijednosti. Time sam, spremajući vrijednosti u svakom vremenskom intervalu,

omogućio prikaz stanja modela u bilo kojem vremenskom intervalu kod opcije ručnog simuliranja modela.

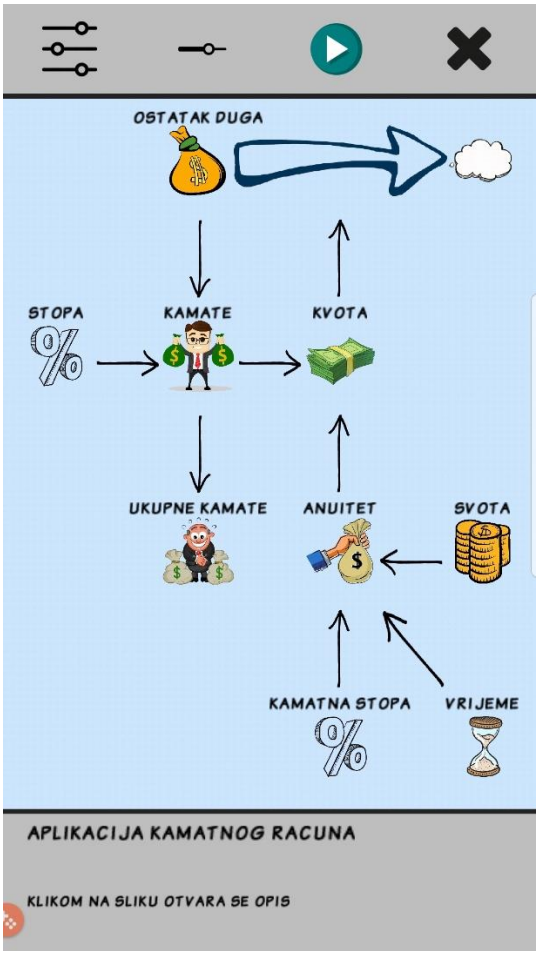


Slika 29. Klasni dijagram aplikacije

Većina klasa za prikaz ekrana aplikacije potječu iz glavne apstraktne klase koju sam nazvao `DLevelMaster`. Ona sadrži većinu grafičkih elemenata korisničkog sučelja kao i logiku za kreiranje izgleda ekrana aplikacije. Na primjer to su varijable i metode: `itemTitle`, `itemDescription`, `timeSlider`, `timeLabel`, `rateSlider`, `rateLabel`, `createContainerTable()`, `createTopBarTable()`, `createBottomBarTable()` i `createMainTable()` koje služe za kreiranje tablica te raspored i umetanje elemenata sučelja. Funkcija `createMainTable()` ne izvršava se u ovoj klasi već se implementira u klasama koje ju proširuju. Ova klasa ne postoji zasebno kao inicirani objekt već ju proširuju tri glavne klase aplikacije: `DLevelSimulation`, `DLevelTutorial1` i `DLevelMainMenu`. Klasa `DLevelSimulation` najopširnija je od svih te se u njoj odvija simulacija modela i stoga ima najviše metoda i varijabli. Nju proširuje klasa `DLevelManual` koja ima funkciju ručnog simuliranja modela i samo nadjačava funkciju za izradu izbornika na vrhu ekrana. Klasa `DLevelTutorial1` služi za izradu prvog ekrana vodiča i ona prilagođava i nadjačava metode roditeljske klase `DLevelMaster` kako bi ih prilagodila svojim potrebama dizajna. Nju proširuju druge klase vodiča kako bi se izbjegla redundancija i nadjačane metode samo izmjenjuju sadržaj ekrana u smislu teksta i slika. Treća klasa koja proširuje glavnu klasu aplikacije je `DLevelMainMenu` koja je vrlo jednostavna i koristi metode za izradu tablica roditeljske klase kako bi prikazala glavni izbornik aplikacije.

Sa tehničkog aspekta u libGDX-u koristio sam se sljedećim klasama za predstavljanje elemenata na ekranu: `Table`, `Actor`, `Label`, `Group`, `TextField`, `Texture`, `Slider`, `Image` i drugima, od kojih je mnogo dio Scene2d komponente. Programski jezik je Java i ne podržava klasične metode za razvoj Android aplikacija ni JavaFX za razvoj desktop aplikacija. Razlog tome je modularnost pa programski kod mora biti moguće pokrenuti na svim platformama koje su odabrane kod kreiranja projekta. Na primjer, ranije navedeni programski okvir *SystemDynamics-Framework* u sebi je sadržavao i JavaFX klase za izradu grafova no nisam ga bio u mogućnosti koristiti jer libGDX ne podržava JavaFX iako bi grafovi bili vrlo korisni za prikaz tokova podataka kroz aplikaciju. Na samom kraju izrade aplikacije, izradio sam slike zaslona kako bi ih dodao u ovaj diplomski rad i kod sam postavio na GitHub kako bi bio javno dostupan za skinuti svima koje zanima ili koji se žele koristit njime. Preveo sam i aplikaciju (.apk) za Android platformu te ju testirao na vlastitom uređaju kako bi

dokazao da libGDX može izraditi Android igru. Niže na slici prikazan je konačan izgleda aplikacije odnosno modela te glavne funkcionalnosti aplikacije.



