UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Silvo Gazvoda

Pregled orodij za agentno modeliranje bioloških sistemov in njihova uporaba na modelu večceličnega biološkega oscilatorja

MAGISTRSKO DELO
ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

doc. dr. Miha Moškon
MENTOR

Ljubljana, 2017

IZJAVA O AVTORSTVU MAGISTRSKEGA DELA

Spodaj podpisani izjavljam, da sem avtor dela in da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali drugem visokošolskem zavodu, razen v primerih, kjer so navedeni viri.
S svojim podpisom zagotavljam, da:

- sem delo izdelal samostojno pod mentorstvom doc. dr. Mihe Moškona,
- so elektronska oblika dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko ter
- soglašam z javno objavo elektronske oblike dela v zbirki "Dela FRI".

— Silvo Gazvoda, Ljubljana, september 2017.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Silvo Gazvoda

Pregled orodij za agentno modeliranje bioloških sistemov in njihova uporaba na modelu večceličnega biološkega oscilatorja

POVZETEK

V magistrski nalogi smo opravili pregled orodij za agentno modeliranje bioloških sistemov. Agentno modeliranje je osredotočeno na modeliranje opazovanega sistema z avtonomnimi entitetami imenovanimi agenti. Tovrstno modeliranje nam omogoča predstavitev biološkega sistema na različnih nivojih abstrakcije. V pregled smo vključili splošno orodje za agentno modeliranje in orodja, ki so specializirana na modeliranje celice kot agent. Orodja smo primerjali in pregledali z vidika načina opisa agentove dinamike, modelirnega programskega jezika, simuliranega virtualnega okolja in z vidika implementiranih celičnih lastnosti, procesov ter dinamike, kot je na primer celična rast in celično gibanje. Podrobneje smo opisali okolja NetLogo, BSim, Gro in CellModeller. Pri vsakem smo opisali interaktivno okolje, modelirni jezik in arhitekturo oziroma implementacijo orodja. Uporabo izbranih orodij smo demonstrirali na vzorčnem primeru večceličnega modela biološkega oscilatorja. V zaključku smo podali predlog ogrodja orodja za agentno modeliranje bioloških sistemov, ki bi uporabnikom še dodatno olajšal postavitev tovrstnih modelov.

Ključne besede: agentno modeliranje, večcelično modeliranje, simulacija, biološki sistemi, skupinsko vedenje, oscilator.

University of Ljubljana
Faculty of Computer and Information Science

Silvo Gazvoda

Overview of Agent-Based Tools for Modelling of Biological Systems and their Application to the Analysis of Multi-Cellular Biological Oscillator

ABSTRACT

In this thesis, we overview computational tools for agent-based modelling of biological systems. Agent-based modelling focuses on the modelling of observed system with individual components called agents. This type of modelling offers a representation of a biological system at multiple levels of abstraction. We overview general-purpose agent-based framework and agent-based tools, which use cells as the primary agents. We compare and examine available tools from different perspectives. We examine how agent dynamics can be modelled, and properties of modelling languages, simulated environments, and built-in complex cellular traits (e.g. growth and movement) are checked. Afterwards, we examine NetLogo, BSim, Gro, and CellModeller framework. For each tool we provide description of their interactive environment, properties of modelling language, and their design and implementation principles. In the last part, we perform analysis of the above mentioned tools on the case study of a multi-cellular biological oscillator model. We conclude with the agent-based framework proposition, which would simplify the process of implementation of multi-agent biological models.

Key words: agent-based modelling, multicellular modelling, simulation, biological systems, emergent behaviour, oscillator.

ZAHVALA

Zahvaljujem se mentorju doc. dr. Mihi Moškonu za strokovno pomoč in potrpljenje pri nastajanju magistrske naloge. Prav tako bi se rad zahvalil staršema, ki sta mi omogočala študij in me spodbujala na študijski poti. Zahvala gre tudi drugim družinskim članom in prijateljem za vse pozitivne želje in spodbude pri študiju in pisanju magistrske naloge.

— Silvo Gazvoda, Ljubljana, september 2017.

KAZALO

1 Uvod

1.1 Motivacija

Napredek računalniških postopkov, s katerimi je mogoče raziskovati delovanje bioloških sistemov, je prispeval k mnogim odkritjem na področju moderne biologije. Na voljo je vedno več orodij, ki uporabnikom omogočajo vzpostavitev tako enostavnih kot zahtevnih modelov bioloških sistemov. V tem delu bomo opravili pregled takšnih orodij, s poudarkom na orodjih za agentno modeliranje (angl. *agent based modelling*). Njihovo uporabo bomo ponazorili na modelu večceličnega biološkega oscilatorja.

Nazorna priča napredka biologije in računalništva je anotacija človeškega genoma na začetku tega tisočletja [1]. Računalniška pomoč biologiji je idejna nit računske biologije. Ta uporablja različne računske pristope pri modeliranju bioloških sistemov in pri procesu snovanja novih ter preoblikovanju obstoječih bioloških organizmov. Področje računske biologije je tesno povezano s področjema sintezne in sistemske biologije. Sintezna biologija vključuje inženirske pristope k načrtovanju in gradnji novih bioloških sistemov ter preoblikovanju obstoječih naravnih sistemov, tako da ti izvajajo novo koristno funkcijo. Sistemska biologija poizkuša odgovoriti na vprašanja o dinamiki procesov znotraj

bioloških sistemov. Sistemska biologija pogosto temelji na mehanističnih pristopih, s katerimi običajno pridobimo matematične opise modelov [2].

Cilj modeliranja bioloških sistemov je postavitev modela, ki opisuje naravni biološki sistem. Računalniški model naravnega sistema je lahko postavljen z enim izmed mnogih orodij za tovrstno modeliranje. Vzpostavitvi dinamičnega modela sledi simulacija, s katero želimo pod določenimi pogoji predvideti odziv sistema. Dinamično modeliranje, h kateremu štejemo tudi agentne modele, nam omogoča izvajanje simulacij pod različnimi robnimi pogoji ali simuliranje dinamike sistema, ki še ne obstaja. Računalniška predstavitev biološkega sistema omogoča hitrejše in cenejše testiranje različnih hipotez pod pogoji, ki jih v laboratoriju stežka pripravimo, ali pa je njihova priprava bistveno dražja ter časovno kompleksnejša. Možnost generiranja obsežnih podatkovnih zbirk o opazovanih sistemih z agentnim modeliranjem in drugimi računalniškimi modeli predstavlja prednost pred pridobivanjem želenih rezultatov z laboratorijskimi eksperimenti. Ugodnejše in hitrejše testiranje hipotez na računalnikih, kot na reagentih, živalih in celicah pa ima tudi svoje omejitve, ki jih bomo spoznali v naslednjih poglavjih.

K računalniškemu modeliranju bioloških sistemov obstaja več pristopov. Podrobneje se bomo spoznali s pristopom agentnega modeliranja, ki sodi v kategorijo prostorsko-časovnega modeliranja (angl. *spatio-temporal modelling*). Ta pri modeliranju upošteva prostorski in časovni aspekt na različnih nivojih abstrakcije. Poleg agentnega modeliranja poznamo še različne razširitve osnovnega prostorsko-časovnega modeliranja. Razdelčni modeli imajo sistem, organiziran v več razdelkov (angl. *compartments*). Razdelek lahko predstavlja posamezno celico ali posamezen del ene celice. Takšni modeli so usmerjeni v predstavitev bioloških lastnosti, kot sta dinamična ureditev razdelkov in prehajanje molekul med razdelki biološkega sistema [3]. Modeli z eksplicitnim opisom prostora imajo za posledico večjo računsko kompleksnost. Primer takšnih so modeli, ki temeljijo na razdelitvi prostora na mrežo (angl. *grid*). Celoten prostor sistema je razdeljen na manjše enote, znotraj katerih je predpostavljena homogenost prostora [4].

V kontekstu agentnega modeliranja se ukvarjamo z vzpostavitvijo modelov, ki so osnovani na množici avtonomnih entitet, imenovani agenti. Agenta si lahko v kontekstu računalništva predstavljamo kot program z definiranimi spremenljivkami in metodami. Vsak agent zaznava svoje okolje in se nanj ustrezno odzove na podlagi definiranih pravil in svojega notranjega stanja. Med simulacijo modela prihaja do hkratnih interakcij med agenti s ciljem posnemanja pojavov obravnavanega sistema. Tovrstno modeliranje ima

dolgo zgodovino na področju socialnih ved in je na tem področju še vedno močno prisotno. Razlog za to je zmožnost opazovanja vedenja ne le posamezne entitete, ampak tudi, kako interakcije vplivajo na vedenje sistema kot celote.

V pričujočem delu se bomo seznanili z aplikacijo tovrstnega modeliranja na biološkem področju. Agentno modeliranje nam omogoča vzpostavitev modelov bioloških sistemov na različnih nivojih abstrakcije. Agent lahko predstavlja celico, tkivo ali celoten organizem. Pozornost bomo usmerili na modele, pri katerih agenti nastopajo kot celice organizma. Vzpostavitev večceličnih modelov bo potekala v programskih orodjih, ki tovrstno modeliranje podpirajo. Opravili bomo analizo vsakega orodja in jih med seboj tudi primerjali. Zanimale nas bodo prednosti in hibe posameznega orodja, uporabili jih bomo tudi pri vzpostavitvi večceličnega modela gensko regulatornega omrežja (angl. *gene regulatory network*). Izbrana orodja bomo demonstrirali na večceličnem oscilatorju, pri čemer želimo vzpostaviti sinhrono delovanje. Oscilacije z različnimi periodami so močno prisotne pri mnogih bioloških ritmih vseh živih bitij. Sinhrono delovanje bomo poskušali doseči z dodatno signalno molekulo, ki bo prehajala med celicami.

1.2 Prispevki magistrske naloge

V zadnjem času je na voljo vrsta orodij, ki omogočajo agentno modeliranje, pri katerih agent nastopa kot celica. Z analizo tovrstnih orodij bomo poudarili prednosti in slabosti posameznih orodij, na podlagi katerih bomo podali predlog o novem orodju, s katerim bi še dodatno olajšali vzpostavitev modelov bioloških sistemov. Analizo orodij bomo opravili na primeru večceličnega modela biološkega oscilatorja.

1.3 Metodologija

V prvem delu naloge se bomo seznanili s področjem agentnega modeliranja bioloških sistemov in orodji za tovrstno modeliranje. Prednosti in slabosti orodij bomo spoznali med vzpostavitvijo modela oscilatorja, ki bo potekala v drugem delu naloge. Za vzpostavitev modela bomo uporabili orodja NetLogo, BSim, Gro in CellModeler. Opis modela bo implementiran v različnih programskih jezikih, saj vsako orodje temelji na svojem jeziku. Testiranje bo tako potekalo v programskih jezikih NetLogo, Java, Gro in Python. Pri vsakem orodju nas zanima, kako je opisana dinamika agentov v implementiranem modelu. Zanima nas tudi, kakšen je model celice in kakšen je model interakcij med

celicami. Podali bomo način predstavitve prostora, ki ga orodje podpira. Seznanili se bomo s programskimi jeziki in njihovo sintakso. Simulacija agentnih modelov običajno pripelje do večjega števila sočasno obstoječih entitet. Preverili bomo, kako orodja podpirajo paralelno naravo modelov in na kakšen način izkoriščajo paralelno arhitekturo. Naposled bomo orodja primerjali po njihovi uporabniški prijaznosti in enostavnosti uporabe. Celotna koda, ki smo jo uporabili za testiranje orodij, je na voljo na naslovu http://lrss.fri.uni-lj.si/bio/material/2017_gazvoda_msc.zip.

1.4 Pregled naloge

V poglavju 2 se bomo seznanili s področjem agentnega modeliranja. Na začetku bomo spoznali prednosti in slabosti tovrstnega modeliranja. Nadaljevali bomo s predstavitvijo uporabe agentnega pristopa k modeliranju bioloških sistemov. V poglavju 3 bomo opisali orodja za agentno modeliranje bioloških sistemov. Orodja, v katerih so agenti obravnavani kot celice, bomo demonstrirali na modelu večceličnega biološkega oscilatorja, ki bo predstavljen v poglavju 4. V zaključku bomo poudarili glavne pomanjkljivosti obstoječih orodij in podali predlog ogrodja orodja za agentno modeliranje tovrstnih modelov.

2 Agentno modeliranje bioloških sistemov

Agentni pristopi k modeliranju so postali prepoznavni na različnih področjih. Agentno modeliranje temelji na ideji samostojnega odločitvenega sistema, imenovanega agent. Agent se nahaja v prostoru, v katerem zaznava svoje okolje in se nanj tudi odziva. Kompleksnost avtonomnega agenta je odvisna tako od narave problema, ki ga rešujemo, kot tudi od števila agentov v okolju. Cilj tega poglavja je predstavitev tistih lastnosti agentnega modeliranja, zaradi katerih je tovrstno modeliranje v zadnjem času postalo prepoznavno tudi na področju biologije. Modele bioloških sistemov lahko razvrstimo v različne kategorije po različnih merilih. Primera različne kategorizacije bioloških modelov predstavita deli [3, 5], ki modele delita po pristopu modeliranja oziroma ločita modele na matematične in računalniške. Ena izmed kategorizacij, ki se je bomo držali v tem delu, je delitev na dinamične in statične modele. Glavna značilnost dinamičnih modelov je sposobnost izvajanja simulacij. Simulacija tovrstnih modelov omogoča analize evolucije dinamike sistema skozi čas. Statični modeli se od dinamičnih modelov razlikujejo predvsem po načinu predstavitve obravnavanega sistema in v svoji predstavitvi ne vključujejo časovne komponente. Primer statičnih modelov so usmerjeni grafi, ki jih lahko analizi-

ramo z različnimi operacijami nad grafi, kot sta iskanje najkrajše poti, povezanost med vozlišči ipd. Najprej se bomo seznanili z dinamičnim modeliranjem bioloških sistemov. Pri tem bomo pregledali tudi bolj grobo delitev modelov na računalniške (angl. *computational*) in matematične modele, povzete po [5]. Nadaljevali bomo s pregledom agentnega modeliranja, katerega modeli sodijo v kategorijo dinamičnih modelov.

2.1 Dinamično modeliranje bioloških sistemov

Sistemska biologija je interdisciplinarno področje, ki med drugim združuje biologijo, kemijo, fiziko in računalništvo. Glavna nit sistemske biologije je analiza bioloških sistemov, katerih dinamike ne moremo pojasniti le z linearno vsoto funkcij sestavnih elementov sistema. V tem kontekstu moramo biološki sistem obravnavati kot celoto. Z razvojem znanosti in tehnike je poleg vse več razlogov za modeliranje bioloških sistemov nastalo tudi več pristopov k modeliranju.

Agentno modeliranje sodi s svojimi značilnostmi med dinamične modele, ki omogočajo prostorsko modeliranje bioloških sistemov. Poleg agentnega poznamo tudi druge pristope k modeliranju, ki pokrivajo specifične aspekte biologije. Modele bioloških sistemov lahko kategoriziramo po različnih kriterijih. Eden izmed načinov delitve bioloških modelov je delitev na računalniške in matematične modele [5]. Matematični modeli so modeli, ki so običajno definirani z diferencialnimi enačbami in pri katerih se računska moč računalnikov izkorišča za analizo matematične odvisnosti med spremenljivkami. Računalniški modeli predstavljajo modele v obliki izvršljivih programov. Na tem mestu lahko potegnemo vzporednice s kategorizacijo na statične in dinamične modele. Podobnost, ki jo najdemo med dinamičnimi in računalniški modeli, je možnost izvajanja simulacij oziroma ponovljivo izvajanje modela.

Biologi so za opisovanje sistemov sprva uporabljali konceptualne modele. S tovrstnimi modeli so celoten sistem opisali s komponentami sistema in njihovo medsebojno interakcijo. S shematskimi modeli so na enostaven način strnili mehanistično razumevanje rezultatov opazovanj sistema [5]. Zaradi enostavnosti dopuščajo takšni modeli preveč nejasnosti pri opisovanju dinamike. Skope informacije modela o dinamiki imajo za posledico statično sliko biološkega procesa. Potreba po dinamični obliki predstavitve modelov, s katerimi je mogoče opisati časovno odvisne procese, skupaj z rastjo kompleksnosti in obsega, je spodbudila uporabo dinamičnih opisov sistemov. Rezultat dinamičnega mo-

deliranja je model, sestavljen iz množice enačb ali pravil, ki opisujejo, kako se sistem spreminja skozi čas. Dinamika biološkega sistema temelji na kvantitativnem ali kvalitativnem opisu, ki pogosto opisuje zakone kemije in fizike. Končni model je mehanistični, saj opisuje različne gonilne mehanizme opazovanega vedenja sistema. V središču obeh vrst modelov, tako dinamičnih kot statičnih, je računalniška znanost. Časovno in materialno potratnost laboratorijskih eksperimentov zmanjšuje dinamično modeliranje. Dinamični modeli bioloških sistemov vsebujejo opis dinamike sistema, ki opisuje spremembe v sistemu. Definicija sistema je sestavljena s stanjem sistema in s pravili, ki določajo spremembe sistema skozi čas. Stanje sistema je definirano s spremenljivkami, ki opisujejo lastnosti opazovanega biološkega sistema.

Osnova matematičnih modelov [5] so prenosne funkcije (angl. *transfer functions*). Prenosna funkcija, ki opisuje relacije med vhodom in izhodom sistema je običajno zapisana v obliki diferencialne enačbe. Kompleksnejši modeli so predstavljeni s kompozitumom prenosnih funkcij, ki tvori sistem medsebojno odvisnih spremenljivk. Neposredno analizo je mogoče izvajati na preprostih sistemih, pri katerih imajo posamezne prenosne funkcije enostavne robne pogoje oziroma so izraženi z linearnimi diferencialnimi enačbami. Pri kompleksnejših modelih, pri katerih imamo opravka z nelinearnimi ali stohastičnimi diferencialnimi enačbami in večjim številom spremenljivk, pa se ne moremo izogniti uporabi računalniških orodij ter simulacij [5]. Aplikacija tovrstnih modelov na področju molekularne biologije ni nov pristop. Na tem področju najdemo vrsto uspešnih aplikacij računalniških modelov, med katerimi je tudi Turingov opis tvorjenja vzorcev v procesu razvoja organizma [6].

Dinamični modeli se glede na kriterij tipa in točnosti podatkov, ki jih želimo z modeliranjem pridobiti, delijo na kvalitativne in kvantitativne modele. Zahtevnost modela se pri definiciji kaže v številu spremenljivk, ki definirajo stanje sistema. Na račun večje kompleksnosti modela in podatkov, iz katerih izhajamo v procesu modeliranja, ti odražajo višjo točnost rezultatov. Naslednji nivo delitve dinamičnih modelov je na deterministične in stohastične modele. Opisana delitev pristopov je le ena izmed mnogih. Različen kriterij delitve pristopov smo opazili pri delitvi na matematične in računalniške modele [5].

Dinamično in statično modeliranje pokrivata širok spekter pristopov k opisu dinamike bioloških sistemov ter obsegata še večji nabor orodij. V naslednjih podpoglavjih se bomo osredotočili na lastnosti agentnega modeliranja. Pristop bomo demonstrirali na modelu večceličnega biološkega oscilatorja. Začeli bomo s splošno predstavitvijo agentov in kaj

abstrakcija agenta predstavlja, skupaj z lastnostmi, ki agentno modeliranje ločuje od drugih računalniških modelov.

2.2 Agentno modeliranje

Agentno modeliranje je način računalniškega modeliranja, pri katerem je model sistema sestavljen iz množice avtonomnih entitet — agentov. Tovrstno modeliranje temelji na definiranih pravilih, na katerih bazirajo odločitve agentov ob dogodkih v modelih. Postavljen model lahko vsebuje enega ali več agentov. Upoštevanje interakcij med množico agentov, postavljenih v skupnem prostoru, ima za posledico široko aplikacijo na mnogih področjih.

Agent je avtonomen računalniški sistem z definiranim notranjim stanjem in lokacijo v nekem simuliranem okolju. Kakšne lastnosti definirajo okolje in kakšne predpostavke so bile sprejete ob postavitvi virtualnega okolja ta hip ni pomembno. Avtonomni agenti opazujejo okolje in se brez zunanjega posredovanja odločijo za ustrezen odziv. Odziv oziroma akcija je odvisna od definiranih pravil agenta. Agenti v okolju izvajajo aktivnosti, ki poleg prej omenjenih pravil in okolice temeljijo tudi na notranjem stanju agenta, ki načeloma ni viden drugim agentom. Pri mnogih aplikacijah agentnega modeliranja nas bolj kot posamezne interakcije med agenti zanima vedenje sistema kot celote. Seveda ne smemo zanemariti pomembnosti lokalnih interakcij med agenti in akcijami agentov, ki so usmerjene k realizaciji zadanih ciljev.

Agenti so postavljeni v okolje, v katerem pride do izraza odzivnost agentnih sistemov. Če se ne omejimo le na simulirano računalniško okolje, je agentovo okolje lahko internet, množica agentov istega tipa ali celo fizično okolje, katerega agent zaznava skozi senzorje in se na zaznane spremembe ustrezno odzove [7]. Odziv agenta ni le odziv na okolje, ampak agent izvaja proaktivno akcijo, s katero se približa zadanim ciljem. Agenti so podvrženi različnemu vedenju glede na sistem, ki ga z modelom predstavljajo. Pri večagentnem sistemu je obravnavan sistem opisan z množico interaktivnih agentov s pravili, opisujoč socialno naravnano dinamiko agentov in odnose med agenti. Poleg pravil in pogojev, s katerimi definiramo vedenje sistema, naprednejši agenti vključujejo nevronske mreže (angl. *neural networks*), evolucijske algoritme (angl. *evolutionary algorithm*) ali druge tehnike učenja, s pomočjo katerih se še dodatno približamo realnemu vedenju ter prilagajanju [8].

Modeliranje z agenti uporablja drugačen pristop od drugih načinov modeliranja. Bistvena razlika je v tem, da pri agentnem modeliranju sistem gradimo od spodaj navzgor, to je modeliranje iz pogleda agenta. V nasprotju z drugimi modeli, agentni modeli niso induktivni [9]. Induktivni modeli svoj proces modeliranja začnejo z zbirko podatkov. Na podlagi podatkov se razvijejo predpostavke za mehanizme, s katerimi bi lahko reproducirali želeno dinamiko. Pri agentnem modeliranju v nasprotju z induktivnimi modeli začnemo proces modeliranja s postavljanjem pravil vedenja in predpostavk ali hipotez o sistemu, ki ga modeliramo. S tovrstnim modeliranjem in izvajanjem simulacij postavljenega modela si želimo ustvariti populacije agentov ter njihovih interakcij v virtualnem okolju z namenom rekonstruiranja vzorcev vedenja opazovanega sistema. Takšen način izvajanja eksperimentov pogosto spremlja izraz *in silico*, ki se v domeni naravoslovnih področij nanaša na proces izvajanja laboratorijskih eksperimentov v računalniško simuliranih okoljih.

Področje računalniškega modeliranja bioloških sistemov zajema različne pristope modeliranja, razvrščene v več kategorij. Poleg agentnega modeliranja, ki sodi v skupino prostorsko-časovnega modeliranja, so nam na voljo tudi druge tehnike modeliranja, kot so Petrijeve mreže in Boolove mreže. Agentno modeliranje se od drugih metod razlikuje po naslednjih predstavljenih lastnostih.

Prva lastnost je možnost enostavne predstavitve prostora. Ta lastnost je posledica izvora agentnega modeliranja s področja celičnih avtomatov (angl. *cellular automata*). V nadaljevanju bomo spoznali, da domala vsa obravnavana orodja za agentno modeliranje omogočajo postavitev modela v 2D ali 3D prostoru. Obstaja več načinov predstavitve prostorske organizacije in strukturne hierarhije, med katerimi je najpogostejša predstavitev sistema z mrežo, obstajajo pa tudi agentni modeli, ki v svojem virtualnem okolju ne uporabljajo fizične predstavitve prostora [9]. Prostorski vidik prinaša v agentno modeliranje lastnost omejenega dojemanja agentove okolice. Agent ima vhodne signale, pogojene z omejitvenimi pravili, ki odražajo sprejemanje lokalnih interakcij.

Agentno modeliranje izkorišča vzporedno računanje (angl. *parallel computing*). Do sedaj smo agenta obravnavali kot abstraktno entiteto, znotraj katere so enkapsulirane lastnosti in pravila obnašanja. Postavitev in razvoj modela sta v kontekstu računalništva analogna razvoju programske opreme.

Nadalje lahko agentno modeliranje primerjamo z objektno orientiranim programiranjem. Tako kot ima vsak agent svoje notranje stanje, pravila in akcije, ima razred

definirane atribute oziroma spremenljivke ter funkcije, ki delajo s podatki objekta in komunicirajo z drugimi objekti. Agenti niso le enostavni objekti, ampak jih lahko poleg odločitvenega sistema obravnavamo kot svojevrstne procese [7]. Ti procesi med sabo komunicirajo in se izvajajo sočasno. Za vsakega agenta v virtualnem okolju je ustvarjen svoj objekt, katerega podatki odražajo agentovo notranje stanje. Med simulacijami modela je agent podvržen različnim lokalnim pogojem, na podlagi katerih se ovrednotijo agentova pravila in izoblikujejo vselej različni vzorci vedenja tako agentov kot celotne populacije sistema. Dinamika jate ptic je najnazornejši primer sistema, v katerem na prvi pogled enostavne interakcije med posameznimi entitetami pripelje do naprednih vzorcev vedenja celotnega sistema [9]. Podobnost agentnega modeliranja z objektno orientiranim programiranjem omogoča intuitivnejšo vzpostavitev modelov, kot nam to omogočajo alternativni pristopi. Za znanstvenike, ki so nekoliko manj vešči matematiki ali računalnikarji, predstavlja vzpostavitev modela z alternativnimi pristopi, kot so navadne diferencialne enačbe, parcialne diferencialne enačbe ali izpeljanke obeh, precejšen izziv.

V dinamiko agentov lahko vključimo njihovo stohastično delovanje. Mnogi sistemi, zlasti biološki, vključujejo na prvi pogled naključno dinamiko. Vtis naključnosti je prisoten le na ravni opazovanja dinamike sistema, odraz takšnega vedenja pa so še vedno deterministična pravila, ki jim sledi posamezen agent. Definiranje determinističnih pravil in začetnih pogojev sistema le na podlagi opazovanja dinamike je zahteven ali celo nedosegljiv cilj. Agentno modeliranje slednjo hibo rešuje s tvorjenjem populacij agentov. Poglejmo si to rešitev še z nekoliko matematičnega vidika. Verjetnosti za določeno dinamiko se določijo za populacijo kot celoto. Verjetnosti za celotno populacijo agentov imajo za posledico neposreden vpliv na določitev funkcije verjetnosti za vedenje posameznega agenta, ki je naposled vključena v agentova pravila. Med izvajanjem modela agenti sledijo poti gibanja, začrtani z izpeljanimi verjetnostnimi pravili odzivanja agenta. Stohastičnost nam omogoča generiranje množice različnih vzorcev dinamike sistema, ki so rezultat izvajanj le enega agentnega modela [9].

Agentno modeliranje ima modularno strukturo, ki pripomore k večji prilagodljivosti modeliranja. Osupljivo prilagodljiva struktura nam omogoča enostavno povečanje števila agentov postavljenega modela, spreminjanje nivoja racionalnosti ali pravil interakcije med agenti [8]. Spremembo kompleksnosti agenta dosežemo z vpeljavo nove informacije, bodisi z vpeljavo novih tipov agentov bodisi s spreminjanjem obstoječih agentovih pravil.

Osnovni model lahko razširimo na različne nivoje abstrakcije. Model, ki je veljaven na enem nivoju abstrakcije, okrepimo s podrobnimi informacijami, s katerimi vpeljemo različne kategorije modela in posledično pridobimo več nivojev, na katerih opazujemo modeliran sistem. Vse vpeljane spremembe agentov in njihovih pravil nimajo vpliva na simulacije in ne potrebujejo dodatnih posegov v načrtovanje simulacije ter samega procesa simulacije [9].

