UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Simulação de Ecossistemas e de Dinâmicas Populacionais: Proposta de um Operador Genético "Elemento Estrangeiro"

Felipe Nelio Souto de Souza

Dissertação de Mestrado do Programa de Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria (MECAI)

ICMC USP SÃO CARLOS

Felipe Nelio Souto de Souza

Simulação de Ecossistemas e de Dinâmicas Populacionais: Proposta de um Operador Genético ”Elemento Estrangeiro”

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria. *VERSÃO REVISADA*

Área de Concentração: Matemática, Estatística e Computação

Orientador: Prof. Dr. Fernando Santos Osório

USP – São Carlos
Fevereiro de 2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

S726s

Souza, Felipe Nelio Souto de
Simulação de Ecossistemas e de Dinâmicas Populacionais: Proposta de um Operador Genético "Elemento Estrangeiro" / Felipe Nelio Souto de Souza; orientador Fernando Santos Osório. -- São Carlos, 2023.
123 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Algoritmo genético. 2. Sistema dinâmico. 3. Dinâmica populacional. 4. Simulação. 5. Elemento estrangeiro. I. Osório, Fernando Santos, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Felipe Nelio Souto de Souza

Simulation of Ecosystems and Population Dynamics: Proposal of a Genetic Operator ”Foreign Element”

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Professional Master's Program in Mathematics Statistics and Computing Applied to Industry, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Mathematics, Statistics and Computing

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
February 2023

Este trabalho é dedicado a todos que,
acreditam em utopias, mas que no dia a dia trabalham
em ações concretas para dar um passo em direção à evolução.

AGRADECIMENTOS

Os agradecimentos principais são direcionados aos meus pais [1], a minha companheira[2] e meus irmãos, da mesma família[3] ou não [4], que me apoiam e acreditam em tudo que faço.

Agradeço muito ao meu orientador [5] que me guiou nesta difícil jornada.

Agradecimentos especiais aos meus tios[6] que investiram em mim no início de minha jornada acadêmica e tornaram tudo isso possível.

Ultimo agradecimento a todos aqueles cujos os trabalhos serviram de base para construção do meu conhecimento e suporte para elaboração deste trabalho.

[1] Maria Francineide Souto Souza e Francisco Nelio Souto de Souza
[2] Michele Pignatari de Mello
[3] Fernando Souto, Anna Luiza Souto e Siciliana Souto
[4] Felipe Calderaro
[5] Fernando Santos Osório
[6] Anna D'Lira e Geroud Schacher

“A desobediência é uma virtude
necessária à criatividade.”
(Raul Seixas)

RESUMO

SOUZA, F. N. S. **Simulação de Ecossistemas e de Dinâmicas Populacionais: Proposta de um Operador Genético "Elemento Estrangeiro"**. 2023. 123 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Este trabalho se apoia em conhecimentos oriundos da biologia e da computação, além utilizar da técnica de simulação para analisar e discutir os efeitos de um operador genético, aqui denominado de *Elemento Estrangeiro* no contexto da teoria de *Algoritmos Genéticos*. Baseado uma ferramenta recriada a partir do `Ecosim`, trabalho de Connor Brooks, executam-se simulações de um ecossistema natural com co-evolução, um sistema dinâmico regido por regras de predação, competição por recursos, e com populações de indivíduos que buscam "sobreviver" ou "dominar" o ambiente onde se encontram. Neste ambiente é possível observar dinâmicas populacionais e como essas são influenciadas pelos parâmetros (características do ambiente). Partindo desse ponto foram encontradas condições nas quais ocorrem regimes de equilíbrio, os quais foram utilizados de base para analisar os impactos e perturbações dos operadores genéticos. Os resultados das análises com operador *elemento estrangeiro* demonstraram que o operador possui propriedades e um impacto interessante no ecossistema, tanto em termos de evolução e dinâmica, quanto em termos de variabilidade da população, porém, se diferenciando de operadores tradicionais, como é o caso do operador de *mutação*.

Palavras-chave: Algoritmo genético, Sistema dinâmico,Dinâmica populacional, Simulação, Elemento estrangeiro.