Slika 30. Konačni izgled Model ekrana

Kod izrade susreo sam se i sa nekoliko problema koji su mi otežavali rad ili su barem postavili izazov koji sam morao proći. Budući da nisam imao iskustva u libGDX-u, spomenut ću neznanje kao prvi problem, iako to zapravo nije problem. Morao sam se upoznati sa sintaksom i korištenjem Scene2d komponente pa je to dosta usporavalo rad. Na primjer, kod kreiranja tablice postavio bih neka svojstva, kao što je `padding`, te se on ne bi pojavljivao na ekranu, da bi se ispostavilo da svaka ćelija tablice ima svoj zasebni `padding` te svaki redak ima svoj. Problem sam riješio korištenjem `.setDebug(true)` metode koji ima gotovo svaka klasa te se može vidjeti na ranije prikazanoj slici modela u razvoju (Slika 23.) gdje se vide rubovi svake ćelije tablice. Budući da libGDX ne posjeduje grafičko sučelje za umetanje elemenata i kontrola na ekran, bio sam primoran koristiti se debugingom. Taj nedostatak grafičkog sučelja

mogu spomenuti kao drugi problem koji mi je otežavao rad s ovim okvirom. Ono bi dosta olakšalo izradu igre, pogotovo zato što je fokus igre upravo na grafičkom dijelu aplikacije, i jedan je od najbitnijih faktora uspješne igre. No to ne znači da je u ovom okviru nemoguće napraviti uspješnu i kvalitetnu igru na razini one napravljene u drugim alatima. Kao treći problem ovog okvira naveo bih nedostatak dokumentacije i primjera zbog ne prevelike popularnosti alata. Službene dokumentacije postoji dovoljno, dobro je objašnjena i kvalitetno napisana, no nedostaje neslužbenih primjera i rješenja problema od strane internetske zajednice. Za neke od jednostavnih specifičnih problema nije moguće pronaći rješenje kratkim „googlanjem“ istog, već je potrebno prolaziti kroz mnogo službene dokumentacije kako bi se pronašlo rješenje. Na primjer, u aplikaciji sam trebao postaviti veličinu fonta u jednoj od kontrola te, nakon dugog istraživanja, u dokumentaciji sam pronašao da je na kontroli moguće povećati skaliranje fonta, no time se gubi na kvaliteti, budući da se font ne mijenja, već samo raste veličina. Da bi se povećala veličina fonta, potrebno je promijeniti datoteke povezane sa skin-om, a to nije baš toliko jednostavno. Druga alternativa bila bi podešavanje kamera na način da se približi, ali to otvara druge probleme pa nije bilo optimalno rješenje. Posljednji problem kojim sam se susreo kod izrade ove igre više se odnosi na sam Java programski jezik nego na libGDX okvir. Kao što sam ranije spomenuo, u ovom poglavlju kod simulacije sam trebao u asinkronom pozivu napraviti brojač koji ažurira stanje polja u jako kratkim intervalima kako bi korisnik dobio prividni osjećaj računanja i pada ili rasta vrijednosti. Koristio sam `Runnable` i `Thread` za ažuriranje vrijednosti, no kada sam postavio jako kratko vrijeme kredita od npr. 1 godine te velike brojeve za kamate, i iznos, kao i brzinu simulacije brojač nije uspio ažurirati grafičko sučelje dovoljno puta u sekundi te se aplikacija počela rušiti. Prvo sam pokušao pronaći neku klasu za animaciju polja, no libGDX kao ni Java ne posjeduju ništa slično, dok Androidovu klasu `ValueAnimator` nisam mogao koristiti. Na kraju sam taj problem riješio promjenom same logike te se vrijednosti ažuriraju puno rjeđe ovisno o trajanju kredita u mjesecima, ali korisnik ima prividan osjećaj da se znamenke stalno mijenjaju.

U ovoj sekciji rada spomenuo sam vlastite probleme sa kojima sam se susreo kod izrade aplikacije, no to ne znači da je sve tako negativno. Alat je kvalitetan i nudi mnogo funkcionalnosti koje olakšavaju izradu igara te bi bez njega bilo mnogo teže napraviti 2D igru u Javi. Neke funkcionalnosti su vrlo dobro riješene, kao na primjer jednostavna promjena ekrana `(Scene)`, metode kojima se upravlja životnim vijekom

aplikacije, struktura projekta je vrlo jasna te pruža izradu igre za više platforma odjednom.

Time sam završio sa opisom aplikacije koju sam izradio te smatram da sam kroz izradu stekao dovoljno znanja i upoznao sam se s alatom libGDX kako bih ga mogao usporediti sa alatom Unity kod izrade dvodimenzionalnih igara. U sljedećem ću poglavlju ovog diplomskog rada napraviti usporedbu tih dvaju alata prema nekim kriterijima koji već postoje i nekima koje ću sam definirati te prema iskustvu iz oba alata pokušati pronaći prednosti i mane svakog alata. Usporedbu ću napraviti na temelju izrade edukativnih igara koje implementiraju sistem-dinamičke modele u tim alatima, budući da je to tema mojeg diplomskog rada. Na kraju ću pokušati donijeti neke zaključke na temelju usporedbe i istraživanja, te tako i zaključiti ovaj diplomski rad.