Naslednja lastnost agentega modeliranja je možnost opazovanja skupinskega vedenja (angl. *emergent behaviour*). Skupinsko vedenje nastane iz interakcij med posameznimi elementi. Skupinsko vedenje je odziv, ki ni le povprečje ali seštevek posameznih elementov. Odziv celote presega seštevek posameznih elementov sistema. Takšen odziv ima za posledico lastnosti, ki niso neposredno povezane ali izpeljane iz lastnosti posameznega elementa in jih je zato težko napovedati. Primer skupinskega vedenja je prometni zamašek. Ta nastane zaradi vedenja in interakcij med vozniki. Odziv, pri katerem se prometna gneča giblje v nasprotni smeri voznikov povzročiteljev, je primer odziva, ki ga opazimo na nivoju celotnega sistema in ga ne moremo predvideti iz obravnave posameznega elementa oziroma voznika [8].

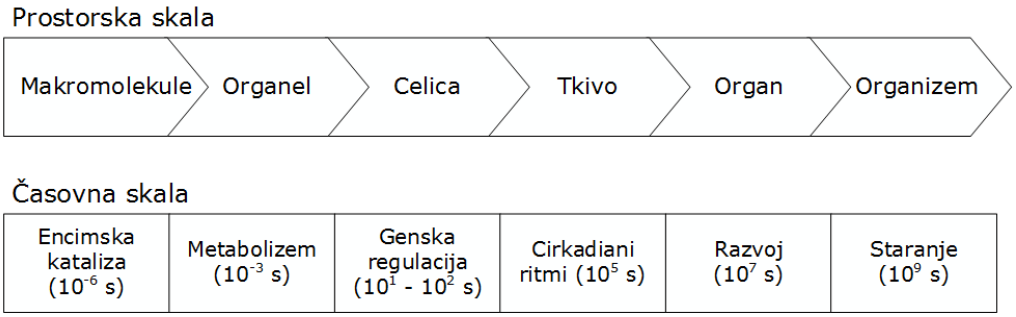
Računska kompleksnost agentnega modeliranja je višja v primerjavi z matematičnimi modeli. Na postavitvi, v kateri programsko ogrodje agentnega modeliranja teče kot paralelni sistem na enonitnem procesorju, agentni model zahteva več iteracij računanja za izpeljavo posameznega dogodka kot pri modelu opisanega z enačbami. Iterativni proces računanja predstavlja ozko grlo, ki ima za posledico omejitev velikosti in kompleksnosti implementacije agentnega modela. Koraki k odstranitvi ozkega grla so bili narejeni z vpeljavo novih računskih okolij za agentno modeliranje. Primer takšnega okolja so platforme, ki za računsko kompleksne operacije izkoriščajo vire grafične procesne enote. Paralelizacijo agentnega modeliranja spremljajo izzivi tudi z drugih področij in ne le s področja računalništva. Ti izzivi se nanašajo na določitev obsega procesa, ki ga poizkušamo paralelizirati ali sočasno izvajati na več procesorjih. Poleg tehničnih vprašanj moramo biti pozorni tudi na posledice takšne paralelizacije na dinamiko modela in ali takšna dinamika ustreza predpostavkam biološkega sistema, ki smo ga vzeli za izhodišče modeliranja sistema [9]. Omejitve pri implementaciji in kako so avtorji pristopili k reševanju vzporednega računanja na izbranih orodjih bomo spoznali v poglavju 3.

Primerov aplikacij agentnega modeliranja je na pretek. Vsako področje, na katerem uporabimo tovrstno modeliranje, ima svoje značilnosti, ki jim moramo posvetiti nekoliko

več pozornosti pri vzpostavitvi modela. Predstavljene splošne lastnosti bomo v nadaljevanju nadgradili z znanjem, pridobljenim pri modeliranju bioloških sistemov.

2.3 Biološki sistemi kot agenti

Raziskave v biologiji potekajo na različnih organizacijskih nivojih. Raziskovalci predstavljajo svoje izsledke raziskav na različnih nivojih biološke organizacije, kot so nivo gena, proteina, celice, tkiva, organa ali organizma. Biološki sistemi so poleg prostorske organizacije opisani tudi s časovnim obsegom, ki se razteza od mikrosekunde za interakcije med molekulami do več desetletij za dosego povprečne življenjske dobe človeka [10]. Dinamiko zapletenih bioloških sistemov sestavljajo biološki procesi, ki se izvajajo na različnih nivojih prostorske in časovne skale. Prostorsko in časovno skalo prikazuje slika 2.1. Znanje, pridobljeno na različnih organizacijskih nivojih, pogosto predstavlja dodaten izziv pri pojasnitvi in aplikaciji mehanističnega znanja, pridobljenega na enem nivoju, na izsledke na višjih nivojih. Poleg pomanjkanja uspešnega prenosa znanja na več organizacijskih nivojev, je na vidiku tudi pomanjkljivost specializiranih raziskovalnih domen, ki se osredotočajo le na procese na določenem nivoju s površnim upoštevanjem vplivov z opazovanemu procesu povezanih pojavov in procesov. Takšne pomanjkljivosti imajo za posledico neorganizirano bazo biološkega znanja in s tem povezanimi ovirami, ki ne omogočajo zadostnega vpogleda v dinamiko celovitega sistema. Vplivi pomanjkanja orodij in pristopov, s katerimi bi povečali sklopljenost med znanstvenimi domenami ter s katerimi bi preprosteje izkoristili obstoječe znanje, se navsezadnje kažejo tudi pri razvoju učinkovitih terapij pri zdravljenju bolezni, ki nastanejo zaradi motenj v notranjem regulatornem procesu. Primeri takšnih bolezni s kompleksnim nelinearnim vedenjem so rak, kronična vnetja in avtoimunske bolezni [9].



Slika 2.1 Prostorska in časovna skala v biologiji [10].

Agentno modeliranje omogoča predstavitev različnih nivojev abstrakcije biološkega sistema. Agenti lahko nastopajo kot celi organizmi, organi, tkiva, celice ali pa celo kot posamezne molekule. Odvisno od kompleksnosti modela in nivoja natančnosti, ki ga želimo doseči, lahko posamezen agent vsebuje tudi druge agente. Nazoren primer agentnega modeliranja bioloških sistemov je modeliranje z agenti, ki so obravnavani kot celice na primarnem nivoju. Takšna predstavitev avtonomnih agentov je analogna celici kot enoti biološke organizacije. Osnovni pristop agentnega modeliranja k postavitvi modela in simuliranja skupinskega vedenja agentov je spodaj navzgor oziroma iz nivoja agentov. Implementacija modela na ta način omogoča postavitev modela na vseh organizacijskih nivojih. Posamezen agent lahko predstavlja eno celico ali večcelični organizem [10]. Postavljanje modela z izhodiščem na nivoju celice podpira dejstvo, da je na voljo bogata zbirka podatkov, pridobljenih z raziskavami, opravljenimi z analizami na nivoju celic [11].

Čeprav agent predstavlja celico, je implementacija modela lahko večnivojska (angl. *multi-scale*). Ti modeli vključuje predstavitev agenta s komponentami iz različnih nivojev prostorske in časovne skale. Agenti pa še vedno eksplicitno definirajo določen biološki nivo ali komponente, kot so denimo celice.

Med številnimi programskimi orodji za agentno modeliranje nas zanimajo orodja, v katerih kot agenti nastopajo celice. Podrobneje nas zanimajo orodja, s katerimi bomo lahko vzpostavili večagentne sisteme, ki predstavljajo modele celičnih populacij. Modeli, predstavljeni na organizacijskem nivoju celic, nam omogočajo razkriti značilnosti in dinamiko celic v celotnem procesu simulacije. S celičnim modelom lahko tako modeliramo celično rast, dinamiko podvojevanja celic in celičnega gibanja z različnimi parametri, kot so velikost celic, geometrija celic in drugi parametri v različnih fazah razvoja celice [3].

Večagentni model tvori množica inteligentnih entitet, ki jih zaznamujejo lastnosti, predstavljene v pričujočem poglavju. Agent je v okolju, v katerem se odziva na spremembe z aktivnostmi. Te aktivnosti so rezultat njegovega notranjega stanja, okolice in definiranih pogojev, pod katerimi se določena aktivnost izvede. Odzivi agentov lahko izražajo stohastičnost, ki jo uporabljamo pri generiranju različnih vzorcev vedenja celotnega sistema. V naslednjem poglavju bomo spoznali, kako predstavljene lastnosti naslavljajo avtorji programskih orodij za agentno modeliranje in do kakšne mere so lastnosti podprte v orodjih NetLogo, BSim, Gro in CellModeller. Pregled izbranih orodij bomo zaključili z diskusijo o možnosti implementacije večagentnega biološkega oscilatorja tako z vidika kompleksnosti implementacije modela kot tudi z vidika uporabniške prijaznosti.

3 Orodja za agentno modeliranje

Z večanjem prepoznavnosti agentnega modeliranja se je pojavila vrsta orodij, ki nam omogočajo vzpostavitev tovrstnih modelov. V pričujočem delu nas zanimajo orodja, osredotočena na modeliranje bioloških sistemov, pri katerih kot agente obravnavamo posamezne celice. Najprej bomo naredili pregled splošnih orodij za agentno modeliranje. Med najprepoznavnejšimi splošnimi orodji za agentno modeliranje najdemo orodje Swarm, sledijo mu Mason, RePast, NetLogo in StarLogo [9]. Swarm predstavlja najstarejše programsko okolje za agentno modeliranje. Izdano je bilo leta 1994 in je bilo v prvi različici napisano v programskem jeziku Objective C. Kasneje mu je bil dodan vmesnik za programski jezik Java, ki omogoča uporabo knjižnic Objective C, napisanih v objektno orientirani razširitvi jezika C v Javi. Izdaji Swarma je sledilo orodje Repast, ki je bilo razvito v Javi. Repast S je zadnja verzija orodja Repast, zasnovana z naborom vizualnih orodij za intuitivnejši razvoj agentnih modelov, specifikacijo agentove dinamike, izvedbo modelov in pregled rezultatov [12]. Mason je bil razvit v Javi kot manjša in hitrejša alternativa orodja Repast s poudarkom na računsko kompleksnih modelih z velikim številom agentov. Pri razvoju so se osredotočili na maksimiranje izvajalne hitrosti in zagotovitev

celovite ponovljivosti izvajanja modelov na različnih vrstah strojne opreme. Razvijalci orodja so si prizadevali za možnost izvajanja daljših simulacij. Glavni cilj je predstavljala možnost zaustavitve izvajanja daljše simulacije in nadaljevati njeno izvajanje na drugih računalnikih [13]. Programsko ogrodje FLAME omogoča postavitev agentnih modelov na paralelnih arhitekturah [14].

Za natančnejše postavitve modelov s specifičnih področij biologije potrebujemo specializirana orodja z implementiranimi pristopi za vključitev vseh posebnih lastnosti in procesov, značilnih za to področje. S tem si občutno olajšamo proces modeliranja, saj so osnovne funkcije obravnavanega sistema v takšnih orodjih že implementirane. Orodja za modeliranje celične dinamike v kontekstu formiranja tkiv oziroma tkivastih sistemov vključujejo kompleksne metode opisa celičnega vedenja in interakcij. Orodje CompuCell3D je osredotočeno na modeliranje procesov formacije tkiv. Orodje vključuje model Glazier-Graner-Hogeweg za opis celične dinamike [15, 16]. Takšna tehnika omogoča modeliranje kompleksnih celičnih morfologij in je bila uspešno uporabljena na več modelih formacije tkiv [17]. Simbiotics je orodje za modeliranje in simulacijo populacije bakterij. Orodje omogoča simulacijo sistemov celic, kolonije in formacijo ter razvoj biofilmov. Simbiotics vsebuje knjižnico modulov z implementacijo celičnih procesov in njene dinamike. Vključena je implementacija modulov za celično geometrijo, fiziko, modeliranje genskih vezij (angl. *genetic circuits*), kemijsko difuzijo in interakcijo celic. Poleg naštetih lastnosti Simbiotics omogoča tudi procesiranje mikroskopskih slik, na osnovi katerih se inicializira virtualni prostor simulacije s konfiguracijo agentov, ki ustreza konfiguraciji oziroma postavitvi celic na originalnih mikroskopskih slikah [18]. DiSCUS (angl. *Discreta Simulation of Conjugation Using Springs*) je orodje za agentno modeliranje, usmerjeno na modeliranje biološkega procesa konjugacije. Konjugacija je prenos DNA med dvema bakterijama, ki sta v neposrednem celičnem stiku. Tovrstna medcelična komunikacija v drugih orodjih ni podprta ali pa je postala podprta z izdajo nove verzije orodja. Primer slednjega je orodje Gro, ki v svoji nadgradnji omogoča tudi modeliranje bakterijske konjugacije [18–20].

Primerjavo med različnimi orodji za agentno modeliranje prikazuje tabela 3.1. V tabeli je prikazano, kakšno dinamiko agentov in celičnih lastnosti orodja podpirajo. Orodja primerjamo tudi po načinu predstavitve simuliranega okolja ter programskem jeziku, v katerem je bilo orodje implementirano in ki se uporablja pri modeliranju.

Domala vsa orodja za agentno modeliranje so implementirana s sinhronim procesom

osveževanja agentovega stanja. Posodobljene vrednosti agenta so drugim agentom takoj razpoložljive. Tovrstna orodja uporabljajo uporabniško definirane razporejevalnike posodobitev agentov, ki delujejo na dva načina. V prvem načinu se agenti dodajajo in posodabljajo po eden naenkrat, kot je to pri orodjih Mason, Repast in Swarm. Drugi način uporablja Netlogo, ki vsem agentom pošlje zahtevo za posodobitev, ta pa se izvrši zaporedno. Takšen pristop ne omogoča vzporednosti asinhronih pristopov posodobitev stanj agentov, ki jo najdemo na primer pri modelih, temelječih na celičnih avtomatih. Nekaterim orodjem je bila možnost vzporednega izvajanja predstavljena v poznejših ali posebnonamenskih različicah [14].

Kljub številčnosti orodij bomo v nadaljevanju naredili pregled štirih orodij in njihovo uporabo demonstrirali na modelu večceličnega biološkega oscilatorja. V pregled bomo vključili splošnonamensko orodje NetLogo, ki med drugimi omogoča modeliranje bioloških sistemov, kot tudi domensko specifična orodja BSim, Gro in CellModeller. Slednja so specializirana za modeliranje celice na nivoju agenta in kot taka že vsebujejo implementacijo modelov določenih celičnih procesov ter njene dinamike, kot je npr. celična rast in celično gibanje. Poleg tega smo v pregled vključili orodje NetLogo, čeprav ni specializirano za modeliranje celice. Razlog za njegovo vključitev je bil obstoj že implementiranih modelov, ki omogočajo enostavno adaptacijo orodja na modeliranje populacije celic.

	Dinamika agentov in lastnosti									Okolje				
Ime	Osnovna pravila	Napredna pravila	ODE	DDE	Stohastičnost	Mobilnost	Kemotaksa	Podvojevanje celic	Celična morfologija	2D	3D	Kemijska difuzija	Programski jezik	Programski jezik modeliranja
BSim	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	Java	Java
CellModeller	✓	✓	✓		✓			✓	✓		✓	✓	Python	Python
CompuCell3D	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	C++/Python	Python/XML
DiSCUS	✓	✓	✓					✓	✓	✓			Python	Python
FLAME	✓	✓			✓			✓		✓			C	C/XML
Gro	✓	✓			✓	✓	✓	✓	✓	✓		✓	C++	Gro
NetLogo	✓					✓		✓		✓	✓		Scala/Java	Logo
Repast HPC	✓	✓								✓	✓		C++	C++
Repast Simphony (Repast S)	✓	✓								✓	✓		Java	Java/Groovy/ReLogo
Simbiotics	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	Java	Java

Tabela 3.1 Primerjava orodij za agentno modeliranje. Stolpci tabele označujejo vsebovane lastnosti orodij z vidika dinamike agentov in lastnosti, okolja ter programskega jezika. Če ima orodje omejen nabor ukazov za nadzor dinamike agentov, je to označeno v stolpcu osnovna pravila. Stolpec napredna pravila označuje, ali so za implementacijo agentove dinamike na voljo vsi ukazi programskega jezika. Stolpca ODE in DDE označujeta, ali je agentovo notranje stanje definirano z navadnimi diferencialnimi enačbami ali diferencialnimi enačbami z zakasnitvijo (angl. *delay differential equations*). Ali je agentovo notranje stanje in interakcija z drugimi agenti stohastična (ob srečanju z drugim agentom, obstaja verjetnost za medsebojno interakcijo) označuje stolpec stohastična dinamika. Mobilnost označuje možnost prostega gibanja agenta v okolju z možnostjo upravljanja agentovih interakcij. Naslednji stolpec označuje obstoj implementacije kemotakse za simulacijo gibanja celic. Možnost replikacije agentov skozi čas označuje podvojevanje celic. Možnost izbire poljubne oblike agenta ali možnost izbire predhodno določenih oblik označuje celična morfologija. Temu sledita stolpca programski jezik, ki je bil uporabljen za razvoj orodja, in programski jezik modeliranja [18, 21, 22].

NetLogo je splošnonamensko orodje za programiranje večagentnih modelov naravnih in družbenih pojavov. Enostavnost uporabe in obsežna knjižnica obstoječih modelov iz različnih znanstvenih domen nam bosta v pomoč pri prvi vzpostavitvi biološkega oscilatorja. Nadaljevali bomo s specializiranim orodjem BSim, ki je bilo razvito za modeliranje bakterijskih populacij na področju sistemske in sintezne biologije. Simuliranje večcelične dinamike v 2D okolju, v katerem so celice deležne vplivov okolja skozi difuzijo signalov in celično rastjo, omogoča orodje Gro. Kot zadnje orodje, s katerim bomo postavili model večceličnega oscilatorja, bomo analizirali CellModeller. Podobno kot BSim in Gro je tudi CellModeller namenjen modeliranju obsežnih večceličnih sistemov, kot je npr. biofilm ali tkivo.

Pri vsakem orodju bomo poleg grobega opisa podali tudi opis funkcionalnosti, uporabniškega vmesnika in simuliranega virtualnega okolja. V vsakem izmed izbranih orodij bomo v ta namen vzpostavili model večceličnega oscilatorja. Mobilnost celic in dinamika modela vključujeta prostorski aspekt agentnega modeliranja, katerega vizualizacija je med simulacijo modela odvisna od načina predstavitve prostora v orodju. Vsako orodje bomo analizirali tudi z vidika programskega jezika. Spoznali bomo, v katerem jeziku je bilo orodje razvito in v katerem poteka modeliranje. V analizo bomo prav tako vključili pregled zmožnosti paralelizacije na paralelnih arhitekturah in ali orodje zaznamujejo kakšne omejitve pri implementaciji modela ter posledično pri generiranju največjega števila entitet. Analizo orodij bomo zaključili s podajanjem njihovih prednosti in slabosti. Izrazite prednosti in hibe orodij bomo uporabili pri predlogu svojega orodja, ki bo združevalo prednosti pregledanih orodij in bo istočasno uporabljalo nove funkcionalnosti pri vzpostavljanju agentnih modelov.

3.1 NetLogo

Analizo programskih orodij bomo začeli s splošnim orodjem NetLogo [23]. Orodje ni specializirano, ampak omogoča postavitev in simulacijo večagentnih modelov iz različnih vsebinskih domen. Orodje spremlja sloves enostavnosti uporabe. Kljub temu orodje omogoča vzpostavitev kompleksnih modelov na področjih biologije, matematike, kemije, fizike itd. Prva različica orodja je bila izdana leta 1999. Od tedaj je orodje tudi v stalnem razvoju, katerega rezultat je stabilno orodje, ki teče na platformah Microsoft Windows, Linux in Mac OS. NetLogo je prosto dostopna namizna aplikacija, napisana

v Scali in Javi. Prve korake v programskem orodju nam poenostavi knjižnica modelov, v kateri najdemo številne vzorčne primere modelov iz različnih vsebinskih sklopov. Knjižnica modelov vsebuje zgledne primere programiranja in dokumentiranja modelov, skupaj s primeri programske kode, s katerimi je prikazana uporaba komponent, ki jih lahko uporabimo pri postavitvi modela. Vsak model v knjižnici lahko raziskujemo skozi njegovo izvršitev in s spreminjanjem programske kode. Poleg bogate knjižnice modelov je uporabnikom na voljo izčrpen uporabniški priročnik. Zaradi naštetih dejstev smo orodje vključili tudi v naš pregled, kljub temu da ni specializirano za agentno modeliranje celic.

Zgodovinsko gledano je NetLogo pomembno orodje v vrsti večagentnih orodij za modeliranje, med katerimi najdemo tudi orodje StarLogo. Orodje NetLogo je nastalo iz programskih jezikov StarLisp in Logo. Programski jezik Logo je, tako kot njegova različica naslednje generacije NetLogo, član družine programskih jezikov Lisp. NetLogo je mišljenje o enostavni uporabi podedoval od Loga, medtem ko je večagentno okolje in sočasno izvajanje podedoval od StarLispa. Avtorji so izdali orodje, v katerem lahko raziskujemo dinamiko modela na nivoju agentov in vzorce vedenja sistema agentov kot celote, ki nastanejo iz interakcij med agenti. Agent je v teh orodjih predstavljen kot želva. V orodju Logo programiramo le enega agenta, v NetLogu pa je obseg mnogo večji, saj postavljeni model lahko vsebuje več tisoč agentov. Okolje, v katerem agenti izvajajo svoje aktivnosti, sestavlja več tipov agentov. V grobem se agenti gibljejo v dveh dimenzijah na mreži celic, ki so tako kot mobilni agenti programabilne.

NetLogo v osnovi podpira postavljanje modelov na ravnini. V zadnjih izdajah orodje vsebuje tudi aplikacijo NetLogo 3D s podprto 3D grafiko. V tej različici orodje omogoča modeliranje dinamike v 3D prostoru. Zasnova orodja omogoča njegovo uporabo v širokem spektru uporabnikov z različnimi strokovnimi ozadji. Orodje je bilo tako zasnovano za uporabnike brez ali z osnovnim znanjem programiranja kot tudi za raziskovalce, ki so izkušeni programerji. Avtorji orodja so s takšnim konceptom omogočili uporabo tako v izobraževanju kot tudi v raziskovalnih krogih. O uporabi na večjem številu področij pričajo modeli, v katerih agenti nastopajo kot molekule, ptice, roboti, avtomobili, celice itd. Model dopolnjuje agentova okolica v obliki mreže celic, ki tako kot agentje svoje poslanstvo opravljajo v različnih vlogah, kot so rakave celice, drevesa, teren, stanovanja itd. [24].

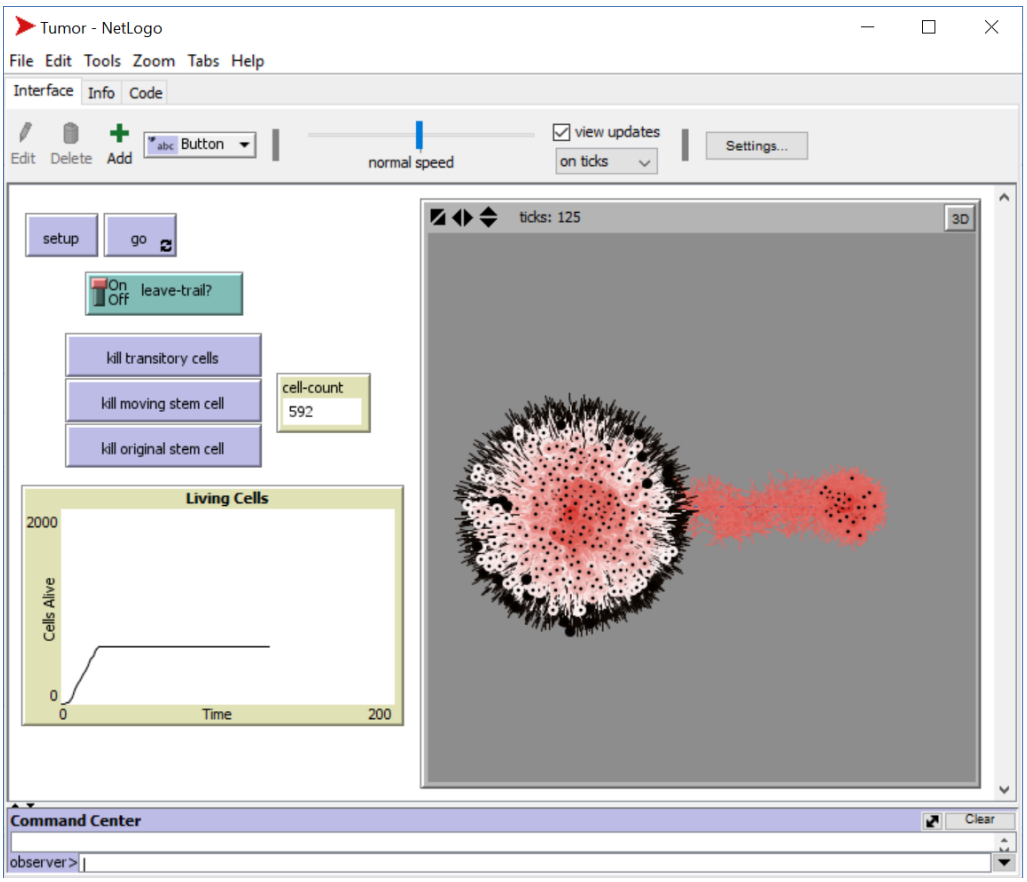
Podroben vpogled v orodje NetLogo sledi v naslednjih podpoglavjih. Pregled funkcionalnosti orodja bomo spoznali v treh delih, in sicer skozi pregled grafičnega vmesnika,

programskega jezika in implementacijo samega orodja. Poleg grobega pregleda orodja bomo v naslednjih podpoglavjih pridobili odgovore na vprašanja o tipu in obliki matematičnih modelov v ozadju modela, vpogled v model celice ter model interakcij med celicami. Določene odgovore o enostavnosti uporabe orodja in podrobnosti o modelu večceličnega biološkega oscilatorja bomo podali pri opisu implementacije vzorčnega modela.

3.1.1 Interaktivno okolje

Grafični uporabniški vmesnik in začetni pogled ob zagonu aplikacije NetLogo je prikazan na sliki 3.1. Zaslonski posnetek vmesnika prikazuje odprt model tumorja [25] iz spremljajoče knjižnice modelov. Model prikazuje rast tumorja in reakcijo le-tega na zdravljenje. V orodju je pregled modela razdeljen na tri dele, ki so predstavljeni na zavihkih *Interface*, *Info* in *Code*. Model je sestavljen iz grafičnega vmesnika v zavihku *Interface* in izvorne kode, zapisane na zavihku *Code*. Orodje omogoča tudi dokumentiranje vzpostavljenega modela v zavihku *Info*. Zaporedje zavihkov predstavlja tipično interakcijo uporabnika z modelom. Privzeti pogled na model je zavihek *Inferface*, na katerem uporabniki najprej zaženemo simulacijo modela. Delovanje modela je dokumentirano na zavihku *Info*. Ko smo seznanjeni z orodjem in z modelom, lahko svoje spremembe ter ideje o modelu implementiramo v izvorni kodi v zavihku *Code* [24].

