Politechnika
Śląska

POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Projekt inżynierski

Modelowanie dynamiki rozwoju infekcji wirusem HIV z wykorzystaniem automatów komórkowych

Autor: Weronika Jargieło

Kierujący pracą: dr hab. inż. Witold Nocoń, prof. PŚ

Gliwice, styczeń 2021

Załącznik Nr 2 do Zarz. Nr 97/08/09

O Ś W I A D C Z E N I E

Wyrażam zgodę/nie wyrażam* zgody na udostępnienie mojej pracy dyplomowej/rozprawy doktorskiej*

……., dnia ……

……

(podpis)

……

(poświadczenie wiarygodności podpisu przez Dziekanat)

** właściwe podkreślić*

Spis treści

Rozdział 1

Wstęp teoretyczny

Po raz pierwszy wirus HIV (*ang. Human Immunodeficiency Virus* - Ludzki Wirus Niedoboru Odporności) został wyizolowany w 1983r., natomiast rok później naukowcy odkryli, że jest on bezpośrednią przyczyną AIDS. W celu zrozumienia złożonej interakcji pomiędzy wirusem HIV, a układem odpornościowym, przeprowadzono wiele badań, aby poznać złożone podłoże biologiczne tego procesu. Poniżej zostały omówione najważniejsze aspekty infekcji wirusem HIV, umożliwiające lepsze zrozumienie procesów, które kryją się za parametrami implementowanego modelu.

Głównym celem ataku wirusa HIV są komórki posiadające na swojej powierzchni receptor CD4. Cząsteczka CD4 występuje głównie na powierzchni komórek, będących subpopulacją limfocytów T, odpowiedzialnych za funkcje pomocnicze, m.in. pomagają one rozpoznawać, atakować i niszczyć bakterie, grzyby, wirusy, jak i inne drobnoustroje, które zakażają organizm. Ekspresję glikoproteiny CD4 stwierdza się również na powierzchni monocytów (a także ich późniejszej formy - makrofagów) oraz komórek dendrytycznych, co oznacza, że mogą one również być celem ataków cząsteczek wirusa HIV.

1.1 Budowa wirusa HIV

HIV to niewielki i stosunkowo prosto zbudowany wirus, który wywołuje chorobę AIDS, czyli zespół nabytego upośledzenia odporności. Wirus ten aktualnie klasyfikowany jest jako członek rodziny retrowirusów, gromadzącej wirusy otoczkowe, zawierające dwie kopie jednoniciowego RNA. Charakterystycznymi cechami retrowirusów jest odwrotna transkrypcja wirusowego RNA na dwuniciowe DNA, a następnie jego integracja z genomem gospodarza.

W skład budowy pojedynczej cząsteczki wirusa wchodzi: materiał genetyczny w postaci dwóch jednoniciowych cząsteczek RNA oraz zestaw enzymów - odwrotna transkryptaza, integraza i proteaza. Składowe wirusa są otoczone płaszczem białkowym - kapsydem oraz zewnętrzną lipidowo-białkową otoczką, w skład której wchodzą glikoproteiny (gp120 i gp41).

RNA wirusa HIV składa się z dziewięciu genów, kodujących łącznie 19 białek. Końcami wirusowego genomu są powtarzalne sekwencje LTR *(ang. long terminal repeat)*, niekodujące żadnych białek, natomiast uczestniczące w połączeniu wirusowych i komórkowych DNA, odgrywając tym samym dużą rolę w procesie transkrypcji. Geny gag, pol i env kodują białka strukturalne wirusa HIV.

1. **Gag (*ang. Group Antygen* - Grupowy Antygen)** – odpowiada za kodowanie białek strukturalnych kapsydu, matriksowych i nukleokapsydu.

   a) **p24** - buduje kapsyd wirusa,

   b) **p17** - jest przymocowany do wewnętrznej strony podwójnej otoczki lipidowej i bierze udział w stabilizacji struktury wirionu. Część cząsteczki p17 znajduje się w głębszych częściach wirionu i odpowiada za transport wirusowego DNA do jądra komórkowego gospodarza,

   c) **p7** - wiąże się z materiałem genetycznym wirusa, umożliwiając jego upakowanie, ułatwia przeprowadzenie odwrotnej transkrypcji oraz pełni rolę w integracji wirusowego materiału genetycznego,

   d) **p6** - zaangażowane w uwalnianie cząsteczek wirusa.

2. **Pol (*ang. Polymerase* - Polimeraza)** – odpowiada za kodowanie prekursorów enzymów wirusowych: odwrotnej transkryptazy (RT), integrazy (IN) i HIV proteazy (PR).

   a) **p10** - proteaza, odpowiada za proteolityczne cięcie białka prekursorowego (p55 oraz p160), kodowanego przez geny Gag oraz Gag-Pol, co skutkuje uwolnieniem enzymów oraz białek strukturalnych.

   b) **p51** - odwrotna transkryptaza, odpowiadająca za transkrypcję wirusowego RNA na DNA.

   c) **p15(66)** - RNaza H, odpowiadająca degradację wirusowego RNA w wirusowym kompleksie replikacyjnym RNA/DNA.

   d) **p32** - integraza, której zadaniem jest integracja prowirusowego DNA z genomem gospodarza.

3. **Env (*ang. Envelope* - Otoczka)** - odpowiada za kodowanie białek otoczki.

4. a) **gp120** - glikoproteina powierzchniowa, odpowiadająca za przyłączenie cząsteczki wirusa do komórki docelowej.

   b) **gp41** - białko transbłonowe, odpowiadające za fuzję wirusa z komórką.

Pozostałe geny kodują białka regulatorowe, biorące udział w infekowaniu komórek, procesie replikacji lub progresji zakażenia.

1. **Tat (*ang. Trans-Activator of Transcription* - Transaktywator Transkrypcji)** – gen kodujący białka, odgrywające kluczową rolę w procesie replikacji wirusa HIV. Produkty powstałe w wyniku ekspresji genu tat wiążą się z sekwencjami genów wirusa HIV i aktywują proces transkrypcji.

2. **Rev (*ang. Regulator of Virion protein expression* - Regulator Ekspresji Białka Wirusowego)** – gen kodujący białka, które poprzez związanie z fragmentami mRNA, reguluje syntezę białek strukturalnych i niestrukturalnych. Dodatkowo ułatwia transport mRNA z jądra do cytoplazmy oraz reguluje intensywność syntezy, zarówno samego siebie, jak i białka Tat.

3. **Vif (*ang. Viral Infectivity Factor* - Czynnik Infekcji Wirusowej)** - koduje białko, pełniące zasadniczą rolę w infekcji wirusem HIV oraz jest znaczącym czynnikiem hamującym oporność komórki gospodarza na zakażenie HIV.

4. **Nef (*ang. Negative Regulatory Factor* - Negatywny Czynnik Regulacyjny)** - koduje białko, posiadające kilka różnych funkcji. Reguluje ono ekspresję receptorów CD4 i antygenów HLA klasy I na powierzchni zakażonej komórki, uniemożliwiając jej zabicie przez limfocyty T-cytotoksyczne. Posiada również właściwości chemotaktyczne, wpływające na migrację spoczynkowych limfocytów T CD4 do miejsca zakażonych makrofagów, co zwiększa prawdopodobieństwo kontaktu tych dwóch rodzajów komórek, nim makrofagi opuszczą układ siateczkowo-śródbłonkowy. Działanie to jest ułatwione poprzez wydzielanie przez makrofagi zakażone wirusem HIV niezidentyfikowanego czynnika aktywującego spoczynkowe LT CD4, co ułatwia zakażenie tych limfocytów. Tłumaczy to istotną rolę makrofagów w rozprzestrzenianiu zakażenia. Białko Nef ma również wpływ na uwalnianie wirusów z komórki przez hamowanie działania teteryny, enzymu uniemożliwiającego uwolnienie wirionu poza obszar komórki.

5. **Vpu (*ang. Viral Protein Unique* - Unikalne Białko Wirusowe)** - koduje białko również odpowiadające za wiele funkcji. Zapobiega połączeniu receptora CD4 i Gp120, zanim białka te zostaną przetransportowane na powierzchnię komórki, ponieważ mogą być syntetyzowane w tym samym czasie. Dodatkowo powoduje proteolizę receptorowego białka CD4, uniemożliwiając niszczenie zainfekowanej komórki przez mechanizmy immunologiczne. Ułatwia również uwalnianie wirusów z komórki gospodarza, najprawdopodobniej poprzez tworzenie kanałów jonowych w błonie plazmatycznej komórki gospodarza.

6. **Vpr (*ang. Viral Protein R* - Białko Wirusowe R)** - koduje białko, występujące w organizmie osoby zarażonej wirusem HIV, w obrębie wirionów, komórek, w surowicy oraz płynie mózgowo-rdzeniowym, a więc istenieje wiele różnych możliwości jego działania. Między innymi odgrywa rolę w prawidłowym przebiegu odwrotnej transkrypcji, wchodząc w skład kompleksu przedintegracyjnego oraz transportując go przez pory błony jądrowej. Przyczynia się również do aktywacji transkrypcji przez wpływ na LTR. Ma istotny wpływ na patogenezę zakażenia wirusem HIV oraz znaczenie w procenie infekcji makrofagów, a także w mniejszym stopniu innych komórek. Naukowcy donoszą również, że wpływa ono na zahamowanie komórki w fazie G2 oraz apoptozę zainfekowanej komórki. Padają również podejrzenia, że ma znaczenie w długotrwałym zakażeniu wirusem HIV, ponieważ może indukować zakażenie spoczynkowych monocytów oraz makrofagów. Dodatkowo białko to hamuje wydzielanie cytokinin (interleukiny 12), pobudzających limfocyty Th1, powodując tym samym przewagę limfocytów Th2 i rozprzestrzenianie infekcji.

1.2 Trójfazowa dynamika infekcji wirusem HIV

U osób zarażonych wirusem HIV można zaobserwować rozwój dynamiki infekcji, złożony z trzech następujących po sobie faz:

Faza I - Ostra

W pierwszym miesiącu po infekcji pacjent znajduje się w pierwszej fazie, zwanej fazą ostrą. Etap ten charakteryzuje się gwałtownym wzrostem stężenia wirusa w krwi obwodowej i spadkiem poziomu limfocytów pomocniczych T (komórek $CD4^+T$). Pacjent odczuwa podobne symptomy, jak w przypadku grypy.

Faza II - Asymptotyczna

W ciągu następnych dwóch lub trzech miesięcy obciążenie wirusem wykazuje gwałtowny spadek, a zawartość komórek $CD4^+T$ w krwi obwodowej pacjenta powraca w przybliżeniu do normalnego poziomu.

Zarażeni pacjenci nie odczuwają wtedy żadnych objawów klinicznych. Oznacza to wejście pacjenta w drugą, bezobjawową fazę, która może trwać od kilku miesięcy do ponad 15 lat (zazwyczaj około 10 lat). Wyróżniane są skrajne grupy pacjentów, u których przebieg zakażenia jest niezwykle szybki, oznaczany skrótem RP *(ang. rapid progressors)* - czas od zakażenia do rozwoju AIDS jest bardzo krótki i wynosi do 2 lat - lub wolny oznaczany jako LTNP *(ang. long term non-progressors)*

W tej fazie, zamiast całkowitej eliminacji, obciążenie wirusem HIV pozostaje stosunkowo niskie, podczas gdy poziom komórek $CD4^+T$ nadal powoli obniża się.

Faza III - AIDS (ang. Acquired Immune Deficiency Syndrome)

Populacja komórek $CD4^+T$ staje się niższa niż wartość krytyczna, a stężenie wirusa ponownie rośnie, co prowadzi do wystąpienia AIDS.

W fazie AIDS układ odpornościowy pacjenta jest tzw. układem upośledzonym, niebędącym w stanie walczyć dłużej z infekcją wirusową oraz z innymi patogenami atakującymi organizm gospodarza. Prowadzi to do śmierci pacjentów na skutek wystąpienia różnych infekcji, które dla zdrowych ludzi nie stanowią śmiertelnego zagrożenia.

Za początek tej fazy przyjmuje się moment, w którym koncentracja zdrowych komórek $CD4^+T$ spada poniżej krytycznego poziomu, określonego jako 30 % normatywnej ilości komórek zdrowych w ludzkim organizmie.

1.3 Cykl replikacyjny wirusa HIV

Cykl życiowy wirusa HIV jest wieloetapowym procesem, z którego każdy etap jest kluczowy do jego ukończenia, a co za tym idzie, do rozprzestrzeniania infekcji w organizmie gospodarza.

Replikacja wirusa HIV rozpoczyna się od połączenia wirionu ze zdrową komórką $CD4^+T$. Wkroczenie wirusa do wnętrza komórki wymaga obecności specyficznych receptorów CD4 w błonie infekowanej komórki oraz koreceptorów takich jak CC5R5 (receptor β-chemokin) lub CXCR4 (receptor α-chemokin). Receptory te wchodzą w interakcję z białkowymi kompleksami obecnymi w otoczce lipidowej wirusa HIV. Kompleksy te składają się z dwóch glikoprotein: błonowej gp120 oraz transbłonowej gp41.

W momencie, gdy wirus HIV spotyka na swojej drodze komórkę $CD4^+T$, glikoproteina gp120 - obecna w błonie lipidowej wirusa - wiąże się z receptorem CD4 oraz

z jednym z wymienionych koreceptorów, które z kolei wywołują zmianę konformacji w białku gp120, co pozwala na odsłonięcie hydrofobowej domeny gp41, która zostaje wprowadzona do błony komórkowej infekowanej komórki. Następnie dochodzi do połączenia i fuzji osłonki wirusowej z błoną komórkową oraz przemieszczenia zawartości kapsydu wirusa do cytoplazmy komórki. Kapsyd wirusa zawiera materiał genetyczny, jak i również zestaw enzymów niezbędnych do dalszego trwania cyklu: integrazę, proteazę oraz odwrotną transkryptazę.

Odwrotna transkryptaza jest odpowiedzialna za rozpoczęcie procesu odwrotnej transkrypcji wirusowego RNA. Posiada dwa katalityczne miejsca: centrum aktywne rybonukleazy oraz centrum aktywne polimerazy.

Centrum aktywne polimerazy odpowiada za przekształcanie jednoniciowego RNA w dwuniciową hybrydę RNA-DNA. Rybonukleaza natomiast odcina nić RNA, po czym polimeraza dobudowuje drugą nić, tworząc dwuniciowe DNA. W trakcie tego mogą powstawać liczne mutacje i nowe warianty wirusa, ponieważ odwrotna transkryptaza nie posiada mechanizmu, ani zdolności korekcji błędów: w transkrypcji powstają przypadkowo z częstością $3,4 * 10^{-5}$ na każdą parę zasad w pojedynczym cyklu replikacyjnym.

Następnie dochodzi do transportu kompleksu złożonego z dwuniciowego HIV-DNA, wirusowego białka Vpr, p7, p17 oraz integrazy do jądra komórkowego oraz integracji z genomem gospodarza. Preferencyjnymi miejscami integracji HIV w jądrowym DNA są aktywne geny i regiony gorących miejsc (hotspots). Prowirus może pozostać nieaktywny (latentny) transkrypcyjnie lub też może ujawniać różne poziomy ekspresji genów, aż do aktywnej produkcji wirusa.

Wbudowane w genom gospodarza wirusowe DNA ulega transkrypcji. Jednak do zainicjowania ekspresji genów wirusowych wymagane są zarówno czynniki komórkowe, jak i wirusowe. Czynniki komórkowe mogą być konstytutywnie wytwarzane przez komórkę lub indukowane przez różnorodne sygnały aktywujące, w tym antygeny, mitogeny, heterologiczne produkty genowe, cytokiny, światło ultrafioletowe i ciepło.

Po aktywacji prowirusa HIV przez czynniki komórkowe, pierwsze geny wirusowe podlegające ekspresji to te, które kodują białka niestrukturalne o funkcjach regulacyjnych. Najważniejszym z tych białek jest Tat, który jest silnym transaktywatorem ekspresji genów wirusa HIV i jest niezbędny do replikacji. Uważa się, że Tat wywiera swój efekt na dwa sposoby: inicjując transkrypcję RNA prowirusa HIV lub przez stymulowanie produkcji transkryptów pełnej długości RNA. W kolejnym etapie powstaje białko Rev, regulujące powstawanie białek strukturalnych i enzymatycznych. W wyniku oddziaływania białka Rev dochodzi do eksportu RNA, niepoddanego procesowi splicing'u (pre-mRNA) lub częściowemu splicing'owi z jądra komórkowego do cytoplazmy, gdzie ulega ono translacji.

Utworzone białka mogą zostać poddane procesom modyfikacji potranslacyjnych - palmitylacji, glikozylacji, fosforylacji. Dodatkowo niektóre z powstałych białek zostają poddane działaniu wirusowej proteazy, bowiem trawi ona długie łańcuchy białkowe do krótszych. Etap ten jest kluczowy dla stworzenia wirionów zdolnych do infekowania zdrowych komórek $CD4^{+}T$.

Ootatni, odbywający się w obrębie błon plazmatycznych zakażonej komórki etap cyklu replikacyjnego wirusa HIV, obejmuje przetransportowanie białka otoczki gp160

z siateczki endoplazmatycznej do aparatu Golgiego, a następnie jego pocięcie przez proteazę na dwie glikoproteiny: gp41 i gp120. Utworzone cząsteczki wpierw są transportowane do błony plazmatycznej zakażonej komórki, a następnie w niej umieszczone. Równolegle w wewnętrznej powierzchni błony plazmatycznej zostaje umieszczone białko p55 (Gag) oraz p160 (Gag-Pol) wraz z wirusowym RNA, rozpoczynając proces tworzenia wirionu, gotowego do opuszczenia komórki gospodarza w wyniku pączkowania. W wyniku dojrzewania wirionu, podczas którego proteazy HIV tną wirusowe białka prekursorowe na mniejsze fragmenty, spełniające rolę enzymów oraz będące białkami strukturalnymi, dochodzi do powstania infekcyjnie dojrzałej cząsteczki wirusowej. Proces dojrzewania wirionu odbywa się w tzw. pączkach lub w niedojrzałych wirionach po opuszczeniu komórki gospodarza. Dojrzały wirus jest zdolny do zakażenia następnej komórki $CD4^{+}T$.