6. Usporedba alata LibGDX i Unity kod izrade edukativnih igra

Na početku ovog poglavlja, pokušat ću približiti alate Unity i LibGDX te pružiti neke bitne informacije o svakom od njih. Objasnit ću općenito što oni predstavljaju, njihovu namjenu, mogućnosti koje pružaju, dostupnost, verzije, licence i slično. Zatim ću te alate međusobno usporediti, kako bih pronašao prednosti, mane, nedostatke i mogućnosti. Na kraju ću pokušati donijetii zaključak na temelju usporedbe i istraživanja te ću izraziti vlastito mišljenje i iskustvo koje sam stekao samim korištenjem tih alata.

Oba ova alata služe za razvoj računalnih igara i to je njihova prvobitna namjena, dok se poprilično razlikuju međusobno, budući da je prvi cjelokupni game engine koji pruža okruženje i alate za razvoj igara. Suprotno od Unityja, LibGDX je programski okvir koji se dodaje u neko postojeće IDE okruženje kao što je Android Studio, Eclipse ili slično i koristi se zajedno s nekom od tih aplikacija te koristi njihove alate. Ostale razlike i jednakosti spomenut ću nakon što približim svaki od alata zasebno.

Prije samog prelaska na alate smatram da je potrebno definirati i objasniti koje igre mogu biti edukativne. Edukativne igre odnose se na igre koje uz zabavu nude i oblik edukacije i učenja te se mogu koristiti u edukativnim okruženjima. One pomažu ljudima savladati i naučiti o nekoj temi i uče ih nove vještine. Iako ih je većina fokusirana na djecu postoje i edukativne igre za odrasle koje obrađuju ozbiljnije i kompliciranije teme. U novije doba stavlja se sve veći naglasak na uključivanje djece s teškoćama u razvoju u obrazovanje kojima je potreban drugačiji, više individualizirani pristup. Osim tog pristupa, potrebno je i prilagođavanje tehnika gdje značajnu ulogu igraju edukativne igre koje služe kao tehnika edukacije djece s teškoćama u razvoju. Na taj način utječe se na njihov psihički i fizički razvoj, dolazi do razvoja sposobnosti i vještina. Igre pomažu i pri razvoju vještine interaktivnog grupnog i individualnog rada. Slično je i kod djece s teškoćama u razvoju. Kod njih se često, osim motoričkih oštećenja, javljaju i druge psihološke i emocionalne smetnje, stoga je vrlo bitno poraditi na razvoju i stvaranju tih vještina i sposobnosti. Dokazano je kako se učenjem kroz igru lakše pridobiva pažnja i održava koncentracija djece. Naravno, edukativne igre važno je prilagoditi dobi i oštećenjima djeteta. Danas se edukativne igre sve više koriste kod edukacije i rada sa djecom s posebnim potrebama u obliku kreiranja svakodnevnih situacija u kojima se oni mogu pronaći te se simuliraju pomoću virtualne

stvarnosti. Njima takav način učenja pomaže jer je pristup kroz igru pristupačniji i zabavniji, lakše je dobiti njihovu pažnju te nemaju osjećaj učenja već igre. Primjenom ovakvog načina edukacije kod učenika ostvaruje se veća motiviranost i lakše usvajanje gradiva.

Edukativne igre najčešće sadrže priču i često zahtijevaju neki input od korisnika kao oblik interakcije te to dovodi do lakšeg savladavanja sadržaja. Razvojem mobilnih uređaja, edukativne igre počele su se pojavljivati i na tim platformama te su danas vrlo popularne na njima. Danas su takve edukativne igre sve popularnije te njihovu potrebu i korist shvaćaju svi, od vlada do učitelja i roditelja. (Igrić L, 2004.)

6.1. Unity

Unity je kompletan game engine, alat namijenjen za razvoj i programiranje računalnih igara za razne platforme. Za razvoj se može koristiti i neko drugo IDE okruženje koje podržava razvoj u C# jeziku kao što su Microsoft Visual Studio, MonoDevelop, JetBrains i drugi. Unity kao alat podržava Windowsi macOS operativne sustave te postoji i eksperimentalna verzija za Linux operativni sustav. On podržava razvoj igara za 27 različitih platformi i to su: iOS, Android, Windows, Universal Windows Platform, Mac, Linux/Steam OS, WebGL, PlayStation 4, PlayStation Vita, Xbox One, Nintendo 3DS, Oculus Rift, Google Cardboard Android & iOS, Steam VR PC & Mac, Playstation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV, Samsung SMART TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple ARKit, Google ARCore i Vuforia. Iz navedenog može se vidjeti da Unity podržava gotovo sve postojeće platforme od računala, igraćih konzola, prijenosnih konzola pa sve do virtualne stvarnosti i televizije. Platforme su grupirane po kategorijama te je moguće razvijati za više platformi unutar iste kategorije u isto vrijeme. Na primjer Unity za konzole u isto vrijeme može razvijati za Playstation 4, Playstation Vitu, Xbox One, Nintendo Switch i Nintendo 3DS. Unity se temelji na programskom jeziku C++ i C# i igre se razvijaju također u tom jeziku te posjeduje vlastiti oblik JavaScripta nazvan UnityScript. Licenciranje kod ovog alata nije besplatno te se on plaća na mjesečnoj bazi po principu pretplate. Postoje tri različite opcije: Personal, Plus i Pro.

Personal licenca besplatan je oblik namijenjen za edukativnu i osobnu uporabu te učenje i upoznavanje s alatom. S takvom licencom radili smo za vrijeme studija i ona ne dopušta zaradu od napravljenog sadržaja i igara. Ima uključene sve osnovne

funkcionalnosti, no postoje ograničenja kao što je na primjer Unity početni zaslon koji je prisutan kod svake aplikacije napravljene ovom licencom, kao i nedostatak nekih funkcionalnosti kao što su analitika i automatski izvještaji o rušenju aplikacije.