ABSTRACT

SOUZA, F. N. S. **Simulation of Ecosystems and Population Dynamics: Proposal of a Genetic Operator "Foreign Element"**. 2023. 123 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

This work is based on knowledge from biology and computing, and uses the simulation technique to analyze and discuss the effects of a genetic operator, here called *Foreign Element* in the context of the theory of *Genetic Algorithms*. Based on a tool recreated from `Ecosim`, work by Connor Brooks, simulations are performed of a natural ecosystem with co-evolution, a dynamic system governed by rules of predation, competition for resources, and with populations of individuals that they seek to "survive" or "dominate" the environment in which they find themselves. In this environment it is possible to observe population dynamics and how these are influenced by parameters (environment characteristics). Starting from this point, conditions were found in which equilibrium regimes occur, which were used as a basis to analyze the impacts and perturbations of genetic operators. The results of the analyzis with the operator *foreign element* showed that the operator has interesting properties and impact on the ecosystem, both in terms of evolution and dynamics, and in terms of population variability, however, different from traditional operators, such as this is the case of the *mutation* operator.

Keywords: Genetic algorithm, Dynamic system, Population dynamics, Simulation, Foreign element.

LISTA DE ILUSTRAÇÕES

Figura 1 – Estrutura básica de Algorítimo Genético 36
Figura 2 – Estrutura de Algoritmo Genético com passo probabilístico de inserção de estrangeiro 42
Figura 3 – Dados de Energia 50
Figura 4 – Dados de Tempo de Vida 50
Figura 5 – Dados de Genética 51
Figura 6 – Dados de População 51
Figura 7 – Interface Gráfica 52
Figura 8 – Simulação Controle 01 57
Figura 9 – Simulação Controle 02 58
Figura 10 – Simulação Metabolismo Elevado 59
Figura 11 – Simulação Metabolismo Reduzido 59
Figura 12 – Simulação Visão Elevada 60
Figura 13 – Simulação Visão Reduzida 60
Figura 14 – Simulação Visão Reduzida 60
Figura 15 – Simulação Espaço Ampliado 61
Figura 16 – Simulação Espaço Reduzido 61
Figura 17 – Curva de Predação 62
Figura 18 – Curva de Predação 62
Figura 19 – Curva de Predação 63
Figura 20 – Simulação mais pressão na predação 63
Figura 21 – Simulação menos pressão na predação 63
Figura 22 – Convergência genética 64
Figura 23 – Convergência genética 65
Figura 24 – Simulação com Mutação e Elemento Estrangeiro 66
Figura 25 – Simulação com Mutação e Elemento Estrangeiro 67
Figura 26 – Simulação com Mutação e Elemento Estrangeiro 68
Figura 27 – Simulação Controle 105
Figura 28 – Simulação Controle 105
Figura 29 – Simulação Controle 106
Figura 30 – Simulação Controle 106
Figura 31 – Simulação Metabolismo - min=0,05 max=0,4 106
Figura 32 – Simulação Metabolismo - min=0,05 max=0,4 107

Figura 33 – Simulação Metabolismo - min=0,05 max=0,4 107
Figura 34 – Simulação Metabolismo - min=0,2 max=1,6 107
Figura 35 – Simulação Metabolismo - min=0,2 max=1,6 108
Figura 36 – Simulação Visão - min=0,2 max=1,6 108
Figura 37 – Simulação Visão - min=0,05 max=0,1 108
Figura 38 – Simulação Visão min=0,05 max=0,1 109
Figura 39 – Simulação Visão - min=0,05 max=0,1 109
Figura 40 – Simulação Visão - min=0,2 max=0,4 109
Figura 41 – Simulação Visão - min=0,2 max=0,4 110
Figura 42 – Simulação Visão - min=0,2 max=0,4 110
Figura 43 – Simulação espaço - min=-2 max=2 110
Figura 44 – Simulação espaço - min=-2 max=2 111
Figura 45 – Simulação espaço - min=-2 max=2 111
Figura 46 – Simulação espaço - min=-0,5 max=0,5 111
Figura 47 – Simulação espaço - min=-0,5 max=0,5 112
Figura 48 – Simulação espaço - min=-0,5 max=0,5 112
Figura 49 – Simulação reprodução - intervalo de tempo=2 112
Figura 50 – Simulação reprodução - intervalo de tempo=2 113
Figura 51 – Simulação reprodução - intervalo de tempo=2 113
Figura 52 – Simulação reprodução - intervalo de tempo=8 113
Figura 53 – Simulação reprodução - intervalo de tempo=8 114
Figura 54 – Simulação reprodução - intervalo de tempo=8 114
Figura 55 – Simulação pressão de predaçao - intervalo=0 à 0,5 114
Figura 56 – Simulação pressão de predaçao - intervalo=0 à 0,5 115
Figura 57 – Simulação pressão de predaçao - intervalo=0 à 0,5 115
Figura 58 – Simulação pressão de predaçao - intervalo=0,5 à 2,0 115
Figura 59 – Simulação pressão de predaçao - intervalo=0,5 à 2,0 116
Figura 60 – Simulação pressão de predaçao - intervalo=0,5 à 2,0 116
Figura 61 – Simulação com mutação 116
Figura 62 – Simulação com mutação 117
Figura 63 – Simulação com mutação 117
Figura 64 – Simulação com mutação 117
Figura 65 – Simulação com mutação 118
Figura 66 – Simulação com mutação 118
Figura 67 – Simulação com elemento estrangeiro 118
Figura 68 – Simulação com elemento estrangeiro 119
Figura 69 – Simulação com elemento estrangeiro 119
Figura 70 – Simulação com elemento estrangeiro 119
Figura 71 – Simulação com elemento estrangeiro 120