Powyższa charakterystyka wirusa HIV oraz przebiegu infekcji wywołana wirusem HIV została oparta na nastepujących publikacjach [4, 6, 7, 8, 9, 10, 12, 13].

1.4 Automat komórkowy

Automat komórkowy to dyskretny układ dynamiczny, który składa się z regularnej sieci określonej liczby komórek, zmieniających swoje stany w zależności od stanów ich sąsiadów, zgodnie z przyjętymi regułami przejść pomiędzy stanami. Stany wszystkich komórek aktualizowane są jednocześnie, przy użyciu tej samej zasady aktualizacji. Proces powtarza się w dyskretnych krokach czasowych.

Okazało się, że zadziwiająco proste reguły aktualizacji komórek mogą realizować niezwykle złożoną dynamikę. Jednym z popularnych przykładów automatu komórkowego jest tzw. Gra w Życie, zaproponowana przez Johna Conwaya.

Konstrukcja każdego automatu komórkowego charakteryzuje się następującymi elementami:

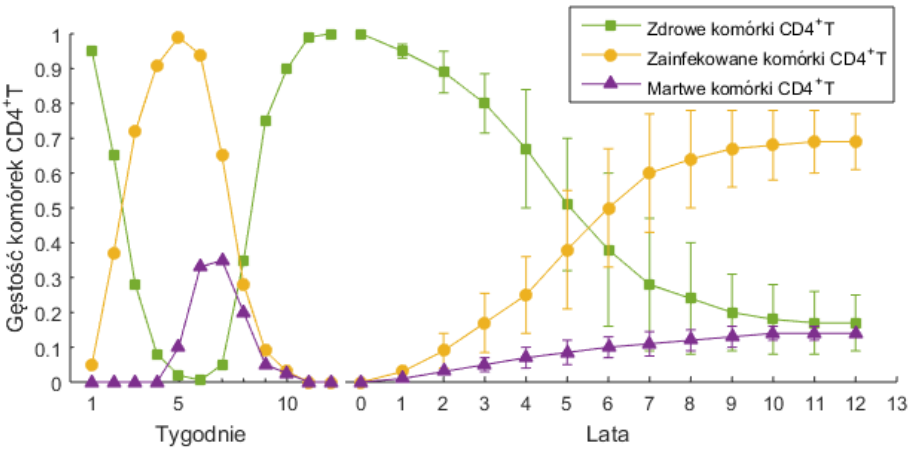
- n-wymiarową, regularną oraz dyskretną siatką komórek,
- zdefiniowanym sąsiedztwem komórek,
- zdefiniowanymi warunkami brzegowymi,
- zbiorem stanów,
- regułami przejść pomiędzy stanami,
- ustalonymi warunkami początkowymi.

Rozdział 2

Przegląd istniejących rozwiązań

2.1 Modelowanie dynamiki infekcji wirusem HIV przy użyciu automatu komórkowego 2D

W 2001r. Rita Maria Zorzenon dos Santos oraz Sérgio Coutinho w jednej ze swoich prac [14] zaproponowali dwuwymiarowy automat komórkowy, jako model realizujący dynamikę infekcji wirusem HIV.



Rysunek 2.1: Przebieg czasowy liczebności komórek zdrowych, zainfekowanych i martwych powstały w oparciu o uśrednienie 500 symulacji (źródło: opracowanie własne według [14]).

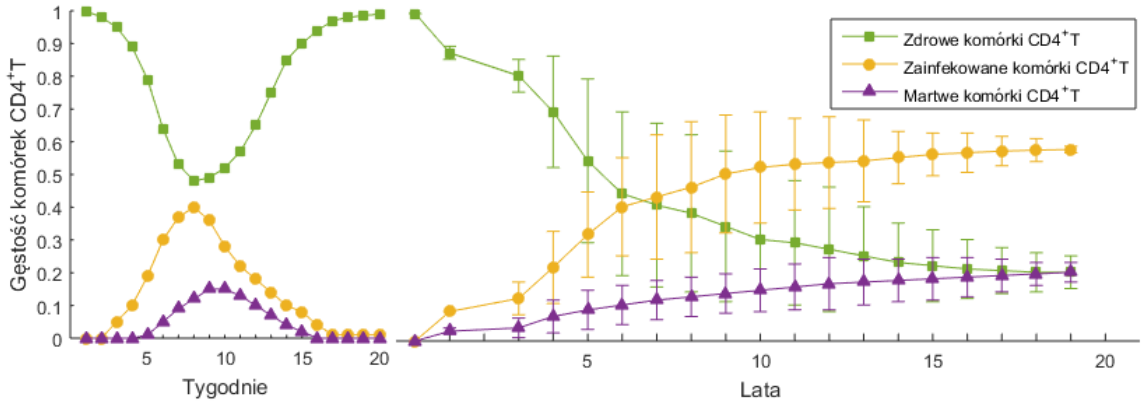
Zastosowane w modelu zasady oraz wartości parametrów opierają się na danych klinicznych, zebranych na podstawie obserwacji oraz biologicznych eksperymentów. Model uwzględnia szereg aspektów biologicznych mających miejsce w trakcie rzeczywistych infekcji wirusem HIV, tj.: wysoki wskaźnik mutacji wirusa. Dodatkowo przestrzenność modelu umożliwia oddanie wpływu aktywacji komórek zainfekowanych przez wirusa HIV, lecz dotychczas przebywających w stanie uśpionym oraz napływ komórek zdrowych z innych regionów węzła chłonnego.

Jak można zauważyć na rysunku 2.1, model odwzorowuje trójfazowy przebieg dynamiki infekcji, począwszy od fazy ostrej, poprzez fazę asymptotyczną, kończąc na fazie AIDS, która osiągnięta zostaje w momencie spadku poziomu zdrowych komórek poniżej

30 % stężenia komórek $CD4^+T$, w organizmie zdrowego człowieka. W pracy ujawniono również proces, odpowiadający za obserwowanie spadku koncentracji niezainfekowanych komórek. Zainfekowane komórki organizują się w struktury przestrzenne, prowadzące do powstania niezanikającego źródła infekcji. Źródło to sukcesywnie generuje fale zainfekowanych komórek, rozprzestrzeniających się na cały system, co prowadzi model ze stanu asymptotycznego do stanu AIDS. Natomiast owy model 2D zapewnia jedynie ograniczoną możliwość odwzorowywania przestrzennej infekcji wirusem HIV w obszarze węzła chłonnego, który w rzeczywistości jest obiektem trójwymiarowym.

2.2 Modelowanie dynamiki infekcji wirusem HIV przy użyciu automatu komórkowego 3D

W artykule Youbin Mo, Bin Ren, Wencao Yang oraz Jianwei Shuai z roku 2013 [11] również poruszono temat modelowania dynamiki infekcji wirusem HIV, z zastosowaniem automatu komórkowego. Zdecydowano się jednak na rozszerzenie wcześniej zaproponowanego rozwiązania o dodatkowy wymiar, konstruując trójwymiarowy automat komórkowy.



Rysunek 2.2: Przebieg czasowy liczebności komórek zdrowych, zainfekowanych i martwych powstały w oparciu o uśrednienie 2000 symulacji (źródło: opracowanie własne według [11]).

Zaproponowany model również odtwarza dynamikę rozwoju infekcji wirusowej, realizując jej wszystkie trzy fazy, obserwowane u pacjentów zakażonych wirusem HIV, poddanym obserwacji w klinikach. Dodatkowo w artykule przedstawiono, że model trójwymiarowy charakteryzuje się lepszą odpornością na zróżnicowanie parametrów niż zastosowany wcześniej automat komórkowy 2D.

Podobnie jak w przypadku dwuwymiarowego modelu, w artykule przedstawiono występującą strukturę źródła fal zakaźnych, rozprzestrzeniających się na cały system, powodującą przejście do fazy AIDS.

Zarówno przypadku pracy opisującej modelu dwuwymiarowego, jak i trójwymiarowy, nie udostępniono programu umożliwiającego odtworzenie przeprowadzonych symulacji, co uniemożliwia sprawdzenie wpływu różnorodnych parametrów modelu na przebieg symulacji. Dodatkowo nie został przedstawiony żaden program realizujący wizualizację trójwymiarowego automatu komórkowego, co umożliwiałoby lepsze zrozumienie oraz ułatwiłoby analizę zaimplementowanego modelu.

2.3 Specyfikacja problemu oraz założenia projektu

Dotychczas naukowcy z dziedziny medycyny i biologii osiągnęli znaczny postęp w zrozumieniu złożonej interakcji między wirusem HIV, a komórkami ludzkiego układu odpornościowego. Niestety dynamika trójfazowego rozwoju infekcji, aż do osiągnięcia fazy AIDS, wciąż pozostaje niejasna i nie tylko wymaga wykonywania większej ilości eksperymentów, ale także wykorzystania modelowania matematycznego.

Po dzień dzisiejszy opracowano kilka modeli matematycznych wykorzystujących zwykłe równania różniczkowe do badania dynamiki zakażenia wirusem HIV. Modele te z powodzeniem odtworzyły dynamikę trzech stanów i wniosły znaczący wkład do zrozumienia mechanizmów infekcji wirusem HIV. Nie mogły one jednak wyjaśnić zachowania przestrzennego oraz stochastycznych skutków dynamiki infekcji wirusem HIV.

Celem niniejszego projektu inżynierskiego jest zrekonstruowanie trójwymiarowego automatu komórkowego, realizującego trójfazową dynamikę infekcji wirusem HIV przedstawionego w punkcie 2.2 oraz zbadanie wpływu zmienności parametrów modelu na jego odpowiedź.

Dodatkowo, w celu lepszego zobrazowania idei opisywanego modelu, zdecydowano się na stworzenie programu, pozwalającego na wizualizację zaimplementowanego automatu komórkowego.

Rozdział 3

Narzędzia

3.1 Python

Automat komórkowy, realizujący dynamikę infekcji wirusem HIV oraz program uruchamiający wizualizację, zostały zaimplementowane w języku Python.

Python jest językiem o bardzo szerokim zakresie zastosowań, pozwalającym na pisanie różnorodnych programów przy użyciu odpowiednich bibliotek lub frameworków, przez co bardzo wiele firm używa go w swoich aplikacjach. Jego interpretery są dostępne dla wielu systemów, co czyni go językiem wieloplatformowym.

Język ten należy do grupy języków interpretowanych, przez co przeważnie charakteryzuje się mniejszą wydajnością w stosunku do języków kompilowanych, m.in.: C++. Jednak w zamierzeniach Pythona, priorytetem nie była wydajność, a stworzenie języka skupiającego inne mocne strony m.in. czytelność, łatwość użycia i produktywność, przez co mniejszym nakładem pracy można stworzyć ten sam program, unikając stosowania bardziej skomplikowanych języków.

Podczas realizacji projektu korzystano z dokumentacji języka Python [3].

3.2 PyOpenGL

W celu implementacji programu do wizualizacji automatu komórkowego, przejrzano dostępne narzędzia i zdecydowano się na wybór tego, które zapewni sprawny i względnie płynny proces wyświetlania komórek w poszczególnych stanach.

Wśród narzędzi współpracujących z językiem Python popularne są takie biblioteki graficzne, jak PyGame, Panda 3D, czy też PyOpenGL. Z wymienionych rozwiązań zdecydowano się na zastosowanie biblioteki - PyOpenGL. PyOpenGL jest to specyfikacja otwartego i uniwersalnego API (*ang. application programming interface* - interfejs programistyczny aplikacji) do tworzenia grafiki. Zawiera zestaw funkcji umożliwiających budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych.

Głównym założeniem tej biblioteki jest tworzenie grafiki komputerowej. Generowanie grafiki zachodzi o wiele szybciej niż w przypadku innych narzędzi. Efekt ten nazywa się przyspieszeniem sprzętowym, osiągnięty jest poprzez realizację poleceń przez procesor graficzny (GPU), przez co tworzenie grafiki następuje szybciej niż innymi sposobami.

Podczas realizacji projektu korzystano z dokumentacji zarówno biblioteki PyOpenGL, jak również biblioteki OpenGL [1, 2].

Rozdział 4

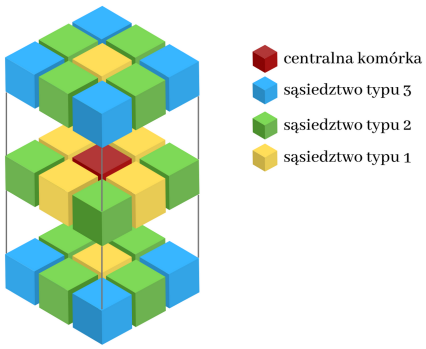
Techniczna realizacja projektu

4.1 Charakterystyka automatu komórkowego

Zaimplementowany automat komórkowy można zilustrować jako trójwymiarową, sześcienną siatkę, o wymiarach 100x100x100 komórek, z których każda reprezentuje jedną komórkę $CD4^{+}T$ ludzkiego układu immunologicznego. Poniżej przedstawiono oraz opisano niezbędne elementy zaimplementowanego automatu komórkowego, których właściwości określono, wzorując się na pracy [11] - omówionej w punkcie 2.2.

4.1.1 Sąsiedztwo komórki

W modelu stan każdej z komórek w kolejnej iteracji determinowany jest poprzez aktualny stan jej 26 sąsiadów. W sąsiedzctwie każdej z komórek wyróżniono trzy rodzaje sąsiedztwa:



Rysunek 4.1: Wizualizacja rodzajów sąsiedztwa komórki

- Sąsiedztwo typu I - tzw. najbliższe sąsiedztwo - należą do niego komórki, które kontaktują się z komórką centralną za pośrednictwem całej powierzchni ściany. Każda komórka posiada sześciu sąsiadów tego rodzaju.
- Sąsiedztwo typu II - tzw. drugie najbliższe sąsiedztwo - należą do niego komórki, które kontaktują się, z komórką centralną za pośrednictwem krawędzi. Każda komórka posiada dwunastu sąsiadów tego rodzaju.

- Sąsiedztwo typu III - tzw. trzecie najbliższe sąsiedztwo - należą do niego komórki, które kontaktują się z komórką centralną za pośrednictwem wierzchołka. Każda komórka posiada ośmiu sąsiadów tego rodzaju.

4.1.2 Przestrzeń stanów

Przestrzeń stanów określono poprzez zdefiniowanie czterech stanów. Każda z komórek automatu komórkowego może znajdować się w danej chwili w jednym ze zdefiniowanych stanów:

- Zdrowa (*ang. Healthy*) - reprezentuje zdrową komórkę $CD4^+T$ ludzkiego układu odpornościowego;
- Zainfekowana typu I (*ang. Infected I*) - reprezentuje zainfekowaną przez wirusa HIV komórkę $CD4^+T$ ludzkiego układu odpornościowego, która posiada nieograniczoną zdolność do rozprzestrzeniania infekcji;
- Zainfekowana typu II (*ang. Infected II*) - reprezentuje zainfekowaną przez wirusa HIV komórkę $CD4^+T$ ludzkiego układu odpornościowego, wobec której została rozwinięta odpowiedź układu immunologicznego;
- Martwa (*ang. Dead*) - reprezentuje zainfekowaną komórkę $CD4^+T$ ludzkiego układu odpornościowego, która została zwalczona na drodze odpowiedzi układu immunologicznego.

4.1.3 Warunki brzegowe

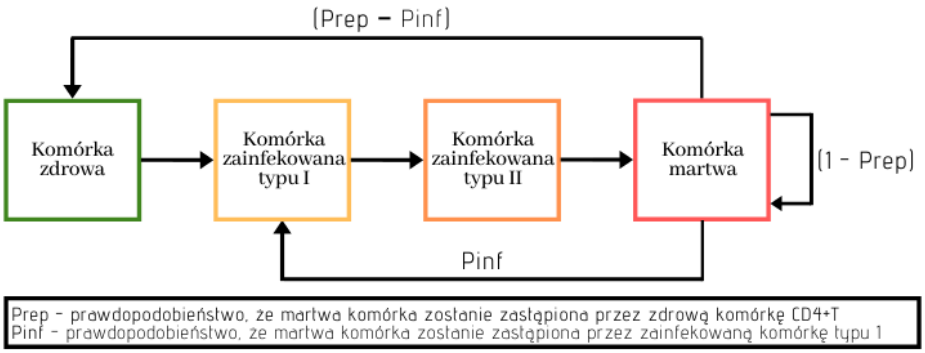
Warunki brzegowe opisywanego aparacie komórkowego zdefiniowano przy zastosowaniu warunku brzegowego Dirichleta (ang. fixed boundary codition).

Jest to warunek pierwszego rodzaju; znajdujący swoje zastosowanie w teorii równań różniczkowych zwyczajnych i cząstkowych. Polega on na założeniu, że funkcja będąca rozwiązaniem danego problemu musi przyjmować określone, z góry zadane wartości na brzegu dziedziny [5].

W przypadku automatu modelującego dynamikę infekcji wirusem HIV przekłada się to na założenie, że na brzegach trójwymiarowej siatki znajdują się tylko komórki w stanie zdrowym, a ich stan nie ulega zmianie w trakcie przeprowadzania symulacji.

4.1.4 Reguły przejść pomiędzy stanami oraz ich podstawy biologiczne

W celu sprecyzowania warunków, umożliwiających zmianę stanów komórek w kolejnych iteracjach przeprowadzanych symulacji, zdefiniowano następujący zbiór reguł przejść.



Rysunek 4.2: Schemat przedstawiający relacje pomiędzy możliwymi stanami komórek.

Reguła I

Jeśli zdrowa komórka spełnia przynajmniej jedną z zasad wymienionych poniżej, w nastepnej iteracji zmienia swój stan i staje się komórką zainfekowaną typu I:

a) w sytuacji gdy posiada przynajmniej jedną komórkę zainfekowaną typu pierwszego w swoim najbliższym lub drugim najbliższym sąsiedztwie,

b) w sytuacji gdy zachodzi koniunkcja trzech poniższych warunków.

Komórka posiada przynajmniej:

- x komórek zainfekowanych typu II w swoim najbliższym sąsiedztwie,
- y komórek zainfekowanych typu II w drugim najbliższym sąsiedztwie,
- x komórek zainfekowanych typu II w trzecim najbliższym sąsiedztwie.