Plus i Pro su plaćene licence alata koje se razlikuju po nekim funkcionalnostima kao i zaradi koju dopuštaju. Unity Plus dopušta zaradu od $200000 godišnje, a ukoliko ona premašuje taj iznos potrebno je kupiti licencu Pro. Cijena se također razlikuje pa je Unity Plus dostupan za $25-$35 mjesečno dok se Pro licenca plaća $125.

Alat podržava razvoj dvodimenzionalnih i trodimenzionalnih igara dok je prvenstveno namijenjen za razvoj u trećoj dimenziji. S obzirom da je fokus ovog diplomskog rada na dvodimenzionalnim igrama, prvenstveno ću promatrati Unity s tog aspekta. Iako je alat namijenjen za treću dimenziju, njegove mogućnosti kod razvoja dvodimenzionalnih igara su na razini kao i drugi alati namijenjeni tome. Projekt se prema zadanim postavkama započinje kao 3D, no postoji opcija stvarnog 2D koja se može uključiti te se projekt ponaša kao 2D. U dvodimenzionalnom razvoju koriste se `Sprite`-ovi koji predstavljaju niz povezanih slika odnosno stanja u kojima slika može biti. Na primjer, kretanje nekog lika gdje se s više slika i njihovim brzim mijenjanjem dobiva efekt animacije. U trećoj dimenziji koriste se `Asseti` koji predstavljaju gotove 3D modele sa integriranim animacijama i kretnjama. Bitno je spomenuti da se i u trećoj dimenziji može razviti i dvodimenzionalna igra kod koje se koristi kamera, kako bi se dobio 2D efekt iako je zapravo projekt u trećoj dimenziji te se koriste `Asseti`. Takav tip igre, gdje su mehanike igre u 2D, a modeli su 3D, naziva se 2.5D. Za bilo koji tip dimenzije alat posjeduje grafičko sučelje kojim se lako dodaju elementi na scenu. `Sprite`-ovi se dodaju na scenu zasebno te ih je moguće grupirati, mijenjati im veličinu, animirati ih i dodavati im Collider-e kako bi imali interakciju i sudarali se sa drugim assetima i okolinom. Dodavanjem i korištenjem sile likovi se mogu kretati i po prostoru. Sve ove akcije dodaju se u grafičkom sučelju alata, dok se pomoću skripti koje se dodaju na objekte i programira logika i dodaju se vrijednosti koje utječu na objekte. Za programere koji si ne mogu priuštiti plaćanje grafičkih dizajnera i nisu iskusni u kreiranju vlastitih slika i modela, Unity nudi vlastiti Asset Store, mjesto gdje se mogu preuzeti besplatni ili plaćeni modeli i slike koji su kreirani od strane korisničke baze.

Korisnička baza Unityja je vrlo jaka i zastupljena te postoji mnogo službenih i neslužbenih izvora u obliku foruma, članaka i primjera iz kojih se može učiti i tražiti rješenja problema. Službena dokumentacija Unityja visoko je kvalitetna, jasna i

opsežna te je dostupna svima besplatno. Grafičko sučelje također je jednostavno za korištenje te time omogućava pristup razvoju igara mlađim generacijama koje se tek susreću s programiranjem.

Ukratko Unity je jedan opsežan alat sa vrlo mnogo mogućnosti koji se trudi biti najbolji u svemu i zadovoljiti sve zahtjeve i potrebe od programera početnika, iskusnih do velikih i malih kompanija koje se bave razvojem igara. (D.Penic, 2018.)

6.2. LibGDX

LibGDX je programski okvir namijenjen za razvoj igara u programskom jeziku Java sa dijelovima C++ komponenti razvijen od kompanije BadLogicGames. Alat nudi mogućnost kreiranja projekta kroz ugrađeni izbornik kojim se generiraju potrebne i željene komponente te se zatim otvara u jednom od IDE okruženja za razvoj aplikacija u Javi kao što su Android Studio, IntelliJ, NetBeans, Eclipse i drugi. S obzirom da LibGDX nije samostalan alat, on se samo dodaje kao programski okvir neovisno o operativnom sustavu, već je bitno samo da razvojno okruženje podržava željeni operativni sustav. Alat omogućava razvoj za različite okoline i platforme kao što su: Windows, Linux, Mac OS X, Android (2.2+), BlackBerry, iOS, Java Applet i Javascript/WebGL. Alat je otvorenog koda i dostupan je svima besplatno bez novčanih naknada i samo je zaštićen licencom Apache 2.0. Alat se može dodati u postojeći projekt korištenjem Gradle alata i starim načinom preuzimanjem alata s interneta i ručnim dodavanjem u projekt.