Figura 72 – Simulação com elemento estrangeiro 120
Figura 73 – Simulação com intervalos 120
Figura 74 – Simulação com intervalos 121
Figura 75 – Simulação com intervalos 121
Figura 76 – Simulação com intervalos 121
Figura 77 – Simulação com intervalos 122
Figura 78 – Simulação com intervalos 122
Figura 79 – Simulação com intervalos 122
Figura 80 – Simulação com intervalos 123

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo alimentação 48

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Ecosim 75
Código-fonte 2 – Classe de configurações do Ecosim 75
Código-fonte 3 – Classe de controle do ecossistema 77
Código-fonte 4 – Classe dos agentes 80
Código-fonte 5 – Classe de apuração das métricas 94
Código-fonte 6 – Plot das métricas de população 101
Código-fonte 7 – Plot das métricas genéticas 102

LISTA DE SÍMBOLOS

AG — Algoritmo genético

Pm — Probabilidade de mutação

Pe — Probabilidade de surgimento de estrangeiro

R_i — Elemento do rebanho, agentes da mesma espécie numa mesma região

Vr — Velocidade de rebanho

C_i — Elemento do conjunto de predados, agentes que servem de comida para determinado agente examinado numa mesma região

P_i — Elemento do conjunto de predadores, agentes que servem se alimentam de determinado agente examinado numa mema região

Av — Vetor bi-dimensional de velocidade

Tr — Tendência de rebanho, fator genético que expressa quanto um agente é propenso a seguir o rebanho

Ap — Vetor bi-dimensional que representa a posição do agente

W — Fator genético de oscilação da função senoidal que incide sobre o movimento

Fpp — Cálculo de rasão de força entre predador e presa

Dpp — Calculo de distância entre predador e presa

Ae — Energia do agente predador

Ce — Energia do agente presa

Am — Característica genética do metabolismo do predador

Cm — Característica genética do metabolismo da presa

Fem — Fator que estabelece peso entre energia e metabolismo na predação

SUMÁRIO

1 INTRODUÇÃO 27
1.1 Contexto 27
1.2 Motivação 27
1.3 Proposta 28
1.4 Organização do trabalho 29

2 CONCEITOS BASE 31
2.1 Sistemas dinâmicos 31
2.2 Simulação 32
2.3 Algoritmos bio-inspirados 32
2.4 Teoria da evolução 34
2.5 Algoritmo genético 35
2.6 Outras definições relevantes 35
2.6.1 Ecossistema *35*
2.6.2 Coevolução *36*
2.6.3 Dinâmica populacional *37*

3 BREVE REVISÃO 39
3.1 Simulador de ecossistemas 39
3.2 Elemento estrangeiro 40

4 METODOLOGIA 41
4.1 Estratégia 41
4.2 Primeiro contato com elemento estrangeiro 41
4.3 Variabilidade de algoritmos genéticos 42
4.3.1 Mutação *43*
4.3.2 Elemento estrangeiro *43*
4.4 Simulador ecosim 43
4.4.1 Introdução ao Ecosim *43*
4.4.2 Instância *43*
4.4.3 Ambiente *44*
4.4.4 Agentes *44*
4.4.5 Parâmetros *44*
4.4.6 Simulação *47*