Pierwsza z zasad (a) odzwierciedla rozprzestrzenianie się infekcji wirusem HIV poprzez kontakt, zanim układ immunologiczny rozwinie specyficzną odpowiedź na infekcję wirusową.

Druga zasada (b) odzwierciedla fakt, że komórki zainfekowane typu II, przed śmiercią, mogą zarazić komórki zdrowe, jeśli ich koncentracja przekracza wcześniej zdefiniowana wartość progową.

Wartość ta określona została przy użyciu parametrów: x, y, z, które zostały dobrane w taki sposób, aby uzyskać jak najlepszą zgodność z danymi, pochodzącymi z badań klinicznych.

W modelu przyjęto następujące wartości parametrów: $x = 5$, $y = 9$ i $z = 4$.

Reguła II

Każda komórka zainfekowana typu I staje się komórką zainfekowaną typu II w kolejnej iteracji symulacji.

Komórki zainfekowane typu II reprezentują prowirusy, przeciwko którym została rozwinięta odpowiedź układu odporonościowego, a ich zdolność do rozprzestrzenia infekcji została ograniczona.

Reguła III

Każda komórka zainfekowana typu II po τ iteracjach staje się komórką martwą.

Parametr τ reprezentuje czas potrzebny do rozwinięcia przez system immunologiczny specyficznej odpowiedzi, w celu uśmiercenia zainfekowanej komórki.

Zwany jest on również opóźnieniem czasowym, które pozwala na uzwględnianie w modelu faktu, że każda nowo zainfekowana komórka reprenzentuje inną odmianę wirusa. Pozwala to na uwzględnienie tempa mutacji wirusa, ponieważ gdy zdrowa komórka $CD4^+T$ zosataje zainfekowana przez wirusa HIV, wykorzystuje on komórkowe DNA w celu replikacji oraz transkrypcji.

Podczas każdej z transkrypcji może wystąpić błąd, powodując średnio jedną mutacje na pokolenie a w związku z czym produkcję nowego linie wirusa, o zróżnicowanej informacji genetycznej względem pozostałych wirionów.

Według badań okres ten trwa od 2 - 6 tygodni. W modelu przyjęto parametr $\tau = 2$, co oznacza, że komórka przechodzi w stan komórki martwej po przebyciu dwóch iteracji w stanie komórki zainfekowanej typu II, ponieważ każda iteracja symulacji oznacza pojedynczy tydzień infekcji.

Reguła IV

W kolejnej iteracji komórka martwa może zostać zastąpiona przez zainfekowaną komórkę typu I z prawdopodobieństwem równym P_{INF} (*ang. Probability of infection* - prawdopodobieństwo zakażenia) lub przez zdrową komórkę z prawdopodobieństwem równym P_{REP} (*ang. Probability of replenishment* - prawdopodobieństwo uzupełnienia). W pozostałych przypadkach komórka pozostaje martwa.

Możliwość zastąpienia komórki martwej komórką zdrową uzwględnia wysoki potencjał ludzkiego układu immunologicznego do odzyskiwania swojego potencjału obronnego po immunosupresji spowodowanej infekcją. Wprowadzenie tejże zasady do modelu umożliwia również uwzględnienie dyfuzji zdrowych komórek $CD4^+T$ w obrębie obszarów układu immunologicznego.

Drugi przypadek, opierający się o wprowadzenie w miejsce martwych komórek, nowo zainfekowanej komórki typu I, mogącej pochodzić zarówno z innych przestrzeni układu odpornościowego, jak i reprezentujące komórki zainfekowane, których aktywność dotychczas była uśpiona.

Wartość parametru P_{INF}, reprezentującego prawdopodobieństwo, że martwa komórka zostanie zastąpiona przez zainfekowaną komórkę typu I została wybrana, bazując na wynikach badań laboratoryjnych.

Ustalono, że białka pochodzące od wirusa ulegają ekspresji w jednej na 10^4 - 10^5 komórek $CD4^+T$, obecnych w krwi obwodowej zakażonego pacjenta. W modelu zastosowana wartość prawdopodobieństwa $P_{INF} = 10^{-5}$.

Prawdopodobieństwo P_{REP} oddaje wysoką umiejętność układu immunologicznego do odzyskiwania potencjału obronnego, poprzez produkcję zdrowych komórek $CD4^+T$. W modelu zastosowana wartość prawdopodobieństwa wynosi $P_{REP} = 0.99$.

4.1.5 Warunki początkowe

Wartość parametru P_{HIV} *(ang. Probability of infection by HIV).*, reprezentuje początkowe prawdopodobieństwo infekcji zdrowej komórki wirusem HIV.

Przed rozpoczęciem symulacji losowany jest początkowy stan każdej komórki poprzez sprawdzenie, czy wylosowana liczba mieści się w zakresie prawdopodbieństwa P_{HIV}. Jeżeli warunek ten jest spełniony, to komórka rozpoczyna symulację będąc komórką zainfekowaną typu I.

Wartość prawdopodobieństwa $P_{HIV} = 5 * 10^{-4}$, zastosowana w modelu, opiera się na obserwacjach eksperymentalnych, że tylko jedna na $10^2 - 10^3$ komórek $CD4^+T$ była nośnikiem wirusowego DNA w trakcie początkowej infekcji.

Opisany powyżej model poddaje się symulacji, przez zadaną liczbę iteracji (najczęściej 1050 iteracji = 20 lat).

W każdej z iteracji stan każdej z komórek oraz stany każdego z sąsiadów komórki podlegają sprawdzeniu, po czym na podstawie powyżej opisanych zasad dochodzi do wyznaczenia nowego stanu dla wszystkich komórek znajdujących się na siatce automatu komórkowego. Na końcu każdej iteracji dochodzi do automatycznego zaktualizowania stanów wszystkich komórek.

4.2 Implementacja automatu komórkowego

Program realizujący ideę opisywanego automatu komórkowego został napisany w oparciu o zasady programowania zorientowanego obiektowo.

Programowanie zorientowane obiektowo jest jednym z paradygmatów programowania, w którym programy definiuje się za pomocą obiektów, scharakteryzowanych przez zbiór pól, określających ich stan oraz metod, opisujących ich zachowanie. Obiekty komunikują się pomiędzy sobą w celu wykonywania zadań.

Podejście to różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Jedną z wielu zalet programowania zorientowanego obiektowo jest znacząco zwiększona kontrola nad projektem, jak również możliwość jego łatwego i szybkiego rozwoju oraz dokonywania zmian. Modularna budowa kodu sprawia, że wytwarzane oprogramowanie jest czytelne, a także pozwala na sprawne wprowadzanie późniejszych modyfikacji.

W realizowanym projekcie stworzono dwie główne klasy: Cell (Komórka) i World (Świat), odwzorowywujące ideę automatu komórkowego. Dodatkowo, w celu ułatwienia posługiwania sie prawdopodobieństwami, zdefiniowanymi w realizowanym modelu, stworzono klasę pomocniczą Probability (Prawdopodobieństwo).

4.2.1 Klasa BoundaryConditions (Warunki Brzegowe)

Jest to prosta klasa, będąca typem wyliczeniowym. Jest to rodzaj typu danych składający się z listy wartości, jakie może przyjąć zmienna tego typu, zdefiniowanych za pomoca tzw. literałów wyliczeniowych.

```
1 from enum import Enum
2
3 class BoundaryConditions(Enum):
4     periodic = 0
5     fixed = 1
```

Listing 4.1: Implementacja klasy BoundaryConditions

Jak widać na powyższej implementacji, klasa ta zawiera listę wartości, reprezentujących możliwe do zastosowania w modelu rodzaje warunków brzegowych. Literał ”0” oznacza wybór warunków brzegowych Dirichleta. Natomiast literał ”1” oznacza wybór warunków periodycznych (które ostatecznie nie zostały wykorzystane przy tworzeniu modelu realizującego dynamikę infekcji wirusem HIV).

Zdecydowano się na wykorzystanie typu wyliczeniowego do reprezentacji rodzajów warunków brzegowych, w celu zapewnienia większej czytelności kodu.

4.2.2 Klasa Cell (Komórka)

Obiekty klasy Cell reprezentują pojedyncze komórki automatu komórkowego - komórki $CD4^+T$ układu immunologicznego.

```
class Cell:
    def __init__(self,**kwargs):

    ### counting number of cell's neighbors in specific state ###
    def numberOfMatesInSpecificState(self,listOfCells,specificState):

    ### checks if cell is a border cell in x, y, z sized world ###
    def setIsBorderCell(self, xMax, yMax, zMax):
```

Listing 4.2: Implementacja klasy Cell

4.2.2.1 Konstruktor klasy Cell

```
def __init__(self,**kwargs):
    ### cordinates and basic parametrs of the cell ###
    self.myX = kwargs.get('myX', -1)
    self.myY = kwargs.get('myY', -1)
    self.myZ = kwargs.get('myZ', -1)

    # at the beginning all cells are healthy
    self.myState = kwargs.get('myState', 0)
    # newState is to update states of cells at the same time
    self.newState = self.myState

    self.numberOfI1Iterations = 1
    self.numberOfI2Iterations = 1
    self.isBorderCell = True

    ### cell's neighbors - list of the cell's neighbors which: ###
    self.wallMates = []  # contact with the cell
    self.lineMates = [] # touch the cell by line
    self.pointMates = [] # touch the cell by point
```

Listing 4.3: Konstruktor klasy Cell

Obiekty klasy Cell charakteryzują się zbiorem właściwości, określonym przez wartości zdefiniowanych pól:

1) **myX, myY, myZ** - zestaw pól, które odpowiadają współrzędnym komórki na siatce automatu komórkowego. Pola domyślnie przyjmują wartość -1, co ma na celu szybką likwidację ewentualnych błędów, ponieważ wartości współrzędnych komórki przyjmują wartość z zakresu $0 - 99$.

2) **myState** - pole przechowujące wartość liczbową, reprezentującą obecny stan komórki. Stany komórek w programie definiowane są kolejno przez cyfry z zakresu $0 - 3$, gdzie 0 oznacza zdrową komórkę, 1 oraz 2 definiuje odpowiednio komórki zainfekowane typu I i II, natomiast cyfra 3 reprezentuje martwe komórki. Pola domyślnie przyjmują wartość 0, co oznacza, że wszystkie obiekty klasy Cell w chwili inicjalizacji reprezentują komórki zdrowe.

3) **newState** - pole to przechowuje stan obiektu, jaki komórka przyjmie w następnej iteracji symulacji. Pole to umożliwia szybką i jednoczesną aktualizację stanu wszystkich komórek automatu komórkowego. Pola domyślnie przyjmują wartość równą wartości pola myState.

4) **numberOfI1Iterations** - pole przechowujące liczbę iteracji, w których komórka przebywa w stanie zainfekowanym typu I. Domyślnie pole przyjmuje wartość 1.

5) **numberOfI2Iterations** - pole przechowujące liczbę iteracji, w których komórka przebywa w stanie zainfekowanym typu II. Domyślnie pole przyjmuje wartość 1.

6) **isBorderCell** - pole przechowujące wartość domyślną, informującą, czy komórka jest położona na brzegu siatki automatu komórkowego. Pole o wartości True oznacza, że obiekt jest komórką graniczną, w przeciwnym wypadku pole przyjmuje wartość False. Pole inicjalizowane jest wartością True, ponieważ z założenia komórka istnieje w odosobnieniu.

7) **wallMates, lineMates, pointMates** - zestaw list zawierających referencje do sąsiadujących obiektów klasy Cell, z uwzględnieniem trzech różnych rodzajów sąsiedztwa, opisanych w punkcie 4.1.1. Listy domyślnie są puste.

4.2.2.2 Metody klasy Cell

Obiekty klasy Cell mogą wykazywać zbiór zachowań, zdefiniowanych za pomocą następujących metod klasy:

1) **numberOfMatesInSpecificState** - metoda zwraca liczebność komórek z listy obiektów Cell, znajdujących się w zadanym stanie. Przyjmuje dwa argumenty: cyfrę z zakresu $0 - 3$, reprezentującą stan komórek, których należy dokonać zliczenia oraz listę zawierającą referencje do obiektów klasy Cell, które mają być poddane zliczaniu.

   Rodzaje przyjmowanych argumentów zapewniają uniwersalność metody, ponieważ w zależności od potrzeb, metodę można wywoływać dla każdej z list: wallMates, lineMates, pointMates, co pozwala na efektywne zliczanie komórek, reprezentowanych przez określony stan.

2) **setIsBorderCell** - metoda, której wywołanie umożliwia określenie i ustawienie wartości jednego z pól klasy Cell - isBorderCell. Przyjmuje trzy argumenty - wartości maksymalnych współrzędnych komórki na siatce automatu komórkowego. Pole to jest kluczowym elementem do implementacji warunków brzegowych Dirichleta opisanych w punkcie 4.1.3.

4.2.3 Klasa Probability (Prawdopodobieństwo)

Obiekty klasy Probability są reprezentacją prawdopodobieństw zastosowanych w modelu, przedstawionych w punkcie 4.1.4.

```
1 from sys import exit
2
3 class Probability:
4     def __init__(self, **kwargs):
5
```

```
    ### brings probabilities to a common denominator ###
    def commonDenominator(self, secondProbability):
```

Listing 4.4: Implementacja klasy Probability

4.2.3.1 Konstruktor klasy Probability

Każdy z obiektów klasy Probability charakteryzowany jest za pomocą dwóch pól:

```
    def __init__(self, **kwargs):

    self.nominator = kwargs.get('nominator', -1)
    self.denominator = kwargs.get('denominator', -1)
```

Listing 4.5: Metody klasy Probability

1) **nominator** - pole, przechowujące wartość licznika prawdopodobieństwa.
2) **denominator** - pole, przechowujące wartość mianownika prawdopodobieństwa.

Domyślnie obu polom zostaje przypisana wartość -1. Działanie to ma na celu wykluczenie błędów, wynikających z braku zdefiniowania jednej z wartości podczas tworzenia obiektu klasy Probability.

4.2.3.2 Metody klasy Probability

Klasa Probability posiada jedną metodę, umożliwiającą sprawne posługiwanie się prawdopodobieństwami używanymi w modelu, poprzez zdefiniowanie przestrzeni zdarzeń elementarnych - wspólnego mianownika rozpatrywanych prawdopodobieństw.

```
### brings probabilities to a common denominator ###
def commonDenominator(self, secondProbability):

    if (self.denominator == 0 or secondProbability.denominator == 0):
        print('One of denominators is 0')
        exit()

    # denominators
    firstDenominator = self.denominator
    secondDenominator = secondProbability.denominator

    # nominators
    firstNominator = self.nominator
    secondNominator = secondProbability.nominator

    commmonDenominator = firstDenominator * secondDenominator

    # updating probabilities
    self.nominator = firstNominator * secondDenominator
    secondProbability.nominator = secondNominator * firstDenominator

    self.denominator = commmonDenominator
    secondProbability.denominator = commmonDenominator
```

Listing 4.6: Metody klasy Probability

Metoda commonDenominator przyjmuje jeden argument, będący obiektem klasy Probability. Zadaniem metody jest obliczenie wspólnego mianownika dla prawdopodobieństwa przekazywanego do metody oraz obiektu klasy Probability, na rzecz którego metoda została wywołana. Metoda dodatkowo sprowadza każde z prawdopodobieństw do wspólnego mianownika poprzez przedefiniowanie wartości ich pól, na podstawie nowo obliczonych wartości liczników, wyznaczonych dla każdego obiektu oraz wspólnego mianownika.

4.2.4 Klasa World (Świat)

Klasa World reprezentuje siatkę komórek automatu komórkowego, co w odniesieniu biologicznym imituje fragment węzła chłonnego, będącego strukturą układu immunologicznego.

```
from classes.Cell import *
from classes.Probability import *
from classes.Visualisation import *
from classes.BoundaryConditions import *
import random
from time import sleep, time
from _thread import start_new_thread
from threading import Semaphore, Thread
from os import path # library to operate on files
from sys import exit

class World:
    def __init__(self,**kwargs):

    ### creation of the world ###
    def createWorld(self):
    ### setting initially infected 1 cells ###
    def setStateOfInitialI1Cells(self):
    # creates semaphores to synchronize simulation & visualisation #
    def createSemaphoresForVisualisation(self):
    ### finds out the state of particular cell ###
    def findOutStateOfCell(self, cell):

    ### HEALTHY --> INFECTED 1 ###
    def checkIfHealthyBecomesI1(self, cell):
    ### INFECTED 1 --> INFECTED 2 ###
    def checkIfI1BecomesI2(self, cell):
    ### INFECTED 2 --> DEAD ###
    def checkIfI2BecomesDead(self, cell):
    ### DEAD --> INFECTED 1/HEALTHY ###
    def checkIfDeadBecomesI1OrHealthy(self, cell):

    ### setting cell's neighbours ###
    def setAllCellsMates(self):
    ### fills cell's neighborhoods ###
    def populateCellNeighborhood(self, cell):
    # adds neighbor to specific kind of neighborhood of the cells #
    def addNeighborToNeighborhoodOfCell(self, cell, neighborCell):

    ### simulation of world ###
    def simulateWorld(self):
    ### performs simulation of whole world ###
```

```
    def calculateWholeSimulation(self, visualisation):
    ### performs single iteration of simulation ###
    def singleIteration(self, visualisation, iterationNumber):
    ### finds out new state of all cells ###
    def findOutAllCellsStates(self):
    ### updating all cell's states at the same time ###
    def updateAllCellsStates(self):
    ### counting the cells in particular state on the grid ###
    def countCellsInSpecificState(self):

    ### creats directory for the results ###
    def createDirForResultFiles(self):
    # saves the results of sigle iteration  to the .txt file #
    def saveResultToFile(self, iteration, listOfCountedCellEachState):
    ### initializes .txt file for saving results of simulation ####
    def initializeFileToSaveResult(self):
```