On podržava razvoj igara u drugoj i trećoj dimenziji, no fokusiran je i nudi mnogo više mogućnosti za razvoj dvodimenzionalnih igara, što ga čini idealnim za razvoj jednostavnijih edukativnih 2D igara. Kod samog razvoja grafike u drugoj dimenziji koristi se Scene2d sustav grafova koji služi za kreiranje korisničkog sučelja kroz hijerarhije elemenata nazvanih jednim imenom – `Actor`. To je klasa koju proširuju svi elementi grafičkog sučelja kao što su slike, oznake, tekstualna polja i drugi (`Image`, `Label`, `TextField`..). Oni služe za davanje inputa u aplikaciju, prikaz informacija na ekranu, izradu izbornika i drugih elemenata korisničkog sučelja. Najbitniji element Scene2d sustava je klasa `Stage` koja sadrži kameru i `SpriteBatch` u koji se spremaju svi Actori na ekranu. Ona 'sluša' za razne inpute kao primjerice dodir na ekran i kada ih primi šalje ih određenom `Actor-u`. Slike se na ekran dodaju putem

tekstura koje ih direktno čitaju iz slika koje su spremljene interno. LibGDX sadrži i 2D Particle Editor kojim se kreiraju čestice kao i Box2D koji služi za dodavanje fizike u igru. Box2D je jedan od najpopularnijih biblioteka za razvoj 2D fizike i nudi klase i metode za pomicanje i interakciju objekata na ekranu.

LibGDX ne posjeduje grafičko sučelje, već se sve odvija kroz Java programski kod, biblioteke i spomenute klase. To može biti i dobra i loša strana ovisno o potrebama i iskustvu. Za same početnike grafičko sučelje dosta olakšava pristup jer sakriva programski kod koji je potrebno razumjeti da bi funkcionirao. Osim toga omogućava lakše postavljanje grafičkih elemenata na scenu i njihovo podešavanje. No time se u isto vrijeme gubi na mogućnostima koje alat može pružiti pa LibGDX nudi mnogo više opcija i mogućnosti naspram drugih alata, iako nije toliko pristupačan početnicima.

Što se tiče korisničke baze ona je postojana, ali ne previše raširena, a alat se pretežito koristi za dvodimenzionalan razvoj kompliciranijih igara. Službena dokumentacija alata dostupna je na službenim stranicama libGDX-a (https://libgdx.badlogicgames.com/documentation/) i vrlo je opširna te su neki dijelovi dosta stručni. Zahtijeva se znanje o programiranju i znanje osnovne sintakse Jave. Neslužbenih izvora ima manje, no to se može povezati sa popularnošću, kompleksnosti i pristupačnosti alata. Postoje forumi i primjeri, a rješenja problema dokumentirana su slabije od ostalih alata i ponekad je teže pronaći neke specifične slučajeve.

Sve u svemu alat je vrlo moćan i nudi vrlo mnogo onima koji to znaju iskoristiti dok za upoznavanje sa programiranjem i razvojem igara nije toliko pristupačan. Igre koje se mogu napraviti mogu biti vrlo napredne i na svim platformama raditi brzo i efikasno. (Slant, 2018.)

6.3. Usporedba

U ovom ću poglavlju usporediti alate, pronaći razlike, prednosti i mane prema kriterijima koje sam definirao. Oni uključuju opseg platforme, pristupačnost, mogućnosti, cijenu i dostupnost, konačan produkt u smislu gotove igre i službene i neslužbene resurse. Na kraju ću iznesti i svoja iskustva i mišljenje na temelju rada i razvoja aplikacija u oba alata.

6.3.1. Opseg platformi

Opseg platformi uzima u obzir platforme za koje se može razvijati u određenom alatu te je taj kriterij vrlo jednostavan. Unity podržava zapanjujućih 27 platformi od mobilnih, desktop, konzola, virtualne stvarnosti pa čak i pametne televizore. S druge strane, LibGDX nudi mogućnost razvoja za samo 7 platformi od kojih se može zaključiti da su one sve najpopularnije. Unity sa svojom širinom spektra platformi daleko pobjeđuje LibGDX, no to je samo u teoriji. U praksi se gotovo nikad ne razvija za sve platforme već se uzima nekoliko relevantnih ili najpopularnijih pa ovaj kriterij nije ni toliko bitan kod odabira alata, ukoliko cilj nije pokrivanje što više platformi.

6.3.2. Pristupačnost

Što se tiče pristupačnosti i dostupnosti ovaj kriterij uzima u obzir koliko je jednostavno služiti se alatom. Unity je sa svojim grafičkim sučeljem i jednostavnim, iako opsežnim pristupom, odličan alat za upoznavanje sa razvojem igara kao i općenito sa programiranjem. On je idealan za srednjoškolske i studijske programe gdje se učenici i studenti prvi puta susreću sa razvijanjem i programiranjem. Postoji mnogo vodiča, dokumentacije, videozapisa i ostale literature koja omogućava još lakše učenje te je moguće napraviti igru sa minimalnim ulaskom u programiranje. Unity je dobar za učenje izrade igara, dok nije idealan za upoznavanje i savladavanje programskog jezika C#, budući da je interakcija sa programskim kodom mala.

Sa druge strane LibGDX nudi daleko kompliciranije sučelje jer nema grafičkog, već se sve odvija kroz programski kod. Za početnike i one, koji se prvi puta susreću sa programiranjem i razvojem igara, to bi mogao biti veliki ili čak preveliki korak, budući da je znanje programskog jezika Java nužno za razvoj. No takav pristup može biti i prednost zato što se odabirom LibGDX-a može naučiti daleko više o sintaksi i programiranju nego kroz Unity.

Ovisno o stupnju znanja, spremnosti na učenje, želji za učenjem i usavršavanjem, oba alata imaju nešto za pružiti korisniku. Početnicima Unity može pružiti lakše učenje sa manjim frustracijama i mogućnosti odustajanja dok LibGDX nudi više znanja, ali i teži pristup i put k znanju i savladavanju.