Zavihek *Interface* je grafični urejevalnik, v katerem urejamo grafične kontrolne elemente modela. V kompleksnih modelih med drugimi najdemo gumbe, ki kličejo podprograme, definirane v zavihku *Code*, stikala za nastavljanje binarnih spremenljivk, drsnike za nastavljanje številčnih spremenljivk, monitorje za prikaz vrednosti spremenljivk med simulacijo in grafe, ki se izrisujejo med simulacijo. Na desni strani pogleda je risalna površina, na kateri se izrisuje dvorazsežno skupno okolje agentov. Okolje je sestavljeno iz manjših celic, imenovanih *patches*, katerih velikost je ravno tako nastavljiva. Površina okolja je nastavljiva skupaj s postavitvijo izhodišča, ki privzeto leži v centru okolja. Predstavitev virtualnega okolja je predstavljena na sliki 3.2 z označeno širino in višino okolja ter označenim izhodiščem. Nastavitve okolja vključujejo tudi različne načine povezav med celicami okolja. Okolje je lahko neomejeno oziroma zvezno ali pa omejeno na robovih. Pri neomejenem okolju se agent ob prečkanju roba pojavi na nasprotni strani. V takšni predstavitvi imajo vse celice enako število sosednjih celic. Agenti so v okolju lahko omejeni le v eni ali obeh smereh na robovih vzdolž osi x in y. Agenti v tem primeru

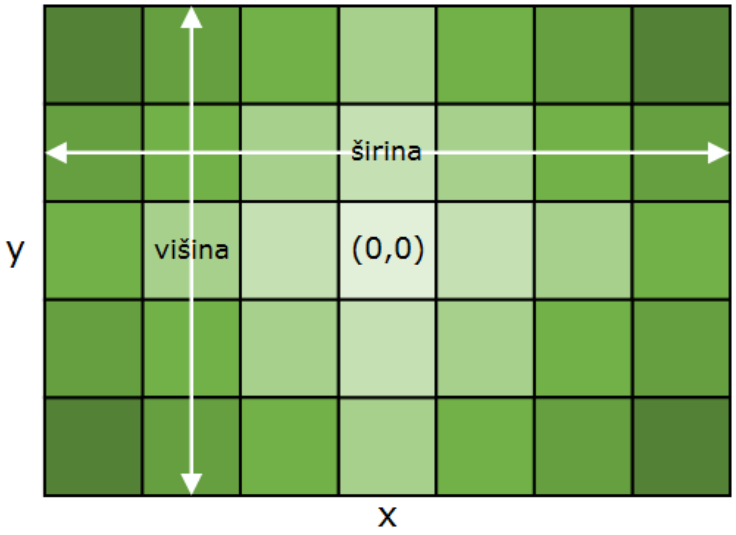


Slika 3.1 Grafični uporabniški vmesnik orodja NetLogo med simulacijo modela tumorja.

roba okolja ne morejo prečkati.

Orodje NetLogo je v eni izmed nadgradenj pridobilo možnost pogleda v treh dimenzijah na okolje agentov, ki se sicer gibljejo vzdolž koordinat x in y. Agent je vedno na celici. Na eni celici lahko prebiva tudi več agentov hkrati. Mobilni agenti se v okolju gibljejo v enoti celice. Agenti se v enem časovnem koraku simulacije v določeno smer lahko premaknejo za eno ali več celic. Agentom je mogoče spremeniti velikost, obliko in barvo. Agentovo obliko lahko izberemo iz predhodno definiranega nabora oblik ali jo oblikujemo sami v urejevalniku, ki je del ogrodja NetLogo [27]. Način predstavitve oblik temelji na vektorski grafiki, katere prednost je ohranjanje kakovosti slike pri transformacijah [24].

Predstavitev modela je zapisana na zavihku *Info*. Obstoj in zasnova zaviha nas spodbujata k zapisu dokumentacije modela v označevalnem jeziku Markdown [28]. Z dokumentacijo lahko poenostavimo začetne korake pri uporabi postavljenega modela in s tem



Slika 3.2 Virtualno okolje je predstavljeno v obliki dvorazsežnega polja celic [26].

posledično zagotavljamo načelo enostavne uporabe orodja, ki ga spoštuje orodje Logo oziroma NetLogo. V dokumentaciji obstoječih modelov običajno zasledimo informacije o modeliranem sistemu, kako model deluje in napotke pri uporabi modela.

Agentove aktivnosti in metode kontrolnih elementov na vmesniku modela je mogoče implementirati v zavihku *Code*. NetLogo je kot programski jezik nadgradil Logo z vpeljavo koncepta agentov in vzporednosti [24]. Osnovne konstrukte pri implementaciji modela bomo spoznali v naslednjem podpoglavju.

3.1.2 Programski jezik

NetLogo temelji na programskem jeziku Logo, ta pa izvira iz jezika Lisp. Logo svojo prepoznavnost pripisuje enostavni uporabi in prijazni sintaksi. Razvito je bilo kot programski jezik, ki bi ga uporabljali otroci, starejši in izkušeni uporabniki [24].

V Logu programiramo aktivnosti le enega agenta v virtualnem okolju. NetLogo koncept agentov posploši in vpelje večje število mobilnih agentov v okolje enega modela. Agenti se gibljejo v ravnini, ki je implementirana kot mreža celic. Kot smo že omenili, so celice mreže ravno tako agenti in so predstavniki drugega tipa agentov. Celice mreže ali stacionarni agenti imajo celoštevilčne koordinate, medtem ko so koordinate mobilnih agentov realna števila. Pozicija mobilnega agenta je tako lahko center celice ali kjerkoli na ravnini celice. Poleg mobilnih in stacionarnih agentov v NetLogu obstajajo še opazovalec

(angl. *observer*) in povezovalci (angl. *links*). V okolju obstaja le en primerek opazovalca, ki nima lokacije. Naloga opazovalca je dodeljevanje ukazov drugim agentom. Četrti tip agentov so povezovalci, ki tako kot opazovalec nimajo lokacije. Primer povezovalca je agent, ki povezuje dve želvi oziroma mobilna agenta. Povezovalec med dvema mobilnima agentoma obstaja le v času obstoja obeh mobilnih agentov. Ob uničenju enega od dveh agentov sledi uničenje agenta povezovalca, ki ju povezuje [26].

Agentova opravila so definirana z dvema vrstama ukazov. Pripadniki prve skupine so ukazi, ob izvedbi katerih agenti izvedejo akcijo in pri tem tvorijo rezultate, ki imajo različen vpliv na agentovo okolje. Drugi tip ukazov, imenovanih *reporters*, pri svoji izvedbi izračuna vrednost, ki jo agent vrne. Predstavniki obeh tipov ukazov se skupaj imenujejo *primitives* in so vgrajeni v NetLogo. Da pa pri programiranju modela nismo omejeni le z naborom ukazov, vključenih v NetLogov slovar ukazov, lahko uporabniki definiramo tudi svoje ukaze (angl. *procedures*). Enkapsuliranje ukazov v podprogram ali funkcijo ni posebnost NetLoga in je znan koncept v programiranju. Podobno kot pri funkcijah v drugih programskih jezikih tudi podprogrami v NetLogu sprejemajo morebitne parametre.

V NetLogu je mogoče definirati, kateri del programske kode izvedejo mobilni agenti, agenti povezovalci ali stacionarni agenti. Privzeto ukaze, če ni definirano drugače, izvede agent opazovalec. Pogosto modeli vsebujejo uporabniško definirani proceduri *to setup* in *to go*. Ti metodi zajemata ukaze za ponastavitev modela v začetno stanje in zagon izvajanja modela. Primer definicije enostavnih procedur *to setup* in *to go* je prikazan v kodi *NetLogoSetupGo*.

Koda 3.1 Primer procedure *to setup* in *to go*. V proceduri *to setup* se ponastavi model v začetno stanje, kreira 10 agentov in nastavi števec simulacijskih korakov na nič. V proceduri *to go* se vsi agenti v svoji smeri premaknejo za en korak. Z ukazom *tick* se števec simulacijskih korakov poveča za ena [26].

```
to setup
  clear-all
  create-turtles 10
  reset-ticks
end

to go
  ask turtles [ fd 1 ]
  tick
```

```
end
```

Telo metode za ponastavitev modela je zaporedje stavkov za povrnitev vrednosti globalnih spremenljivk, izbris izhodnih podatkov in uničenje mobilnih agentov skupaj s ponastavitvijo privzetih vrednosti spremenljivkam stacionarnih agentov. Ob izbrisu populacije agentov se ponastavi tudi številčenje agentov. Agent ima vgrajeno spremenljivko *who*, ki je celo število, večje ali enako 0, in služi kot oznaka za identifikacijo agenta. Ob izbrisu vseh agentov se naslednjemu generiranemu agentu dodeli številka 0. Življenjski cikel enega agenta ni vedno enak življenjski dobi, ki jo ponazarja celoten tek simulacije. Agenti med simulacijo umirajo in se kreirajo. Novo kreiranemu agentu se nikoli ne dodeli identifikator umrlega agenta. Klic metode za ponastavitev stanja se prav tako odraža v očiščenju risalne površine virtualnega okolja v grafičnem vmesniku, v katerem populacija mobilnih agentov ne šteje nobenega člana. Preden se lahko začne naslednja simulacija modela, običajno potrebujemo enega ali več mobilnih agentov. V primeru takšne zahteve je procedura *to setup* primerno mesto za klic ukaza, ki generira enega ali več agentov. Ponastavitev začetnega stanja je hipen dogodek pred začetkom izvajanja simulacije. Potek simulacije in posledično trajanje simulacije je odvisno od zaporedja ukazov, ki jih agent izvaja iterativno v svoji življenjski dobi. Programsko kodo za različne vrste agentov je mogoče organizirati v različne podprograme. Te nato kličemo iz vmesnika ali drugih metod.

NetLogo loči različne tipe agentov. Agenti in povezovalne agente je mogoče razširiti z definiranjem svojih podtipov. Ti podtipi nam omogočajo lažjo implementacijo dinamike za različne tipe agentov. Za primer lahko vzamemo kar svoj model biološkega oscilatorja, v katerem bomo definirali tip agenta z lastnostmi, ki predstavljajo gensko regulatorno omrežje oscilatorja.

Vsa števila v NetLogu so predstavljena v plavajoči vejici z dvojno natančnostjo v standardu IEEE 754. Poleg natančnosti je pri modeliranju znanstvenih problemov zaželena ponovljivost rezultatov. Poleg psevdonaključnega generatorja algoritmi razvrščanja izvajanja agentov uporabljajo determinizem. Generatorji naključnih števil in podrobnosti pri mehanizmu razvrščanja izvajanja agentov se med verzijami razlikujejo. Posledica tega je, da se isti model lahko vede drugače. Enako zaporedje naključnih števil pri določenem semenu generatorja psevdonaključnih števil in ponovljivost rezultatov dosežemo z uporabo orodja enake verzije [26].

3.1.3 Izvajanje simulacij

Simulacija se pri mnogih obstoječih modelih zažene preko vmesnika s klicem procedure *to go*. Procedura *to go* se pokliče s posebnim gumbom, ki svoje zaporedje ukazov izvaja iterativno. Če simulacijo modela v NetLogu pogledamo z nekoliko abstraktnega zornega kota, je ta sestavljena iz zaporedja časovnih korakov, v katerih se preverja izpolnjenost pogojev in izvajanje ustreznih agentovih akcij. Eden izmed načinov, kako bi to implementirali v drugih jezikih, je z globalno zanko. V NetLogu podobno funkcionalnost dosežemo z uporabo namenskega gumba, ki periodično kliče določeno metodo. Predstavitev časovnih korakov pa bi dosegli z deklaracijo globalne spremenljivke, katere vrednost se poveča ob vsaki ponovitvi globalne zanke. Za opisano funkcijo ima NetLogo vgrajen števec. Čas v NetLogu teče v diskretnih korakih, imenovanih *ticks*. Vrednost števca se v večini modelov začne z 0 in se med simulacijo povečuje za 1. Števec privzeto zavzame celo število, lahko pa zavzame tudi decimalna števila. Slednje nam pride prav, če želimo prikazati zvezno gibanje.

Grafični vmesnik vsebuje pogled, ki prikazuje hipno stanje agentov modela. Za prikaz stanja agentov v naslednjem trenutku je treba pogled posodobiti in ponovno izrisati. NetLogo omogoča dva načina vizualne posodobitve modela: zvezni in diskretni način. V simulaciji je pomembno, kdaj se izvede posodobitev, saj s tem vplivamo na pogled modela in kolikokrat se posodobitev izvede. Nizka perioda posodobitev zahteva več časa za posodobitev pogleda. Posledica pogostih posodobitev je tudi počasnejše izvajanje modela. Pri zveznem načinu se posodobitve izvajajo z določeno frekvenco. Število posodobitev v eni sekundi je nastavljiva vrednost, ki vpliva na hitrost izvajanja modela. Diskretni način izvajanja posodobitev temelji na diskretnih korakih *ticks*. Privzeto izvajanje posodobitev pri diskretnem načinu je ena posodobitev pogleda v enem diskretnem koraku, takojšnje posodobitve pa je mogoče sprožiti tudi v programski kodi. Prednost drugega načina izvajanja posodobitev v primerjavi z zveznim je konsistentno in predvidljivo vedenje modela pri posodobitvah pogleda, ki je neodvisno od računalnika, na katerem model teče ter se ne spreminja med izvajanji modela. Neprekinjene posodobitve so uporabne pri modelih, pri katerih izvajanje ni sestavljeno iz krajših diskretnih faz. Prav nam pride tudi v procesu iskanja napak v programski kodi, saj v tem načinu delovanja vidimo, kaj se z modelom dogaja znotraj enega diskretnega koraka. Drugače kot pri posodobitvah ob diskretnih korakih, kjer je viden le končen rezultat enega koraka, z upočasnitvijo modela

vidimo tudi premik enega agenta naenkrat. Na frekvenco posodobitev prikaza vpliva nastavitev števila slik na sekundo (angl. *frames per second*). Ta neposredno določa periodo posodobitev pri zveznem načinu. Ko se posodobitve izvajajo v diskretnih korakih, nastavitev frekvence predstavlja zgornjo mejo števila posodobitev v eni sekundi. Nastavljena frekvenca posodabljanja se šteje kot privzeta hitrost modela.

3.1.4 Implementacija programskega orodja

Prve verzije orodja so bile napisane v Javi zaradi podpore tako jezika kot grafičnih knjižnic na več platformah. Poznejši verziji 5.0 in aktualna verzija 6.0, izdana v decembru 2016, sta napisani v Scali in Javi. Programska koda v Scali se prevede v javansko bitno kodo, ki je interoperabilna z Javo in drugimi jeziki Javinega virtualnega stroja. Orodje je bilo v začetku kombinacija prevajalnika in interpreterja. Ker se programska koda ni neposredno interpretirala, je bila dosežena višja zmogljivost. Orodje je bilo implementirano na način, kjer je prevajalnik analiziral, anotiral in organiziral strukturo v obliko, ki se interpretira učinkoviteje. Pozneje se je izkazalo, da je za povečanje zmogljivosti orodja korak v pravo smer prevajanje NetLogo kode v javansko bitno kodo namesto lastne vmesne predstavitve [24, 29]. Dokumentacija orodja verzije 6.0 opisuje, da orodje še vedno uporablja interpreter in prevajalnik [26]. Prevajalnik kodo pretvori v bitno kodo, ker pa ta še ne podpira celega jezika, del kode ostane interpretiran.

NetLogo pri izvajanju modela izkorišča en procesor oziroma jedro [26]. Orodje podpira sočasno izvajanje med agenti. Če ukažemo skupini agentov, da se premakne za 10 enot, ne želimo, da en agent opravi pot 10 enot, medtem ko se drugi agentje še niso premaknili. Izvajanje želimo izvesti v korakih. Vsi agentje tako najprej izvedejo en korak, nato še drugi korak in tako naprej. NetLogov simulacijski pogon teče v eni niti. Agenti se sočasno gibljejo le navidezno, njihovo gibanje se izvaja v določenem vrstnem redu po en agent naenkrat. Sočasno izvajanje poteka tako, da NetLogov pogon preklaplja med agentovimi stanji oziroma konteksti po opravljeni enoti dela. Vedenje modela ob takšnem izvajanju ostane deterministično, saj je časovna razporeditev preklopov med agentovimi konteksti deterministična [24]. NetLogo večjedrnost sodobnih procesorjev uporablja v NetLogo integriranem orodju BehaviorSpace. Slednje omogoča izvajanje eksperimentov z modelom na način, v katerem se model izvede večkrat, vsakič z drugimi nastavitvami, saj v modelih pogosto nastopajo parametri z različnimi vrednostmi. Vse možne kombinacije vrednosti predstavljajo obsežen prostor, v katerem določene kombinacije parame-

trov ne privedejo do pričakovanega rezultata izvajanja modela. Orodje BehaviorSpace nam omogoča raziskati prostor potencialnih rezultatov izvajanja modela in določiti obseg vrednosti parametrov, ki vodijo do želenih vzorcev vedenja modela. Večjedrnost se tu izkoristi za sočasno izvajanje modelov, in sicer vsako jedro izvaja po en model [26].

3.2 BSim

Splošnonamenska orodja Repast, FLAME in NetLogo so orodja, ki imajo vgrajene le osnovne funkcionalnosti, povezane z modeliranjem bioloških sistemov. Namesto specializiranih funkcionalnosti spološnonamenska orodja omogočajo napredno konfiguracijo agentove dinamike in agentovega okolja. Takšna orodja omogočajo simulacijo modelov na različnih področjih, ampak pomanjkanje z biologijo povezanih funkcionalnosti predstavlja dodatno oviro pri vzpostavitvi delujoče biološko relevantne simulacije. Kompleksne celične lastnosti, kot so rast, podvojevanje, gibanje s fiziko okolja, v katerem se posnema gibanje in interakcija med celicami, je mogoče neposredno implementirati s specializiranimi orodji, ki so običajno ozko osredotočena na reševanje določenega problema. Poleg zelo specifičnih orodij obstaja vrsta orodij z osnovnimi biološkimi elementi, ki poenostavljajo razvoj novih modelov. Eno izmed takšnih orodij je tudi BSim [30].

Orodje BSim vključuje tridimenzionalno okolje, ki implementira Brownowo gibanje, difuzije kemičnih polj in možnost vključitve različnih oblik agentov znotraj ene simulacije. Različne predstavitve agentove dinamike so podprte s simulatorji za navadne diferencialne enačbe, diferencialne enačbe z zakasnitvijo in za dinamiko, definirano s splošnimi pravili. BSim vsebuje nabor primerov simulacij, ki so prilagodljive in katerih skupek rešuje številne primere agentnega modeliranja [21]. Področje uporabe orodja BSim so 2D ali 3D simulacije s stohastičnimi interakcijami bakterijskih populacij in delcev v tekočini ter primeri večceličnega računalništva v sistemski in sintezni biologiji. Orodje je osredotočeno na razvoj modelov velikega obsega v programskem jeziku Java [22].

Splošnonamenska orodja, kot sta FLAME in NetLogo, vsebujejo minimalni nabor funkcionalnosti z namenom enostavne vključitve uporabnikovih funkcionalnosti. Takšna usmerjenost poenostavlja uporabo programskega orodja in omogoča širok nabor aplikacij. V kontekstu bakterijskih sistemov takšen pristop ni priporočen. Splošne lastnosti bakterij so skupne mnogim modelom. Eden izmed primerov tovrstnih modelov je kemotaksa. Uporabnikom pri implementaciji takšnih modelov vzpostavitev okolja z zajetimi

lastnostmi celičnih sistemov in zapletenih procesov od začetka predstavlja dodatno oviro. Naslednja pomanjkljivost uporabe splošnonamenskih orodij v tej domeni je odsotnost implementacije okolja s fiziko, ki posnema gibanje celic in njihovih medsebojnih interakcij. Pomanjkljivost obstoječih orodij za modeliranje celičnih populacij je njihova specifičnost. Tovrstna orodja so osredotočena na vidike vedenja, ki so najpomembnejši za obravnavani problem. Primera takšnih študijskih primerov sta produkcija biomase in biokemično regulatorno omrežje kemotakse [31].

Avtorji orodja BSim so naslovili pomanjkljivosti specifičnosti orodij in predstavili BSim kot skupno ogrodje za razvoj agentnih modelov celičnih populacij. BSim predstavlja skupek obstoječega dela na tem področju, ki prihranja čas in trud s ponovno uporabljivostjo vgrajenih lastnosti celic ter z omogočanjem natančnega opisa kompleksnih mikrostruktur in okolja. Orodje omogoča analizo različnih nivojev abstrakcije, od individualnih agentov do zveznih polj. V orodje je vključena tudi realistična znotrajcelična dinamika celic preko simulacije gensko regulatornih in signalih omrežij v obliki navadnih diferencialnih enačb. Uporabniki se z orodjem spoznajo preko vrste obstoječih simulacij, katerih funkcionalnosti je mogoče prilagoditi in razširiti s tehnikami objektno orientiranega programiranja. BSim v nasprotju s splošnimi orodji, kot sta FLAME in NetLogo uporabnikom nudi nabor temeljnih funkcionalnosti za hitro ter učinkovito vzpostavitev delujoče simulacije bakterijske dinamike in vključitve matematičnih modelov, ki opisujejo dinamiko individualnih celic [31].

Predstavitev orodja bomo začeli s pregledom načina uporabe pričujočega programskega orodja. Pregledali bomo tudi simulacijsko okolje, programski jezik modeliranja in podrobnosti implementacije orodja.

3.2.1 Interaktivno okolje

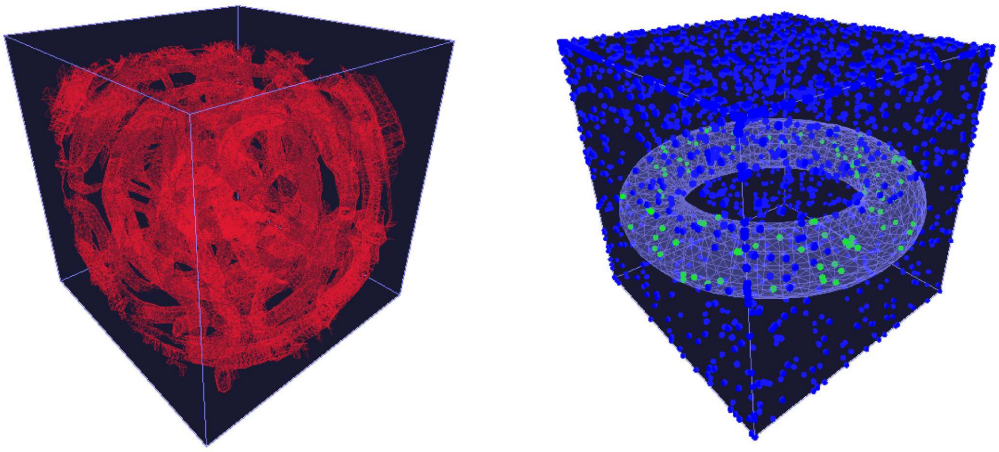
Orodje nima samostojnega interaktivnega grafičnega uporabniškega vmesnika, kot ga ima orodje NetLogo. V BSimu so modeli napisani v programskem jeziku Java, ki jih lahko implementiramo z navadnim urejevalnikom teksta ali z vrsto paketov za razvoj programske opreme, kot je Eclipse [32] ali IntelliJ [33].

3.2.2 Simulacijsko okolje

Simulacije v BSimu potekajo v tridimenzionalnem, s tekočino napolnjenem okolju. Obseg okolja je definiran s kvadrom in lastnostmi tekočin, kot sta viskoznost ter temperatura, ki

ju lahko spreminjamo, da dosežemo želene eksperimentalne pogoje. Za dosego različnih eksperimentalnih pogojev je mogoče definirati robove okolja na več načinov. Prvi tip robov so kompaktni robovi, od katerih se agenti in drugi elementi okolja med simulacijo odbijajo. Naslednji tip robov je rob, ki fizično omejuje gibanje. Zvezni ali periodični robovi omogočajo vzpostavitev večjega prostora. Katerikoli objekt, ki prečka zvezni rob, se pojavi na isti relativni poziciji na nasprotni strani simuliranega okolja [31].

V realnem okolju bakterije srečujejo in so v interakciji s kompleksnimi strukturami podobnega reda velikosti. BSim omogoča vključitev mikrostruktur v simulirano okolje kot generične objekte, ki jih lahko generiramo s programsko kodo ali jih uvozimo. Ti objekti definirajo poljubno oblikovno področje v prostoru in predstavljajo del simuliranega okolja modela. Z vpeljavo takšnih objektov in uporabe metod za detekcijo trkov dosežemo zapletene geometrijske meje ter prostorsko spremenljive parametre okolja ali vedenja [31]. Slika 3.3 prikazuje simulirano okolje v orodju BSim s kompleksnimi objekti, ki vplivajo na dinamiko celic.



Slika 3.3 Simulirano okolje z orodjem BSim. Levi del slike prikazuje okolje z vpeljanim prostorskim poligonom. Desni del slike prikazuje posnetek okolja med simulacijo, v kateri objekt v obliki torusa vpliva na vedenjske lastnosti celic [31].

Veliko truda na področjih sintezne in sistemske biologije je posvečenega raziskovanju ter razumevanju gensko regulatornih omrežij, ki so podrobno opisana v [34, 35] in predstavljena v poglavju 4. Dinamika bakterij na osnovi gensko regulatornih omrežij je v orodju BSim definirana kot sistem navadnih diferencialnih enačb. Gensko regulatorna omrežja je mogoče vključiti v agente, ki lahko vzajemno delujejo z lokalnim okoljem

preko izločanja ali zaznavanja kemijskih zvrsti in preko neposrednega kontakta z drugimi agenti ali objekti. Simulacija dinamike poteka z numeričnim reševanjem sistema navadnih diferencialnih enačb z Eulerjevo metodo, metodo Runge-Kutta reda 2 ali metodo Runge-Kutta reda 4, s katerimi dosežemo natančnost in robustnost pri integraciji različnih časovnih skal, prisotnih v gensko regulatornih omrežjih [30]. Funkcije za numerično reševanje diferencialnih enačb, povezane z različnimi agenti, tečejo neodvisno, kar omogoča učinkovito ločitev različnih časovnih skal. Takšen pristop omogoča uporabo različnih časovnih korakov. Bakterije, ki izražajo počasnejšo dinamiko, imajo večji časovni korak in je ni potrebno osveževati z enako frekvenco kot tiste, ki izražajo hitrejšo dinamiko ter imajo manjši časovni korak. Obstoječe delo na področju gensko regulatornih omrežij je osredotočeno na znotrajcelično dinamiko. BSim s svojimi funkcijami predstavlja orodje, s katerim je mogoče nasloviti, kako se regulatorna omrežja vedejo na nivoju populacije. Naslavlja tudi vpliv prostorskega pozicioniranja, vplive difuzije znotrajceličnih signalov in kakšen vpliv prinaša bakterijska heterogenost [31, 36].

3.2.3 Programski jezik

Agent je v orodju predstavljen kot objekt Java z enkapsuliranimi metodami za obdelavo njegovih lastnosti ob fizični interakciji in integraciji dinamike, predstavljene z gensko regulatornim omrežjem. BSim je zasnovan na ideji, da je vsak neodvisen element modela zajet v popolnoma ločenem modulu. Module je mogoče razširi z novimi funkcijami, lahko pa jim spremenimo osnovno implementacijo. Takšna modularna zasnova BSima omogoča nadgraditev starejših modelov z novo izdanimi moduli. Modularna zasnova velja tudi za avtonomne agente, ki predstavljajo bakterije, proteine ali celo posamezne molekule. Namesto implementacije neodvisnih tipov agentov se uporablja koncept delca, ki vključuje skupne lastnosti vseh agentov.

Definicija delca je definirana na predpostavki, da so delci sferične oblike s pripadajočimi lastnostmi velikosti, orientacije in pozicije. Delec predstavlja osnovi tip agenta. Novo definirani agenti podedujejo skupne lastnosti, ki so definirane v delcu, iz katerega izhajajo in po potrebi razširjajo osnovni tip agenta z dodatno definiranimi specifičnimi funkcijami. BSim je na tovrsten način razširljiv z novimi tipi agentov, kar omogoča razvoj novih funkcionalnosti orodja. Zaradi dedovanja skupne implementacije delca je te agente mogoče vključiti in pravilno modelirati njihovo fizično vedenje. Prednost takšnega pristopa je možnost definiranja različnih družin agentov s specifičnimi lastnostmi. Agent,

ki predstavlja bakterijo, izraža svojo dinamiko, medtem ko spremenjene ali mutirane različice dinamike pridobimo s spreminjanjem lastnosti tega agenta. To nam prihrani čas pri razvoju modela in izboljša kakovost modela s ponovno uporabo že preizkušenih funkcionalnosti [31].