Listing 4.7: Implementacja klasy World

4.2.4.1 Konstruktor klasy World

```
def __init__(self,**kwargs):

    ### simulation parameters ###
    self.rows = kwargs.get('rows', 100)
    self.cols = kwargs.get('cols', 100)
    self.layers = kwargs.get('layers', 100)

    self.numberOfCells = self.rows * self.cols * self.layers
    self.numberOfIterations = kwargs.get('numberOfIterations', 30)

    self.visualisation_ON = kwargs.get('visualisation_ON', False)
    self.saveSimulation_ON = kwargs.get('saveSimulation_ON', False)

    self.nameOfFile = kwargs.get('nameOfFile', "noNameFile" + str(
    random.randint(0,100))
    self.nameOfDirForResults = 'results_of_simulations'

    ### states ###
    self.healthy = 0
    self.infected1 = 1
    self.infected2 = 2
    self.dead = 3

    ### Initial conditions ###
    # from Alive to I1
    self.pHIV = kwargs.get('pHIV', [5, 10000])
    self.pInitialHIV = Probability(nominator = self.pHIV[0],
                                   denominator = self.pHIV[1])

    ### boundary conditions ###
    self.boundaryCondition = kwargs.get('boundaryCondition',
    BoundaryConditions.fixed)

    # RULES 1 FOR: Healthy Cell --> Infected 1 Cell
    self.numI1Cell_WallMates = kwargs.get('numI1Cell_WallMates', 1)
    self.numI1Cell_LineMates = kwargs.get('numI1Cell_LineMates', 1)
```

```
35
36      self.numI2Cell_WallMates = kwargs.get('numI2Cell_WallMates', 5)
37      self.numI2Cell_LineMates = kwargs.get('numI2Cell_LineMates', 9)
38      self.numI2Cell_PointMates = kwargs.get('numI2Cell_PointMates', 4)
39
40      # RULES 2 FOR: Infected 1 Cell --> Infected 2 Cell
41      self.numberOfIterationsInI1State = kwargs.get('
    numberOfIterationsInI1State', 1)
42
43      # RULES 3 FOR: Infected 2 Cell --> Dead Cell
44      self.numberOfIterationsInI2State = kwargs.get('
    numberOfIterationsInI2State', 2)
45
46      # RULES 4 FOR:
47      # probability that DEAD cell is replenished by HEALTHY cell
48      self.pRep = kwargs.get('pRep', [99, 100])
49      self.pReplenision = Probability(nominator = self.pRep[0],
50                                      denominator = self.pRep[1])
51
52      # probability that DEAD cell becomes INFECTED 1 cell
53      self.pInf = kwargs.get('pInf', [974, 100000000])
54      self.pInfection = Probability(nominator = self.pInf[0] ,
55                                    denominator = self.pInf[1])
56
57      # make common denominator
58      self.pInfection.commonDenominator(self.pReplenision)
59
60      # creating dir and file to save simulation
61      self.createDirForResultFiles()
62
63      # list of list of cells in specific state that should be displayed
64      # [0] - Infected1, [1] - Infected2, [2] - Dead
65      self.cellsListToDisplay = [[],[],[]]
66
67      # creation of World
68      self.cellsList = []
69      self.createWorld()
70      self.setStateOfInitialI1Cells()
71      self.setAllCellsMates()
72
73      # creating semaphores for visualisation & simulation
74      self.createSemaphoresForVisualisation()
```

Listing 4.8: Konstruktor klasy World

Klasa World posiada następujący zbiór pól:

1) **rows, cols, layers** - zestaw pól określających wielkość siatki automatu komórkowego w każdym z trzech wymiarów. Wartość każdego z pól domyślnie ustawiona zostaje na 100.

2) **numberOfCells** - pole przechowujące informacje o liczbie wszystkich komórek automatu komórkowego.

3) **numberOfIterations** - pole pozwalające na określenie liczby iteracji symulacji. Domyślnie symulacja opisywanego automatu komórkowego wykonywana jest w 1050 iteracjach.

4) **visualisation_ON** - pole przechowujące wartość typu logicznego. W przypadku wartości True, zostaje przedstawiona wizualizacja automatu komórkowego obrazując każdy z kroków symulacji. Domyślnie pole przyjmuje wartość False.

5) **saveSimulation_ON** - pole przechowujące wartość typu logicznego. W przypadku wartości True, zostaje tworzony folder oraz plik tekstowy, w których zostają zapisywane poszczególne kroki przeprowadzanej symulacji. Domyślnie pole przyjmuje wartość False.

6) **nameOfFile** - pole przechowujące łańcuch znaków, będacy nazwą pliku tekstowego z przebiegiem całej symulacji. Domyślnie plik nazywa się noNameFileX, gdzie X jest jest wylosowaną liczbą z zakresu $0 - 99$.

7) **nameOfDirForResults** - pole przechowujące łańcuch znaków, będący nazwą folderu, w którym przechowywane są wszystkie wynikowe pliki tekstowe. Domyślnie folder, przechowujący wyniki symulacji nazywa się results_of_simulations.

8) **healthy, infected1, infected2, dead** - pole reprezentujące liczbową interpretacje poszczególnych stanów komórek automatu komórkowego. Pola ustawiane są wewnątrz klasy, nie można ich modyfikować z poziomu konstruktora.

9) **pHIV, pRep, pInf** - zbiór pól, gdzie każde jest dwuelementową listą, zawierającą kolejno licznik oraz mianownik każdego z prawdopodobieństw. Domyślnie pola są kolejno określane przez następujące listy: [5, 10000], [99, 100] oraz [974, 100000000].

10) **pInitialHIV, pReplenision, pInfection** - zestaw pól, gdzie każde reprezentuje określone prawdopodobieństwo, jako obiekt klasy Probability.

11) **numI1Cell_WallMates, numI1Cell_LineMates** - pola, określające liczbę zainfekowanych komórek typu I, w sąsiedztwie typu I oraz II, potrzebnych do tego, aby zdrowa komórka w kolejnej iteracji stała się komórką zainfekowaną typu I. Domyśle obydwa pola przyjmuja wartość 1.

12) **numI2Cell_WallMates, numI2Cell_LineMates, numI2Cell_PointMates** - pola, określające liczbę zainfekowanych komórek typu II, w każdym z typów sąsiedztw komórki (parametry x, y, z), determinujących przejście zdrowej komórki w komórkę zainfekowaną typu I, w kolejnej iteracji. Domyślnie pola przyjmują następujące wartości: 5, 9, 4.

13) **cellsListToDisplay** - lista list obiektów klasy Cell, która umożliwia sprawny proces wizualizacji automatu komórkowego. Każda z wewnętrznych list zawiera referencje do obiektów klasy Cell, w konkretnych stanach. Pierwsza z list zawiera referencje do zainfekowanych komórek typu I, następna zainfekowanych typu II, natomiast ostatnia przechowuje referecje do komórek martwych.

14) **cellsList** - jest to lista list list komórek, przechowująca wszystkie komórki (obiekty klasy Cell) automatu komórkowego, odpowiednio w każdym z trzech wymiarów.

Dodatkowo konstruktor wywołuje zestaw metod, niezbędnych do inicjalizacji automatu komórkowego. Wszystkie metody zostały opisane szczegółowo w punkcie 4.2.4.2.

4.2.4.2 Metody klasy World

Klasa World posiada znacznie większą ilość funkcjonalności względem pozostałych klas. Wynika to z faktu, że obiekt tej klasy odpowiada za określanie wielu właściwości automatu komórkowego, tj.: stany komórek w kolejnych iteracjach symulacji, czy też przypisanie sąsiadów każdej z komórek, znajdującej się na siatce automatu komórkowego.

1) **createWorld**
   Metoda odpowiadająca za tworzenie oraz uzupełnienie listy cellsList obiektami klasy Cell.

2) **setStateOfInitialI1Cells**
   Metoda odpowiadająca za wylosowanie komórek zainfekowanych przez wirusa HIV, w oparciu o wartość prawdopodobieństwa P_{HIV} oraz nadanie odpowiedniej wartości (1, odpowiadającej komórka zainfekowanym typu I) polom myState oraz newState.

3) **createSemaphoresForVisualisation**
   Metoda odpowiadająca za stworzenie semaforów, służących do synchronizacji dwóch wątków programu: wizualizacji (w sytuacji, gdy użytkownik zadeklaruje chęć jej zobaczenia, $visualisation\_ON = True$) oraz symulacji, czyli procesu obliczeń stanów komórek w każdej kolejnej iteracji.

   Zastosowanie wielowątkowości umożliwia efektywniejsze działanie wizualizacji automatu komórkowego. Jednak dzieje się tak tylko ideowo, jako że interpreter języka Python wszystkie wątki wykonuje naprzemiennie i nie ma możliwości równoczesnego wykonywania kilku różnych wątków.

4) **findOutAllCellsStates**
   Metoda, której zadaniem jest określenie stanu komórki w kolejnej iteracji symulacji. Jej działanie sprowadza się do wywołania, w zależności od aktualnego stanu komórki, jednej z poniższych metod.

   - **checkIfHealthyBecomesI1**

```
###    HEALTHY --> INFECTED 1    ###
def checkIfHealthyBecomesI1(self, cell):
    if((self.boundaryCondition == BoundaryConditions.periodic)
        or (cell.isBorderCell == False)):
        if(cell.numberOfMatesInSpecificState(cell.wallMates,
            self.infected1) >= self.numI1Cell_WallMates
            # I1 cell, wall neighborhood
        or  cell.numberOfMatesInSpecificState(cell.lineMates,
            self.infected1) >= self.numI1Cell_LineMates
            # I1 cell, line neighborhood
        or (
            cell.numberOfMatesInSpecificState(cell.wallMates,
            self.infected2) >= self.numI2Cell_WallMates
            # I2 cell, wall neighborhood
        and cell.numberOfMatesInSpecificState(cell.lineMates,
            self.infected2) >= self.numI2Cell_LineMates
            # I2 cell, line neighborhood
        and cell.numberOfMatesInSpecificState(cell.pointMates
            ,self.infected2) >= self.numI2Cell_PointMates
            # I2 cell, point neighborhood
```

```
21                 )
22             ):
23             cell.newState = self.infected1
24             cell.numberOfI1Iterations = 1
```

Listing 4.9: Metoda klasy World - checkIfHealthyBecomesI1

Metoda zostaje wywoływana w sytuacji, gdy aktualny stan komórki to zdrowa. Sprawdza ona, czy zostały spełnione warunki Reguły I, opisanej w punkcie 4.1.4. Metoda ustawia pole newState komórki - przyjmowanej jako argument funkcji - na wartość 1, co oznacza, że w kolejnej iteracji komórka przechodzi w stan zainfekowanej typu I.

- **checkIfI1BecomesI2**

```
1 ### INFECTED 1 --> INFECTED 2 ###
2 def checkIfI1BecomesI2(self, cell):
3     if cell.numberOfI1Iterations >=
4     self.numberOfIterationsInI1State:
5         cell.newState = self.infected2
6         cell.numberOfI2Iterations = 1
7     else:
8         cell.numberOfI1Iterations = cell.numberOfI1Iterations+1
```

Listing 4.10: Metoda klasy World - checkIfI1BecomesI2

Metoda realizująca regułę II, punktu 4.1.4. Metoda zmienia wartość pola newState komórki - przyjmowanej jako argument funkcji - na wartość 2, co oznacza, że w kolejnej iteracji komórka przechodzi w stan zainfekowanej typu II.

- **checkIfI2BecomesDead**

```
1 ### INFECTED 2 --> DEAD ###
2 def checkIfI2BecomesDead(self, cell):
3     if cell.numberOfI2Iterations >= self.
    numberOfIterationsInI2State:
4         cell.newState = self.dead
5     else:
6         cell.numberOfI2Iterations = cell.numberOfI2Iterations+1
```

Listing 4.11: Metoda klasy World - checkIfI2BecomesDead

Metoda realizująca regułę III, punktu 4.1.4. Metoda zmienia wartość pola newState komórki - przyjmowanej jako argument funkcji - na wartość 3, co oznacza, że w kolejnej iteracji komórka staje się komórką martwą.

- **checkIfDeadBecomesI1OrHealthy**

```
1 ### DEAD --> INFECTED 1/HEALTHY ###
2 def checkIfDeadBecomesI1OrHealthy(self, cell):
3     randomProbability = random.randrange(0, self.pInfection.
    denominator)
4
5     # DEAD --> INFECTED 1
6     if randomProbability < self.pInfection.nominator:
7         cell.newState = self.infected1
8     # DEAD --> HEALTHY
9     # (Prep-Pinf) remaining probability
10    elif randomProbability < self.pReplenision.nominator:
11        cell.newState = self.healthy
```

```
12
```

Listing 4.12: Metoda klasy World - checkIfDeadBecomesI1OrHealthy

Metoda realizująca regułę IV, punktu 4.1.4. W ciele metody generowane jest losowe zdarzenie (liczba z zakresu, ograniczonego z góru przez wartość wspólnego mianownika rozpatrywanych w modelu prawdopodobieństw). W zależności od wyniku losowania, metoda może ustawić pole newState komórki przyjmowanej jako argument funkcji na wartość 0, 1 lub pozostawić je niezmienione.

5) **setAllCellsMates**
Metoda odpowiedzialna za zapełnienie list sąsiadów (wallMates, pointMates, lineMates), każdej z komórek znajdujacej się na trójwymiarowej siatce automatu komórkowego, zgodnie z podziałem określonym przez warunki sąsiedztwa, opisane w punkcie 4.1.1. Opiera się o wywołanie dla każdej komórki z listy $self.cellsList$ metody populateCellNeighborhood, opisanej poniżej.

6) **addNeighborToNeighborhoodOfCell**

```
1  ### adds neighbor to specific kind of neighborhood of the cells ###
2  def addNeighborToNeighborhoodOfCell(self, cell, neighborCell):
3      # number of different indices for specific neighbors:
4      wallNeighbors = 1
5      lineNeighbors = 2
6      pointNeighbors = 3
7
8      differentIndexes = (neighborCell.myZ != cell.myZ) + (
       neighborCell.myX != cell.myX) + (neighborCell.myY != cell.myY)
9      if differentIndexes == pointNeighbors:
10         cell.pointMates.append(neighborCell)
11     elif differentIndexes == lineNeighbors:
12         cell.lineMates.append(neighborCell)
13     elif differentIndexes == wallNeighbors:
14         cell.wallMates.append(neighborCell)
```

Listing 4.13: Metoda klasy World - addNeighborToNeighborhoodOfCell

Metoda, której zadaniem jest umieszczanie przekazanego jako argument obiektu klasy Cell w odpowiedniej liście sąsiadów.

Algorytm jej działania polega na sprawdzeniu, ile współrzędnych komórki sąsiedniej różni się od współrzędnych komórki centralnej.

Przykładowo komórka sąsiadująca całą powierzchnią ściany z inną komórką, leży w tej samej warstwie oraz tym samym wierszu, bądź kolumnie co ta komórka. W związku z czym liczba różnych współrzędnych wynosi 1 (zmienna wallNeighbors). Komórki sąsiadujące przez krawędź leżą tylko w tej samej warstwie, różniąc się wartością dwóch współrzędnych. Komórki sąsiadujące przez wierzchołek nie posiadają żadnych wspólnych wartości.

Sprawdzaniu ile różnych wierzchołków posiada sąsiadująca komórka służy zmienna differentIndexes, która jest sumą trzech warunków, z których każdy sprawdza jedną ze współrzędnych komórki. Na podstawie jej wartości komórka jest umieszczana na odpowiedniej liście sąsiedztwa komórki cetralnej.

7) **populateCellNeighborhood**

```
### fills cell's neighborhoods ###
def populateCellNeighborhood(self, cell):
    l = cell.myZ
    r = cell.myX
    c = cell.myY

    lRange = list(range(l-1, l+2))  # create range [first, last)
    rRange = list(range(r-1, r+2))
    cRange = list(range(c-1, c+2))

    if l+1 == self.layers: # checking if value is not out of range
        lRange[2] = 0
    if r+1 == self.rows:
        rRange[2] = 0
    if c+1 == self.cols:
        cRange[2] = 0

    for ll in lRange:
        for rr in rRange:
            for cc in cRange:
              neighborCell = self.cellsList[ll][rr][cc]
              self.addNeighborToNeighborhoodOfCell(cell,
  neighborCell)
```

Listing 4.14: Metoda klasy World - populateCellNeighborhood

Metoda przyjmuje jako argument obiekt klasy Cell. Odpowiada za odnalezienie wszytkich sąsiadów każdej z komórek.

Algorytm metody opiera się na pobraniu współrzędnych przekazanej komórki. Następnie na ich podstawie definiowane są zmienne $lRange$ (sąsiedzi w wartswach), $rRange$ (sąsiedzi w wierszach) oraz $cRange$ (sąsiedzi w kolumnach) przechowujące wstępne listy indeksów sąsiadów. W każdej liście znajdują się trzy wartości. W zależności od wymiaru, dla którego definiowana jest konkretna lista, jej drugi element jest wartością współrzędnej komórki, którą przekazano do ciała metody. Pozostałe dwa elementy są indeksami danego wymiaru, które określają położenie sąsiadów komórki. Obliczane są poprzez odjęcie lub dodanie wartości 1 od współrzednej komórki centralnej. Przykładowa, jeśli komórka centralna leży w czwartej warstwie siatki automatu komórkowego, to jej sąsiedzi będą znajdować się w wartwie trzeciej, czwartej oraz piątej, a więc zmienna $lRange$ w tej sytuacji będzie wyglądać następująco $lRange = [3, 4, 5]$.

Należy jednak zwrócić uwagę na wartości graniczne współrzędnych komórki. Zakładając, że wartość drugiego elementu jest równa $k = 0$, pierwszy z elementów tablicy przyjme wartość $k - 1 = -1$. Może wydawać się to problematyczne, jednak język Python umożliwia indeksowanie tablicy od tyłu liczbami ujemnymi, w związku z czym taka komórka, będzie miała sąsiadów po przeciwnej stronie siatki.

Może również zdarzyć się sytuacja, w której jeden z indeksów komórki będzie równy $k = 99$, co przy wykonaniu działania $k + 1 = 100$, spowoduje wykroczenie poza obszar listy cellsList, ponieważ listy w jeżyku Python indeksowane są od wartości zero. Taka sytuacja spowoduje błąd programu. Aby uniknąć wystąpienia opisanego problemu, zaimplementowano w kodzie przedstawionym na listingu 7, linie 11-16. Sprawdzają one, czy wartość jakiejkolwiek współrzędnej zwiększonej o jeden wykracza poza zakres tablicy, jeśli tak, w odpowiedniej liście, przechowującej zakres

indeksów sąsiadów, podmienia ostatnią wartość na 0. Tym sposobem komórka leżąca na końcu siatki posiada sąsiadów po jej przeciwległej stronie.