6.3.3. Mogućnosti

Izuzev širine platformi koju alati podržavaju, ovaj kriterij mogućnosti predstavlja širinu alata i opcije koje pruža za razvoj. Unity je pretežito centriran na razvoj trodimenzionalnih igara, no to ne znači da je druga dimenzija zapostavljena. On podržava grafiku, fiziku, skriptiranje, multiplayer, mrežni rad, zvuk, animacije, UI, alternativne stvarnosti, Asset Store i mnogo drugih funkcionalnosti. Samim time alat je vrlo opsežan i pokušava zadovoljiti sve potrebe različitih korisnika i više cilja na kvantitetu nego na kvalitetu.

Nasuprot tome LibGDX je alat koji se fokusirao na izradu dvodimenzionalnih igara te mu je to primarni cilj, dok također podržava i razvoj u trećoj dimenziji. Alat podržava i mnogo funkcionalnosti kao što su 2D grafika, 3D grafika, zvuk, upravljanje memorijom, rukovanje unosom, mrežni rad, lokalizacija i ostale mogućnosti kao editor za 2D čestice, editor za teksture i editor za puteve. Alat je vrlo snažan, iako je za iskorištavanje toga potrebno znanje iskustvo te igre koje su napravljene njima rade jednako kvalitetno i dobro na različitim platformama.

6.3.4. Cijena i dostupnost

Parametar cijene i dostupnosti bitan je za korisnike i ovisan o tipu igre koja se razvija. Kod ovog kriterija javlja se i pitanje vrijednosti jer je LibGDX samo programski okvir, dok Unity nudi cijelo okruženje za razvoj igara. Samim time alati su različite prirode te je jedan besplatan i otvorenog koda, dok se drugi plaća i nudi cijelo okruženje.

Unity je dostupan u tri različite varijante: Personal, Plus i Pro. Personal licenca je besplatna za neprofitno korištenje, učenje i akademsko obrazovanje, dok se druge dvije plaćaju ovisno o zaradi od napravljenog sadržaja. LibGDX je kompletno besplatan, potrebno je samo spajanje na internet te preuzimanje alata. Licenciran je pod licencom Apache 2.0.

U slučaju edukativnih igara, one su najčešće jednostavnije prirode te su razvijene od strane manjih timova ili čak pojedinaca. Samim time potrebno je da alat bude besplatan ili što jeftiniji te je prema su alati u ovom slučaju podjednaki. Unity se preporuča za razvoj igara pod Personal licencom ako se od igre ne planira ostvarivati

dobit jer je u tom slučaju i on besplatan. Za veće igre i zaradu od igara Unity će tražiti naknadu za licencu te tada LibGDX postaje privlačnija opcija, kada se u obzir uzima kriterij cijene.

6.3.5. Konačan produkt

Konačan produkt odnosi se na rezultat samog programiranja, odnosno gotovu igru gdje kriteriji za mjerenje kvalitete nisu tako jednostavni. Podržavanjem 27 platformi konačna igra u Unityju može zadovoljiti daleko širu korisničku bazu ako je to cilj igre. Kvaliteta same igre je teško mjeriva, budući da ulogu igraju i druge varijable, kao na primjer kvaliteta koda. Samim time teško je procijeniti hoće li se ista igra napisana u različitim alatima ponašati bolje ili lošije na istoj platformi. Isto vrijedi i za edukativne igre te, bez obzira koji tip igre se razvija, vrlo je teško procijeniti sam utjecaj alata na kvalitetu.

6.3.6. Resursi i dokumentacija

Kao što sam ranije spomenuo u zasebnim poglavljima svakog alata, dokumentacije postoji za oba alata. Službene dokumentacije vrlo su kvalitetne, detaljne, opsežne i jasne i tu nema prevelikih razlika između alata. No Unity je daleko zastupljeniji alat te, samim time, neslužbenih izvora ima mnogo više nego za LibGDX. Postoji mnogo više uputa, videozapisa, članaka, foruma i drugih oblika resursa za Unity nego što postoji za LibGDX i to je usko povezano s popularnošću alata, korisničkom bazom i primjenom samih alata. Unity je alat sa širom upotrebom i kao takav je bolje dokumentiran. LibGDX je pretežito namijenjen za razvoj u 2D i onima koji su savladali osnove te žele napraviti što kvalitetniju igru.

6.3.7. Vlastita iskustva

Na samome kraju usporedbe ova dva alata iznijet ću vlastita mišljenja nakon programiranja u ovim alatima te nakon stečenog iskustva u oba. Sa Unity -jem sam se bavio kroz kolegije za vrijeme studija i to je alat velikog opsega te mi se iz početka

činio dosta kompliciran. Koristili smo besplatnu verziju za edukativnu uporabu – Personal edition. Za vrijeme nastave, upoznao sam se sa osnovama alata gdje smo radili u 2D i 3D te sam zatim svoje znanje proširio izradom grupnog projekta, odnosno igre, kroz trajanje od jednog semestra. Igra je bila namijenjena za Windows računala i to je bio moj prvi susret sa programiranjem i dizajnom računalnih igara. Najviše sam se bavio izradom i programiranjem animacija, pisanjem skripti i izradom sadržaja u smislu `Asseta` i slika za igru. Najteže mi se činilo programiranje skripti za projektile koji su se ispaljivali iz lika kojim je upravljao igrač. Tu se javljalo dosta problema, u smislu greški i bug-ova i morao sam naučiti sintaksu C#, budući da sam bio slabo upoznat sa njime. Grafičko sučelje alata bilo mi je vrlo jasno i pristupačno te sam se u njemu vrlo brzo snašao. Dokumentacije je bilo vrlo mnogo te sam na internetu uspio pronaći rješenje za gotovo sve probleme koji su me snašli. Pomoglo je i to što smo imali iskusnijeg kolegu koji je ranije radio s alatom te je imao više iskustva od mene. U konačnici, bio sam zadovoljan kako je projekt prošao i igrom koju smo napravili kao tim od četiri člana. Naučio sam dosta o samom razvoju igara, o samome alatu i timskom radu.