Tip	Ime	Opis
		bsim
Razred	*BSim*	Vsebuje vse informacije, povezane s simulacijo, vključno z mejami in tipi mej okolja, temperaturo itd.
Razred	*BSimTicker*	Nadzira glavno zanko in posodablja simulacijo ob vsakem časovnem koraku.
Razred	*BSimThreadedTicker*	Razred za večnitno izvajanje posodobitev simulacije - vzporedno izvajanje posodobitev primernih delov simulacije.
		bsim.particle
Razred	*BSimParticle*	Temeljna avtonomna entiteta v simulaciji.
Razred	*BSimBacterium*	Razširitev osnovne entitete z lastnostmi bakterij.
		bsim.geometry
Razred	*BSimMesh*	Definicija poljubne površine v simulaciji.
Razred	*BSimCollision*	Metode za zaznavanje trkov.
		bsim.ode
Razred	*BSimOdeSolver*	Metode za numerično reševanje navadnih diferencialnih enačb.
Vmesnik	*BSimOdeSystem*	Vmesnik z metodami za definiranje sistema navadnih diferencialnih enačb.
		bsim.draw
Razred	*BSimDrawer*	Predloga za grafični prikaz simulacije brez implementacije, pripravljen za razširitev.
Razred	*BSimP3DDrawer*	Razširitev *BSimDrawer* za risanje z uporabo knjižnice *Processing*.

Tabela 3.2 Razredi in vmesniki orodja BSim, razdeljeni po paketih programske kode [30, 31].

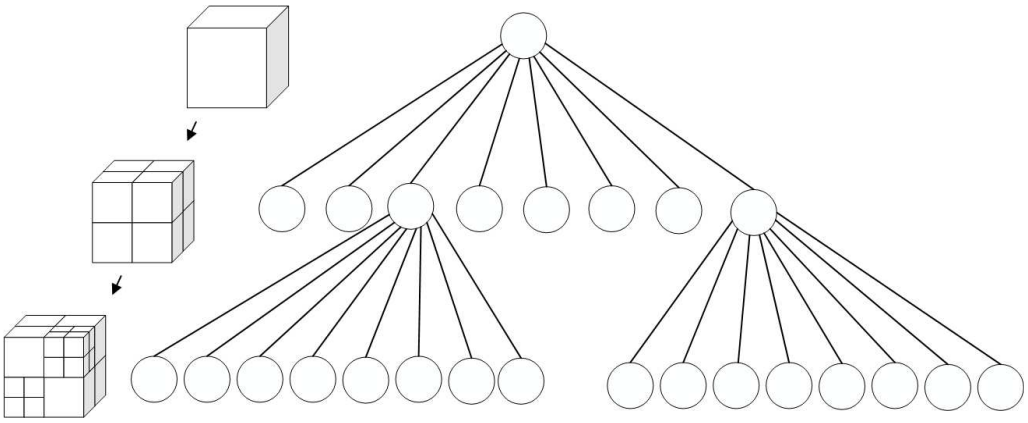
Tabela 3.2 prikazuje nekatere razrede in vmesnike orodja BSim s kratkimi opisi.

3.2.4 Implementacija orodja

Arhitektura orodja BSim je bila razvita z objektno usmerjenim programskim jezikom Java in je na voljo na platformah Windows, Linux in Mac OS. Za ustvarjanje animiranih 3D grafičnih objektov se uporablja knjižnica *Processing*. Ta skrbi za postopek, v katerem se v videu ali animaciji uveljavijo vse grafične ureditve v programski ali strojni opremi. Knjižnica omogoča tudi izvoz animacij v datoteko in vsebuje podporne funkcije za kreiranje grafičnih objektov [36].

Modeliranje in simuliranje fizičnih procesov in kompleksnih prostorskih okolij je podprto s podatkovnimi strukturami, kot je osmiško drevo (angl. *octree*), ki se uporablja za učinkovito hranjenje večnivojske predstavitve v okolju. Osmiško drevo je hierarhična podatkovna struktura, sestavljena iz nivojev gnezdenih vozlišč. Vozlišča v podatkovni strukturi predstavljajo območja prostora, kot je prikazano na sliki 3.4. Korensko vozlišče predstavlja celotno okolje. Vozlišče je povezano z nasledniki, ki predstavljeno območje naprej delijo na osem enakih delov. Za predstavitev BSimovega okolja vsako vozlišče v drevesu hrani koncentracijo kemijskih zvrsti, difuzijsko konstanto in povezave do nasle-

dnikov ter korena poddrevesa [37].



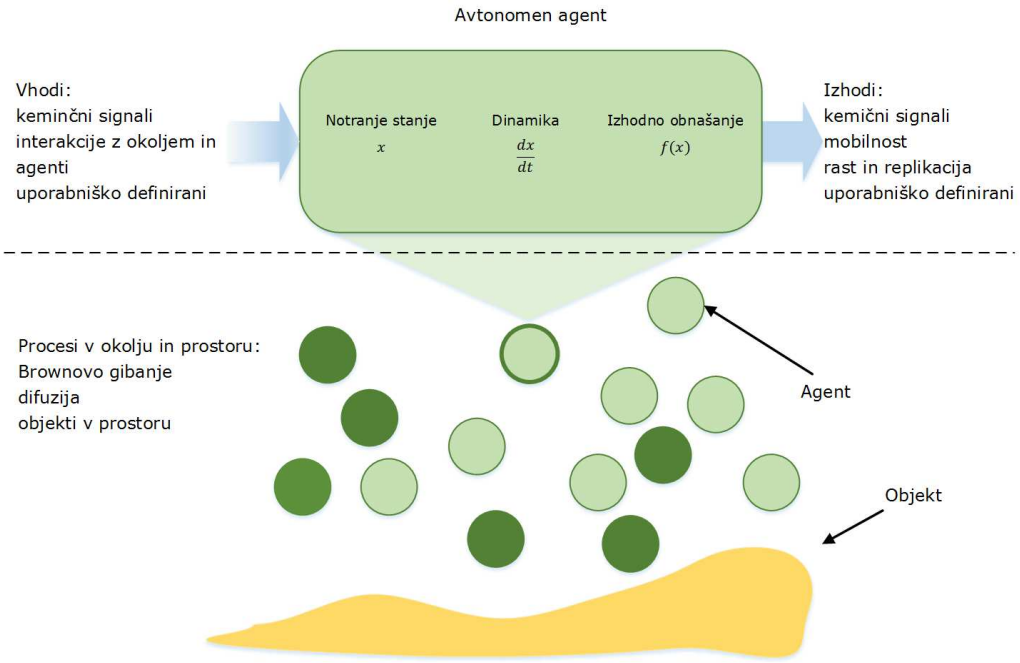
Slika 3.4 Prostor, predstavljen z osmiškim drevesom [37].

Struktura modela v orodju BSim je predstavljena na sliki 3.5. Zgornji del slike prikazuje zgradbo avtonomnega agenta, spodnji del pa deljeno okolje, v katerem bivajo agenti. Agenti vsebujejo vektor notranjega stanja, ki se skozi čas spreminja. Orodje podpira predstavitev agentove dinamike z navadnimi diferencialnimi enačbami ali z uporabniško definiranimi pravili. Agent zaznava različne faktorje okolja kot vhodne podatke in generira izhod v lokalnem okolju. Okolje zagotavlja skupen medij, v katerem se agenti gibljejo, komunicirajo preko kemičnih signalov in medsebojno vplivajo skozi fizični kontakt z drugimi agenti ali objekti [31].

Orodje BSim je bilo deležno nadgradnje z izdajo nove verzije v letu 2017. Nova verzija vključuje vse lastnosti prvotne verzije, hkrati pa dodaja in nadgradi funkcionalnosti s podporo opisa znotrajcelične dinamike z diferencialnimi enačbami z zakasnitvijo in navadnimi diferencialnimi enačbami, realno celično morfologijo pri rasti, delitvi ter interakciji celic in realne fizične parametre simuliranega okolja v 3D, vključujoč lastnosti kemotakse [36].

3.3 Gro

Naslednje specializirano orodje za modeliranje celice kot agent je orodje Gro [38]. Orodje omogoča implementacijo modelov bakterijske rasti z lastnim višje programskim jezikom, imenovanim Gro, v katerem definiramo parametre simulacije in agentova pravila. Orodje omogoča analizo dinamike globalne geometrije mikrokolonije, ki nastane iz lokalnih inte-



Slika 3.5 Shematski prikaz modela v orodju BSim. Model je sestavljen iz dveh nivojev: individualnih agentov in deljenega okolja [31].

rakcij celic.

Programski jezik Gro je osnovan na način enostavnega izražanja visokonivojskih pravil in hkrati omogoča implementacijo kakršnihkoli omrežij kemijskih reakcij ali modelov, temelječih na gensko regulatornih omrežij [21]. Pričujoče orodje je že bilo uspešno uporabljeno za opis raznih celičnih pravil v koloniji celic. Kolonija je morfološka struktura, ki nastane z delitvijo ene ali več celic. Jezik Gro je bil tako uporabljen za implementacijo modelov rasti celic kolonije različnih morfologij in implementacijo modelov, v katerih celice zaznavajo svoj položaj znotraj kolonije [38].

Orodje Gro je bilo razvito za specifikacijo, snovanje in simulacijo večceličnih modelov v 2D okolju s prisotnimi dejavniki okolja, ki so posledica celične rasti, visoke koncentracije celic in difuzije signalov. Orodje teče na platformi Mac OS in Windows. Arhitektura orodja je razvita v programskem jeziku C++ v razvojnem okolju Qt, ki se uporablja predvsem za razvoj grafičnih uporabniških vmesnikov in aplikacij, ki morajo teči na več platformah. Orodje Gro združuje pristopa porazdeljenih sistemov in vzporednega računanja za simulacijo rasti do nekaj tisoč bakterijskih celic v 2D okolju. Fizika v simuliranem okolju temelji na 2D pogonu za fiziko, implementiranem v programskem jeziku C.

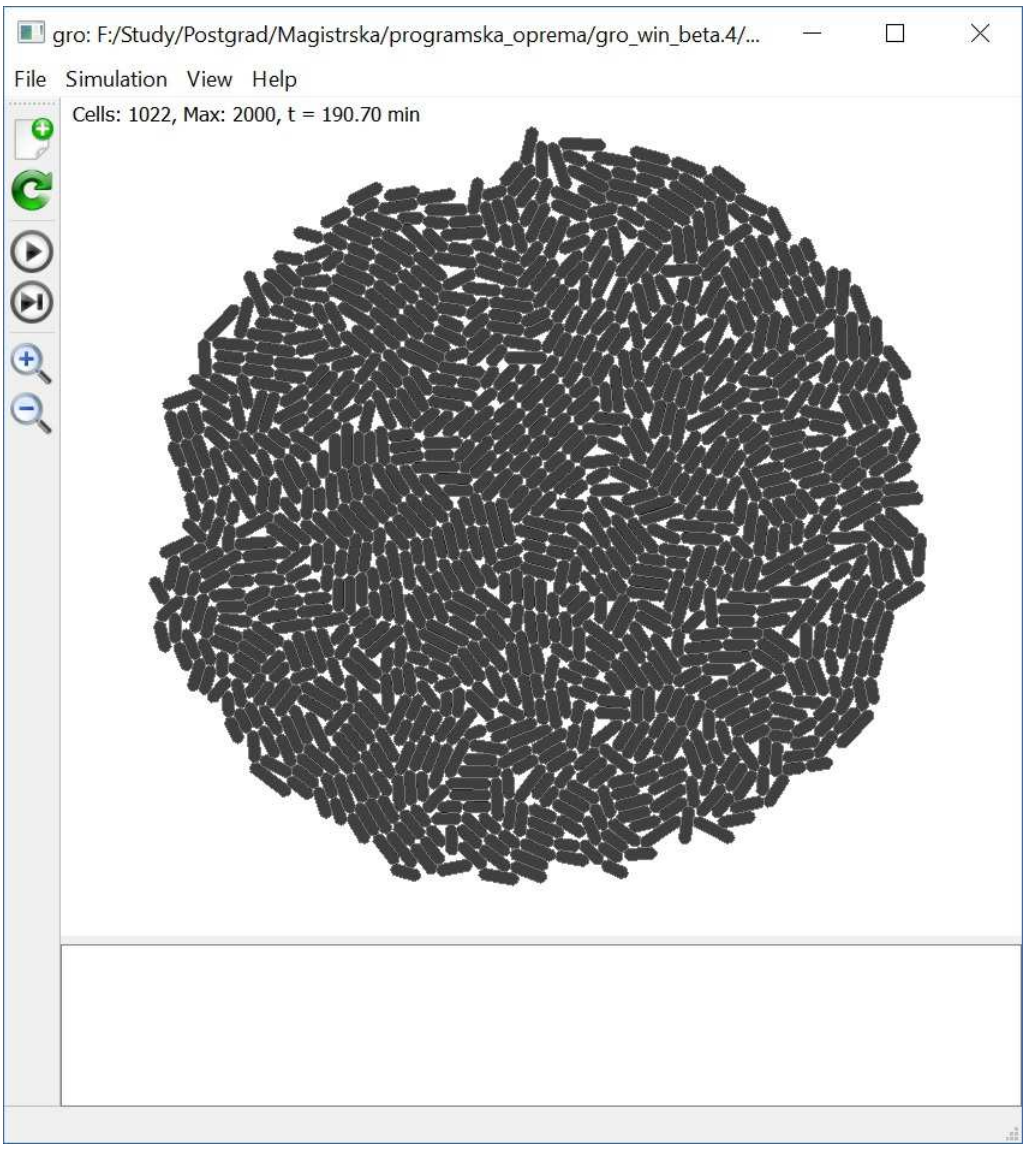
Orodje je osredotočeno na simulacijo rasti mikrokolonij bakterij s podobnimi lastnostmi, kot jih ima bakterija *Escherichia coli* (*E. coli*). Vizualizacija rasti bakterij je prikazana na enem nivoju s pogledom, kot bi ga imeli, če bi dinamiko bakterij opazovali s fluorescenčnim mikroskopom, ki se uporablja za analizo in preučevanje celic, njihove strukture in prisotnosti kemijskih spojin s procesom vzbujanja fluorescence snovi v mikroskopskem preparatu. Avtorji orodja se v simulacijah osredotočajo na dogajanje v razvoju prvih generacij bakterij v procesu formacije kolonije, saj naj bi po okoli 10 generacijah rasti prišlo do visoke koncentracije bakterij v sistemu in posledično do ravnovesja [38].

V naslednjih poglavjih bomo predstavili orodje Gro. Spoznali bomo grafični uporabniški vmesnik in način predstavitve vizualizacije simulacije. Pri tem bomo spoznali programski jezik, njegove lastnosti in izvorno kodo. Predstavili bomo arhitekturo orodja in način izvajanja simulacij.

3.3.1 Interaktivno okolje

Orodje bomo začeli spoznavati skozi grafični uporabniški vmesnik, ki je prikazan na sliki 3.6. Vmesnik omogoča uporabnikom simuliranje že definiranih modelov, vključenih v distribuciji orodja ali uporabniško definiranih modelov. Na sliki 3.6 je prikazan vmesnik med simulacijo preprostega modela rasti celic iz vključenega nabora definiranih primerov v imeniku *examples*. Po odprtju programa je simulator Gro pripravljen na simulacijo. Orodna vrstica na levem robu programskega okna vsebuje gumbe za zagon, začasno ustavitev simulacije, simuliranje enega časovnega koraka in spreminjanje velikosti prikaza vizualizacije simulacije. Prikaz 2D simuliranega okolja zasede večji del površine grafičnega vmesnika s pogledom iz zornega kota, kot ga imamo pri opazovanju kolonije z mikroskopom.

Simulacija modela, prikazanega na sliki 3.6, se začne z eno celico *E. coli*, inicializirano na poziciji (0,0) v simuliranem okolju. S pritiskom na gumb za zagon simulacije celica začne rasti. Celica raste do približno dvokratnika prvotnega volumna. Ko doseže takšen volumen, se ob delitvi celice ustvarita dve po velikosti približno enaki celici. Simulacija teče, dokler je ne ustavimo ali dokler ne dosežemo nastavljene koncentracije celic. V pričujočem modelu se simulacija ustavi, ko je v populaciji 2000 ali več celic. Simulacija ne poteka v realnem času ali s konsistentnim časovnim korakom. Število celic narašča eksponentno, kar pomeni, da skupaj s takšno rastjo celic tudi orodje izvaja eksponentno več računanja. Sčasoma simulacija postane vse počasnejša [39].

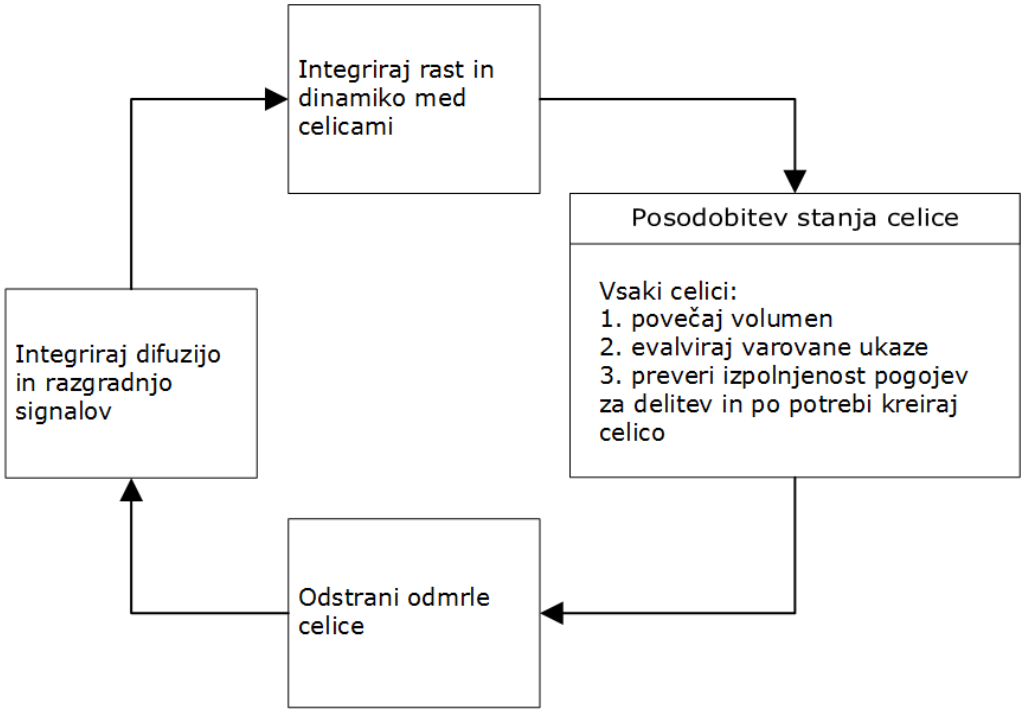


Slika 3.6 Grafični uporabniški vmesnik orodja Gro. Vmesnik sestavlja stranska orodna vrstica z gumbi za nadzor poteka simulacije in grafičnega prikaza simulacije. Uporabniški vmesnik je prikazan med simulacijo modela *growth.gro* [39].

3.3.2 Simulacijsko okolje

Orodje Gro omogoča realističen simulator formacije bakterijskih mikrokolonij, ki posnema dinamiko celice *E. coli* na enojni celični plasti (angl. *monolayer*) in je opazovana skozi mikroskop. Simulator vsebuje implementacijo procesov rasti in delitve, vpliva kontaktnih sil med celicami ter difuzijo majhnih molekul. Shema povezanosti in usklajenosti

procesov je prikazana na sliki 3.7. Glavno zanko orodja sestavljajo štirje koraki. V vsaki iteraciji glavne zanke se posodobi notranje stanje celic. Med drugim se poveča volumen posamezne celice in evalvirajo ukazi, ki so povezani z določeno celico. Kot v realnem svetu tudi v simulaciji prihaja do umiranja celic. V simulaciji se korak odstranitve odmrlih celic izvede po posodobitvi stanja celic. Iteracija zanke izvajanja simulacije se nadaljuje z integracijo vplivov difuzije in degradacije signalov. V zadnjem koraku se integrira še celična rast in vpliv celične rasti oziroma sil med celicami, ki nastanejo zaradi rasti celic.



Slika 3.7 Zanka korakov delovanja orodja Gro. V vsaki iteraciji zanke se posodobi notranje stanje celic in izvede delitev celic. V naslednjem koraku se odstranijo vse odmrle celice. Temu sledi vključitev vpliva difuzije in degradacije signalov z numerično integracijo. V zadnjem koraku se z numerično integracijo upošteva tudi rast celic in vpliv rasti le-teh na druge celice v okolju [38].

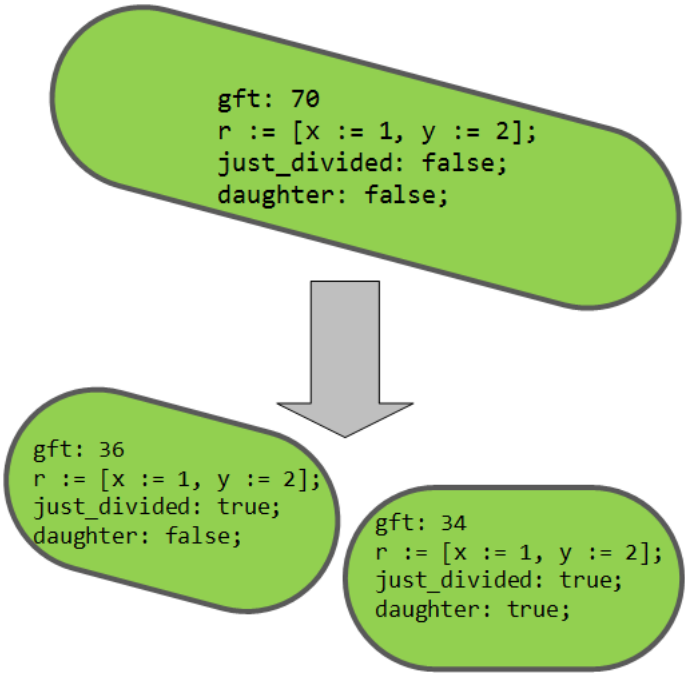
Za celice je predpostavljeno, da so cilindrične oblike s polmerom r=0,5 μm in začetno dolžino l=2 μm. Časovna ločljivost vsakega podprocesa simulacije je nadzorovana s parametrom časovnega koraka *dt*. Časovni korak je nastavljiv parameter v programski kodi. Začetni volumen celice v Gro je V=$\pi r^2 l$ = 1,57 fL. Hitrost rasti ali rastni faktor celice k je nastavljiv. Privzeta vrednost hitrosti rasti celice je 0,0081 fL/min. Takšna hitrost rasti prinese 85-minutni čas podvojitve celice, kar pomeni, da ena celica raste približno 85 minut, nakar se celica razdeli na dve celici. Rast celice je modelirana po

diferencialni enačbi 3.1.

$$\frac{dV}{dt} = kV \tag{3.1}$$

Enačba je numerično integrirana s časovnim korakom dt, vzporedno s simulacijo drugih podprocesov. Število celic in volumen mikrokolonije tako rastejo eksponentno. Simulirana rast mikorkolonije v Gro se kvalitativno ujema z rastjo dejanske celice *E. coli* [38].

Vsaka celica raste, dokler ne doseže dvokratnika prvotnega volumna, kar smo spoznali že pri modelu rasti celice, prikazanem na sliki 3.8. Ko ima celica približno dvakrat večji volumen, se ta razdeli na približno dve enaki celici. Delitev celice je shematično prikazana na sliki 3.8. Vsaka celica je povezana, ali pa ji je dodeljen določen program oziroma zaporedje ukazov in spremenljivk. Slika 3.8 prikazuje, kako se vrednosti spremenljivk priredijo ob delitvi celic v matični in hčerinski celici.



Slika 3.8 Shematski prikaz delite celice. Pri delitvi celice se vrednosti spremenljivk matične in hčerinske celice ponovno priredijo. Numeričnim spremenljivkam se vrednosti približno razpolovijo, medtem ko vrednosti spremenljivk drugih tipov ostanejo nespremenjene. Spremenljivki tipa boolean *just_divided* in *daughter* se dodelita po delitvi. Spremenljivki označujeta, da je prišlo do dogodka delitve celice [38].

Pri delitvi celice smo spoznali, da se celica deli, ko približno doseže dvokratnik svoje

prvotne lastnosti. Kdaj točno pride do delitve celice, določata nastavljiva parametra povprečna vrednost μ in varianca σ^2 velikosti delitve. Privzeti vrednosti spremenljivk sta μ = 3,14 fL in σ^2 = 0,005 fL^2. Število celic v Gro tako narašča po diskretnem stohastičnem procesu [38].

Simulirane celice so omejene na eni plasti. Fizika rasti in kontaktnih sil je modelirana z 2D pogonom za fiziko Chipmunk [40]. Celice so modelirane kot 2D poligoni. Kontaktne sile med celicami so nastavljene tako, da celice med seboj drsijo in formirajo nasičeno geometrijo, kot jo lahko opazimo v realnem gojenju mikrokolonije. V vsakem koraku glavne zanke simulacije Gro, ki je prikazana na sliki 3.7, se volumen celic ustrezno prilagodi zahtevanim kontaktnim silam, ki odražajo opisano dinamiko formiranja mikrokolonije. Potem se pokliče pogon za fiziko, ki simulira ustrezne kontaktne sile in ustvari gibanje za *dt* časovnih enot.

Orodje Gro omogoča simuliranje še enega biološkega pojava, značilnega za celice, to je komunikacija med celicami preko molekul. Komunikacija med celicami je v Gro opisana s parcialnimi diferencialnimi enačbami. Za simulacijo dinamike signalov uporablja metodo končnih elementov (angl. *finite element method*) [41]. Metoda končnih elementov je ena izmed numeričnih metod za iskanje rešitev integralskih enačb in parcialnih diferencialnih enačb, ki probleme, definirane definirane v kompleksnih geometrijah. Za simulacijo dinamike komunikacije se uporablja Eulerjeva metoda z enakim korakom *dt* časovnih enot, kot se uporablja pri simulaciji rasti in gibanja. Difuzijski dejavnik in dejavnik degradacije se podata ob deklaraciji nove signalne molekule. Celice lahko oddajajo, zaznavajo in absorbirajo molekule. Poleg celice *E. coli* Gro omogoča vmesnik za definiranje novih tipov celic s poljubno geometrijo [38].

3.3.3 Programski jezik

Vsaka celica v simulaciji izvaja program, napisan v programskem jeziku Gro. Jezik Gro je močno tipiziran, kar pomeni, da je v spremenljivke mogoče zapisati vrednosti določenega tipa. Programski jezik Gro sodi v skupino interpretiranih programskih jezikov. Tip spremenljivke v kodi je lahko eden od naslednjih primitivnih tipov, ki jih poznamo tudi v drugih programskih jezikih:

- *boolean*,
- *integer*,

- *real* – števila predstavljena z dvojno natančnostjo,
- *string*,
- *list* – homogeni seznami,
- zapisi (angl. *records*) – grupiranje podatkov,
- *lambda* izrazi – neimenovane funkcije.

Primer preprostega izraza lambda je prikazan v kodi 3.2, ki prejetemu številu negira vrednost.

Koda 3.2 Primer izraza lambda v programskem jeziku Gro. Izraz negira vrednost podanemu argumentu.

```
lambda x . -x;

// podajanje argumenta izrazu lambda
(lambda x . -x) 10;
```

V pregledu orodja smo že spoznali, da ima vsaka celica v simulaciji program. Program v Gro je zaporedje inicializacij in varovanih ukazov (angl. *guarded commands*). Slednji so oblike *g:c*, varovalo *g* je logični izraz, *c* pa ukaz. Ukaz je lahko seznam prireditev ali klic funkcij. Varovani ukazi omogočajo modeliranje vzporednosti. Za lažjo interpretacijo konstrukta program si poglejmo primer izvorne kode 3.3.