Ostatecznie, przy użyciu trzech pętli for, przeszukuje się listę wszystkich komórek siatki automatu komórkowego (cellsList) i przy pomocy metody addNeighborToNeighborhoodOfCell przedstawionej na listingu 6, umieszcza się kolejno sąsiadujące komórki w odpowiadnich listach sąsiadów komórki.

8) **simulateWorld**

```
### simulation of world ###
def simulateWorld(self):
    if self.visualisation_ON:
        visualisation=Visualisation(
                    displaySemaphore=self.displaySemaphore,
                    simulationSemaphore=self.simulationSemaphore,
                    cellsList=self.cellsList,
                    cellsListToDisplay = self.cellsListToDisplay
                    )
        try:
            thread = Thread(
                target = self.calculateWholeSimulation,
                args = (visualisation, )
                )
            thread.start()
        except:
            print("Unable to start thread for simulation")
            exit()

        visualisation.startDisplayingWorld()
    else:
        self.calculateWholeSimulation(None)
```

Listing 4.15: Metoda klasy World - simulateWorld

Metoda, służąca do uruchamiania symulacji automatu komórkowego. Jeśli w konstruktorze klasy World wartość argumentu $visualisation\_ON = True$, w ciele metody tworzony jest obiekt klasy Visualisation. Następnie w bloku try-catch tworzony jest nowy wątek programu, który wywołuje metodę calculateWholeSimulation, po czym wywoływana jest metoda klasy Visualisation, odpowiadająca za rozpoczęcie wizualizacji. Gdy wizualizacja nie jest wyświetlana, wywoływana jest funkcja kolejno obliczająca kroki symulacji.

9) **singleIteration**

```
### performs single iteration of simulation ###
def singleIteration(self, visualisation, iterationNumber):
    if self.saveSimulation_ON:
        countedCellsInEachState = self.countCellsInSpecificState()
        self.saveResultToFile(iterationNumber,
   countedCellsInEachState)

    if self.visualisation_ON:
        self.simulationSemaphore.acquire()

    try:
        loopTime = time()

```

```
13          self.findOutAllCellsStates()
14
15          self.updateAllCellsStates()
16
17          loopTime = time() - loopTime
18          print(iterationNumber, " loop time: ", loopTime)
19
20          if self.visualisation_ON:
21              visualisation.refreshDisplay(self.cellsListToDisplay)
22
23      finally:
24          if self.visualisation_ON:
25              self.displaySemaphore.release()
```

Listing 4.16: Metoda klasy World - singleIteration

Metoda wykonująca pojedynczą iterację symulacji. W ciele metody dochodzi do zapisywania informacji odnośnie liczby komórek w poszególnych stanach do pliku tekstowego. Jeśli w trakcie uruchomienia programu zachodzi jednoczesna wizualizacja semafor, symulacji zostaje przejęty, przez co zachodzi obliczenie nowych stanów wszystkich komórek, a następnie do klasy wizualizacji zostaje wydane polecenie odświeżania okranu. Ostatecznie następuje zwolnienie semafora wyświetlania.

10) **calculateWholeSimulation**
Metoda, w której to w pętli obliczane są kolejne kroki sumulacji automatu komórkowego, z zastosowaniem metody singleIteration.

11) **findOutAllCellsStates**
Metoda, której zadaniem jest przemierzenie całej listy komórek siatki automatu komórkowego i określenie nowego stanu każdej z nich w kolejnej iteracji, poprzez wywołanie każdorazowo metody findOutStateOfCell.

12) **updateAllCellsStates**

```
1  ### updating all cell's states at the same time ###
2  def updateAllCellsStates(self):
3      if self.visualisation_ON:
4          self.cellsListToDisplay = [[], [], []]
5
6      for layer in self.cellsList:
7        for row in layer:
8         for cell in row:
9              cell.myState = cell.newState
10             # dont draw healthy cells
11             if self.visualisation_ON and cell.myState:
12               self.cellsListToDisplay[cell.myState-1].append(cell)
```

Listing 4.17: Metoda klasy World - updateAllCellsStates

Metoda uaktualniająca wartość pola myState na podstawie wartości przechowywanej przez pole newState. Działanie to ma na celu jednoczesne uaktualnienie stanów wszystkich komórek. Dodatkowo aktualizowana jest lista komórek do wyświetlenia - cellsListToDisplay.

13) **countCellsInSpecificState**
Metoda zliczająca liczbę komórek w każdym ze stanów. Zwraca ona listę czterech

wartości, gdzie pod indeksem odpowiadającemu stanowi komórki znajduje się liczba komórek w zadanym stanie.

```
### counting the cells in particular state on the grid ###
def countCellsInSpecificState(self):
  stateCounter = [0, 0, 0, 0] # H, I1, I2, D
  for layer in self.cellsList:
     for row in layer:
       for cell in row:
           stateCounter[cell.myState]=stateCounter[cell.myState]+1
  return stateCounter
```

Listing 4.18: Metody klasy World - countCellsInSpecificState

14) **createDirForResultFiles**
Metoda, której zadaniem jest utworzenie folderu, w którym będą mieściły się wszystkie wynikowe pliki tekstowe.

15) **saveResultToFile**
Metoda, zapisująca ilość danych komórek oraz numer iteracji do pliku tekstowego.

16) **initializeFileToSaveResult**
Metoda inicjalizująca plik do zapisu danych wynikowych z symulacji.

4.3 Testowanie automatu komórkowego

Program realizujący automat komórkowy, modelujący dynamikę infekcji wirusem HIV, został skrupulatnie sprawdzony pod względem poprawności jego działania. Przetestowano następujące aspekty logiki działania programu:

- reguły przejść komórek pomiędzy poszczególnymi stanami,
- wybór odpowiednich sąsiadów, wraz z uwzględnieniem rodzaju sąsiedztwa,
- prawidłowe uaktualnienie bieżących stanów komórek w ostatniej fazie każdej iteracji.

Dla każdego z wymienionych celów testów, zrealizowano sprawdzanie z zastosowaniem czterech różnych metod:

- Testy automatyczne - specjalny skrypt testowy, służący do automatycznej weryfikacji, sprawdzający wybrany fragment programu.
- Testy manualne - sprawdzanie stanów komórek świata w każdej iteracji na podstawie wypisywanych zmiennych obiektu klasy World, a dokładniej listy przechowującej wszytkie komórki (obiekty klasy Cell). Każdą z komórek reprezentowano przez cyfrę z zakresu od zera do trzech (imitująca stan komórki), wypisywaną w konsoli systemowej, w celu weryfikacji poprawności stanu komórki w danej iteracji symulacji.
- Testy wizualne - sprawdzanie stanów komórek świata w każdej iteracji na podstawie wizualizacji automatu komórkowego.
- Rewizja kodu - kompletny przegląd zaimplementowanego rozwiązania przez dodatkowe osoby, posiadające doświadczenie w dziedzinie programowania.

4.4 Wizualizacja automatu komórkowego

Dodatkowym celem niniejszego projektu było stworzenie programu wizualizującego automat komórkowy realizujący rozwój dynamiki infekcji wirusem HIV. Głównym problemem podczas realizacji tego etapu projektu było znalezienie odpowiedniego narzędzia, które pozwoli na względnie szybkie i płynne, graficzne przedstawienie miliona komórek o trójwymiarowej strukturze, w poszczególnych iteracjach symulacji. Ostatecznie zdecydowano się na zastosowanie biblioteki PyOpenGL, krótko opisanej w rozdziale 3.

Trójwymiarowa siatka automatu komórkowego reprezentuje obszar węzła chłonnego, będącego elemetem ludzkiego układu immunologicznego, który podczas wizualizacji zobrazowany został w postaci dużego sześcianu, zbudowanego za pomocą białych krawędzie.

Natomiast komórki automatu komórkowego, będące reprezentacją komórek $CD4^{+}T$ ludzkiego układu immunologicznego, są reprezentowane jako małe sześciany. Kolor sześcianu zależy od stanu w jakim aktualnie znajdują się komórka, która dany sześcian reprezentuje.

Komórki zdrowe są niewidoczne w trakcie symulacji, komórki zainfekowane kolejno typu I i II, w trakcie symulacji, reprezentowane są przez sześciany koloru żółtego i pomarańczowego, natomiast komórki martwe wyświetlane są jako czerwone sześciany.

4.4.1 Klasa Visualisation (Wizualizacja)

Klasa Visualisation zawiera zestaw niezbędnych pól oraz metod, pozwalających na realizację postawionego celu, jakim jest zobrazowanie wyglądu automatu komórkowe w poszczególnych iteracjach przeprowadzanych symulacji.

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from OpenGL.arrays import vbo
from time import sleep, time
from threading import Semaphore
from array import array
from numpy import array as numpyArray
from numpy import uint32

class Visualisation():
    def __init__(self,**kwargs):

    def InitGL(self):

    ### --- Creates all cubes vertices as VBO arrays --- ###
    def initCubes(self):

    def initVbos(self):

    def initVertexVboWithColor(self, vertexesColors):

    def initCellsSurfaces(self):

    def createSurfacesForCell(self, x, y, z):

    def ReSizeGLScene(self, Width, Height):
```

```
28
29      def idle(self):
30
31      def displayWorld(self):
32
33      def drawBigCube(self):
34
35      ### --- draws cubes using VBO --- ###
36      def drawSmallCubes(self, indexesToDisplay, currentVertexVbo):
37
38      def startDisplayingWorld(self):
39
40      def refreshDisplay(self, cellsListToDisplay):
41
```

Listing 4.19: Implementacja klasy Visualisation

4.4.1.1 Konstruktor klasy Visualisation

```
1  def __init__(self,**kwargs):
2
3      self.windowWidth = kwargs.get('windowWidth', 800)
4      self.windowHeight = kwargs.get('windowHeight',600)
5      self.cellsInAxis = kwargs.get('cellsInAxis', None)
6      self.lenBigCube = kwargs.get('bigCubeLength', 3)
7      self.lenSmallCube = self.lenBigCube / self.cellsInAxis
8
9      self.cellsListToDisplay = kwargs.get('cellsListToDisplay', None)
10
11     self.simulationSemaphore = kwargs.get('simulationSemaphore', None)
12     self.displaySemaphore = kwargs.get('displaySemaphore', None)
13
14     self.numberOfVertexesPerCube = 8
15     # coefficients needed to find proper vertex position in VBO
16     self.xCoefficient = self.numberOfVertexesPerCube*self.cellsInAxis*
17                         self.cellsInAxis
18     self.yCoefficient = self.numberOfVertexesPerCube*self.cellsInAxis
19     self.zCoefficient = self.numberOfVertexesPerCube
20
21     self.xTranslation = +self.lenBigCube/8
22     self.yTranslation = -self.lenBigCube/2
23     self.zTranslation = -3 * self.lenBigCube
24
25     self.currentRotation = 300.0
26     self.rotatingSpeed = 00.0
27
28     self.colBigCube = [255, 255, 255]
29
30     self.colAInf1 = [[0.6,0.6,0],[0.9,0.9,0],[0.9,0.9,0],[0.8,0.8,0],
31                      [0.8,0.8,0],[1,1,0],[0.95, 0.95,0],[1,1,0]]
32     self.colAInf2 = [[0.6,0.3,0],[0.9,0.5,0],[0.9,0.5,0],[0.8,0.4,0],
33                      [0.8,0.4,0],[1,0.6,0],[0.95,0.55,0],[1,0.6,0]]
34     self.colADead = [[0.6,0,0],[0.9,0,0],[0.9,0,0],[0.8,0,0],
35                      [0.8,0,0],[1,0,0],[0.95,0,0],[1,0,0]]
36     self.colAList = [self.colAInf1, self.colAInf2, self.colADead]
37
38     self.bigCubeEdgesIndexes = array('B', [0,1, 0,3, 0,4, 2,1,
```

```
                                        2,3, 2,7, 6,3, 6,4, 6,7,
                                        5,1, 5,4, 5,7]).tobytes()
    self.numberOfEdgesVertexes = 24
    self.byteStrideBetweenVertexes = 24
    # big cube vertices
    self.verticesBigCubeA = array('f',[
            self.lenBigCube,0,0,
            self.lenBigCube,self.lenBigCube,0,
            0,self.lenBigCube,0,
            0,0,0,
            self.lenBigCube,0,self.lenBigCube,
            self.lenBigCube,self.lenBigCube,self.lenBigCube,
            0,0,self.lenBigCube,
            0,self.lenBigCube,self.lenBigCube
            ]).tobytes()

    self.cellsSurfacesIndexes = None
    self.vertexVbos = None
```

Listing 4.20: Konstruktor klasy Visualisation

Obiekt klasy Visualisation charakteryzuje się następującymi polami:

1) **windowWidth, windowHeight** - pola określające kolejno szerokość oraz wysokość okna symulacji.

2) **lenBigCube** - pole określające długość boku dużego sześcianu, reprezentującego obszar węzła chłonnego układu immunologicznego.

3) **lenSmallCube** - pole precyzujące długość boku małych sześcianów, reprezentujących komórki układu immunologicznego. Pole obliczane jest na podstawie stosunku długości boku dużego sześcianu oraz ilości symulowanych komórek w jednym wymiarze.

4) **cellsInAxis** - pole określające długość boku dużego sześcianu, określone za pomocą liczby komórek w jednym z wymiarów trójwymiarowego obszaru węzła chłonnego.

5) **cellsListToDisplay** - pole reprezentujące listę, zawierającą trzy listy wewnętrzne, gdzie każda zawiera kolejno referencje do obiektów klasy Cell, kolejno w trzech stanach: komórki zainfekowane typu I, komórki zainfekowane typu 2, komórki martwe. Utworzenie takiej listy umożliwia szybszy proces wizualizacji.

6) **simulationSemaphore** - semafor zwalniany przez wizualizację i przejmowany przez symulację, w celu wykonania obliczeń.

7) **displaySemaphore** - semafor zwalniany przez symulację i przejmowany przez wizualizację, w celu wyświetlania aktualnego stanu świata.

8) **numberOfVertexesPerCube** - liczba wierzchołków przypadająca na pojedynczy sześcian.

9) **xCoefficient, yCoefficient, zCoefficient** - współczynniki odpwiadające za przesunięcie wierzchołków kolejnych ścian sześcianów.

10) **xTranslation, yTranslation, zTranslation** - wartości przesunięcia obrazu, w celu wyświetlenia obiektu na środku okna wizualizacji.

11) **currentRotation** - kąt obrotu wszystkich sześcianów.

12) **colBigCube** - lista zawierająca kolory krawędzi dużego sześcianu.

13) **colAInf1** - lista zawierająca kolory wierzchołków sześcianów, reprezentujących komórki zainfekowane typu I.

14) **colAInf2** - lista zawierająca kolory wierzchołków sześcianów, reprezentujących komórki zainfekowane typu II.

15) **colADead** - lista zawierająca kolory wierzchołków sześcianów, reprezentujących komórki martwe.

16) **colAList** - lista zawierająca listy, reprezentujące kolory wierzchołków sześcianów dla poszczególnych stanów komórek.

17) **bigCubeEdgesIndexes** - tablica indeksów wierzchołków, odpowiadających kolejno wyświetlanym liniom (krawędziom) dużego sześcianu, przekonwertowana na ciąg bajtów, w celu uniknięcia każdorazowej konwersji przy wyświetlaniu.

18) **numberOfEdgesVertexes** - całkowita liczba wierzchołków, wykorzystywana do rysowania krawędzi dużego sześćianu.

19) **byteStrideBetweenVertexes** - liczba bajtów jaką zajmuje pojedynczy wierzchołek małego sześcianu w (*ang. Vertex Buffer Object* - obiekt bufora wierzchołków).

20) **verticesBigCubeA** - tablica wierzchołków dużego sześcianu przekonwertowana na ciąg bajtów, w celu uniknięcia każdorazowej konwersji przy wyświetlaniu.

21) **cellsSurfacesIndexes** - lista list list, zawierająca tablice indeksów wierzchołków dla kolejnych ścian każdego sześcianu w VBO.

22) **vertexVbos** - lista zawierająca VBO dla każdego stanu (koloru). W każdym VBO znajduje się tablica wszystkich wierzchołków oraz danego koloru każdej z symulowanych komórek.

4.4.1.2 Metody klasy Visualisation

Klasa Visualisation posiada następujące metody:

1) **InitGL** - metoda inicjalizująca okno aplikacji oraz VBO potrzebne do wyświetlania sześcianów.

2) **initCubes** - metoda inicjalizująca listę obiektów VBO oraz indeksów sześcianów potrzebnych do wyświetlania.

3) **initVbos** - metoda inicjalizująca zmienną vertexVbos,

4) **initVertexVboWithColor** - metoda tworząca VBO dla koloru podanego w argumencie wejściowym metody.

5) **initCellsSurfaces** - metoda inicjalizująca zmienną cellsSurfacesIndexes.

6) **createSurfacesForCell** - metoda inicjalizująca tablicę indeksów wierzchołków dla sześcianu określonego przez argumenty wejściowe: x, y, z.

7) **idle** - metoda wywoływana w stanie spoczynku pętli wyświetlającej.

8) **displayWorld**

```
def displayWorld(self):
    # take semaphore to display world, so simulation will have to
   wait
    self.displaySemaphore.acquire()
    try:
        displayTime = time()

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        self.drawBigCube()

        glLoadIdentity()
        glTranslatef(self.xTranslation,
                     self.yTranslation,
                     self.zTranslation)
        glRotatef(self.currentRotation, 0, 1, 0)

        for cellsInOneState, currentVertexVbo
                in zip(self.cellsListToDisplay,self.vertexVbos):