Za LibGDX sam čuo za vrijeme kolegija Mobilne Aplikacije i odmah me zainteresirao, budući da sam imao iskustva sa razvojem mobilnih aplikacija u Javi. Tada sam shvatio da je računalnu igru moguće napraviti i u Java programskom jeziku te sam se odlučio da bih mogao to i napraviti kroz diplomski rad. U programiranje igre nisam krenuo bez znanja Jave pa to nije bila prepreka mojem razvoju aplikacije. Isprva mi je alat bio pomalo ograničavajući, budući da sam do tada igru radio većinom pomoću grafičkog sučelja te me njegov nedostatak me pomalo iznenadio i pomalo mi otežao razvoj. Kasnije sam shvatio da to i nije preveliko ograničenje zato što je sav potreban dizajn moguće i napraviti programskim putem. Puno sam koristio tablice i koordinatni sustav, kako bi pozicionirao elemente i kontrole na ekran te kako bi im mijenjao veličine i svojstva. Razvijao sam mobilnu igru za Android mobilnu platformu te je to platforma na kojoj imam najviše iskustva, stoga tu nisam imao problema. Najprije sam nacrtao okvirni izgled igre te sam nakon toga izradio sistem-dinamički model na kojem se aplikacija bazirala u grafičkom obliku. Grafički dio potkrijepio sam kodom koristeći ranije spomenuti okvir za sistemsku dinamiku. Na kraju sam animirao model, dodao navigaciju po ekranima i kreirao simulaciju modela. Dodao sam i glavni izbornik igre i vodič u kojemu se korisnik upoznaje sa igrom te sam time upotpunio igru. Prilikom razvoja susreo sam se s nekoliko problema koje sam detaljnije objasnio

u prethodnom poglavlju i koji su mi otežali razvoj. Na kraju sam ih sve uspio riješiti uz pomoć dokumentacije. Dokumentacija za ovaj alat nije opsežna kao za Unity, ali ima je i više nego dovoljno da se razvije kompletna igra. Koristio sam službenu dokumentaciju te sam ju dopunio temama sa foruma i videozapisima za specifične slučajeve problema. Zadovoljan sam aplikacijom koju sam napravio i drago mi je što sam se odlučio isprobati novu tehnologiju razvoja igara koja to radi na malo drugačiji način od onog s kojim sam se sreo u Unityju. Preporučam Unity za upoznavanje sa programiranjem, razvojem igara i onima koji nisu toliko iskusni u tome te zatim kada se stekne iskustvo i upozna sa osnovama razvoja preći na drugi alat. Ukoliko želite isprobati novu tehnologiju, drugačiji pristup izradi ili novi programski jezik LibGDX je odličan programski okvir za učiniti baš to. On nudi vrlo snažno okruženje za izradu 2D i 3D igara za koje je potrebno dosta znanja, ali to rezultira u kvalitetnim i funkcionalnim konačnim rezultatom - igrama.

7. Zaključak

Ovaj diplomski rad obrađuje temu implementacije sistem-dinamičkih modela u edukativnim računalnim igrama uz korištenje LibGDX okvira. Kako bih pristupio implementaciji sistem-dinamičkih modela u edukativnim igrama, najprije sam odlučio pojasniti i približiti sistemsku dinamiku, njezin nastanak kroz povijest, pojavu u svijetu računala i širenje na cijeli svijet. Spomenuo sam programske jezike koji su se koristili i one koji se još danas koriste. Jedan od njih – Vensim – sam i isprobao kreirajući sistem-dinamičke modele.

U sljedećem sam poglavlju pojasnio što su to generičke strukture kod izrade sistem-dinamičkih modela, odnosno molekule. Zaključio sam da je njihova svrha standardiziranje dijelova sistem-dinamičkih modela kako bi se koristili unificirani dijelovi i time se olakšala izrada, dijeljenje i učenje sistem-dinamičkih modela. Neke od molekula sam detaljnije razradio kreirajući ih u alatu Vensim, pronalaženjem primjene u praksi i prikazivanjem matematičkih formula od kojih su sačinjene i koje ih podupiru.

Kako bih sistemsku dinamiku povezao sa programiranjem i LibGDX okvirom, općenito istražio alat, napravio usporedbu i donio neke zaključke, odlučio sam napraviti model koji implementira molekule u tome alatu. Odlučio sam se za izradu kamatnog računa na temelju sistem-dinamičkog modela u LibGDX-u kao edukativna igra koja pomaže pri svladavanju istog te pomaže kod računanja troškova i uzimanja kredita. Kroz izradu upoznao sam se sa LibGDX okvirom i stekao dovoljno znanja da mogu napraviti usporedbu sa drugim alatom za istu namjenu – Unityjem.