Koda 3.3 Primer programa, ki prejme en argument. Program vsebuje inicializacijo spremenljivke *t* in dva varovana ukaza. Prvi varovani ukaz povečuje vrednost spremenljivke *t*, drugi pa jo postavi na 0, ko je ta večja ali enaka vrednosti prejetega argumenta *x*. Inicializacijski stavek se izvede le enkrat, medtem ko se varovana ukaza evalvirata v vsaki iteraciji posodobitve celice [39].

```
program p(x) := {
  t := 0;

  true :   { t := t + dt }
  t >= x : { t := 0      }
};
```

V izvorni kodi 3.3 je prikazan primer inicializacije spremenljivke t in primer dveh varovanih ukazov: prvi vrednost spremenljivke t povečuje, drugi vrednost postavi na 0, ko je ta večja ali enaka vrednosti spremenljivke x. Ukazi inicializacije se izvedejo samo ob zagonu programa. Ponovno se ne izvedejo niti ob delitvi celice, ko se program matične

celice kopira v hčerinsko celico. Varovani ukazi se izvedejo v vsakem koraku simulacije in ob vsaki posodobitvi stanja celice, če se izraz oziroma varovalni pogoj evalvira v logično vrednost *true*. Cilj takšnega pristopa je, da vsak ukaz z varovalom predstavlja ločen proces v celici in da se ti izvedejo sočasno. Kot v standardnih pristopih modeliranja vzporednosti je vrstni red izvajanja ukazov z varovalom nedoločen [38, 39].

Stohastične dogodke v celici je mogoče modelirati s posebno funkcijo *rate*, ki sprejme en argument in naključno vrne *true* ali *false*. Izraz *rate (r)* vrne *true* po določeni oceni z verjetnostjo $r \cdot dt$ in *false* v drugih primerih. Parameter dt je časovni korak simulacije.

Interpretacija programa v Gro se začne z razčlenitvijo in preverjanjem podatkovnih tipov. Programi, povezani s celicami, so inicializirani. To pomeni, da se izvedejo vsi ukazi, ki niso del ukazov z varovalom. V tem koraku se inicializirata tudi začetna pozicija in orientacija celic. Izvajanje simulacije se izvaja po korakih, predstavljenih na sliki 3.7:

- V vsaki celici se posodobi volumen in izvede program varovanih ukazov. Preverijo se tudi pogoji delitve celice. Če so pogoji izpolnjeni, pride do delitve celice. Pri delitvi se priredijo spremenljivke matični in hčerinski celici. Hčerinska celica prejme kopijo programa in notranje stanje matične celice.
- Odstranijo se odmrle celice.
- Koncentracije signalnih molekul se posodobijo z numerično integracijo reakcijskih/difuzijskih procesov.
- Dinamika rasti in kontaktne sile med celicami so numerično integrirane z integriranim 2D pogonom za fiziko.

Programi lahko različno vplivajo na obnašanje gostiteljeve celice. Spoznali smo različne funkcionalnosti orodja Gro in kaj te omogočajo. Simuliramo lahko dinamiko rasti celic, ki oddajajo ali zaznavajo signale. S programi lahko dosežemo delitev celic ne glede na volumen ali velikost celice. Nismo pa omejeni le na programiranje opisanih celičnih lastnosti; napišemo lahko poljubno logiko, program ali stohastični proces.

3.3.4 Implementacija orodja

Programsko orodje Gro je razvito v programskem jeziku C++. Dinamika celice je implementirana s knjižnico funkcij za fiziko v 2D prostoru. Knjižnica vsebuje funkcije za

operacije s polinomi, kontaktnimi silami in za trenje teles v prostoru. Simuliranje signalov je implementirano z modificirano metodo končnih elementov. Grafika v simulaciji in uporabniški vmesnik sta implementirana v razvojnem okolju Qt [38].

Orodje Gro je v letu 2017 dobilo nadgradnjo. V novi verziji so naslovili pomanjkljivosti prve verzije in orodje razširili z novimi funkcionalnostmi. Večjo pomanjkljivost prve verzije predstavlja algoritem, ki upravlja z mehaniko celic v koloniji. Algoritem se je izkazal kot potraten in predstavlja omejitev pri simulaciji večjega števila celic. Hitrost izvajanja simulacije prvotne verzije se pri približno 3000 celicah v koloniji občutno zmanjša. V novi verziji so vključeni novi pogoni in knjižnice, ki razširjajo implementacijo obstoječih celičnih procesov. Dodana je bila tudi nova oblika medcelične komunikacije z bakterijsko konjugacijo [20].

3.4 CellModeller

Programsko orodje CellModeller je osredotočeno na simulacijo formacij sintetičnih biofilmov. Orodje temelji na knjižnici OpenCL, ki omogoča učinkovito simulacijo kolonije z več kot 30000 celicami. Napredek na učinkovitem izkoriščanju virov GPE za izvajanje računsko intenzivnih procesov omogoča izvajanje simulacij vzporednih problemov velikega obsega z vzporednim izvajanjem večjega števila niti. OpenCL je ogrodje, ki razvijalcem omogoča pospešitev aplikacij z vzporednim izvajanjem operacij na heterogenih platformah, sestavljenih iz CPE, GPE in drugih procesorjev [42]. CellModeler omogoča specifikacijo in simulacijo agentove dinamike, opisane s pravili ali pravili v obliki diferencialnih enačb [21].

Mikrobni biofilmi so kompleksne združbe bakterij, sposobne samoorganizacije. Zanje sta značilno fiziološko sodelovanje in prostorska organizacija, ki povečujeta tako metabolično učinkovitost kot odpornost na spremembe v lokalnem okolju. Slednje lastnosti so vzrok za privlačnost biofilmov v znanstvenih krogih, še posebno na področju produkcije kemikalij, kot so sestavine farmacevtskih izdelkov in biogoriva. Biofilmi so pogosto tudi glavni dejavniki trajnih okužb. Boljše razumevanje njihove organizacije bi pripomoglo k razvoju novih pristopov tovrstnega zdravljenja. Takšna dinamika biofilmov je koordinirana z različnimi oblikami sporazumevanja med celicami. Sporazumevanje poteka preko zaznavanja kvoruma (angl. *quorum sensing*), produkcije in zaznavanja majhnih molekul, zmožnih difuzije ter sporočanja na osnovi kontaktov [43].

Snovanje sintetičnega biofilma predstavlja izziv zaradi mnogih medsebojnih dejavnikov transkripcijske regulacije, medceličnega sporočanja in celične biofizike. Računalniško modeliranje na tem področju stremi k razvoju orodja, s katerim je mogoče modelirati dinamiko biofilmov in predvideti njihovo vedenje. Biofilm je kompleksen sistem, ki je sestavljen iz velikega števila celic. Slednja značilnost predstavlja omejitev oziroma veliko računsko zahtevnost pri računalniškem modeliranju števila celic, ki ga najdemo v realnih sistemih [43].

Orodje je osredotočeno na izvajanje morfoloških lastnosti kolonij in simulacijo realnih rastnih pogojev, ki jih najdemo v eksperimentalnih postavitvah. Druge lastnosti orodja bomo spoznali v pregledu uporabniškega vmesnika, simulacijskega okolja, programskega jezika modeliranja in implementacije orodja.

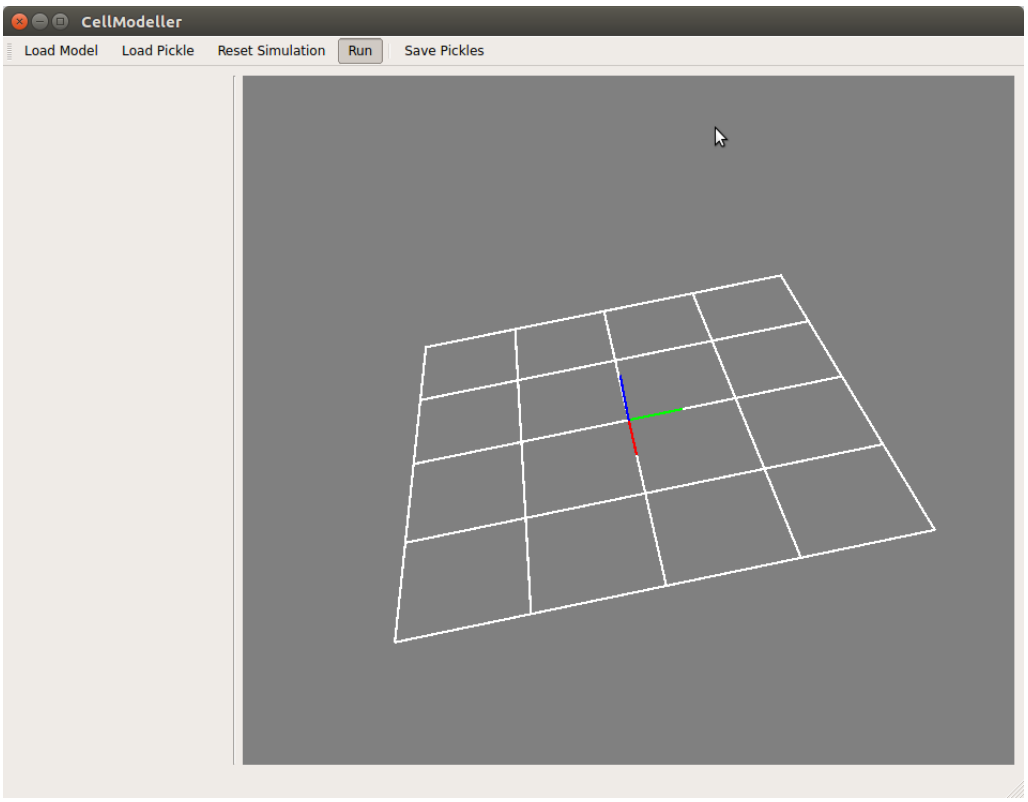
3.4.1 Interaktivno okolje

Orodje CellModeller vsebuje enostaven grafični uporabniški vmesnik, ki je prikazan na sliki 3.9. Vmesnik sestavlja orodna vrstica z gumbom *Load Model* za odpiranje modelov, gumba *Load Pickle* za nalaganje simulacije iz shranjenega stanja, gumba *Run* za zagon simulacije, gumba *Reset Simulation* za postavitev simulacije v začetno stanje in stikala *Save Pickle*, ki je aktivno, če se shranjujejo podatki simulacije. Orodje v osnovni distribuciji vsebuje implementirane primere modelov v imeniku *Examples*. Uporabniški vmesnik sestavlja tudi 3D pogled na simulirano virtualno okolje.

3.4.2 Simulacijsko okolje

CellModeller je modularno ogrodje za izvajanje modelov znotrajceličnih procesov, medceličnega sporočanja in biofizike celic. Vsaka od teh komponent različno vpliva na notranje stanje celice ali zunanje okolje. Nastale spremembe vplivajo tudi na druge komponente preko povratnih zank. Denimo, da celica zaznava lokalno koncentracijo signala in aktivira transkripcijo. Slednji proces vpliva na rast in gibanje celice ter spremembo lokalne koncentracije signala. Naprej lahko sprememba koncentracije signala vpliva na sam proces transkripcije in tako predstavlja primer povratne zanke ali vzročno-posledične zveze [43].

V simulaciji biofilmov je vsaka celica preko biofizičnih interakcij in signalov povezana z več celicami. Ker rast celic poteka počasneje kot biofizične interakcije, se posodobitev rasti in znotrajceličnih procesov ter sistemov sporočanja izvaja ločeno. Po vsakem koraku



Slika 3.9 Grafični uporabniški vmesnik programskega orodja CellModeller.

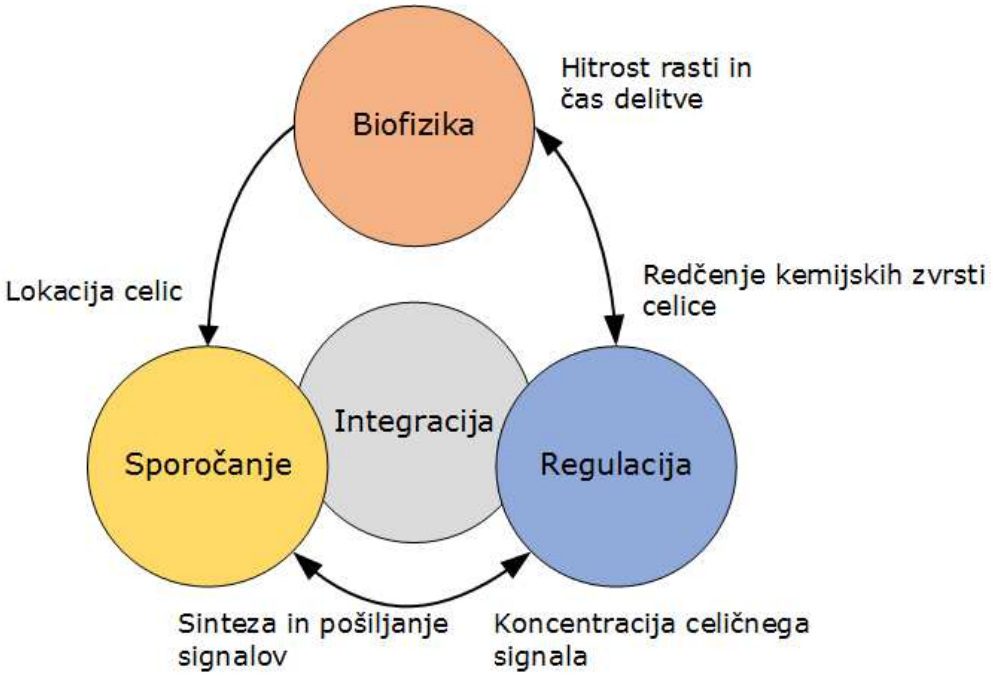
biofizičnega procesa se stanje celice, ki je definirano z volumnom, pozicijo itd., posodobi. Znotrajcelični procesi in sporočanje se posodobijo z ustreznim časovnim korakom. Celice istega tipa so deležne izvajanja posodabljanja svojega stanja z enakim naborom pravil in sistemom diferencialnih enačb. CellModeller slednje rešuje vzporedno s pristopom SIMD (angl. *single instruction multiple data*) za izračun vpliva vsake celice na celotno simulacijo [43].

Izvajanje simulacije povezuje različne module orodja v naslednjih korakih:

- klic metode *update* za aplikacijo definiranih pravil vsaki celici;
- delitev celic, ki imajo spremenljivko *divideFlag*, nastavljeno na *True*;
- integracija rasti celic z določenim časovnim korakom Δt z upoštevanjem omejitev okolja;
- integracija kemijskih zvrsti in zunajceličnih signalov vsake celice za časovni korak Δt;

- posodobitev spremenljivke stanja vsake celice ter ponovni klic metode *update*.

Struktura orodja CellModeller je shematsko prikazana na sliki 3.10. Prikazani moduli so zajeti v objektu *Simulator*, ki izvaja simulacijo in kreira različne module.



Slika 3.10 Struktura orodja CellModeller. Vsebovani implementirani moduli biofizike, sporočanja in regulacije v orodju za modeliranje in simuliranje večceličnih sistemov [44].

Glavni moduli so *CLBacterium*, *CLEulerIntegrator*, *CLCrankNicIntegrator* in *GridDiffusion*. Našteti moduli so implementirani v svojih razredih. Razred *CLBacterium* zajema implementacijo biofizike, ki izvaja kontrolo nad lastnostmi fizike in interakcijo celic. Implementacija razreda *CLEulerIntegrator* omogoča reševanje navadnih diferencialnih enačb in posledično omogoča simulacijo izražanja dinamike genov ter kemijskih zvrsti znotraj celice. Razred *GridDiffusion* zajema implementacijo modela za medcelično signaliziranje, ki izvaja proces difuzije signalov na 3D prostorski mreži. Razred *CLCrankNicIntegrator* implementira integrator za reševanje parcialnih diferencialnih enačb dinamike kemijskih zvrsti znotraj celice in difuzije zunanjih signalov [43, 44].

3.4.3 Programski jezik

Modeli so implementirani v programskem jeziku Python. Modeli sestavljajo funkcije, ki inicializirajo objekt *Simulator* z zahtevanimi moduli, parametri, začetnimi pogoji in pravili za nadzor izvajanja sistema.

Agenti kot celice so v orodju paličaste oblike. Stanje vsake celice lahko opišemo z vektorjem v enačbi 3.2, pri čemer je $(c_x, c_y, c_z)^T$ pozicija celice, $(\phi_x, \phi_y, \phi_z)^T$ je orientacija, L pa dolžina celice.

$$\vec{x} = (c_x, c_y, c_z, \phi_x, \phi_y, \phi_z, L)^T \tag{3.2}$$

Vsaka celica vsebuje nabor spremenljivk stanja, navedenih v tabeli 3.3. Poleg naštetih lahko definiramo tudi svoje spremenljivke.

Ime	Opis
pos	masno središče celice
dir	orientacija celice
length	dolžina (μm)
radius	polmer (μm)
volume	volumen (μm^3)
species	seznam koncentracij notranjih kemijskih zvrsti
signals	seznam koncentracij signalov
cellType	celoštevilski identifikator tipa celice
growthRate	relativna hitrost rasti celice (min^{-1})
dividedFlag	sproži delitev celice, ko je *True*

Tabela 3.3 Celica je definirana s spremenljivkami stanja [43].

Dinamiko celic lahko opišemo na več načinov. Obstaja vrsta kompleksnih mehanizmov, ki določajo dinamiko celice. Obstajajo dinamike takšne narave, kjer ni treba eksplicitno modelirati kompleksnih mehanizmov dinamike, ampak za takšne procese zadostujejo že empirično izpeljana pravila. Takšna pravila lahko v CellModeller zapišemo v funckijo *update*. Primer enostavnih pravil, definiranih v funkciji *update*, prikazuje koda 3.4. V funkciji *update* se sprehodimo čez vse celice in preveimo, če je dosegla ciljno velikost *cell.targetVol*. Če je celica dosegla velikost delitve, se celici priredi spremenljivka stanja *cell.divideFlag* vrednost *True* in se jo tako označi za delitev, ki se izvede v nada-

ljevanju procesa simulacije. Ob delitvi celice nastaneta dve hčerinski celici, ki privzeto podedujeta vrednosti spremenljivk stanja matične celice. Drugačen model delitve celic lahko definiramo v funkciji *divide(parent, daughter1, daughter2)*. Primer specifičnega modela delitve celic je nastavitev spremenljivke *cell.targetVol* oziroma velikosti celice, pri kateri želimo izvesti delitev. Naslednji primer eksplicitne implementacije delitve modela je tudi naključna razdelitev kemijskih zvrsti matične celice med hčerinski celici.

Koda 3.4 Funkcija *update* z definiranimi pravili dinamike celic. V funkciji za vsako celico preverimo izpolnjenost pogoja za delitev celice. Če je pogoj izpolnjen oziroma je volumen celice večji od volumna za delitev, se celica označi za delitev s nastavitvijo spremenljivke *divideFlag* [44].

```
def update(cells):
        for (id, cell) in cells.iteritems():
                if cell.volume > cell.targetVol:
                        cell.divideFlag = True
```

Osnovno strukturo modela poleg funkcij *update* in *divide* sestavljata še funkciji *setup* in *init*. V funkciji *setup* se izvede inicializacija modulov z ustreznimi parametri. Katere module in s kakšnimi parametri bomo inicializirali, je odvisno od modela, ki ga želimo razviti. Glavne module orodja smo spoznali že v 3.4.2. Če želimo modelirati enostavno rast celic, inicializiramo modul za biofiziko *CLBacterium*. Za kompleksnejše modele, kot je model ekspresije genov, inicializiramo poleg osnovnega modula biofizike *CLBacterium* tudi integratorski modul *CLEulerIntegrator* za reševanje diferencialnih enačb. Ta omogoči simulacijo dinamike genov in kemijskih zvrsti. V funkciji *init* se nastavi stanje celice. Stanje celice se nastavi kot začetni pogoj na začetku simulacije, ko je celica kreirana ali ob delitvi celice. Implementacija funkcij *setup* in *init* modela enostavne rasti celice je prikazana v kodi 3.5.

Koda 3.5 Funkciji *setup* in *init*, v katerih inicializiramo potrebne module za izvajanje simulacije oziroma nastavimo stanje celic ob ustvaritvi le-teh [44].

```
def setup(sim):
        #nastavi modul za biofiziko
        biophys = CLBacterium(sim, jitter_z=False)

        #nastavi modul za regulacijo
        regulator = ModuleRegulator(sim, sim.moduleName)

        #init. objekta Simulator z modulom za biofiziko
```

```
        #in regulacijo
        sim.init(biophys, regulator, None, None)

        #dodaj celico v simulacijo z lokacijo in orientacijo
        sim.addCell(cellType=0, pos=(0,0,0), dir=(1,0,0))

        #vizualizacija simulacije
        therenderer = Renderers.GLBacteriumRenderer(sim)
        sim.addRenderer(therenderer)

def init(cell):
        #povp. vrednost in porazdelitev
        #velikosti delitve celice
        cell.targetVol = 3.5 + random.uniform(0.0,0.5)
        #hitrost rasti celice
        cell.growthRate = 1.0
        cell.color = (0.0,1.0,0.0)
```

V pričujočem primeru modela rasti celic smo spoznali način definiranja preprostih agentovih pravil. Uporaba različnih nivojev abstrakcije pri definiranju agentovih pravil je odvisna od sistema, ki ga želimo modelirati. Orodje je osredotočeno na modeliranje biofilmov in če želimo simulirati dinamiko transkipcijskih omrežij v koloniji celic, potrebujemo model z natančnejšim opisom dinamike. Procesi znotraj celic so običajno opisani z diferencialnimi enačbami. Primer takšnega procesa predstavlja oscilatorno omrežje, sestavljeno iz transkripcijskih regulatorjev [45]. Kot primer si poglejmo sistem, opisan z diferencialnimi enačbami, pri katerem vsaka enačba opisuje spremembo kemijske zvrsti. Sprememba kemijskih zvrsti $\vec{u}$ naj bo predstavljena v obliki enačbe 3.3, pri čemer funkcijo f definira uporabnik.

$$\frac{d\vec{u}}{dt} = f(\vec{u}) \tag{3.3}$$

Iskanje rešitev sistemov diferencialnih enačb je računsko draga operacija. V orodju CellModeller se uporablja pristop, v katerem se za reševanje sistema enačb uporabi OpenCL, ki vzporedno za vsako celico i izračuna $f(u_i)$. V izvorni kodi modela se sistem enačb implementira v posebni funkciji. Primer implementacije tovrstne funkcije je zapisan v kodi 3.6. Primer kode predstavlja enostaven sistem, v katerem je dinamika ke-

mijskih zvrsti opisana z diferencialno enačbo. Za ilustracijo primera predpostavimo, da je celica ob rojstvu inicializirana z določeno koncentracijo kemijske zvrsti. Koncentracija kemijskih zvrsti celice se priredi namenski spremenljivki *species*. Koda 3.6 ponazarja sistem z eno kemijsko zvrstjo *x0*, ki se proizvaja znotraj vsake celice s hitrostjo produkcije *k1*.

Koda 3.6 Funkcija *specRateCL* z implementirano dinamiko, opisano z diferencialno enačbo. Koda v funkciji se v izvajanje poda OpenCL [44].

```
def specRateCL():
    return '''
    const float k1 = 2.f;
    float x0 = species[0];
    rates[0] = k1;
    '''
```

Izvorna koda znotraj funkcije *specRateCL* je napisana v programskem jeziku C. Ta koda je podana OpenCL in prevedena med izvajanjem. OpenCL poskrbi, da se koda izvede na CPE ali GPU [43].

3.4.4 Implementacija orodja

Orodje je razvito s programskim jezikom Python in ogrodjem OpenCL za platformi Mac OS in Linux. Pri implementaciji so bili uporabljeni naslednji paketi:

- *pyopencl* za dostop do programskega vmesnika OpenCL za vzporedno računanje ter
- *Numpy* in *Scipy* za izvajanje matematičnih operacij oziroma znanstveno obdelavo podatkov v Pythonu.

Uporaba ogrodja OpenCL omogoča izvajanje simulacij v orodju CellModeller na širokem naboru arhitektur CPE in GPE [43].

4 Agentno modeliranje večceličnih bioloških oscilatorjev

Oscilatorna dinamika predstavlja temelj mnogih bioloških sistemov. Oscilatorji regulirajo osnovne fiziološke procese, kot so cirkadiani ritmi, dihanje, izločanje inzulina in srčne funkcije [46]. Raziskave tovrstne dinamike na področju sintezne biologije so pripeljale do sintetičnih genskih vezij, ki izražajo oscilacije. Edend prvih uspešnih primerov sintetičnega oscilatorja je represilator [45]. To je gensko vezje, sestavljeno iz obroča oziroma zanke genov, v kateri vsak izraža represorski protein za naslednji gen v zanki. Preden se lotimo podrobnejšega pregleda represilatorja, bomo predstavili osnove gensko regulatornih omrežij. Njihova podrobna predstavitev sledi v naslednjem podpoglavju. Obravnavana orodja bomo demonstrirali na primerih agentnih modelov medcelične komunikacije in oscilatorne dinamike na nivoju gensko regulatornih omrežij. Ker vsako orodje temelji na svojem modelirnem jeziku, bomo pri vsakem opisali izvorno kodo, ki implementira ključno dinamiko modela.

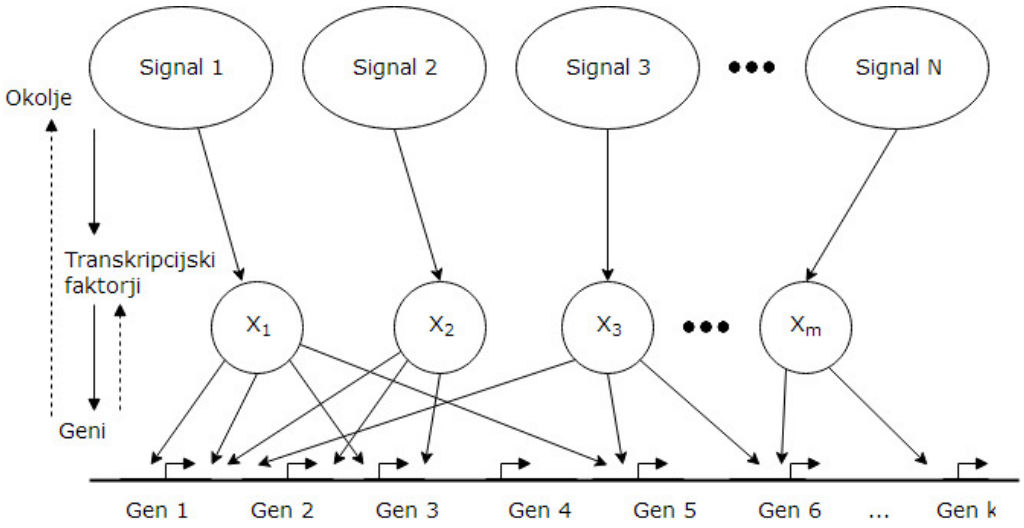
4.1 Gensko regulatorna omrežja

Sistemska in sintezna biologija v zadnjih letih usmerjata veliko pozornosti k razumevanju bioloških procesov na nivoju gensko regulatornih omrežij in k razvoju novih genskih vezij, ki implementirajo različne funkcije. Primeri so gensko regulatorna omrežja, ki implementirajo logične funkcije, pomnijo stanje, inducirajo oscilacije in izvajajo medcelično komunikacijo. V pregledu orodij za agentno modeliranje smo spoznali, da orodja, specializirana za modeliranje celičnih procesov, omogočajo opis dinamike gensko regulatornih omrežij znotraj posameznih agentov. Gensko regulatorno omrežje sestavlja skupina vzajemno delujočih genov ali delov genov, ki nadzirajo izvajanje specifične celične funkcije. Vse celične funkcije v grobem temeljijo na proteinih. Produkcija proteinov, ki jih potrebuje celica, se izvaja v procesu izražanja genov (angl. *gene expression*). Vsebovanost proteinov v celici je odvisna predvsem od hitrosti produkcije, ta pa je upravljana s posebnimi proteini. Kako proteini vplivajo na produkcijo drugih proteinov in kako poteka interakcija med geni, lahko predstavimo z gensko regulatornim oziroma transkripcijskim omrežjem [34].