            # check if list not empty
            if cellsInOneState:

                indexesToDisplay = array('I', [])
                for cell in cellsInOneState:
                   if cell.isBorderCell or not all(cell.wallMates):
                   # print border cell/any neighbour if is healthy
                      indexesToDisplay.extend(
                      self.cellsSurfacesIndexes[cell.myX]
                                               [cell.myY]
                                               [cell.myZ])

                currentVertexVbo.bind()
                self.drawSmallCubes(indexesToDisplay,
                                    currentVertexVbo)
                currentVertexVbo.unbind()

        glutSwapBuffers()
        displayTime = time(vertexvertex) - displayTime
        print("display time: ", displayTime)

    finally:
    # end of displaying world, release semaphore for simulation
        self.simulationSemaphore.release()
```

Listing 4.21: Metody klasy Visualisation - displayWorld

Metoda wyświetlająca aktualny stan symulowanego obszaru węzła chłonnego.

9) **drawBigCube** - metoda odpowiadająca za wyświetlenie dużego sześcianu.

10) **drawSmallCubes** - metoda odpowiadająca za wyświetlenie małych sześcianów w danym stanie, określonych poprzez argumenty funkcji.

11) **startDisplayingWorld**

```
def startDisplayingWorld(self):
    # take simulation semaphore to initialize itself and display
   first step
    self.simulationSemaphore.acquire()

    self.InitGL()

    glutDisplayFunc(self.displayWorld)
    glutIdleFunc(self.idle)

    glutMainLoop()
```

Listing 4.22: Metody klasy Visualisation - startDisplayingWorld

Metoda wywoływana w celu rozpoczęcia wizualizacji automatu komórkowego. Musi zostać wywołana w głównym wątku programu.

12) **refreshDisplay**

```
def refreshDisplay(self, cellsListToDisplay):
    self.cellsListToDisplay = cellsListToDisplay
    glutPostRedisplay()
```

Listing 4.23: Metody klasy Visualisation - refreshDisplay

Metoda odpowiadająca za dodanie żądania odświeżenia okna wizualizacji, zgodnie z listą komórek do wyświetlenia przekazaną w argumencie.

4.4.1.3 Zastosowane usprawnienia

W celu zapewnienia względnie szybkiego wyświetlania automatu komórkowego, starano się w pełni wykorzystać potencjał użytej biblioteki oraz zastosować kilka usprawnień, aby zapewnić szybsze odświeżanie okna wizualizacji.

Wyświetlanie komórek jest realizowane za pomocą wywołań funkcji glDrawElements z biblioteki PyOpenGL. Wyświetlanie rozpoczyna się od wyrysowania dużego sześcianu za pomocą zmiennych bigCubeEdgesIndexes i verticesBigCubeA. Przy pomocy jednego wywołania funkcji glDrawElements, uruchamianej w trybie rysowania linii, pomiędzy każdymi kolejnymi dwoma wierzchołkami jest rysowana prosta. Natomiast przy rysowaniu małych sześcianów funkcja jest wywoływana w trybie rysowania powierzchni - każde kolejne cztery wierzchołki, reprezentują jedną powierzchnie (bok sześcianu).

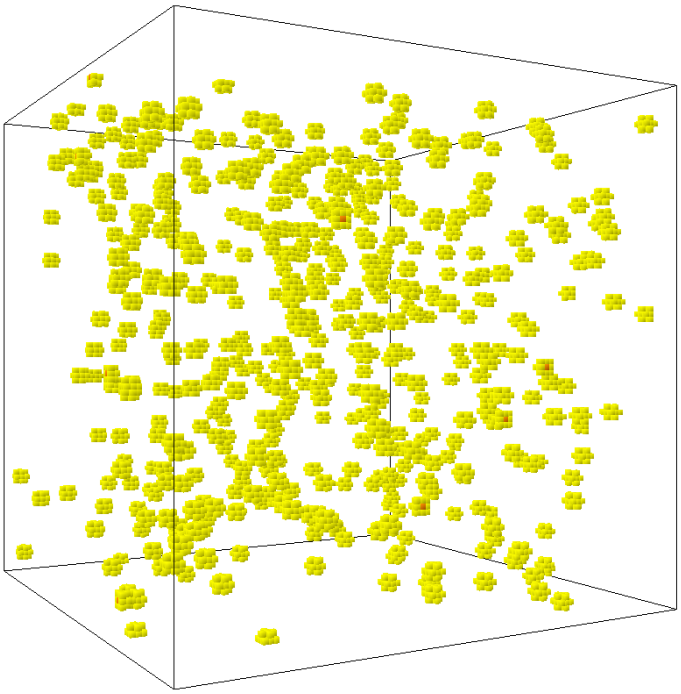
Dodatkowo, zamiast tablicy z wierzchołkami, kopiowanej przy każdym wywołaniu glDrawElements, jak ma to miejsce w przypadku dużego sześcianu, funkcja ta wykorzystuje obiekt bufora wierzchołków (dalej zwane VBO - Vertex Buffer Obcject).

Podejście to umożliwia znaczne zmniejszenie czasu wyświetlenia pojedynczej klatki (iteracji symulacji), gdyż wszystkie dane dotyczące wierzchołków i ich kolorów są skopiowane jednorazowo i zapamiętane na karcie graficznej. Do rysowania małych sześcianów stworzono trzy VBO, po jednym dla każdego możliwego koloru.

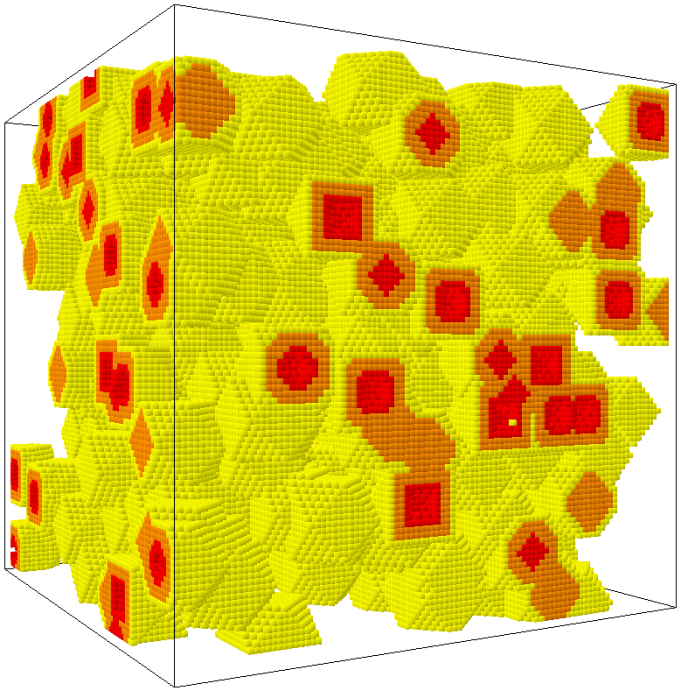
Kolejnym usprawnieniem wyświetlania jest rysowanie wszystkich komórek za pomocą maksymalnie trzech wywołań glDrawElements, po jednym dla każdego koloru. Usprawnienie to wynika z faktu, że zwiększanie liczby wywołań tejże funkcji jest bardziej czasochłonne od zwiększenia liczby wierzchołków rysowanych podczas jednego jej uruchomienia. Wywołując funkcję należy jedynie wskazać odpowiedni VBO, przekazać listę indeksów wierzchołków z VBO do wyświetlenia oraz liczbę rysowanych wierzchołków.

Ostatnim usprawnieniem jest nierysowanie komórek, dla których w łatwy sposób można sprawdzić, że nie będą one widoczne - komórka rysowana jest tylko wtedy, gdy jest komórką graniczną obszaru symulacji lub ma w najbliższym sąsiedztwie przynajmniej jedną zdrową komórkę.

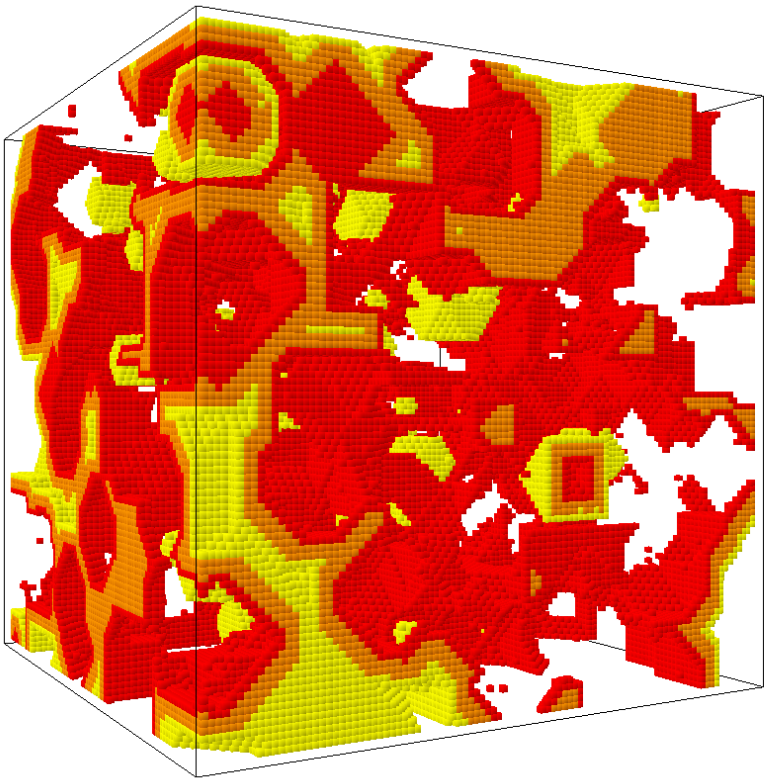
4.4.1.4 Zrzuty ekranu wybranych kroków procesu wizualizacji przykładowej symulacji automatu komórkowego



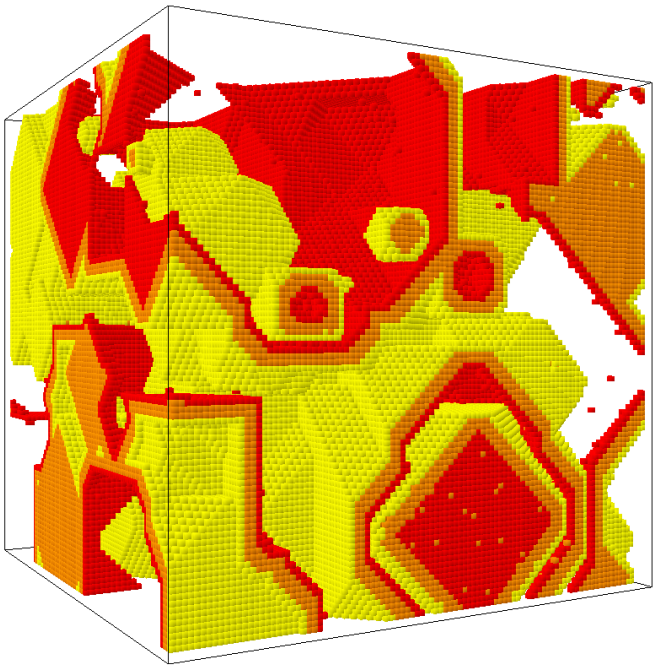
Rysunek 4.3: Wizualizacja automatu komórkowego obrazującego pierwszy miesiąc dynamiki infekcji wirusem HIV



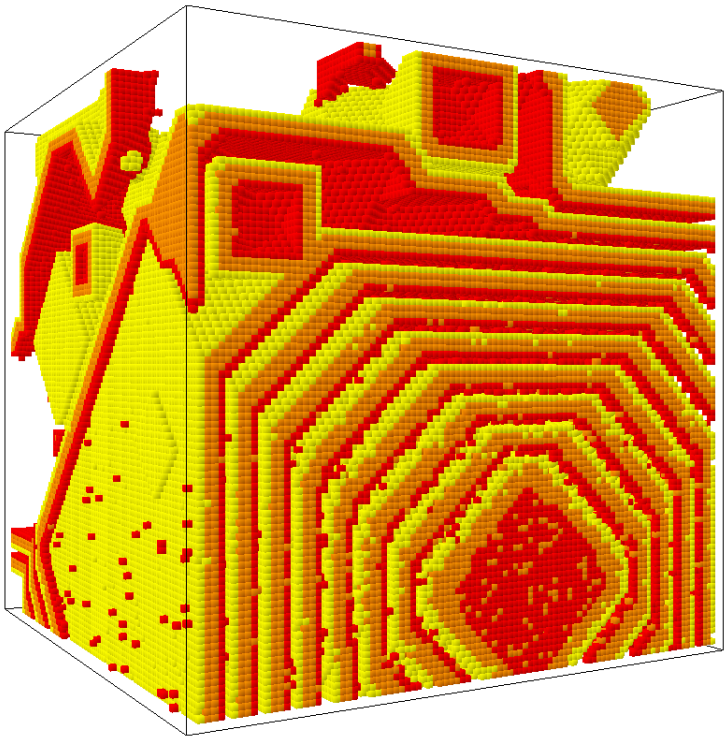
Rysunek 4.4: Wizualizacja automatu komórkowego obrazującego drugi miesiąc dynamiki infekcji wirusem HIV



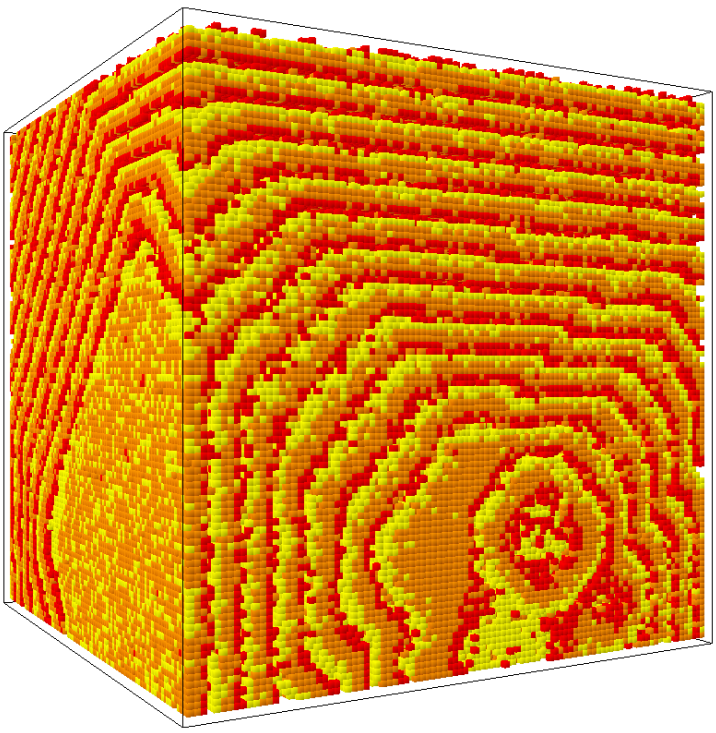
Rysunek 4.5: Wizualizacja automatu komórkowego obrazującego czwarty miesiąc dynamiki infekcji wirusem HIV



Rysunek 4.6: Wizualizacja automatu komórkowego obrazującego drugi rok dynamiki infekcji wirusem HIV



Rysunek 4.7: Wizualizacja automatu komórkowego obrazującego trzeci rok dynamiki infekcji wirusem HIV



Rysunek 4.8: Wizualizacja automatu komórkowego obrazującego ósmy rok dynamiki infekcji wirusem HIV

Na rysunku 4.3 przedstawiony został automat komórkowy reprezentujący fragment ludzkiego węzła chłonnego w pierwszym miesiącu infekcji wirusem HIV. W tym okresie na siatce automatu komórkowego przeważają komórki zainfekowane typu I. W dalszym etapie postępu infekcji, który można obserwować na rysunku 4.4, ich liczba stopniowo wzrasta, aż do około czwartego miesiąca infekcji (rysunek 4.5), w którym to poziom zainfekowanych komórek spada, co oznacza częściowe zwalczenie infekcji przez układ immunologiczny oraz wyjście z ostrej fazy dynamiki infekcji wirusem HIV.

Na rysunkach 4.6, 4.7, 4.8 przedstawione zostały kolejne lata infekcji wirusowej. Poczynając od rysunku 4.6, można zauważyć utworzenie charakterystycznego źródła, odpowiedzialnego za generację fal infekcji. W rezultacie liczba zdrowych komórek $CD4^{+}T$ nieprzerwanie spada, powodując rozprzestrzenienie się infekcji wirusowej po całym systemie, a tym samym wprowadzając model w fazę AIDS.

4.5 Skrypt uruchamiający symulację