4.4.7 Saídas . . . 49
4.4.8 Outras considerações . . . 52
4.4.9 Considerações finais deste capítulo . . . 53

5 EXPERIMENTOS . . . 55
5.1 Simulação de controle . . . 55
5.1.1 Parâmetros usados . . . 55
5.1.2 Dados das simulações . . . 57
5.2 Efeito dos parâmetros . . . 58
5.2.1 Efeito do metabolismo . . . 58
5.2.2 Efeito da visão . . . 59
5.2.3 Pressão do espaço . . . 61
5.2.4 Pressão de reprodução . . . 61
5.2.5 Pressão de predação . . . 62
5.3 Simulação dos efeitos dos operadores . . . 64
5.3.1 Operador de mutação . . . 64
5.3.2 Operador de elemento estrangeiro . . . 64
5.3.3 Simulação com intervalos intercalados . . . 65
5.4 Considerações finais deste capítulo . . . 68

6 CONCLUSÃO . . . 71
6.1 Efeito do elemento estrangeiro . . . 71
6.2 Resultados do simulador . . . 71
6.3 Principais contribuições e trabalhos futuros . . . 72

REFERÊNCIAS . . . 73

APÊNDICE A CÓDIGO FONTE ECOSIM . . . 75

APÊNDICE B CÓDIGO PLOT DAS MÉTRICAS . . . 101

APÊNDICE C AMOSTRAS DE DADOS . . . 105
C.1 Dados de controle . . . 105
C.2 Experimentos metabolismo . . . 106
C.3 Experimentos visão . . . 108
C.4 Experimentos pressão de espaço . . . 110
C.5 Experimentos pressão de reprodução . . . 112
C.6 Experimentos pressão de predação . . . 114
C.7 Experimentos mutação . . . 116
C.8 Experimentos elemento estrangeiro . . . 118
C.9 Experimentos intervalos intercalados . . . 120

CAPÍTULO

1

INTRODUÇÃO

1.1 Contexto

Ao longo do tempo, o universo, o planeta Terra, os ecossistemas, os seres vivos, a humanidade e a ciência passam por constantes transformações, por vezes esse processo é chamado de evolução. Em alguns contextos essa evolução é mensurada e submetida a uma régua com juízo de valor, da qual a progressão nesta régua representa a avaliação da evolução. Há momentos, em que o caminho da evolução parece convergir para um ponto diferente do almejado, neste momento é comum que haja uma ruptura que possa perturbar o processo e mudar o rumo da evolução. Da segunda para a terceira década do século XXI, é possível acompanhar a inexorável evolução da ciência ao mesmo tempo em que a organização social humana passa por turbulências. Neste ambiente a ciência[1] tem papel fundamental de com seus métodos (que apesar de falíveis, são aperfeiçoados e colocados à prova constantemente) iluminar os caminhos da sociedade humana. Desde há muito a ciência busca inspiração na natureza para encontrar soluções para os problemas enfrentados ou inventados pela humanidade, mais recentemente na história os computadores têm sido ferramenta fundamental para resolução e otimização problemas. Neste contexto se situa esse trabalho que reúne conhecimento de algumas disciplinas, da biologia à computação, e se utiliza do método científico, para propor a validade da pauta de um novo operador genérico e análise de seu impacto em dinâmicas populacionais.

1.2 Motivação

A otimização de problemas não-lineares por algoritmos inspirados na natureza, por vezes enfrenta dificuldades em encontrar o melhor ponto do espaço de soluções, decorrente do grande número de soluções ótimas locais que o problema possa a ter (SILVEIRA; OLIVEIRA; SILVA,

[1] Como entidade que representa todos aqueles que de alguma maneira trabalham para o crescimento do conhecimento coletivo

2007). Costumeiramente, para resolução de um problema de otimização, a solução desejada é um ótimo global, não só um ótimo local, e quando por influência de uma dessas soluções a otimização converge para regiões distantes da solução ótima, o resultado final tende a ser insatisfatório. Motivado pela implementação de um operador genético aplicado em um algoritmo de otimização baseado em simulação de populações (algoritmo genético), o presente trabalho examina os efeitos do operador genético “elemento estrangeiro” em dinâmicas populacionais com intuito de discutir sua viabilidade na utilização em implementações de algoritmos genéticos. A fim de propor mais uma alternativa de solução para o problema de convergência, comum em algoritmos de otimização, através da vasta exploração do espaço de soluções.