Na temelju iskustva iz Unity-ja koje sam stekao tokom studija i znanja o LibGDX-u prilikom izrade ove aplikacije napravio sam usporedbu tih alata. Oba alata su besplatna za osobnu i edukativnu upotrebu. Unity je alat više fokusiran na pokrivanje što više mogućnosti te se ne fokusira na isticanje u jednoj. On nastoji zadovoljiti što veću korisničku skupinu od početnika do profesionalaca. Time u isto vrijeme ostvaruje jednostavnost koju traže početnici, no ograničava profesionalce koji žele više mogućnosti i napredne alate. Nasuprot tome, LibGDX je programski okvir koji se ubacuje u drugo razvojno okruženje i ne posjeduje grafičko sučelje, već se služi tablicama i koordinatnim sustavnom za razmještaj kontrola po ekranu. On nudi mnogo, ali i traži dosta znanja i vremena za svladavanje. Ako se tek upoznajete sa

programiranjem i razvojem igara, Unity vam nudi i više nego što je potrebno. Kada želite nešto više od alata za programiranje, isprobati novo okruženje, naučiti novi jezik ili kreirati vrhunsku 2D igru LibGDX je alat koji valja isprobati.

8. Literatura

1. ClubOfRome, (2018.), About us & history, dostupno 13.08.2018. na https://www.clubofrome.org/about-us/history/
2. Colleen Lannon, (2018.), Causal loop construction: The basics, dostupno 13.08.2018. na https://thesystemsthinker.com/causal-loop-construction-the-basics/
3. Craig W. Kirkwood, (1998.), System Behaviorand causal loop diagrams, dostupno 13.08.2018. na http://www.public.asu.edu/~kirkwood/sysdyn/SDIntro/ch-1.pdf
4. Daniel H. Kim, (1992.), Guidlines for Drawing Causal Loop Diagrams, dostupno 13.08.2018. na http://www.cs.toronto.edu/~sme/SystemsThinking/2014/GuidelinesforDrawingCausalLoopDiagrams.pdf
5. Daniel H. Kim, (1999.), Introduction to systems thinking, dostupno 13.08.2018. na https://thesystemsthinker.com/introduction-to-systems-thinking/
6. Dario Penic, (2018.), How to choose an Android game engine: LibGDX vs Unity, dostupno 13.09.2018. na http://dariopenic.com/how-to-choose-an-android-game-engine-libgdx-vs-unity/
7. Ed Cunfill, (2018.), Connecting systems thinking and action, dostupno 13.08.2018. na https://thesystemsthinker.com/connecting-systems-thinking-and-action/
8. ETHW, (2017.), Jay W. Forrester Biography, dostupno 13.08.2018. na https://ethw.org/Jay_W._Forrester
9. Gene Bellinger, (2004.), Balancing Loop, dostupno 13.08.2018. na http://www.systems-thinking.org/theWay/sba/ba.htm
10. Gene Bellinger, (2004.), Reinforcing Loop, dostupno 13.08.2018. na http://www.systems-thinking.org/theWay/sre/re.htm
11. Igrić L, (2004.), Društveni kontekst, posebne potrebe/invaliditet/teškoće u razvoju i edukacijsko uključivanje
12. Jadranka Božikov, (2006.), Modeliranje i simulacija, dostupno 13.08.2018. na https://bib.irb.hr/datoteka/347082.modeliranje_i_simulacija_-_v2a2.pdf

13. Jay W. Forrester, (1996.), The Begining of System Dynamics, dostupno 13.08.2018. na http://web.mit.edu/sysdyn/sd-intro/D-4165-1.pdf
14. Jay W. Forrester, (2011.), Chapter 20: Profile sin Operations Research, dostupno 13.08.2018. na https://www.researchgate.net/publication/227267121_Jay_Wright_Forrester
15. Jim Hines, (2015.), Molecules of Structure, dostupno 13.08.2018. na https://www.systemdynamics.org/assets/docs/MOLEC2_03.pdf
16. Mckinsey Quarterly, (1995.), The beginning of system dynamics, dostupno 13.08.201.8 na https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/the-beginning-of-system-dynamics
17. Slant, (2018.), Libgdx vs Unity comparison, dostupno 13.09.2018. na https://www.slant.co/versus/1047/1050/~unity_vs_libgdx
18. Sondoss ElSwah, (2013.), Generic and reusable structure in systems cynamics modelling: roadmap of literature and future research questions, dostupno 13.08.2018. na https://www.systemdynamics.org/assets/conferences/2015/proceed/papers/P1078.pdf
19. Systemdinamics, (2018.), Introduction to System Dynamics, dostupno 13.08.2018. na http://lm.systemdynamics.org/what-is-s/
20. Systemdynamics, (2018.), Infographic – System dynamics: 60 years of progress, dostupno 13.08.2018. na https://www.systemdynamics.org/assets/Infographic/SDS_infographic-WEB4.png
21. Vensim, (2018), Modeling with Molecules, dostupno 13.08.2018. na https://vensim.com/modeling-with-molecules-2-02/
22. Vensim, (2018.), Vensim History, dostupno 13.08.2018. na http://vensim.com/vensim-history/
23. Wikipedia, (2018.), STELLA_(programming_language), dostupno 13.08.2018. na https://en.wikipedia.org/wiki/STELLA_(programming_language)
24. Wikipedia, (2018.), System dynamics, dostupno 13.08.2018. na https://en.wikipedia.org/wiki/System_dynamics

9. Popis slika