Celico si lahko predstavljamo kot integrirano strukturo, sestavljeno iz nekaj tisoč tipov vzajemno delujočih proteinov. Protein je sistem velikostnega reda nanometra, ki opravlja določeno funkcijo z visoko natančnostjo [35]. Preprosta bakterija *E. coli* vsebuje nekaj milijonov proteinov okoli 4000 različnih tipov. Celice sodelujejo v različnih procesih, v katerih potrebujejo različne proteine. Primer takšnega procesa je odpravljanje posledic poškodbe celice, ki generira proteine, s katerimi se celica obnovi. Celica neprestano spremlja svoje okolje in uravnava količine potrebnih proteinov različnih tipov. Ta neprestani proces, s katerim se določa raven produkcije posameznega tipa proteinov, se v veliki meri odvija v gensko regulatornih omrežjih [35].

Celice se nahajajo v kompleksnem okolju, v katerem zaznavajo različne signale. Poleg signalov celice zaznavajo tudi fizične parametre, kot so temperatura, tlak in prisotnost koristnih hranil [35]. Prav tako je pomembna informacija o notranjem stanju celice z informacijami o poškodbah, npr. DNA, membrane ali proteinov in z informacijo o stanju metabolizma. Celice se na signale odzivajo s produkcijo ustreznih proteinov. Za predstavitev različnih stanj okolja celice uporabljajo posebne proteine, imenovane transkripcijski faktorji (angl. *transcription factors*). Transkripcijski faktorji so običajno zasnovani tako, da hitro prehajajo med aktivnim in neaktivnim molekularnim stanjem.

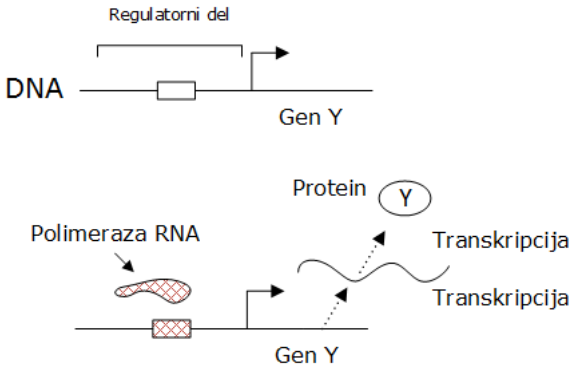
To upravljajo posebni signali iz okolja. Vsak aktiven transkripcijski faktor se lahko veže na DNA in tako regulira hitrost izražanja določenega gena. Proces izražanja gena je proces, v katerem se gen prepiše v informacijsko RNA ali mRNA (angl. *messenger RNA*), ki se nato prevede v protein. Proces je predstavljen tudi na sliki 4.1 [34, 35].



Slika 4.1 Preslikava med signali iz okolja, transkripcijskih faktorjev znotraj celice in geni, ki jih regulirajo. Signali iz okolja aktivirajo določene transkripcijske faktorje. Aktivirani transkripcijski faktorji se vežejo na DNA in regulirajo hitrost prepisovanja genov v mRNA oziroma hitrost transkripcije določenih genov. Generirani mRNA se nato prevede v protein [35].

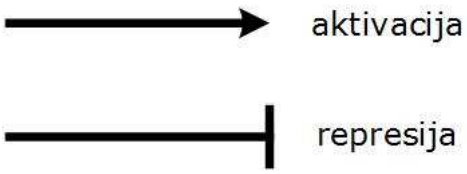
Interakcija med transkripcijskimi faktorji in geni je opisana s transkripcijskimi oziroma gensko regulatornimi omrežji. Glavne elemente omrežij predstavljajo geni in transkripcijski faktorji. DNA vsebuje celotni dedni zapis živega organizma, ki je sestavljen iz množice genov. Geni so nosilci posamezne značilnosti organizma in tako vsebujejo informacije, potrebne za proces produkcije proteinov. Transkripcija gena je proces, v katerem polimeraza RNA sestavi molekulo mRNA, ki ustreza kodnemu zapisu gena. Sintetizirana mRNA se nadalje prevede v protein [35]. Proces tvorjenja proteina je prikazan na sliki 4.2.

Računalniški pogled na proces regulacije ekspresije gena razkriva obstoj vhodnih podatkov v obliki transkripcijskih faktorjev in izhodnih podatkov v obliki izhodnih proteinov. Proces izražanja proteina je odvisen od transkripcijskih faktorjev in njihovih stanj. *Aktivatorski transkripcijski* (angl. *activators*) faktorji z vezavo na regulatorni del gena aktivirajo izražanje proteinov. *Represorski transkripcijski* (angl. *repressors*) fak-



Slika 4.2 Vsak gen je običajno sestavljen iz regulatornega dela in ekspresijskega dela. Regulatorni del vsebuje posebno sekvenco DNA, ki ob prisotnosti aktivatorjev oziroma odsotnosti represorjev privlači polimerazo RNA (RNAp). Polimeraza RNA je skupek proteinov, ki tvorijo encime za sintezo mRNA. Ta se veže na t. i. promotor gena. Proces sintetiziranja mRNA se imenuje transkripcija, prepis mRNA v ciljni protein pa translacija [35].

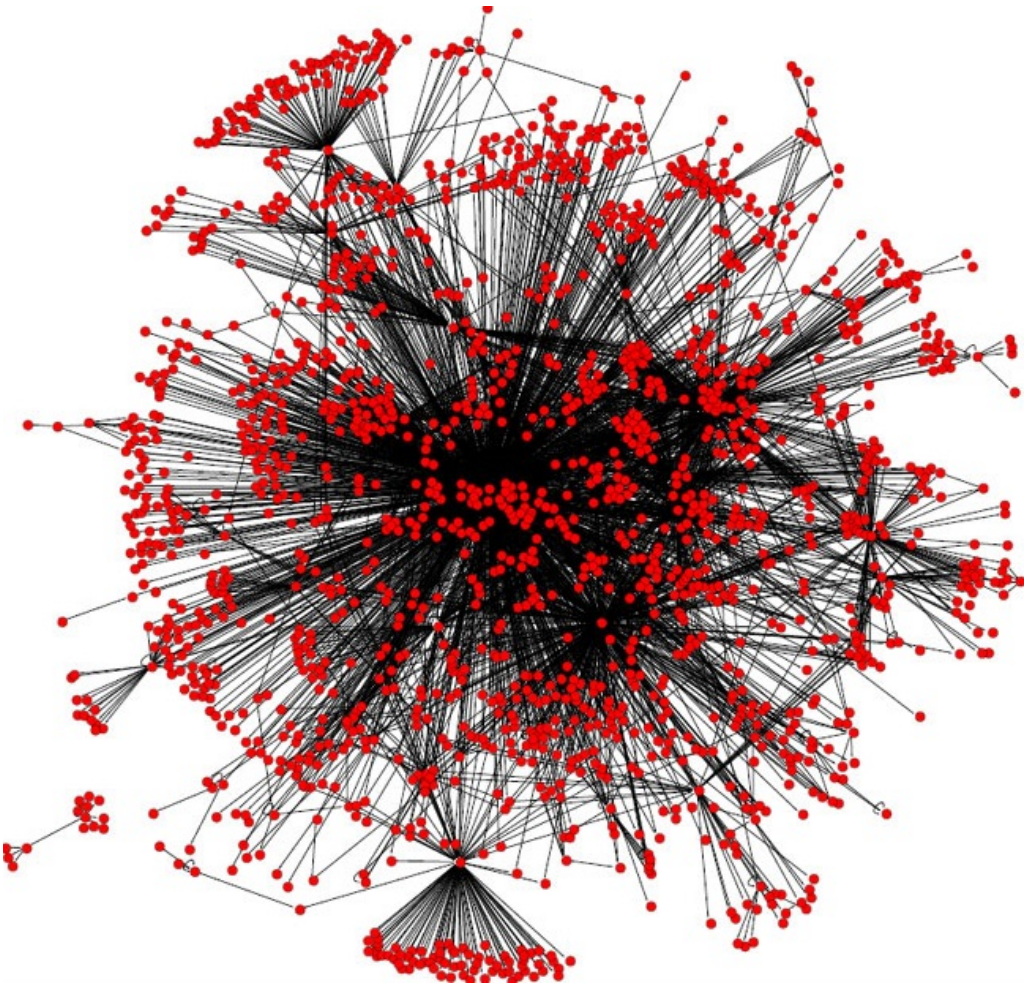
torji izražanje genov zavirajo [34]. Grafična upodobitev aktivacije z aktivatorjem oziroma represije z represorjem prikazuje slika 4.3.



Slika 4.3 Grafična upodobitev aktivacije oziroma represije genskega izražanja.

Tvorjeni proteini lahko nastopijo kot transkripcijski faktorji za drug gen. Izhodi teh genov imajo lahko vlogo transkripcijskih faktorjev za druge gene. Takšen način interakcije tvori gensko regulatorno omrežje. Celotno omrežje bakterije *E. coli* je prikazano na sliki 4.4. V omrežju vozlišča predstavljajo gene, povezave pa transkripcijsko regulacijo gena s proteinom, ki je nastal kot produkt drugega gena [35].

Najenostavnejši primer gensko regulatornega omrežja je en gen, ki regulira svojo aktivnost. V takšnem samoregulatornem genskem vezju proteini, ki so nastali v koraku translacije procesa izražanja gena, regulirajo izražanje gena kot aktivator ali kot represor. Kompleksnejši primeri genskih vezij so genska stikala (angl. *genetic switches*), oscilatorna genska omrežja, genska vezja, ki implementirajo medcelično komunikacijo, in genska vezja, ki implementirajo logična vrata [34].



Slika 4.4 Kompleksno transkripcijsko omrežje bakterije *E. coli*. V prikazani vizualizaciji vozlišča predstavljajo gene, povezave pa regulacijo gena z izhodom drugega gena [47].

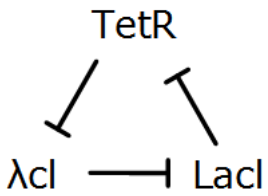
4.1.1 Represilator

Oscilatorna genska omrežja so genska vezja, ki generirajo obstojne oscilacije. Tovrstna omrežja celicam omogočajo vzdrževanje notranjih ur, s katerimi napovedujejo oziroma predvidijo periodične spremembe pogojev. Primer takšne spremembe je cikel med dnevom in nočjo oziroma cirkadiani ritem [46].

Represilator je sintetično gensko regulatorno omrežje, ki izraža oscilatorno dinamiko. Implementacija represilatorja v bakteriji *E. coli* je predstavljena v [45]. Gensko omrežje represilatorja periodično inducira sintezo zelenega fluorescenčnega proteina (angl. *green fluorescent protein, GFP*), ki se uporablja kot indikator stanja posameznih celic. Osci-

lacije se tako odražajo v koncentraciji izraženega GFP v celicah.

Topologija omrežja represilatorja je prikazana na sliki 4.5. Prvi represorski protein LacI zavira izražanje drugega represorskega proteina TetR, ki zavira izražanje proteina λ cI. Zanka se sklene z zaviranjem izražanja LacI s strani λ cI. Delovanje omrežja je odvisno od več dejavnikov, kot so hitrost transkripcije, hitrost translacije in hitrost razgradnje proteinov ter mRNA. Odvisnost od vrednosti naštetih dejavnikov oziroma parametrov vodi do ravnovesega ali oscilatornega stanja sistema, ki se odraža v spremenljivi dinamiki oscilacij z različnimi amplitudami, periodami in fazami [21, 34, 45].



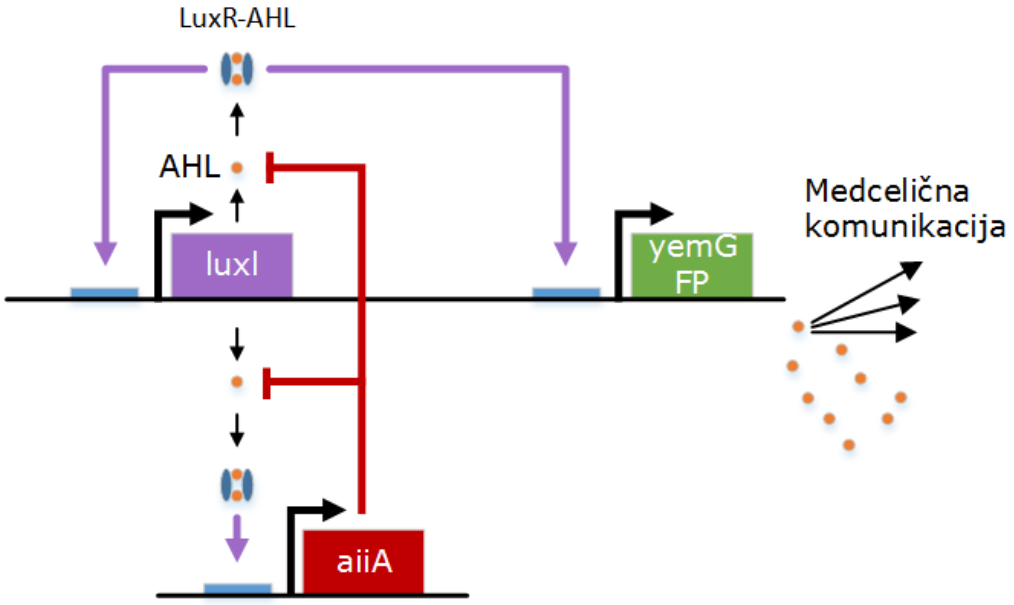
Slika 4.5 Gensko regulatorno omrežje represilatorja. Represilator je sestavljen iz treh genov, *TetR*, λ *cI* in *LacI*, povezanih v povratni zanki, v kateri vsak gen represira izražanje naslednjega gena v zanki [45].

4.1.2 Enostaven večcelični oscilator

Pri implementaciji represilatorja v celici *E. coli* so se osredotočili na opazovanje oscilacij znotraj ene celice. Variabilnost dinamike oscilacij z različnimi amplitudami in periodami je bila naslovljena pri implementaciji genskega vezja, ki omogoča induciranje trajnih oscilacij v populaciji celic [46]. Razvito gensko omrežje z medcelično komunikacijo generira sinhronizirane oscilacije v rastoči populaciji celic. Sklopljenost med variabilnimi celicami so avtorji dosegli z mehanizmom zaznavanja kvoruma (angl. *quorum sensing*) s signalno molekulo, ki prosto prehaja med celicami.

Gensko regulatorno omrežje oscilatorja z medcelično komunikacijo je prikazano na sliki 4.6. Zasnova sinhroniziranega oscilatorja temelji na elementih medcelične komunikacije mehanizma zaznavanja kvoruma v bakterijah *Vibrio fischeri* in *Bacillus Thuringiensis*. Omrežje sestavljajo gen *luxI* iz bakterije *V. fischeri*, gen *aiiA* iz bakterije *B. Thuringiensis* in gen *yemGFP*. Gen *luxI* tvori majhno molekulo AHL (angl. *acyl-homoserine lactone*), ki prehaja skozi celično membrano v okolje in tako prehaja med celicami. Znotraj celice se molekula veže na protein LuxR. Skupek LuxR-AHL služi kot transkripcijski aktivator za promotor *luxI*, ki je prisoten pri vseh treh genih: *luxI*,

aiiA in *yemGFP*. Izraženi proteini gena *aiiA* posredno represirajo promotorski del preko povečevanja hitrosti oziroma degradacije molekul AHL [46].



Slika 4.6 Gensko regulatorno omrežje sinhroniziranega oscilatorja. Oscilator je sestavljen iz treh genov: *lexI*, *aiiA* in *yemGFP*. Medcelična komunikacije poteka preko majhne molekule AHL, ki prehaja med celicami in aktivira promotor *luxI*. Promotor *luxI* je prisoten oziroma regulira aktivnosti ekspresije *lexI*, *aiiA* in *yemGFP* [46].

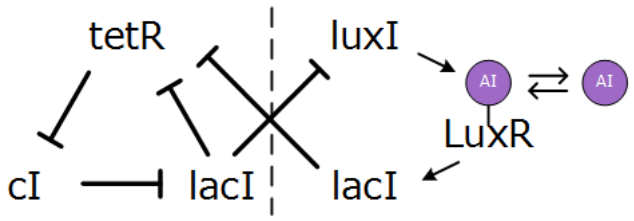
Eksperimenti s tovrstnim oscilatorjem so pokazali, da oscilatorna dinamika nastopi ob zadostni gostoti celic v populaciji. Majhno število individualnih celic ne proizvede dovolj povzročiteljev ali induktorjev za aktivacijo ekspresije preko promotorja *luxI*. Ko populacija celic doseže določeno gostoto, pride do ekspresije genov *LuxI*, *aiiA* in GFP. V tem času se kopiči aiiA, ki povzroči degradacijo AHL. Sčasoma se promotorji vrnejo v neaktivno stanje. Produkcija aiiA se upočasni in tako dovoli ponovni cikel kopičenja AHL in aktiviranja promotorjev [46].

4.1.3 Večcelični represilator

Implementacija represilatorja je pokazala, da individualne celice izražajo nestabilno dinamiko oscilacij. Rezultati so pokazali razlike v periodi oscilacij med različnimi celicami, kot tudi spremenljivo periodo osciliranja znotraj ene celice. Populacijo represilatorjev so uspešno stabilizirali in sinhronizirali z aplikacijo medcelične komunikacije. Interkacija represilatorjev preko mehanizma zaznavanja kvoruma ima za posledico sinhronizirano

osciliranje sistema [48].

Nadgradnja modela represilatorja z medcelično komunikacijo je prikazana na sliki 4.7. Represilator je sestavljen iz treh genov, katerih izraženi proteini zavirajo izražanje naslednjega gena v ciklu. Medcelična komunikacija temelji na dveh proteinih. Prvi protein LuxI tvori majhno molekulo AI, ki lahko prehaja skozi celično membrano. Ko se drugi protein LuxR veže na molekulo AI, skupaj tvorita kompleks LuxR-AI, ki aktivira izražanje gena *lacI*. Model represilatorja je razširjen tudi z dodatnim genom *lacI*, tako da je njegovo izražanje inducirano z LuxR-AI in genom, ki izraža protein LuxI pod regulacijo represorskega proteina LacI [48].



Slika 4.7 Gensko regulatorno omrežje represilatorja, razširjeno z modulom za medcelično komunikacijo. Omrežje represilatorja je predstavljeno na levi strani črtkane črte, modul za medcelično komunikacijo pa je na desni strani navpične črtkane črte [48].

Za modeliranje izražanja genov v populaciji celic potrebujemo informacijo o koncentraciji mRNA in koncentraciji proteinov v posamezni celici. Dinamika mRNA je regulirana z degradacijo in represijo vseh genov represilatorja skupaj z genom *lacI*, ki predstavlja modul za sinhronizacijo. Dinamika mRNA v celici i je opisana z enačbo 4.1. Parameter a_i predstavlja koncentracijo mRNA, parameter C_i pa koncentracijo proteinov v celici. Zaporedje indeksov (v enačbi 4.1 a in C) označuje vrstni red reguliranja genov ($i \in \{1, 2, 3\}$). Črka a oziroma A ustreza genu *tetR*, črka C pa genu *lacI*, ki regulira gen *tetR*. Parameter n predstavlja Hillov koeficient, parameter α pa predstavlja hitrost transkripcije.

$$\frac{da_i}{dt} = -a_i + \frac{\alpha}{1 + C_i^n} \tag{4.1}$$

Dinamika proteinov je opisana z enačbo 4.2. Parameter β je definiran z razmerjem življenjske dobe mRNA in proteinov.

$$\frac{dA_i}{dt} = \beta(a_i - A_i) \tag{4.2}$$

Poleg dinamike mRNA in proteinov je v modelu opisana tudi dinamika molekul za medcelično komunikacijo. Koncentracija slednje kemijske zvrsti je v celici odvisna od njene hitrosti razgradnje in sinteze ter difuzije v medceličnem mediju. Sprememba koncentracije signalnih molekul znotraj posamezne celice je opisana z enačbo 4.3. Parameter S_I predstavlja koncentracijo signalne kemijske zvrsti znotraj posamezne celice, η je hitrost difuzije signalne kemijske zvrsti v celični membrani, parameter S_E pa predstavlja koncentracijo signalnih molekul v okolju. Dinamika je odvisna še od parametra k_{s0}, ki predstavlja hitrost degradacije signalne kemijske zvrsti znotraj celice in parametra k_{s1}, ki predstavlja hitrost sinteze signalnih molekul v odvisnosti od izraženega gena *luxI*.

$$\frac{dS_I}{dt} = -k_{s0}S_I + k_{s1}A_i - \eta(S_I - S_E) \tag{4.3}$$

V nadaljevanju sledijo opisi implementacij agentnih modelov večceličnih bioloških oscilatorjev v orodjih NetLogo, BSim, Gro in CellModeller, ki smo jih podrobno preučili v poglavju 3.

4.2 Testiranje in rezultati

Orodja bomo testirali na modelu večceličnih oscilatorjev. Pri tem bomo izhajali iz modelov, ki jih orodja vsebujejo v svoji zbirki vzorčnih primerov. Naknadno bomo v izbranih orodjih izvedli implementacijo večceličnega represilatorja, predstavljenega v 4.1.3.

Orodje NetLogo bomo v prvi fazi testirali na modelu *Quorum_Sensing* [49]. V modelu je implementiran mehanizem zaznavanja kvoruma za medcelično komunikacijo [50]. Dinamika agentov opisuje biološki proces bioluminiscence, pri katerem gre za oddajanje svetlobe pri živih bitjih. Celice ali bakterije med seboj komunicirajo s signalnimi molekulami. V orodju NetLogo so agenti preproste entitete brez bioloških lastnosti celic, kot je volumen, koncentracija kemijske zvrsti in implementirane dinamike, ki je značilna za celice. Model je postavljen v 2D prostoru. Orodje ni specializirano za modeliranje celic, zato nima implementirane celične dinamike in celičnih procesov, kot jih imajo BSim, Gro in CellModeller. Orodje nima definiranega tipa celice s pripadajočimi lastnostmi, zato moramo razred oziroma tip in dinamiko celičnih procesov definirati sami. Posledica tega je, da simulacija ne posnema realnega gibanja oziroma se težko približamo realnemu gibanju celic po prostoru. V drugi fazi bomo obstoječi model razširili z implementacijo večceličnega represilatorja.

Programsko orodje BSim bomo testirali na modelu sinhroniziranega oscilatorja in večceličnega represilatorja. Orodje v svoji distribuciji vključuje primer modela oscilatorja *BSimQuorumOscillator* in model večceličnega represilatorja *BSimQuorumRepressilator* [30]. Celice oziroma agenti so modelirani v s tekočino napolnjenem 3D okolju. Mehanizem zaznavanja kvoruma, s katerim dosežemo sinhronizacijo oscilacij agentov, je implementiran s koncentracijo kemijske zvrsti v simulacijskem okolju. Dinamika kemijske zvrsti je pogojena s hitrostjo difuzije in hitrostjo razgradnje. Gibanje celice v prostoru je simulirano z modelom, ki je implementiran na osnovi realnega gibanja celic v tekočini. Agent je predstavljen z objektom definiranega javanskega razreda z lastnostmi celice, kot so npr. pozicija, polmer in sile, ki delujejo na celice v prostoru. Agent je v okolju predstavljen s kroglo polmera 1 μm v zveznem 3D okolju. Lastnosti prostora so nastavljive s parametrom viskoznosti *visc*, ki privzeto znaša 0,0027 $Pa \cdot s$, in temperature *temperature*, ki je privzeto nastavljena na 305 K. Proces celične rasti in celičnega podvojevanja ni modeliran, zato je število agentov v okolju med simulacijo enako.

Orodje Gro bomo demonstrirali na modelu oscilatorja *coupled_oscillator* [39] in modelu večceličnega represilatorja. V osnovi orodje vključuje implementacijo večceličnega oscilatorja, pri katerem je gibanje celic ali agentov omejeno. V naši implementaciji rasti kolonije in gibanja celic ne omejujemo. Oscilacije v agentu se odražajo s koncentracijo GFP. Sinhronizacija med celicami je implementirana z medceličnimi signali. Orodje omogoča modeliranje v 2D prostoru. Orodje je osredotočeno na modeliranje celic, ki temeljijo na lastnostih bakterije *E. coli*. Začetni volumen celice v modelu je 1,57 fL. V modelu je simuliran proces celične rasti in proces delitve celic. Celična rast je določena s parametrom hitrosti rasti *ecoli_growth_rate*, delitev pa je odvisna od parametra povprečna vrednost μ in varianca σ^2 velikosti delitve. Vrednosti parametrov sta $\mu = 3{,}14$ fL in $\sigma^2 = 0{,}005\ fL^2$. Simulacijo začnemo z enim agentom, ki pa se z opisanim procesom delitve celic razdeli na dva agenta. Simulacijo oscilatorja končamo, ko populacija šteje 1000 agentov. Model oscilatorja bomo spremenili v model represilatorja z implementacijo dinamike mRNA in proteinov, opisanih v 4.1.3.

Orodja CellModeller nismo uspeli zagnati, zato bomo v pričujočem poglavju opisali samo implementacijo enostavnega oscilatorja, ki je vključen v osnovni distribuciji orodja. Model demonstrira uporabo orodja za simulacijo dinamike celic in genov, katerih dinamika je opisana s sistemom diferencialnih enačb. Za reševanje sistemov diferencialnih enačb se uporablja modul *CLEulerIntegrator*.

Celotna koda, ki smo jo uporabili za testiranje orodij, je na voljo na naslovu http://lrss.fri.uni-lj.si/bio/material/2017_gazvoda_msc.zip.

4.2.1 NetLogo

Testiranje orodja smo začeli z modelom *Quorum_Sensing*. Izvorna koda 4.1 prikazuje deklaracijo tipa agenta z definiranimi lastnostmi. To storimo s ključno besedo *cell-own*. Privzeto agentje do teh lastnosti še ne morejo dostopati, saj jih agentom ali določenemu tipu agentov še nismo definirali. Kateri tip ali *breed* agentov dostopa do definiranih uporabniških lastnosti, določimo z ukazom *set breed*. V pričujočem primeru se lastnosti celice *cell* priredijo s klicem ukaza *set breed cell*, kot je prikazano v proceduri *to cel* v kodi 4.1.

Koda 4.1 Definiranje lastnosti celice in procedura *to cel*, v kateri se agentom določijo privzeta oblika, barva, tip ali *breed*, velikost in druge lastnosti, definirane v *cell-own*. Z ukazom *set breed cell* se agentu določijo tudi lastnosti v *cell-own* [49].

```
cell-own
[
  Energy
  LuxR
  LuxI
  Complex
  duplic_time
]
to cel
  set-default-shape cell "circle 2"
  crt 1 [ setxy random-pxcor random-pycor
          set color white
          set breed cell
          set Energy 1000
          set duplic_time 30 + random 30
          set LuxR 1
          set LuxI 1
          set size 2
        ]
end
```

Začetno stanje simulacije inicializiramo v proceduri *to setup*, v kateri ponastavimo števec simulacijskih korakov, vizualizacijo simulacije in kreiramo agente. Pri inicializaciji agentov kličemo proceduro *to cel*, predstavljeno v kodi 4.1.