1.3 Proposta

Inspirado por trabalhos anteriores como *"A Genetic Method for Evolutionary Agents in a Competitive Environment"*(MORIWAKI *et al.*, 1997) e *"Co-evolution of Antagonistic Intelligent Agents using Genetic Algorithms"*(ROSA *et al.*, 2013), a ferramenta implementada e utilizada para este estudo estudo é um simulador de ecossistema. Trata-se de uma simulação de um ecossistema com agentes antagônicos que competem e atravessam um processo de coevolução. Neste contexto, o trabalho busca discutir a aplicação do operador genético *elemento estrangeiro*. Este operador inspirado em observações empíricas, que como o próprio nome sugere, representa a inserção no ambiente de agentes que não derivam da população presente. A proposta é também debater a aplicação em algoritmos de otimização bio-inspirados. Como resultado, é esperado conseguir observar e mensurar os impactos deste operador. Como resultado adicional há a construção de uma ferramenta de simulação que pode ser extensível para aplicação em outras modelagens de ecossistemas e de sistemas dinâmicos populacionais.

Considerando um ecossistema local um sistema dinâmico natural, porém sendo até um certo ponto "fechado ou limitado em suas fronteiras", onde dentro de suas propriedades e do seu contexto local, é bem usual que seja atingido um estado de "equilíbrio"(regime de funcionamento que se estabiliza após certo tempo). Inclusive, alguns desses sistemas podem até mesmo levar a "extinção"de alguns dos participantes desse sistema local e assim atingir um estado estável. Por exemplo, em um sistema onde temos muitos predadores e presas, podemos rapidamente acabar com as presas de um certo tipo no ambiente (as fáceis de serem alcançadas). A introdução de um "elemento estrangeiro", que perturba o regime atual do sistema dinâmico, pode alterar consideravelmente o estado desse sistema, por exemplo, se um "elemento estrangeiro"perturbar este equilíbrio/desequilíbrio do ecossistema local. Uma mudança no regime de funcionamento do sistema dinâmico pode conduzir ele para novos estados, novas situações, que não seriam alcançadas (considerando condições específicas de pressão seletiva que podem levar há uma convergência prematura) se não tivesse sido introduzido este novo componente no sistema, se não tivesse sido incluído um "elemento estrangeiro"que irá perturbar a convergência do sistema como um todo para um estado em particular. Este tipo de situação ocorre na natureza, mas

também em outros sistemas dinâmicos, naturais ou artificiais, e merece ser melhor estudado. E esse é o contexto desse trabalho que está sendo proposto.

1.4 Organização do trabalho

O trabalho se divide em 5 capítulos:

- O primeiro, e presente capítulo, é uma *Introdução* que tem a função de contextualizar o problema e apresentar os temas que serão discutidos no trabalho;
- Na sequência são aprofundados *Conceitos base*, ou seja o arcabouço teórico que funciona como alicerce para todo o trabalho;
- Em *Breve revisão* são contemplados os pontos de destaque da pesquisa bibliográfica mais recentes para os temas chave do trabalho;
- No capítulo de *Metodologia* é descrita a abordagem feita em relação ao problema em estudo, o processo de construção do simulador de ecossistema e os detalhes sobre como é conduzido o estudo;
- Complementado o capítulo anterior, a seção de *Experimentos* apresenta as medições realizadas e o resultado de suas interpretações;
- Por fim, na *Conclusão* temos o fechamento que reúne e sintetiza todo o processos e resultados do trabalho, além de apresentar uma perspectiva de continuidade para este estudo.

CAPÍTULO

2

CONCEITOS BASE

Nesta seção serão abordados os conceitos e definições relevantes para entender e sustentar o trabalho.