```
from classes.World import *
from classes.Cell import *
from classes.BoundaryConditions import *

cellsNumberInAxis = 100
numberOfIterations = 1050
i1ToI2 = 1
i2ToDead = 2
pInf = [974, 100000000]
pHiv = [5, 10000]
pRep = [99, 100]
bc = BoundaryConditions.fixed
nameOfFileSim = "simulation_test"

world = World(rows = cellsNumberInAxis,
              cols = cellsNumberInAxis,
              layers = cellsNumberInAxis,
              numberOfIterations = numberOfIterations,
              numberOfIterationsInI1State = i1ToI2,
              numberOfIterationsInI2State = i2ToDead,
              pRep = pRep,
              pInf = pInf,
              pHIV = pHiv,
              boundaryCondition = bc,
              visualisation_ON = True,
              saveSimulation_ON = True,
              nameOfFileSimulation = nameOfFileSim)

world.simulateWorld()
```

Listing 4.24: Skrypt uruchamiający symulację - main.py

W pliku main.py, uruchamiającym symulację automatu komórkowego, modelującego dynamikę infekcji wirusem HIV zostaje stworzony obiekt klasy World, z określonymi przez użytkownika parametrami (w przeciwnym wypadku parametry przyjmują wartości domyślne). Następnie na rzecz obiektu wywoływana zostaje metoda *simulateWorld*, uruchamiająca całą symulacje.

W skrypcie uruchamiającym pojedynczą symulację automatu komórkowego możliwe jest zdefiniowanie parametrów automatu komórkowego:

1) **cellsNumberInAxis** - liczba komórek automatu komórkowego w pojedynczym wymiarze automatu komórkowego,

2) **numberOfIterations** - liczba iteracji,

3) **i1ToI2** - liczba iteracji, w których komórka musi przebyć w stanie zainfekowanym typu I, aby mogła przejść w stan komórki zainfekowanej typu II,

4) **i2ToDead** - liczba iteracji, w których komórka musi przebyć w stanie zainfekowanym typu II, aby mogła przejść w stan komórki martwej,

5) **pInf** - dwuelementowa lista, zawierającą licznik i mianownik prawdopodobieństwa, że martwa komórka zostanie zastąpiona przez zdrową komórkę,

6) **pHiv** - dwuelementowa lista, zawierającą licznik i mianownik prawdopodobieństwa, że w chwili początkowej infekcji komórka zdrowa zostanie zainfekowana,

7) **pRep** - dwuelementowa lista, zawierającą licznik i mianownik prawdopodobieństwa, że martwa komórka zostanie zastąpiona przez komórkę zainfekowaną typu I,

8) **boundaryCondition** - warunki brzegowe,

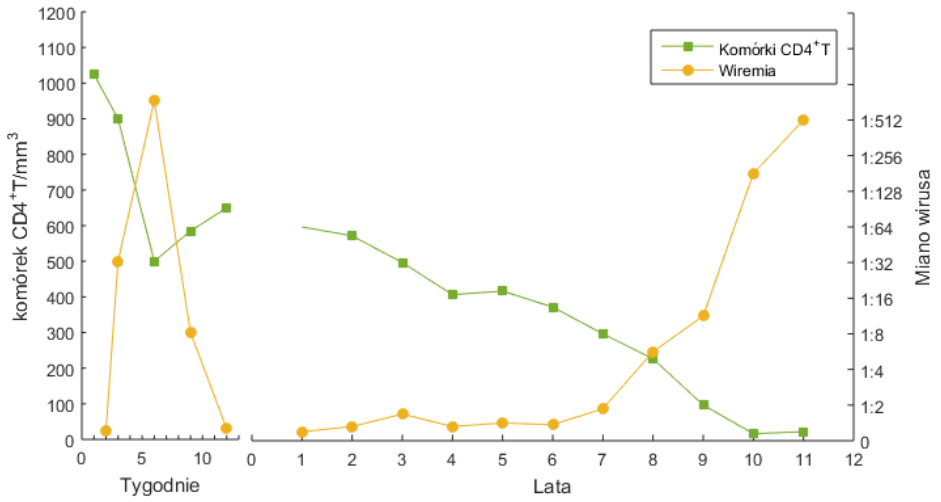
9) **nameOfFile** - nazwa pliku wynikowego symulacji, z liczba komórek w poszczególnym stanie, w każdej iteracji symulacji.

Ostatecznie uzyskano zadowalające wyniki odnośnie czasu wizualizacji pojedynczej iteracji symulacji. Uśredniony czas wyświetlania pojedynczej klatki wynosi 0.9380 sekundy, natomiast mediana równa jest 0.7669 sekundy. W poniższym punkcie 4.4.1.4 przedstawiono obraz automatu komórkowego w wybranych iteracjach pojedynczej symulacji.

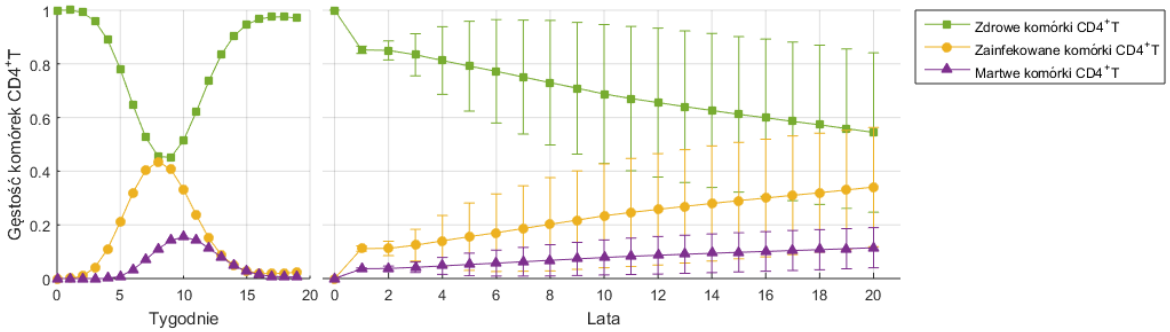
Rozdział 5

Analiza wyników

W celu oceny jakości otrzymanego modelu, wyniki uzyskane na podstawie skonstruowanego modelu zostały porównane z przebiegami czasowymi gęstości komórek poszczególnych rodzajów, obserwowanych wśród pacjentów.

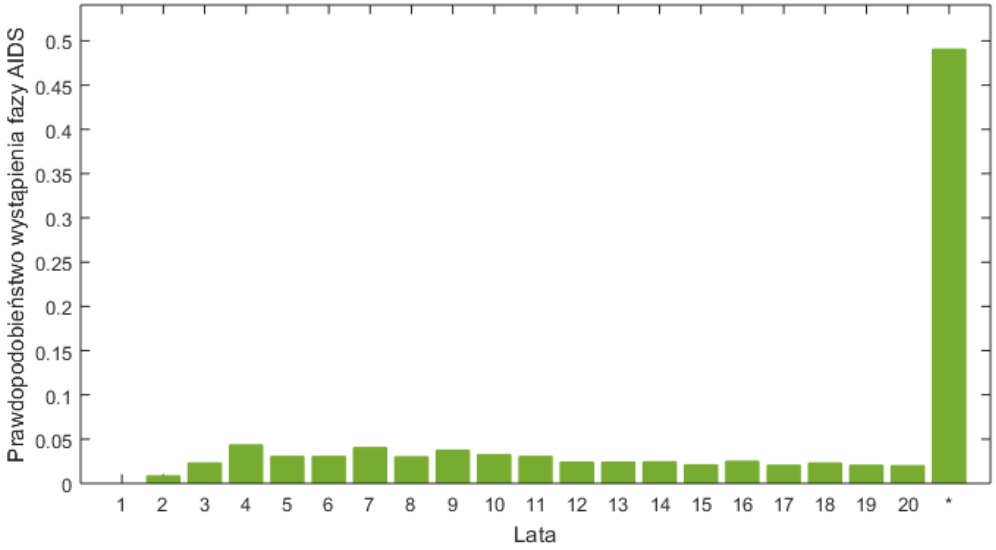


Rysunek 5.1: Przebieg czasowy gestości zainfekowanych komórek odnotowany u pacjentów (źródła: praca własna według [12]).



Rysunek 5.2: Przebieg czasowy gęstości komórek zdrowych, zainfekowanych i martwych uzyskany poprzez uśrednienie wyników z 2000 niezależnych symulacji skonstruowanego modelu.

Uzyskany model, którego wyniki symulacji zostały przedstawione na rysunku 5.2, nie realizuje w pełni dynamiki infekcji wirusem HIV. Model umożliwia częściowe odwzorowanie jedynie pierwszej fazy infekcji - fazy ostrej. Faza ostra, zgodnie z rysunkiem 5.2, wystąpiła w ósmym tygodniu infekcji, podczas gdy według doniesień literaturowych ([7]), faza ta występuje w szóstym tygodniu. Wyniki zaprezentowane na rysunku 5.2 dla pierwszych dziesięciu tygodni są zgodne z wynikami przedstawionymi w publikacji [11] - rysunek 2.2.



Rysunek 5.3: Rozkład prawdopodobieństwa wystąpienia AIDS (* - przypadki, w których nie osiągnięto fazy AIDS w badanym czasie symulacji).

W przeciwieństwie do pierwszej fazy infekcji, uzyskany model zdecydowanie nie odzwierciedla fazy AIDS. Uśredniona gęstość komórek zdrowych (rysunek 5.2) stopniowo spada, jednak nie przekracza wartości krytycznej w badanym czasie symulacji. Jest to dobrze widoczne na rysunku 5.3, gdzie można zauważyć, iż w niecałej połowie przypadków nie występuje wkroczenie w fazę AIDS.

Otrzymane wyniki sugerują, że automat komórkowy wraz z przedstawionym zestawem parametrów, proponowany w [11], nie jest odpowiedni do modelowania dynamiki infekcji wirusem HIV.

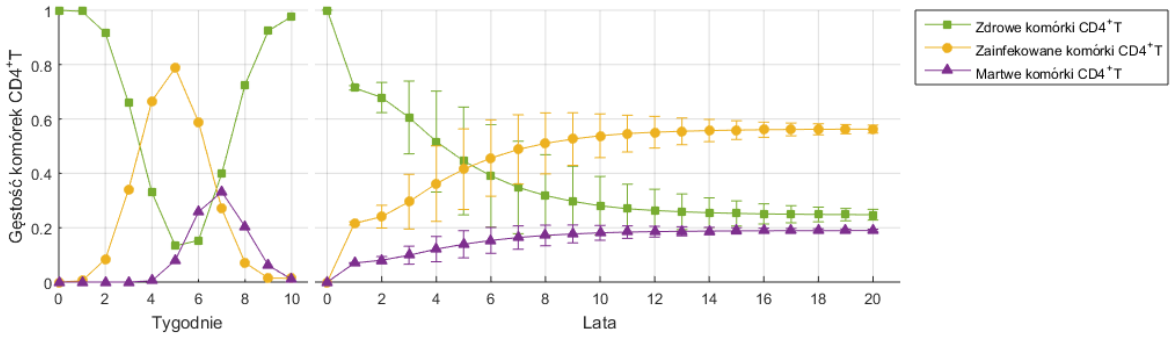
Rozdział 6

Modyfikacja parametrów

W związku z faktem, że model odtworzony na podstawie parametrów wybranych w artykule [11] nie odwzorowuje dynamiki infekcji wirusem HIV, zdecydowano się na zaproponowanie własnego modelu. Model ten oparto na obserwacjach klinicznych, opisanych w punkcie 4.1.4, z wykorzystaniem konstrukcji automatu komórkowego zaproponowanego w pracy [11]. Do nowego modelu wprowadzono następujące modyfikacje parametrów:

1. $P_{HIV} = 5 * 10^{-3}$
   Zdecydowano się na zwiększenie wartości parametru P_{HIV}, reprezentującego początkowe prawdopodobieństwo infekcji zdrowej komórki wirusem HIV.
2. $P_{INF} = 1 * 10^{-4}$
   Zdecydowano się na zwiększenie wartość parametru P_{INF}, reprezentującego prawdopodobieństwo, że martwa komórka zostanie zastąpiona przez zainfekowaną komórkę typu 1.

Pozostałe parametry oraz elementy konstrukcyjne są zgodne z oryginalnym modelem, przedstawionym w punkcie 4.1.4.



Rysunek 6.1: Przebieg czasowy liczebności komórek zdrowych, zainfekowanych i martwych, uzyskany poprzez uśrednienie wyników z 2000 niezależnych symulacji skonstruowanego modelu.

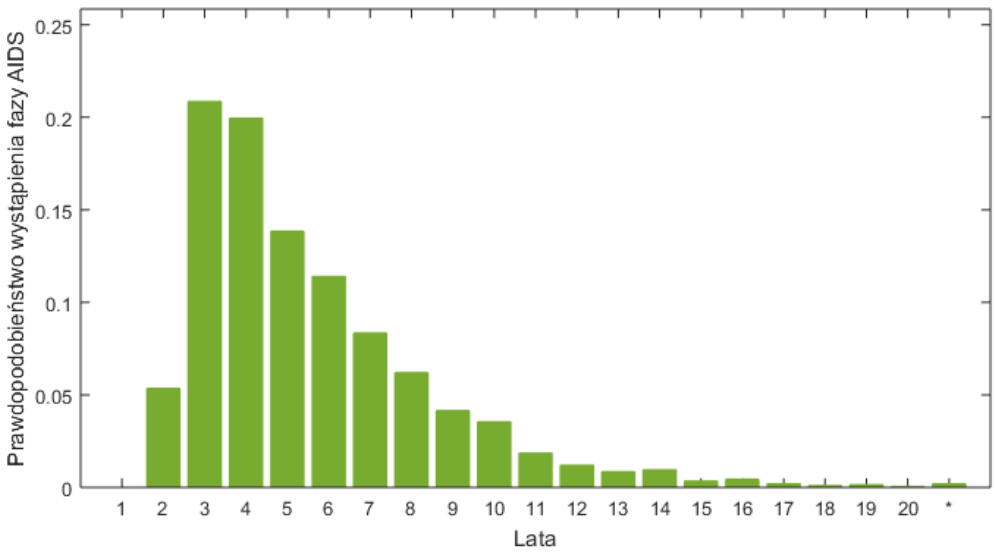
Analizując wyniki zebrane na rysunku 6.1, można stwierdzić, iż zaproponowany model dobrze odwzorowywuje dynamikę infekcji wirusem HIV.

W pierwszych dziesięciu tygodniach dobrze odwzorowany jest przebieg fazy ostrej, uzyskanej - podobnie jak w przypadku obserwacji klinicznych, przedstawionych na rysunku 5.1 - w piątym tygodniu od początkowej infekcji wirusowej. Uzyskany model

dobrze realizuje rownież dwie pozostałe fazy infekcji. Po uśrednieniu liczby komórek z każdej symulacji, widocznym na rysunku 6.1, koniec fazy asymptotycznej w tak uzyskanym modelu można odnotować w dziesiątym roku od początkowego zarażenia wirusem HIV. Moment ten jednoznacznie określa osiągnięcie trzeciej fazy infekcji - AIDS. Uzyskane wyniki odnośnie czasu rozwijania się infekcji są w zgodne z przedstawionymi na rysunku 5.1 obserwacjami klinicznymi.

Dodatkowo zauważyć można, iż otrzymane przebiegi pokrywają się z wynikami uzyskanymi w pracy [14], które również odnajdują odzwierciedlenie w badaniach klinicznych pod względem stężenia komórek poszczególnych rodzajów w danej fazie infekcji. Obserwowaną różnicą pomiędzy danymi klinicznymi (rysunek 5.1), a wynikami badanego modelu (rysunek 6.1), jest spadek gęstości komórek zdrowych podczas fazy ostrej poniżej poziomu 0.2, gdzie w przypadku danych klinicznych odnotowany jest spadek do połowy wartości początkowego stężenia. Pomimo różnicy w osiąganym poziomie, jest ona mniejsza niż dla wyników uzyskanych w publikacji [14], co przemawia za lepszym odwzorowaniem infekcji w niniejszej pracy.

W początkowej fazie infekcji (rysunek 6.1), słupki błędów, obrazujące odchylenie standardowe od wartości średniej gęstości komórek, są niezauważalne. Fakt ten oznacza, że dla wszystkich przeprowadzonych symulacji, rożnice w przebiegu fazy ostrej infekcji były znikome. Potwierdzają to przesłanki literaturowe, obecne w [7], z których wynika, że największe różnice w przebiegu infekcji wirusowej, odnotowywane są w długości fazy asymptotycznej oraz w charakterze jej przebiegu. Opisane zróżnicowanie fazy asymptotycznej jest również dobrze odwzorowane w przebiegu uzyskanym po modyfikacji parametrów, w postaci znaczących słupków błędów, widocznych w okresie od dwóch do piętnastu lat po zakażeniu.



Rysunek 6.2: Rozkład prawdopodobieństwa wystąpienia AIDS (* - przypadki, w których nie osiągnięto fazy AIDS w badanym czasie symulacji).

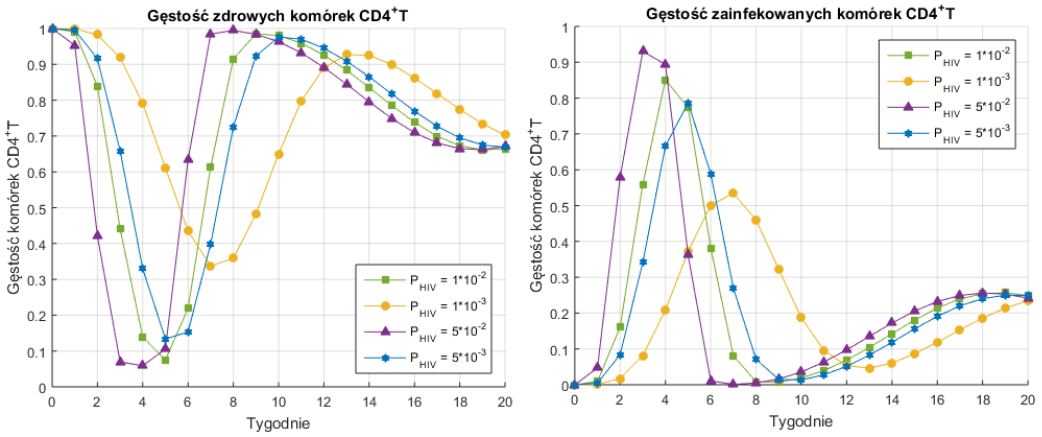
Na rysunku 6.2 można zaobserwować, że dla uzyskanego modelu największe prawdopodobieństwo wystąpienia fazy AIDS skupia się wokół trzeciego, czwartego i piątego roku od początkowej infekcji wirusem HIV. Mediana momentu wkroczenia w fazę AIDS wynosi pięć lat, natomiast średnia wynosi w przybliżeniu 5,5 roku.

Rozdział 7

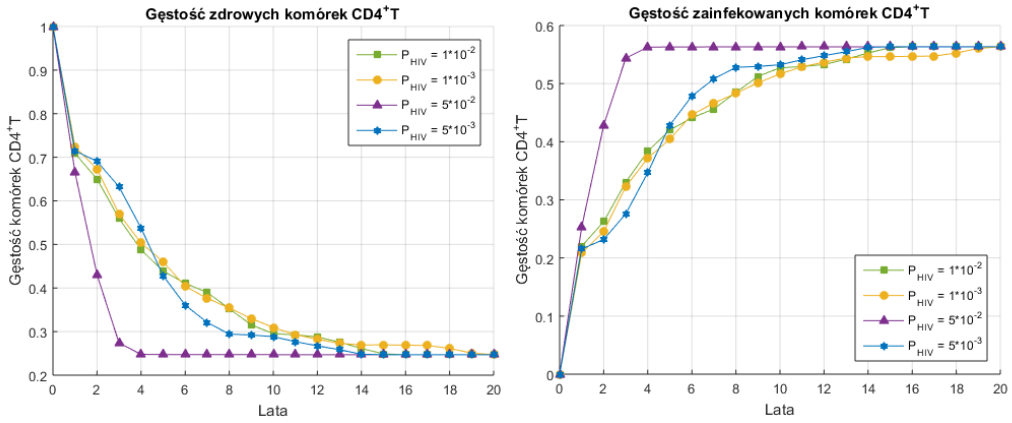
Wpływ zmienności parametrów

Badanie wpływu zmienności parametrów modelu z punktu 6 przedstawiono w oparciu o uśrednienie dwudziestu symulacji dla każdego realizowanego zestawu parametrów.

7.1 Zmienność P_{HIV}



Rysunek 7.1: Wpływ zmienności prawdopodobieństwa P_{HIV} na przebieg fazy ostrej.

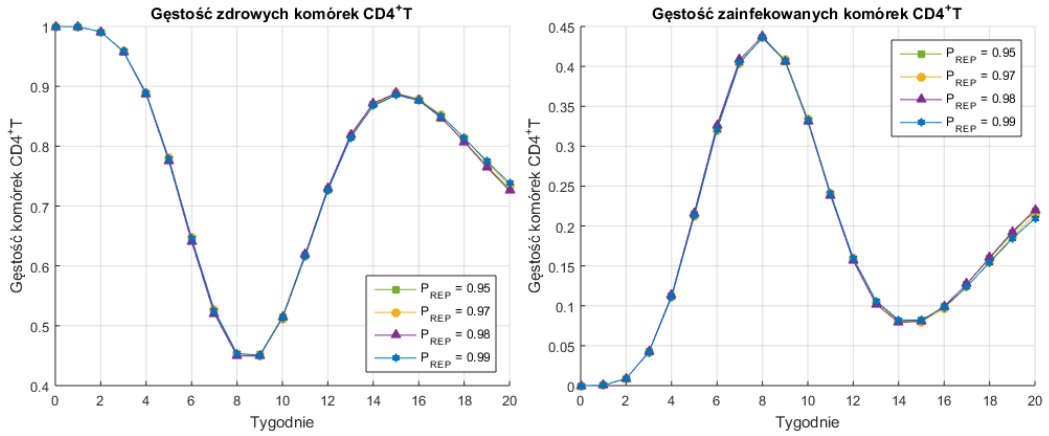


Rysunek 7.2: Wpływ zmienności prawdopodobieństwa P_{HIV} na przebieg fazy asymptotycznej.

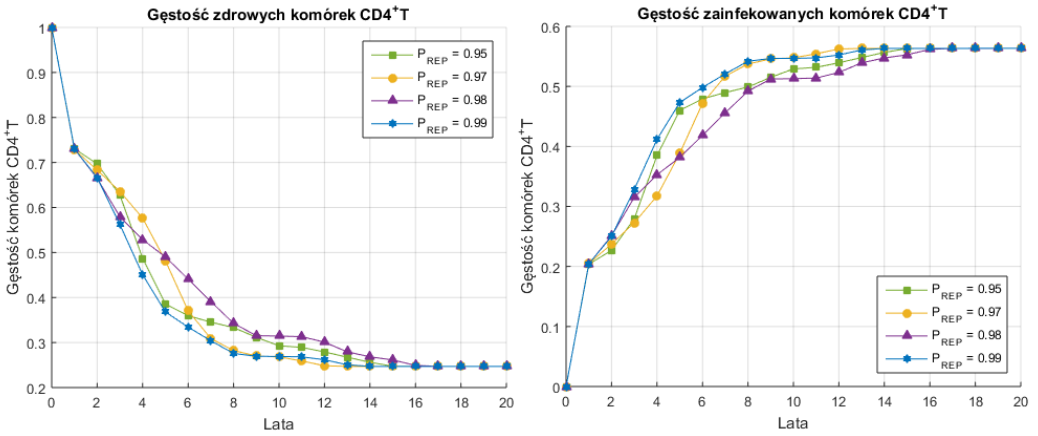
Jak można zaobserwować na rysunku 7.1, zwiększenie wartości prawdopodobieństwa P_{HIV}, oznaczającego zwiększenie liczby komórek zarażonych wirusem HIV w trakcie początkowej infekcji, powoduje szybsze wystąpienie fazy ostrej.

Na rysunku 7.2 można zauważyć, że faza AIDS zostaje osiągnięta najszybciej w przypadku największej wartości prawdopodobieństwa P_{HIV}. Natomiast w przypadku pozostałych, mniejszych wartości P_{HIV}, osiągnięcie fazy AIDS obserwuje się w podobnym czasie, różnią się one jednak charakterem przebiegu infekcji.

7.2 Zmienność P_{REP}