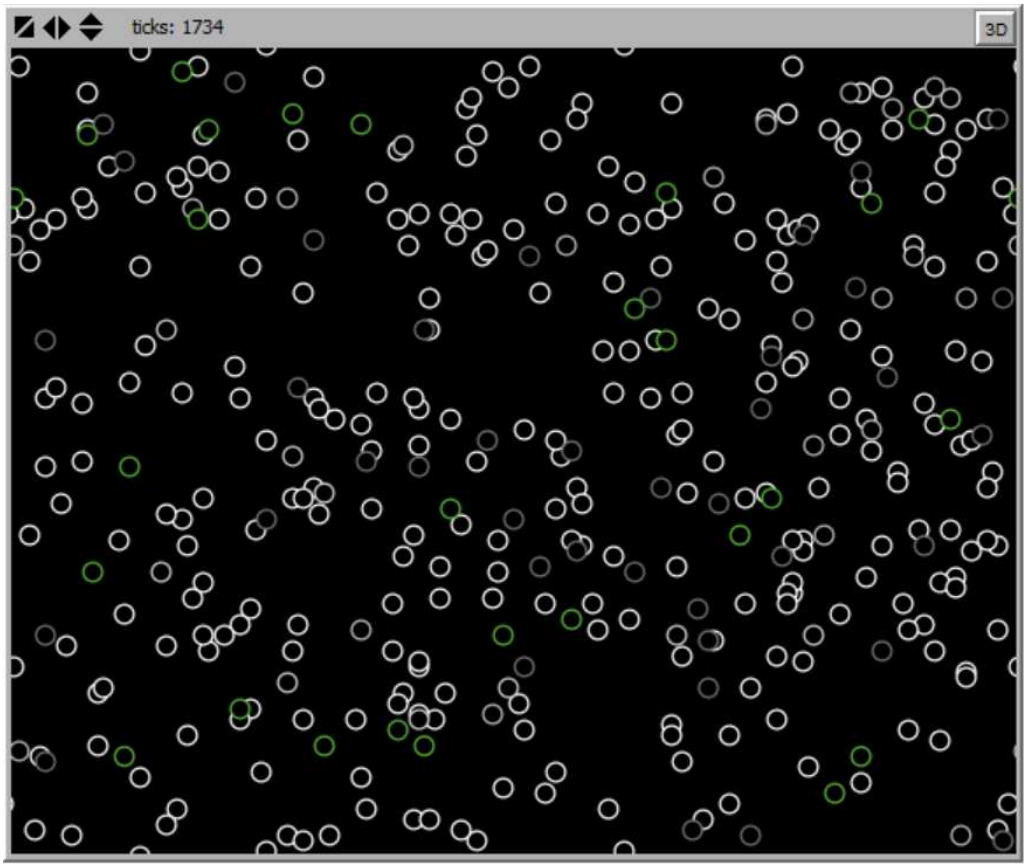
Dinamika agentov in posodobitev vizualizacije se izvede v proceduri *to go*, ki se kliče v vsakem simulacijskem koraku. Na začetku procedure se kliče procedura, ki posodobi pozicijo agentov v prostoru. Druga orodja, ki so specializirana na celično modeliranje, privzeto vsebujejo implementacijo določenega celičnega gibanja, kot je na primer Brownowo gibanje. V pričujočem modelu je gibanje agenta implementirano s kodo *NetLogoMove*. Vsak agent preveri svojo okolico. Izmed sosednjih osmih polij se izberejo naključna in se preveri, ali je na tem polju že kakšen agent. Če je polje prazno, se agent premakne na izbrano polje, v nasprotnem primeru se agent v tem simulacijskem koraku ne premakne.

Koda 4.2 V proceduri *move* se pridobi seznam koordinat sosednjih polj agenta. Izmed osmih sosednjih polj se naključno izbere enega. Agent se na izbrano polje premakne, če je prazno [49].

```
to move
 let ne [ list pxcor pycor ] of neighbors
 let value one-of ne
 if ( not any? turtles-on patch ( item 0 value ) ( item 1 value ) )
  [ move-to patch ( item 0 value ) ( item 1 value ) ]
end
```

V definiciji agenta smo med drugimi podali spremenljivke *LuxI*, *LuxR* in *Complex*. Spremenljivki *LuxI* in *LuxR* hranita koncentracijo izraženih proteinov, ki regulirata bioluminiscenco. Spremenljivka *Complex* hrani koncentracijo označevalca v agentu. Vsak agent vsebuje spremenljivko *duplic_time*, katere vrednost predstavlja čas, ko pride do podvojitve celice. Podvojitev celice se implementira s klicem ukaza *hatch-cell*, kateremu podamo število novo nastalih agentov in lastnosti agenta. Ukaz za prvi argument prejme število ena, saj pri podvojevanju nastane samo en nov agent.

Dinamika agentov je odvisna od koncentracije signalnih molekul. Ob nizki koncentraciji se hitrost produkcije signalnih molekul in hitrost izražanja genov *LuxR* in *LuxI* razlikujeta od hitrosti produkcije ter izražanja pri višjih koncentracijah signalnih molekul. Koncentracija označevalca v agentu se povečuje le ob visokih koncentracijah signalnih molekul. Simulacija opisanega modela je prikazana na sliki 4.8. Ob visoki koncentraciji signalnih molekul agenti oziroma celice začnejo oddajati svetlobo. To je prikazano z različno intenzivnostjo bele barve agentov.

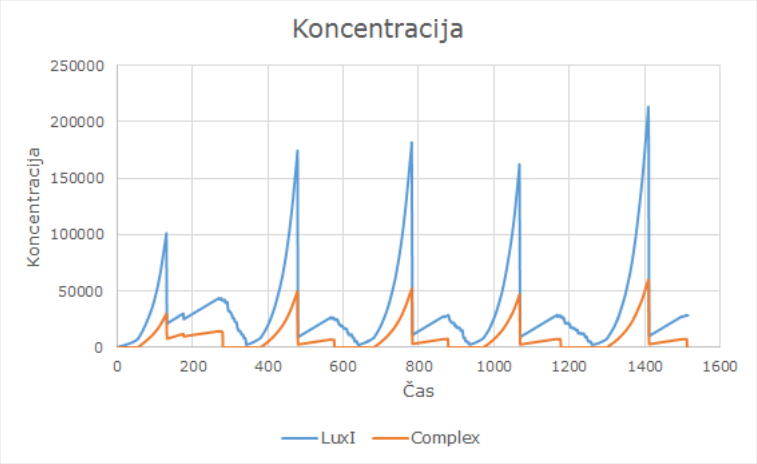


Slika 4.8 Vizualizacija simulacije modela *Quorum_Sensing*. Zeleni agenti ne vsebujejo označevalca, drugi agenti imajo različno intenzivnost bele barve, ki odražajo koncentracijo označevalca.

Koncentracija izraženega gena *LuxI* in koncentracija označevalca *Complex* sta prikazani na grafu 4.9. Graf prikazuje vsoto koncentracije vseh agentov v okolju. Os x označuje simulacijske korake, os y pa koncentracijo kemijske zvrsti. Pri visoki koncentraciji signalnih molekul se začne hitrejše izražanje gena *LuxI* in povečevanje označevalca v agentih. Populacija agentov v simulaciji je omejena navzgor. Ko pride do prekoračitve tega praga, se sproži uničenje določenih agentov, ki povzročijo padce koncentracije na grafu 4.9.

Oscilatorno dinamiko smo implementirali z razširitvijo modela *Quorum_Sensing*. Pri vzpostavitvi oscilatorja smo izhajali iz gensko regulatornega omrežja večceličnega represilatorja, opisanega v 4.1.3. Dinamiko agenta smo opisali s sistemom enačb, ki opisujejo dinamiko gensko regulatornega omrežja.

Notranje stanje agenta je definirano s koncentracijo mRNA, proteinov in kemijske



Slika 4.9 Graf koncentracije izraženega gena *LuxI* in koncentracija označevalca *Complex* med simulacijo. Graf prikazuje vsoto koncentracije vseh agentov. Na osi x so prikazane časovne enote simulacije, na osi y pa koncentracija kemijske zvrsti.

zvrsti za medcelično komunikacijo. Agentovo stanje je implementirano v kodi 4.3.

Koda 4.3 Vsak agent v modelu je definiran s koncentracijo mRNA, proteinov in kemijsko zvrstjo za medcelično komunikacijo.

```
cell-own
[
  ;;koncentracija mRNA genov tetR, cI in lacI
  tetR_mRNA
  cI_mRNA
  lacI_mRNA
  ;;koncentracija proteinov genov tetR, cI in lacI
  tetR_Protein
  cI_Protein
  lacI_Protein
  AHL
]
```

Na začetku simulacije stanje agenta inicializiramo z naključnimi vrednostmi koncentracij. Dinamika agenta je implementirana v proceduri *to go*, ki se izvede v vsakem simulacijskem koraku. Koda 4.4 prikazuje implementacijo dinamike vsakega agenta. V kodi je prikazana dinamika mRNA in proteina gena *tetR* ter dinamika kemijske zvrsti za medcelično komunikacijo AI oziroma AHL. Dinamika za druge gene je definirana na enak način. Orodje NetLogo privzeto ne vsebuje metod za reševanje diferencialnih enačb, zato

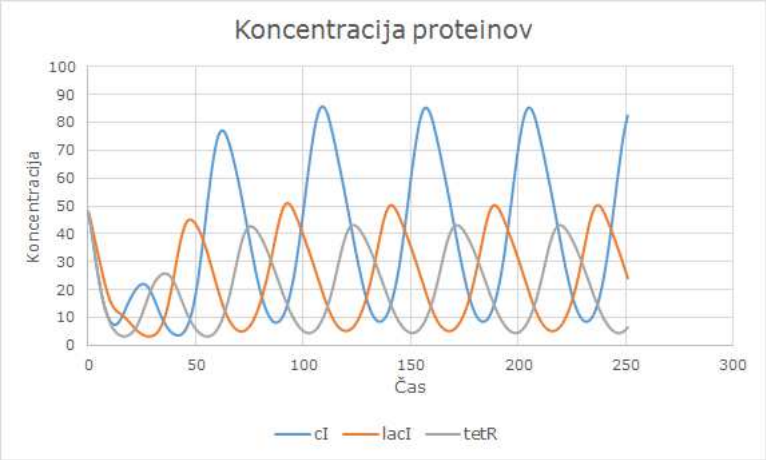
smo dinamiko implementirali z Eulerjevo metodo reševanja diferencialnih enačb. Pri tem smo uporabili časovni korak dt $0,01$.

Koda 4.4 Implementacija dinamike mRNA in proteina gena *tetR* ter kemijske zvrsti AHL.

```
ask cell
    [
      let tetR_m tetR_mRNA
      let tetR_P tetR_Protein
      set tetR_mRNA tetR_mRNA + dt * ((-1 * tetR_mRNA) +
      (alpha / (1 + lacI_Protein ^ n)))

      set tetR_Protein tetR_Protein +
      dt * (beta * (tetR_m - tetR_Protein))

      set AHL AHL + dt * ( - k_s0 * AHL + k_s1 * tetR_P -
      eta * (AHL - AHL_external))
    ]
```



Slika 4.10 Graf koncentracije proteinov cI, lacI in tetR za 200 agentov. Na osi x je prikazan čas, na osi y pa povprečna koncentracija proteinov. Parametri modela so reskalirani, tako da so koncentracije in čas podani brez enot.

V simulaciji smo uporabili vrednosti parametrov $\alpha = 216$, $\beta = 1$, $n = 2$, $k_{s0} = 1$, $k_{s1} = 0,01$ in $\eta = 2$ kot v delu [48]. Pri navedenih parametrih smo dobili oscilacije, prikazane na sliki 4.10. Graf prikazuje povprečno koncentracijo proteinov cI, lacI in tetR za vse agente v populaciji. Simuliranih je bilo 250 časovnih enot s populacijo 200 agentov.

4.2.2 BSim

Model večceličnega oscilatorja predstavlja agentni model z enostavnim opisom agentove dinamike. V modelu je demonstrirana medcelična komunikacija z mehanizom zaznavanja kvoruma, predstavljenega v 4.1.2. Kemijska zvrst v okolju je implementirana z razredom *BSimChemicalField*. Ta razdeli prostor v manjše razdelke z vsebovano koncentracijo kemijske zvrsti, ki z določeno hitrostjo razpada in prehaja v sosednje razdelke z določeno hitrostjo difuzije. Hitrost difuzije in hitrost razpada sta nastavljiva parametra, ki se inicializirata na začetku simulacije skupaj z drugimi parametri simulacije. Inicializacijo simulacijskih parametrov prikazuje koda 4.5.

Koda 4.5 Koda prikazuje nastavitev parametrov simulacije. Kreiramo objekt tipa *BSim* in nastavimo hitrost difuzije ter razpad kemijske zvrsti, hitrost produkcije, zakasnitev produkcije, prag kemijske zvrsti in objekt *BSimChemicalField* [44].

```
BSim sim = new BSim();


final double diffus = 600;
final double decay = 0.05;
final double prodRate = 1e9;
final double prodDelay = 1;
final double threshold = 1e4;
final BSimChemicalField field =
new BSimChemicalField(sim, new int[]{10,10,10}, diffus, decay);
```

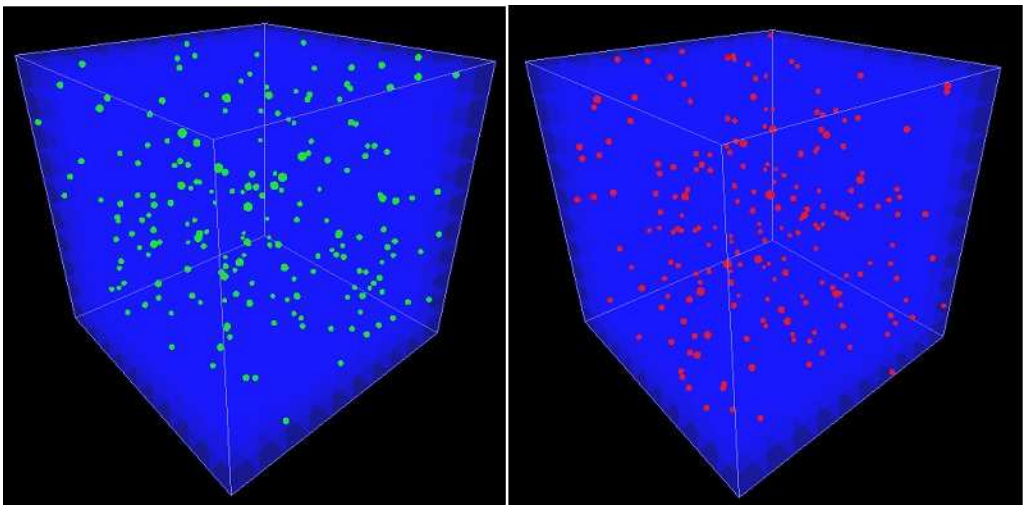
V naslednjem koraku implementiramo bakterijo ali agenta. BSim vsebuje definirane razrede z implementacijo različnih dinamik gibanja agentov. Primer takšnega razreda je *BSimBacterium*. V definiranem razredu definiramo metodo *action*, v kateri določimo pogoje aktivnega in neaktivnega stanja agenta. V aktivno stanje agent preide, če je na razdelku s koncentracijo kemijske zvrsti, ki je višja od definirane s spremenljivko *threshold*. V nasprotnem primeru agent ostane v neaktivnem stanju in v okolje sprosti kemijsko zvrst z določeno hitrostjo produkcije. Opisana funkcionalnost je implementirana v kodi 4.6.

Koda 4.6 Koda prikazuje metodo *action*, v kateri lahko definiramo poljubno dinamiko. V metodi se preveri, ali je v razdelku na poziciji agenta koncentracija višja od definirane s spremenljivko *threshold*. V primeru izpolnjenosti pogoja se agent označi kot aktiven, v nasprotnem primeru pa se v okolje sprosti kemijska zvrst [44].

```
public void action() {
 super.action();
```

```
 if (field.getConc(position) > threshold) {
  activated = true;
  lastActivated = sim.getTime();
 } else {
  activated = false;
  if (lastActivated == -1 ||
  (sim.getTime() - lastActivated) > prodDelay) {
   //sprostitev kemijske zvrsti
   field.addQuantity(position, prodRate * sim.getDt());
  }
 }
}
```

Agente na začetku simulacije postavimo na naključne pozicije v prostoru. Dinamika simulacije se posodablja z metodo *setTicker*, v kateri kličemo metode *action*, *updatePosition* za posodobitev pozicije agentov in metodo *update* za posodobitev koncentracij kemijske zvrsti v prostoru. Simulacija z 200 agenti je prikazana na sliki 4.11. Slika prikazuje agente v neaktivnem in aktivnem stanju. Vizualizacijo simulacije modela omogoča razred *BSimP3DDrawer*.



Slika 4.11 Simulacija modela sinhroniziranega oscilatorja. Levi del slike prikazuje agente v neaktivnem stanju. Desni del slike prikazuje agente v aktivnem stanju. Agent postane aktiven, če koncentracija flourescenta preseže definirani prag.

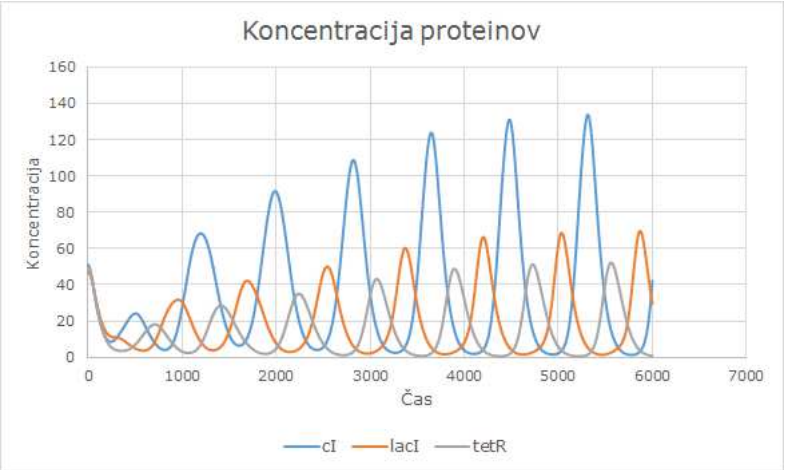
Če simulacijo zaženemo z majhnim številom agentov in pri tem ne spreminjamo dru-

gih parametrov, kot sta hitrost produkcije ali hitrost difuzije, agentje svojega stanja ne spreminjajo sinhronizirano. Pri večjem številu agentov v prostoru ti svoje stanje spreminjajo enotno. Takšno vedenje je posledica visoke koncentracije kemijske zvrsti v prostoru. Na oscilatorno dinamiko vplivamo tudi s parametri hitrosti difuzije, hitrosti degradacije, hitrosti produkcije kemijske zvrsti in praga koncentracije, pri kateri agent spremeni stanje. S hitrostjo difuzije in degradacije vplivamo na širitev kemijske zvrsti v prostoru, hitrost produkcije pa določa, kakšna koncentracija kemijske zvrsti se sprošča v okolje.

BSim vključuje tudi implementacijo kompleksnejšega modela *BSimCoupledRepressilators* sklopljenih represilatorjev z mehanizmom zaznavanja kvoruma. Večcelični represilator je opisan v 4.1.3 in shematsko predstavljen na sliki 4.7. V modelu je demonstrirana funkcionalnost opisa agentove dinamike z diferencialnimi enačbami, ki se v vsakem simulacijskem koraku integrirajo s temu namenjenimi funkcijami. V pričujočem modelu se za integracijo uporablja metoda *BSimOdeSolver.rungeKutta45*, ki implementira numerično metodo Runge-Kutta. Gensko regulatorno omrežje represilatorja je implementirano z razširitvijo razreda *BSimBacterium*. Sistem diferencialnih enačb, ki temeljijo na delu [48], je implementiran z vmesnikom *BSimOdeSystem*. Represilatorji se med sabo sinhronizirajo s pomočjo signalnih molekul. Te so v modelu implementirane enako kot pri modelu enostavnega oscilatorja *BSimQuorumOscillator* z razredom *BSimChemicalField*.

Vsak agent vsebuje implementacijo represilatorja. Represilator je opisan s sistemom diferencialnih enačb, predstavljenih v 4.1.3 in [48]. Sistem sestavlja sedem enačb, ki opisujejo dinamiko mRNA, proteinov in molekul za medcelično komunikacijo znotraj enega agenta. Začetni pogoji za sistem diferencialnih enačb so določeni naključno. Sistem enačb se reši v vsakem koraku simulacije za časovni korak *dt*. Ta je privzeto in v opisanem primeru nastavljen na $0,01$. Represilator je definiran v razredu, ki implementira vmesnik *BSimOdeSystem* za definiranje sistemov diferencialnih enačb.

Oscilacije znotraj agentov so prikazane na grafu 4.12. Graf 4.12 prikazuje povprečno koncentracijo proteinov cI, lacI in tetR za vse agente v populaciji. Graf prikazuje oscilacije 200 agentov za simuliranih 6000 sekund. V simulaciji smo uporabili vrednosti parametrov $\alpha = 216$, $\beta = 1$, $n = 2$, $k_{s0} = 1$, $k_{s1} = 0,01$ in $\eta = 2$, tako kot pri testiranju orodja NetLogo. Parametri so reskalirani, tako da so čas in koncentracije opazovanih kemijskih zvrsti podani brez enot.



Slika 4.12 Koncentracija proteinov cI, lacI in tetR za 200 agentov v 6000 časovnih enotah. Povprečna koncentracija proteinov je prikazana na osi y, časovni koraki pa na osi x.

4.2.3 Gro

Simulacijo modela začnemo z enim agentom, ki predstavlja celico *E. coli*. Inicializacija agenta z lastnostmi celice *E. coli* je prikazana v kodi 4.7. Agent je inicializiran s pozicijo (0,0) v 2D prostoru. Dinamika agenta je opisana s pravili, definiranimi v programu *oscillator*. Agent je poleg pozicije inicializiran tudi s programom, v katerem je opisana oscilatorna dinamika.

Koda 4.7 Inicializacija agenta z pozicijo (0,0) v 2D prostoru s programom *oscillator*. Program *oscillator* prejme argument, ki se v nadaljevanju uporabi za določitev koncentracije zelenega fluorescenčnega proteina *gfp*. [39].

```
ecoli ( [x:=0, y:=0], program oscillator(0));
```

Fluorescenčni označevalec v agentu inicializiramo v programu *oscillator* s spremenljivko *gfp*. Koncentracija slednjega se posodobi v vsakem koraku simulacije. Kako hitro nastaja in kako hitro se GFP v agentu razgradi je odvisno od koncentracije signala s, volumna celice in parametra $p.x$. V vsakem koraku se parameter $p.x$ evalvira z določeno verjetnostjo. Ta verjetnost je izračunana z ukazom *rate()* in je odvisna od koncentracije signala s v agentu. Opisani koraki so prikazani v izvorni kodi 4.8.

Koda 4.8 Inicializacija agenta z zelenim fluorescenčnim proteinom. Program *oscillator* prejme argument *g0*, ki se uporabi pri izračunu koncentracije GFP v agentu. V spodnji kodi opazimo tudi primer zapisa *record* in varovane ukaze. Zapis *p* vsebuje definirane pomožne parametre za kontrolo pošiljanja signalov. Spremenljivka *GO* se uporablja za kontrolo posodobitve *p.x* in *p.mode*, spremenljivka *p.mode* pa za kontrolo toka programa, v katerem ponastavimo *p.x* in sprostimo signal v okolje [39].

```
program oscillator(g0) := {
```

```
gfp := 0.5*volume*g0;

//record
p := [mode := G0, t := 0, x := g0];
//posodobi koncentracijo gfp
true : {gfp := 0.5*volume * p.x}

//pogojno posodobi p.x
p.mode = G0 & rate (k0) : {p.x := p.x+0.01*(150-p.x)}
p.mode = G0 & rate(kb*get_signal(s)) :
        {p.x := p.x+0.01*(150-p.x)}
```

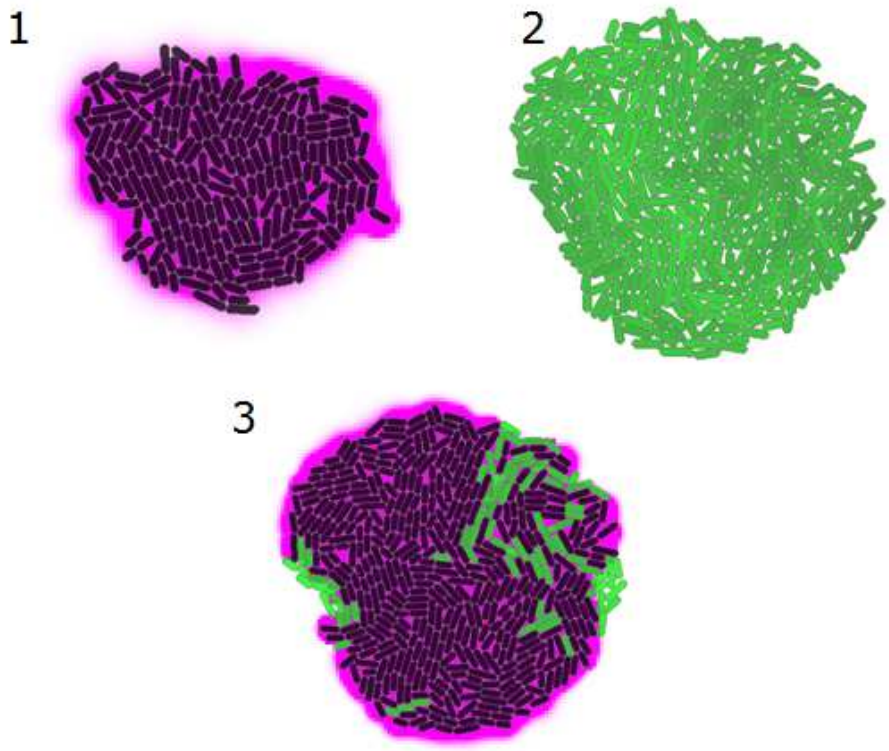
Koncentracijo signala preberemo s klicem funkcije *get_signal(s)*, ki kot parameter prejme enolični identifikator signala, dodeljen ob inicializaciji signala. Agent sprosti signal v okolje ob klicu funkcije *emit_signal(s, se)*, v kateri povemo vrsto signala in koncentracijo, ki jo agent na svoji poziciji sprosti v okolje. Primer sprostitve signala v okolje je prikazan v kodi *groProgramSignal*.

Koda 4.9 Sprostitev signala v okolje s klicem funkcije *emit_signal(s,se)* [39].

```
p.x > 100 : {
    p.x := 0,
    emit_signal(s, se)
}
```

Simulacija se začne z enim agentom. Med simulacijo se celice delijo in kolonija posledično eksponentno raste. Po delitvi celice matična celica in nova hčerinska celica vsebujeta enak program z opisom agentove dinamike. Na sliki 4.13 je prikazan potek simulacije opisanega večceličnega oscilatorja. Na sliki so prikazani trije posnetki simulacije. Prva kolonija šteje 225 agentov. Vijolična barva prikazuje koncentracijo sproščenega signala v okolje. Sčasoma se produkcija fluorescenčnega označevalca poviša in pride do stanja, ki ga prikazuje kolonija 2 s 400 agenti. Ko agentje presežejo določen prag produkcije *gfp*, se označevalec začne degradirati in agentje začnejo sproščati signal, kot je prikazano v koloniji 3, ki šteje 489 agentov. Opisan način osciliranja se tako ponavlja med neprestano rastjo celic.

Implementacije represilatorja smo se lotili s spremembo opisa oscilatorne dinamike. Osilatorna dinamika gensko regulatornega omrežja represilatorja, predstavljenega v 4.1.3,



Slika 4.13 Vizualizacija modela sinhroniziranega oscilatorja. Slika prikazuje posnetek kolonije v procesu oddajanja signala (1), produkcije zelenega fluorescenčnega proteina (2) in degradacije fluorescenčnega označevalca ter ponovnega sproščanja signalnih molekul v okolje (3).

je implementirana z varovanimi ukazi, ki se evalvirajo v vsakem koraku implementacije s časovnim korakom $0,01$. Stanje agenta smo definirali v programu, ki ga prejme celica ob njeni inicializaciji, z definicijo zapisa *rep*. Definicija zapisa je prikazana v kodi 4.10, v kateri koncentracijo mRNA, proteinov in AHL inicializiramo naključno.

Koda 4.10 Definiranje stanja celice oziroma agenta z inicializacijo zapisa *rep*. Koncentracije mRNA, proteinov in AHL se generira naključno.