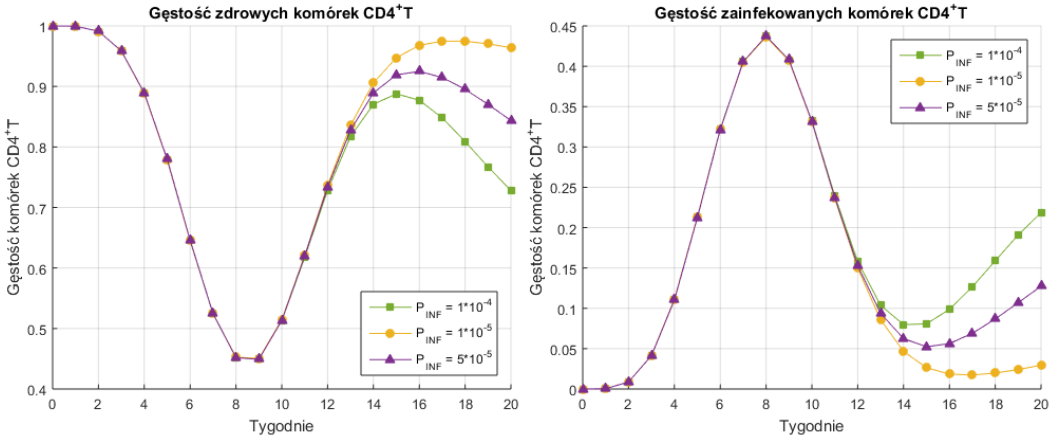
Rysunek 7.3: Wpływ zmienności prawdopodobieństwa P_{REP} na przebieg fazy ostrej.



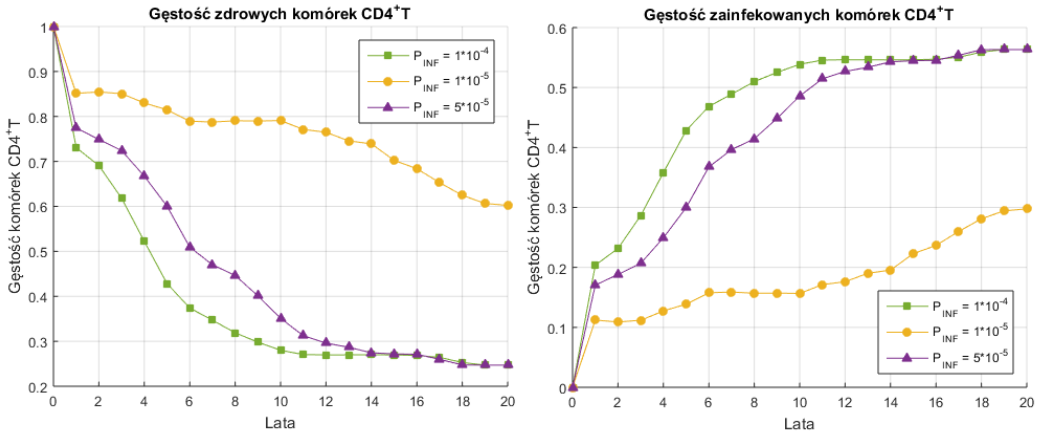
Rysunek 7.4: Wpływ zmienności prawdopodobieństwa P_{REP} na przebieg fazy asymptotycznej.

Jak można zaobserwować na rysunku 7.3, zmiana wartości prawdopodobieństwa P_{REP} nie wpływa na przebieg fazy ostrej infekcji. Zwiększanie wartości omawianego parametru modelu, charakteryzuje się również niewielkim wpływem na przebieg fazy asymptotycznej. Ewentualne różnice mogą zostać odnotowane w drobnym czasowym przesunięciu momentu osiągnięcia fazy AIDS ($P_{REP} = 0.95 P_{REP} = 0.98$) oraz w charakterach przebiegów.

7.3 Zmienność P_{INF}



Rysunek 7.5: Wpływ zmienności prawdopodobieństwa P_{INF} na przebieg fazy ostrej.



Rysunek 7.6: Wpływ zmienności prawdopodobieństwa P_{INF} na przebieg fazy asymptotycznej.

Zmienność wartości prawdopodobieństwa P_{INF} nie wpływa na osiagnięcie fazy ostrej infekcji. Natomiast, jak można zaobserować na rysunku 7.5, zwiększenie wartości prawdopodobieństwa P_{INF}, czyli szansy, że martwa komórka zostanie zastąpiona przez zainfekowaną komórkę typu I, powoduje przyspieszenie spadku komórek żywych, a więc zgodnie z przypuszczeniami, które potwierdza rysunek 7.6, skrócenie fazy asymptotycznej choroby.

Rozdział 8

Podsumowanie

Celem niniejszego projektu inżynierskiego było zrekonstruowanie trójwymiarowego automatu komórkowego, realizującego trójfazową dynamikę infekcji wirusem HIV, przedstawionego w punkcie 2.2 oraz zbadanie wpływu zmienności parametrów modelu na jego odpowiedź.

Efekt został uzyskany poprzez utworzenie programu w języku Python, implementującego omawiany automat komórkowy oraz bazującego na paradygmacie programowania zorientowanego obiektowo. Wzorując się na aparacie komórkowym, opublikowanym w pracy [11], nie udało się zrealizować modelu w pełni oddającego dynamikę infekcji wirusem HIV. Spośród znanych trzech faz infekcji wirusowej, model oddaje jedynie częściowo realia fazy ostrej, osiągniętej w ósmym tygodniu od początkowej infekcji. Jest to wynik akceptowalny, jednak w danych klinicznych wystąpienie tej fazy jest obserwowane dwa tygodnie wcześniej. Po wejściu w fazę asymptotyczną, gęstość zdrowych komórek $CD4^{+}T$ stopniowo spada, jednak nie przekracza wartości krytycznej, a więc tak zdefiniowany model nie wkracza w fazę AIDS (rysunek 5.3). Spore słupki błędów, uzyskane w uśrednionych wynikach modelu, wynikają z faktu, że dla prawie połowy generowanych przypadków, w badanym czasie symulacji faza AIDS nie została osiągnięta. Jako przyczynę niepowodzenia w odwzorowaniu modelu można było rozważać błędy w implementacji elementów konstrukcyjnych oraz zasad i warunków automatu komórkowego. Możliwość ta jednak została odrzucona, ponieważ jak zostało to opisane w punkcie 4.3, kluczowe aspekty logiki stworzonego programu zostały przetestowane zarówno z zastosowaniem testów manualnych, jak i również automatycznych.

W związku z niepowodzeniem w odtworzeniu modelu, zdecydowano się na modyfikację parametrów w oparciu o oryginalny automat komórkowy [11] oraz podstawy biologiczne, opisane w punkcie 4.1.4. Wprowadzone modyfikacje parametrów, omówione w rozdziale 6, umożliwiły utworzenie modelu realizującego dynamikę rozwoju infekcji wirusem HIV. Po uśrednieniu wyników z dwóch tysięcy symulacji, fazę ostrą można zaobserwować w piątym tygodniu infekcji. Następnie gęstość komórek powraca do prawidłowego poziomu, po czym zaczyna stopniowo spadać. Okres ten przedstawia fazę asymptotyczną. Ostatecznie, w dziesiątym roku infekcji, gęstość komórek spada poniżej wartości krytycznej, co oznacza wkroczenie w fazę AIDS.

Przeprowadzono również symulację z różnymi zestawami parametrów, co pozwoliło zbadać wpływ poszczególnych parametrów modelu na charakter dynamiki infekcji wirusem HIV. Zważywszy na fakt, że parametry modelu reprezentują określone aspekty biologiczne, dotyczące zarówno układu immunologicznego, jak i istoty infekcji wirusem HIV, poprzez badanie zmienności parametrów, można było obserwować jak zmiana wa-

runków biologicznych (w ujęciu matematycznym) wpływa na rozwój infekcji wirusowej.

Dodatkowo przyjętym założeniem pracy było stworzenie programu pozwalającego na wizualizację zaimplementowanego automatu komórkowego. Działanie to pozwala na lepsze zobrazowanie idei wykonanego automatu oraz umożliwia obserwowanie powstawania charakterystycznego źródła, odpowiedzialnego za generacje fal infekcji. Od tego momentu gęstość zdrowych komórek $CD4^+T$ stopniowo spada. Opisane źródło odpowiada za rozprzestrzenianie infekcji wirusowej po całym systemie, powodując przejście modelu z fazy asymptotycznej do fazy AIDS.

Natura wirusa HIV i jego interakcja z komórkami $CD4^+T$ jest niezwykle złożona. Prezentowany model nie ma na celu omówienia pełnej złożoności tejże skomplikowanej dynamiki. Jego zadaniem jest omówienie wybranych aspektów oddziaływań pomiędzy komórkami odpornościowymi układu immunologicznego, a wirusem HIV. Dodatkowo w pracy pokazano, że stosując tak prostą konstrukcję, jaką jest automat komórkowy oraz zbiór nieskomplikowanych zasad i parametrów, możliwa jest realizacja problemu o złożonej naturze. Ponadto otrzymany model, po modyfikacji parametrów, w zadowalający sposób oddaje realizm dynamiki infekcji wirusem HIV. Jednak należy zaznaczyć, że w zaimplementowanym aparacie komórkowym zostały uwzględnione jedynie niektóre aspekty infekcji wirusem HIV, z uwagi na fakt, iż realizowany projekt inżynierski miał pozwolić na lepsze zrozumienie dynamiki rozwoju infekcji wirusem HIV. Modelując dynamikę uwzględniono tylko komórki $CD4^+T$, podczas gdy w rzeczywistości występuje więcej typów komórek podatnych na infekcję. Według literatury [7], celem infekcji wirusa są również makrofagi, komórki dendrytyczne, komórki prekursorowe szpiku oraz grasicy, jak i limfocytów B, komórek NK (*ang. Natural Killers* - naturalni zabójcy), itp. Co więcej, komórki w symulacji mogły znajdować się w jednym z czterech możliwych stanów, lecz w ujęciu biologicznym, liczba możliwych stanów komórek jest większa. Jednym z dodatkowych stanów, który można by zaimplementować, jest tzw. stan spoczynkowy limfocytów T CD4. Komórka w tym stanie jest określona jako taka, która nie napotkała aktywującego ją antygenu. Dodatkowym typem komórki, podatnej na infekcję wirusem HIV, mogącym być wprowadzonym do modelu jest makrofag. Procesem biologicznym, który łączyłby dodany stan i typ komórek, jest uzwględnienie w modelu wydzielania przez zainfekowane makrofagi bliżej niezidentyfikowanego czynnika aktywującego spoczynkowe limfocyty T CD4, co ułatwia zakażenie tych limfocytów.

Różnorodność komórek, w których może zostać zarchiwizowany wirus HIV w postaci latentnej, ma przełożenie na tworzenie rezerwuarów wirusa, co z kolei ma znaczący wpływ na intensyfikację rozwoju infekcji. Wpływ poszczególnych komórek infekowanych przez wirusa HIV oraz ich rola w odpowiedzi odpornościowej, zarówno swoistej jak i nieswoistej, skierowana przeciwko wirusowi HIV, została szerzej opisana w pracy [7].

Znaczący wpływ na dynamikę infekcji ma również migracja wirusa HIV pomiędzy węzłem chłonnym, a przedziałami układu krwionośnego, podczas gdy w przedstawionej pracy zamodelowano jedynie pojedynczy obszar węzła chłonnego. Przedstawione powyżej aspekty, nieuwzględnione w modelu, to jedynie podzbiór procesów biologicznych, mających wpływ na złożoność wirusa HIV. Zatem w celu skontruowania dokładniejszego biologicznie modelu dynamiki rozwoju infekcji wirusem HIV, należałoby uwzględnić jeszcze wiele elementów, które wykraczają poza zakres niniejszego projektu. Należy mieć rownież na uwadze, że jak największe urzeczywistnienie modelu mogłoby pozwolić na rzetelne zamodelowanie terapii antywirusowej. Mimo to, przedstawione powyżej aspekty stanowią solidną podstawę do rozbudowy otrzymanego modelu i dokładniejszego zbadania natury wirusa HIV.

Bibliografia

[1] *Dokumentacja biblioteki OpenGL.* https://www.opengl.org/documentation/. Dostępne dnia 14.01.2020.

[2] *Dokumentacja biblioteki PyOpenGL.* http://pyopengl.sourceforge.net/documentation/. Dostępne dnia 14.01.2020.

[3] *Dokumentacja języka Python.* https://www.python.org/doc/. Dostępne dnia 14.01.2020.

[4] Blood, G. A. C. et al. (2016). Human immunodeficiency virus (hiv). *Transfusion Medicine and Hemotherapy*, 43(3).

[5] Cheng, A. H.-D. and Cheng, D. T. (2005). Heritage and early history of the boundary element method. *Engineering Analysis with Boundary Elements*, 29(3).

[6] Gładysz, A. (2007). *Zakażenia HIV/AIDS: poradnik dla lekarzy praktyków.* Wydawnictwo Continuo.

[7] Gładysz, A. and Knysz, B. (2009). *Diagnostyka, profilaktyka, klinika i terapia zakażeń HIV/AIDS: współczesne możliwości i problemy.* Wydawnictwo Continuo.

[8] Grzeszczuk, A. (2014). *HIV/AIDS.* PZWL.

[9] Hauser, S. L., Josephson, S. A., English, J. D., Engstrom, J. W., Prusiński, A., and Belniak, E. (2012). *Harrison. Neurologia w medycynie klinicznej. Tom I.* Wydawnictwo Czelej, 2 edition.

[10] Hope, T. and Trono, D. (2000). Structure, expression, and regulation of the hiv genome. *AIDS*, 12.

[11] Mo, Y., Ren, B., Yang, W., and Shuai, J. (2014). The 3-dimensional cellular automata for hiv infection. *Physica A: Statistical Mechanics and its Applications*, 399.

[12] Pantaleo, G., Graziosi, C., and Fauci, A. S. (1993). The immunopathogenesis of human immunodeficiency virus infection. *New England Journal of Medicine*, 328(5).

[13] Rosenberg, Z. F. and Fauci, A. S. (1991). Immunopathogenesis of hiv infection. *The FASEB Journal*, 5(10).

[14] Zorzenon dos Santos, R. M. and Coutinho, S. (2001). Dynamics of hiv infection: A cellular automata approach. *Physical review letters*, 87(16).