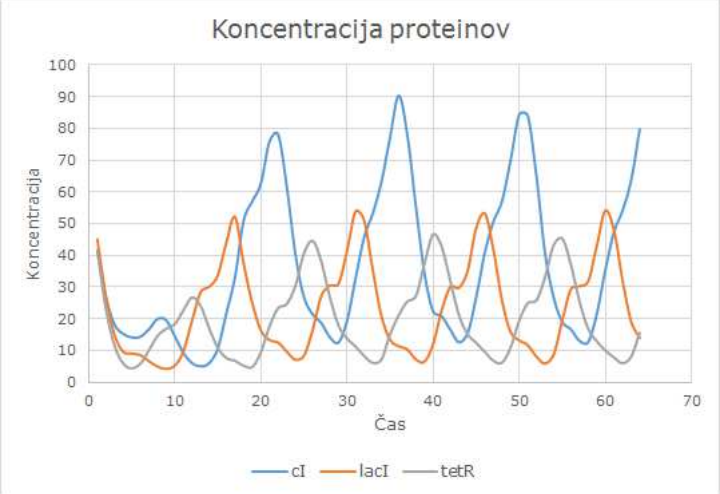
```
rep := [tetR_mRNA := rand(100), cI_mRNA := rand(100),
lacI_mRNA := rand(100), tetR_Protein := rand(100),
cI_Protein:= rand(100), lacI_Protein := rand(100),
AHL := rand(10) * 0.5];
```

Varovani ukaz za posodobitev koncentracije mRNA gena *tetR* je prikazan v kodi 4.11. Koncentracija drugih mRNA, proteinov in AHL je implementirana na enak način.

Koda 4.11 Varovani ukaz za posodobitev koncetracije mRNA gena *tetR*. Zaradi izpolnitve pogoja se ukaz izvede v vsakem koraku simulacije.

```
true : {rep.tetR_mRNA := rep.tetR_mRNA + dt *
* ((-1 * rep.tetR_mRNA) +  alpha/(1 + rep.lacI_Protein ^ n) )}
```

Pri implementaciji smo predpostavili, da pri deljenju celic hčerinske celice podedujejo vrednosti parametrov matičnih celic. Spomnimo se, da se pri deljenju celic vrednosti parametrov razpolovijo. Temu se lahko izognemo tako, da parametre definiramo v zapisu.

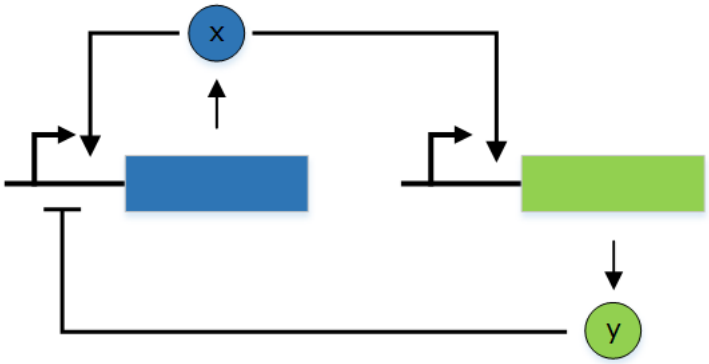


Slika 4.14 Koncentracija proteinov cI, lacI in tetR za vse agente v populaciji. Povprečna koncentracija proteinov je prikazana na osi y, časovni koraki pa na osi x.

Oscilacije koncentracij proteinov so prikazane na grafu 4.14. Zaradi procesa delitve celic smo simulacijo začeli s samo 10 agenti in jo končali po pretečenih 65 časovnih enotah s populacijo 80 agentov. V simulaciji smo uporabili vrednosti parametrov $\alpha = 216$, $\beta = 1$, $n = 2$, $k_{s0} = 1$, $k_{s1} = 0,01$ in $\eta = 2$, tako kot pri testiranju orodij NetLogo in BSim. Graf prikazuje povprečno koncentracijo proteinov cI, lacI in tetR za vse agente. Parametri so reskalirani, tako da so čas in koncentracije opazovanih kemijskih zvrsti podani brez enot. Na osi y je prikazana koncentracija, na osi x pa časovni koraki simulacije.

4.2.4 CellModeller

Večcelični oscilator, ki je v orodju že implementiran, je sestavljen iz dveh genov. Prvi gen izraža aktivator x, ki se veže na regulatorni del obeh genov. Drugi gen izraža represor y za prvi gen. Protein x regulira oba gena, medtem ko protein y zavira aktivator. Topologija oscilatorja je prikazana na sliki 4.15.



Slika 4.15 Gensko regulatorno omrežje preprostega oscilatorja. Oscilator je sestavljen iz dveh genov. Modri izraža aktivatorje za oba gena, zeleni pa izraža represor za prvi gen [44].

Implementacija modela vključuje definicijo naslednjih funkcij: *setup*, *init*, *update*, *divide* in *specRateCL*. V funkciji *setup* inicializiramo vse module, ki jih potrebujemo med simulacijo. V pričujočem primeru v funkciji *setup* inicializiramo modul za regulacijo simulacije *ModuleRegulator*, modul za biofiziko celic *CLBacterium*, modul za integracijo dinamike, opisane z diferencialnimi enačbami *CLEulerIntegrator* in modul, ki skrbi za vizualizacijo simulacije *Renderers.GLBacteriumRenderer*. Inicializacija modulov v funkciji *setup* je prikazana v kodi 4.12.

Koda 4.12 Definicija funkcije *setup* z inicializacijo modula fizike in dinamike celic ter modula za reševanje diferencialnih enačb. Simulacija se začne z eno celico na poziciji (0,0,0) [44].

```
def setup(sim):
    # biofizika, integrator
    biophys = CLBacterium(sim, max_cells=max_cells, jitter_z=False)
    integ = CLEulerIntegrator(sim, 2, max_cells)

    # regulator
    regul = ModuleRegulator(sim)

    sim.init(biophys, regul, None, integ)

    # celica in njena pozicija
    sim.addCell(cellType=0, pos=(0,0,0))

    # vizualizacija simulacije
```

```
    therenderer = Renderers.GLBacteriumRenderer(sim)
    sim.addRenderer(therenderer)
```

V funkciji *init* definiramo velikost celice, pri kateri pride do podvojevanja oziroma delitve celice in hitrosti rasti celice. Oscilator, ki ga želimo implementirati, je sestavljen iz dveh genov. Gena v programski kodi obravnavamo kot kemijski zvrsti, ki ju inicializiramo v spremenljivko celice *cell.species*. Primer inicializacije kemijskih zvrsti je prikazan v kodi 4.13.

Koda 4.13 Funkcija *init* z inicializacijo velikosti celice, pri kateri pride do delitve, hitrosti rasti celice in koncentracije kemijskih zvrsti [44].

```
def init(cell):
    cell.targetVol = 3.0 + random.uniform(0.0,0.5)
    cell.growthRate = 0.6

    # koncentracija kemijskih zvrsti
    cell.species[:] = [0,0]
```

Posodobitev stanja modela v vsakem simulacijskem koraku se izvaja v funkciji *update*. V funkciji se z zanko sprehodimo po vseh celicah in jih označimo za delitev. Celica je označena za delitev, če je njen volumen večji od velikosti delitve celice *cell.targetVol*, ki smo jo definirali v funkciji *init*.

Ko je celica označena za delitev, se deli na dve hčerinski celici, ki privzeto podedujeta vrednosti spremenljivk matične celice. Svoj model delitve celice lahko implementiramo z definicijo funkcije *divide*. V modelu oscilatorja se v funkciji *divide* nastavi le velikost, pri kateri se zgodi delitev celice *cell.targetVol*.

Model oscilatorja ima implementirano tudi metodo *specRateCL*. Funkcija vrne telo funkcije kot niz, ki se prevede med izvajanjem programa in je nato predana v izvajanje v orodje OpenCL. Koda v telesu funkcije je napisana v programskem jeziku C in opisuje dinamiko oziroma hitrost spremembe posamezne kemijske zvrsti. Implementacija spremembe kemijskih zvrsti, opisane z enačbami, je prikazana v kodi 4.14. Kemijske zvrsti celice, inicializirane v funkciji *init*, se prepišejo v spremenljivki x in y, ki ustrezata aktivatorju ter represorju, prikaznih na sliki 4.15.

Koda 4.14 Funkcija *specRateCL* z izvorno kodo, ki opisuje dinamiko znotraj posamezne celice. Koda vsebuje enačbi, ki opisujeta hitrost spremembe posamezne kemijske zvrsti celice, definirane v funkciji *init* [44].

```
def specRateCL():
```

```
    return '''
    /* inicializacija parametrov k1, k2, rho ... */
    float x = species[0];
float y = species[1];
rates[0] = delta*(k1*(1+rho*x*x)/(1+x*x+sigma*y*y)-x);
rates[1] = delta*k2*(1+rho*x*x)/(1+x*x)-y;
    '''
```

Orodje CellModeller smo testirali na platformi Linux. Na prvo oviro pri uporabi orodja smo naleteli že pri njegovi namestitvi. Namestitvena skripta orodja vsebuje neažurne naslove repozitorijev paketov, ki jih potrebuje CellModeller pri svojem izvajanju. Po posodobitvi naslovov repozitorijev skripta namesti pakete novejših verzij, pri katerih se orodje ne zažene. Orodje smo uspešno zagnali ob namestitvi starejših verzij določenih paketov. Kljub zagonu orodja simulacije modelov nismo uspeli zagnati. Neuspešne zagone simulacije gre pripisati pomanjkanju nameščenih programskih paketov in knjižnic, ki jih uporablja okolje OpenCL za pospešitev izvajanja simulacije z izkoriščanjem virov GPE. Posledica tega je tudi neuspešen zagon simulacije in testiranje pričujočega primera enostavnega oscilatorja.

4.3 Primerjava orodij

V pričujočem poglavju smo demonstrirali uporabo izbranih orodij na večagentnih modelih, ki simulirajo medcelično komunikacijo in oscilatorno dinamiko. Demonstracije orodij smo se lotili z vidika uporabniške prijaznosti in se tako osredotočili na podprte funkcionalnosti in že implementirane biološke lastnosti, ki uporabnikom poenostavijo vzpostavitev modelov. V orodju NetLogo smo analizirali implementacijo modela medcelične komunikacije zaznavanja kvoruma. Orodje v osnovi ne vsebuje implementiranih lastnosti celice, biofizike in dinamike celičnih procesov. V NetLogu je implementacija dinamike agentov in njegovih lastnosti odvisna od uporabnika. Kljub temu da v orodju obstajajo vzorčni primeri implementacij modelov bioloških sistemov, z orodjem ne moremo na enostaven način implementirati dinamike celic, ki simulira realno biofiziko celičnih sistemov. V orodjih BSim, Gro in CellModeller je implementacija fizike celic, celične rasti, dinamika medcelične komunikacije in drugih lastnosti skrita uporabnikom ter že pripravljena na uporabo.

NetLogo je izmed izbranih orodij edino orodje z uporabniškim vmesnikov, ki vsebuje

tako grafične gradnike za spreminjanje parametrov simulacije kot urejevalnik izvorne kode modela. Orodji Gro in CellModeller vsebujeta le osnovni grafični vmesnik, v katerega naložimo model in zaženemo njegovo simulacijo, medtem ko orodje BSim interaktivnega vmesnika ne vsebuje. Modele v orodjih BSim, Gro in CellModeller lahko programiramo v urejevalnikih teksta ali v namenskih razvojnih okoljih. Orodji NetLogo in BSim sta se izkazali kot stabilni in zanesljivi, saj pri izvajanju simulacij in splošni uporabi ni prihajalo do nepredvidenih napak. Enako ne moremo trditi za orodje CellModeller. Neposodobljena namestitvena skripta se izvede neuspešno in ne namesti vseh programskih paketov, ki jih orodje zahteva za izvajanje modelov. Kljub alternativnim pristopom namestitve orodja z upravljalniki paketov, z orodjem nismo uspeli zagnati nobenega vzorčnega modela, vključenega v orodje. Pri orodju Gro smo opazili počasno delovanje pri večjem številu agnetov. Kombinacija majhnega časovnega koraka simulacije in izpisa na standardni izhod povzroči netekoče delovanje ali celo sesutje programa Gro.

Z orodji NetLogo, BSim in Gro smo implementirali večcelični represilator. Orodja med seboj po simulacijskih rezultatih težko primerjamo. Modeli se med seboj razlikujejo po implementaciji. Vsako orodje je osredotočeno na določeno področje, na katerega je prilagojen način implementacije modelov. Posledica tega so različni opisi dinamik agentov in medcelične komunikacije ter podpora različnim funkcijam, ki jih uporabljamo pri implementaciji modelov. Pri implementaciji modelov v Gro moramo biti na primer pozorni na proces delitev celic in upoštevati delitev vrednosti parametrov celic pri delitvi. V orodju NetLogo moramo funkcije za reševanje diferencialnih enačb razviti sami ali uvoziti iz zunanjih knjižnic.

Pregledana orodja temeljijo na različnih modelirnih jezikih. Orodja so zaenkrat še v takšni fazi, da od uporabnika zahtevajo vsaj določen nivo znanja programiranja. NetLogo kljub enostavni uporabi ne omogoča implementacije natančnih modelov bioloških sistemov. Orodja BSim, Gro in CellModeller temeljijo na jezikih Java, Gro in Python. BSim, Gro in CellModeller so specializirana za modeliranje kompleksnejših celičnih modelov. V osnovni vsebujejo vzorčne primere implementacij modelov, ki pokrivajo določene aspekte celične dinamike. Kljub temu uporabniki potrebujejo dovolj znanja programiranja za razumevanje in nadgraditev obstoječih modelov ali razvoj svojih modelov.

5 Zaključek

Razvoj računalništva in razvoj računalniških postopkov, ki omogočajo preučevanje bioloških sistemov, je prispeval k razvoju ter mnogim odkritjem na področju moderne biologije, medicine in farmacije. Razvoj računalniških postopkov je prinesel vrsto orodij, ki postavljanje modelov bioloških sistemov omogočajo tudi računalniškim laikom. V pričujočem delu smo opravili pregled orodij, ki so osredotočena na agentno modeliranje tovrstnih sistemov. Agentno modeliranje je način računalniškega modeliranja, pri katerem je model sistema sestavljen iz množice avtonomnih entitet, imenovanih agentje. Osredotočili smo se na splošnonamenska orodja in na orodja, pri katerih agentje nastopajo kot celice ter kot taka že vsebujejo implementacijo lastnosti in dinamiko celičnih procesov, ki jo opazimo v bioloških sistemih. Tovrstna orodja od uporabnika zahtevajo vsaj določen nivo znanja programiranja. Prvo orodje, ki smo ga vključili v pregled, je NetLogo. To je splošno orodje, ki ni specializirano na modeliranje celic. Kljub temu vsebuje implementirane module, ki omogočajo enostavno adaptacijo na modeliranje bioloških sistemov. Modeliranje poteka preko intuitivnega grafičnega uporabniškega vmesnika v programskem jeziku NetLogo. Nadaljevali smo s pregledom orodja BSim. To

orodje je specializirano na modeliranje bakterijskih populacij. BSim vključuje referenčno implementacijo lastnosti bakterij z namenom hitre vzpostavitve novih modelov, ki so delno zgrajeni iz že obstoječih modelov. Programsko ogrodje nima samostojnega uporabniškega vmesnika, zato implementacija modelov v programskem jeziku Java poteka v urejevalnikih teksta ali v enem izmed integriranih razvojnih okolij. Za razliko od orodja NetLogo, orodje BSim omogoča definiranje agentove dinamike na osnovi gensko regulatornih omrežij z metodami za reševanje diferencialnih enačb. V pregled orodij smo vključili tudi orodje Gro. Slednje omogoča vzpostavitev modelov celičnih populacij, v kateri celične lastnosti temeljijo na lastnostih bakterije *E. coli*. Vzpostavitev modelov poteka v višjem programskem jeziku Gro. Orodje vključuje tudi enostaven uporabniški vmesnik z gumbi za nadzor poteka simulacije, vmesnik pa ne vsebuje urejevalnika za izvorno kodo modelov. Zadnje orodje, ki smo ga vključili v pregled, je CellModeller. Osredotočeno je na simulacijo formacije biofilmov in omogoča pospešeno izvajanje simulacij z izkoriščanjem GPE. Orodje vključuje enostaven uporabniški vmesnik za vizualizacijo simulacij modelov, napisanih v Pythonu. CellModeller, Gro in BSim vsebujejo implementacijo določenih celičnih procesov ter celične dinamike. Orodja CellModeller z razliko od drugih orodij nismo uspeli zagnati.

Orodja smo demonstrirali na enostavnih večagentnih modelih, ki do neke mere simulirajo medcelično komunikacijo in oscilatorno dinamiko znotraj celic ter sistema kot celote. Za najenostavnejše orodje za uporabo se je izkazalo orodje NetLogo. Žal ne omogoča postavitve kompleksnih celičnih modelov, kot npr. orodji BSim in Gro. S specializiranimi orodji za modeliranje celic se približamo postavitvi realnih bioloških sistemov v laboratorijih, čemur so takšna orodja namenjena. Obravnavana orodja temeljijo na različnih modelirnih jezikih. Za razvoj svojih in nadgraditev obstoječih vzorčnih primerov orodja zahtevajo določen nivo znanja programiranja ter poznavanje modelirnih konceptov, ki jih orodje realizira. Vsa orodja imajo pripadajočo dokumentacijo. Najbolje je dokumentirano orodje NetLogo z izčrpnimi primeri programske kode in opisi ukazov, ki mu sledita orodji BSim in Gro. Nekoliko manj dosledna je dokumentacija orodja CellModeller, ki nima posodobljenega namestitvenega postopka in določenih implementiranih modulov.

Uporabniki so pri izbiri ustreznega orodja za modeliranje specifičnega problema soočeni z odločitvijo izbire med vse večjim naborom specifičnih orodij, ki pa zahtevajo veliko znanja in časa za osvojitev. Optimalno orodje za modeliranje bioloških orodij bi vključevalo način modeliranja, pri katerem uporabnik ni pogojen z naprednim znanjem programira-

nja in poznavanjem podrobnosti določenega programskega jezika. Korak k temu je opis bioloških modelov na standarden in enoličen način. Takšen opis in izmenjavo načrtovanih bioloških sistemov ponujata jezika SBML (angl. *system biology markup language*) [51] ter SBOL (angl. *synthetic biology open language*) [52]. Adaptacija takšnega opisa modelov na agentno modeliranje bi omogočila vzpostavitev modelov z agentovo dinamiko, opisano z obstoječimi modeli in testiranje njihove dinamike v večceličnem okolju. Spoznali smo, da obstaja vrsta orodij za modeliranje agentnih modelov bioloških sistemov. Razlog za takšno številčnost je pokrivanje določenih področij biologije in njihova prilagoditev ter optimiziranost na specifične funkcionalnosti, ki jih zahteva obravnavano področje. Orodja temeljijo na različnih programskih jezikih in tako še dodatno otežujejo uporabnikom izbiro ustreznega orodja za modeliranje določenega sistema. Oteženo je tudi prehajanje med različnimi orodji. Možno bi bilo razviti orodje, ki bi pokrivalo večji nabor celičnih lastnosti in celičnih procesov ter hkrati omogočiti enostavno uporabo. Kljub potencialnemu razvoju univerzalnega orodja bodo v prihodnosti še vedno obstajala specializirana orodja, ki bodo nudila optimizirano izvajanje simulacij konkretnega sistema pri konkretnih pogojih.

Literatura

[1] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al., The Sequence of the Human Genome, Science 291 (5507) (2001) 1304–1351.

[2] E. Klipp, W. Liebermeister, C. Wierling, A. Kowald, R. Herwig, Systems Biology: A Textbook, John Wiley & Sons, 2016.

[3] E. Bartocci, P. Lió, Computational Modeling, Formal Analysis, and Tools for Systems Biology, PLoS Comput Biol 12 (1) (2016) e1004591.

[4] H. Resat, M. N. Costa, H. Shankaran, Spatial Aspects in Biological System Simulations, Methods in enzymology 487 (2011) 485.

[5] J. Fisher, T. A. Henzinger, Executable cell biology, Nature biotechnology 25 (11) (2007) 1239–1249.

[6] P. K. Maini, T. E. Woolley, R. E. Baker, E. A. Gaffney, S. S. Lee, Turing's model for biological pattern formation and the robustness problem, Interface focus 2 (4) (2012) 487–496.

[7] M. Wooldridge, Agent-based software engineering, IEE Proceedings-software 144 (1) (1997) 26–37.

[8] E. Bonabeau, Agent-based modeling: Methods and techniques for simulating human systems, Proceedings of the National Academy of Sciences 99 (suppl 3) (2002) 7280–7287.

[9] G. An, Q. Mi, J. Dutta-Moscato, Y. Vodovotz, Agent-based models in translational systems biology, Wiley Interdisciplinary Reviews: Systems Biology and Medicine 1 (2) (2009) 159–171.

[10] J. O. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integrative Biology 3 (2) (2011) 86–96.

[11] D. C. Walker, J. Southgate, The virtual cell—a candidate co-ordinator for 'middle-out'modelling of biological systems, Briefings in bioinformatics 10 (4) (2009) 450–461.

[12] C. Macal, M. North, Introductory tutorial: Agent-based modeling and simulation, in: Proceedings of the 2014 Winter Simulation Conference, IEEE Press, 2014, pp. 6–20.

[13] S. F. Railsback, S. L. Lytinen, S. K. Jackson, Agent-based simulation platforms: Review and development recommendations, Simulation 82 (9) (2006) 609–623.

[14] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework, in: High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, IEEE, 2012, pp. 538–545.

[15] F. Graner, J. A. Glazier, Simulation of biological cell sorting using a two-dimensional extended Potts model, Physical review letters 69 (13) (1992) 2013.

[16] J. A. Glazier, F. Graner, Simulation of the differential adhesion driven rearrangement of biological cells, Physical Review E 47 (3) (1993) 2128.

[17] M. H. Swat, G. L. Thomas, J. M. Belmonte, A. Shirinifard, D. Hmeljak, J. A. Glazier, Multi-Scale Modeling of Tissues Using CompuCell3D, Methods in Cell Biology 110 (2012) 325–366.

[18] J. Naylor, H. Fellermann, Y. Ding, W. K. Mohammed, N. S. Jakubovics, J. Mukherjee, C. A. Biggs, P. C. Wright, N. Krasnogor, Simbiotics: A Multiscale Integrative Platform for 3D Modeling of Bacterial Populations, ACS Synthetic Biology (2017).

[19] A. Goni-Moreno, M. Amos, DiSCUS: A Simulation Platform for Conjugation Computing, in: International Conference on Unconventional Computation and Natural Computation, Springer, 2015, pp. 181–191.

[20] M. Gutiérrez, P. Gregorio-Godoy, G. Pérez del Pulgar, L. E. Muñoz, S. Sáez, A. Rodríguez-Patón, A New Improved and Extended Version of the Multicell Bacterial Simulator Gro, ACS Synthetic Biology (2017).

[21] T. E. Gorochowski, Agent-based modelling in synthetic biology, Essays in biochemistry 60 (4) (2016) 325–336.

[22] S. Abar, G. K. Theodoropoulos, P. Lemarinier, G. M. O'Hare, Agent Based Modelling and Simulation tools: A review of the state-of-art software, Computer Science Review (2017).

[23] U. Wilensky, I. Evanston, NetLogo: Center for connected learning and computer-based modeling, Northwestern University, Evanston, IL (1999) 49–52.

[24] S. Tisue, U. Wilensky, NetLogo: Design and implementation of a multi-agent modeling environment, in: Proceedings of agent, Vol. 2004, 2004, pp. 7–9.

[25] U. Wilensky, NetLogo Tumor model, http://ccl.northwestern.edu/netlogo/models/Tumor, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1998).

[26] U. Wilensky, NetLogo user manual version 6.0.1 (2017).

[27] E. Sklar, NetLogo, a multi-agent simulation environment (2007).

[28] J. Gruber, Markdown language, https://daringfireball.net/projects/markdown/ (pridobljeno 23. 4. 2017) (2004).

[29] S. L. Lytinen, S. F. Railsback, The evolution of agent-based simulation platforms: a review of NetLogo 5.0 and ReLogo, in: Proceedings of the fourth international symposium on agent-based modeling and simulation, 2012.

[30] BSim, http://bsim-bccs.sourceforge.net/ (pridobljeno 25. 10. 2016).

[31] T. E. Gorochowski, A. Matyjaszkiewicz, T. Todd, N. Oak, K. Kowalska, S. Reid, K. T. Tsaneva-Atanasova, N. J. Savery, C. S. Grierson, M. Di Bernardo, BSim: An Agent-Based Tool for Modeling Bacterial Populations in Systems and Synthetic Biology, PloS one 7 (8) (2012) 1–9.

[32] Eclipse Integrated Development Environment, https://www.eclipse.org/ (pridobljeno 3. 8. 2017).

[33] IntelliJ IDEA, https://www.jetbrains.com/idea/ (pridobljeno 3. 8. 2017).

[34] B. P. Ingalls, Mathematical Modelling in Systems Biology: An Introduction, MIT Press, 2013.

[35] U. Alon, An Introduction to Systems Biology, Chapman & Hall, 2007.

[36] A. Matyjaszkiewicz, G. Fiore, F. Annunziata, C. Grierson, N. Savery, L. Marucci, M. di Bernardo, BSim 2.0: An Advanced Agent-Based Cell Simulator, ACS Synthetic Biology (2017).

[37] D. Meagher, Geometric modeling using octree encoding, Computer graphics and image processing 19 (2) (1982) 129–147.

[38] S. S. Jang, K. T. Oishi, R. G. Egbert, E. Klavins, Specification and simulation of synthetic multicelled behaviors, ACS Synthetic Biology 1 (8) (2012) 365–374.

[39] Gro, http://depts.washington.edu/soslab/gro/ (pridobljeno 25. 10. 2016).

[40] Chipmunk2D, https://chipmunk-physics.net/ (pridobljeno 3. 8. 2017).

[41] G. Dhatt, G. Touzot, et al., Finite element method, John Wiley & Sons, 2012.

[42] J. E. Stone, D. Gohara, G. Shi, OpenCL: A parallel programming standard for heterogeneous computing systems, Computing in science & engineering 12 (3) (2010) 66–73.

[43] T. J. Rudge, P. J. Steiner, A. Phillips, J. Haseloff, Computational modeling of synthetic microbial biofilms, ACS Synthetic Biology 1 (8) (2012) 345–352.

[44] CellModeller, http://haselofflab.github.io/CellModeller/ (pridobljeno 22. 10. 2016).

[45] M. B. Elowitz, S. Leibler, A synthetic oscillatory network of transcriptional regulators, Nature 403 (6767) (2000) 335–338.

[46] T. Danino, O. Mondragon-Palomino, L. Tsimring, J. Hasty, A synchronized quorum of genetic clocks, Nature 463 (2010) 326–330.

[47] The extreme complexity of the E. coli transcriptional regulatory network, https://www.nature.com/scitable/content/the-extreme-complexity-of-the-e-coli-14457504 (pridobljeno 2. 8. 2017).

[48] J. Garcia-Ojalvo, M. B. Elowitz, S. H. Strogatz, Modeling a synthetic multicellular clock: Repressilators coupled by quorum sensing, Proceedings of the National Academy of Sciences 101 (30) (2004) 10955–10960.

[49] U. Wilensky, NetLogo Voting model, http://ccl.northwestern.edu/netlogo/models/Voting, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1998).

[50] N. A. Whitehead, A. M. Barnard, H. Slater, N. J. Simpson, G. P. Salmond, Quorum-sensing in Gram-negative bacteria, FEMS microbiology reviews 25 (4) (2001) 365–404.

[51] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, et al., The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models, Bioinformatics 19 (4) (2003) 524–531.

[52] M. Galdzicki, K. P. Clancy, E. Oberortner, M. Pocock, J. Y. Quinn, C. A. Rodriguez, N. Roehner, M. L. Wilson, L. Adam, J. C. Anderson, et al., The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology, Nature biotechnology 32 (6) (2014) 545–550.