2.1 Sistemas dinâmicos

Uma importante definição para essa discussão é a de sistema dinâmico. Uma possível definição é: “Um sistema pode ser definido como um conjunto de objetos agrupados por uma interação ou interdependência... Um sistema é dinâmico quando algumas grandezas que caracterizam seus objetos constituintes variam no tempo.” (MONTEIRO, 2006)

A origem da teoria dos *Sistemas Dinâmicos* coincide com a da *Mecânica Clássica*, porém o interesse em examinar e descrever esses sistemas já podia ser observado em postulações de Aristóteles.(MONTEIRO, 2006). Saltando no tempo, no final do século XX, observamos uma mudança do paradigma mecanicista para o ecológico, de diferentes formas e com diferentes velocidades nos vários campos científicos (CAPRA, 1996). Neste segundo paradigma dois conceitos são importantes para destaca-lo do primeiro, um é compreender que as partes não explicam o funcionamento do todo, isso se observada a partir da organização das estruturas das partes e não simplesmente a soma das partes segregadas. Outro é que os objetos não estão desassociados do meio, ou seja, existe uma relação entre os objetos e meio que está sua volta(CAPRA, 1996).

Ao analisar diversos sistemas dinâmicos, muitos problemas se desdobram em uma ou mais equações diferenciais a serem resolvidas. Quando estes levam à equações diferenciais lineares, encontramos técnicas analíticas para resolve-las. Por sua vez, as equações diferenciais não-lineares são mais complicadas pois não existem técnicas gerais de resolução. Na segunda metade do século XX, a evolução dos computadores impulsionou a resolução de problemas não-lineares usando métodos numéricos(VILLATE, 2007). O advento da computação numérica

permitiu um grande avanço para a análise de sistemas dinâmicos e, por meio de simulações computacionais, pode-se verificar o comportamento desse conjunto(MONTEIRO, 2006).

2.2 Simulação

Desde o século XVI a metodologia de se fazer ciência está associada a dois principais paradigmas, o *experimental* e o *teórico*. No primero realizam-se observações, medições e qualificações do fenômeno de interesse afim de entender, responder perguntas ou validar hipóteses. Já no segundo as hipóteses são definições fundamentais baseada em ancoragem lógica de conhecimentos prévios(HOEKSTRA; KROC; SLOOT, 2010). Apesar de em alguns contextos (como a matemática) o processo teórico possa se manter neste contexto sendo apoiado por provas e postulados, em outros contextos ele pode se encontrar com a metodologia experimental para demonstrar ou refutar sua validade. Pode se dizer que, desde a Segunda Guerra Mundial, um terceiro paradigma científico surgiu, o *computacional*, no qual estudamos os fenômenos através de simulações em computador (HOEKSTRA; KROC; SLOOT, 2010).

Falar de simulação implica em falar de modelos, esse é o termo que designa a entidade conceitual ou física que se assemelha, imita, descreve, prevê ou transmite informações sobre o comportamento de algum processo ou sistema. O benefício de ter um modelo é poder explorar o comportamento intrínseco de um sistema em um maneira econômica e segura(KLEE; ALLEN, 2011).

O comportamento de sistemas dinâmicos, de maneira geral, pode ser descrito por equações e fórmulas matemáticas, que representam seus princípios científicos e/ou observações empíricas. A esse conjunto de equações e parâmetros, que podem ser usados para descrever o comportamento de um sistema dinâmico, é dado o nome de *modelo matemático do sistema*. O paradigma computacional se tornou fundamental em situações em que as descrições por modelos matemáticos são de difícil tratamento e também quando a experimentação direta não está ao alcance (KLEE; ALLEN, 2011).

Um modelo de computador é um modelo abstrato implementado em um programa de computador. Modelos de computador de sistemas naturais (como parte da modelagem matemática) são amplamente utilizados em física, biologia, ciências sociais, economia e engenharia. Com seu poder já bastante explorado no campo das exatas (como física, matemática, química, engenharia) esse paradigma vê sua importância aumentando continuamente em áreas como biologia, medicina, sociologia e psicologia (HOEKSTRA; KROC; SLOOT, 2010).

2.3 Algoritmos bio-inspirados

Outro conceito bastante explorado no contexto computacional é de algoritmos. Esse pode ser descrito como um procedimento passo a passo de fornecer cálculos ou instruções. Uma

das classes de algoritmos são os *Algoritmos de otimização*, que são procedimentos iterativos designados a encontrar a melhor solução, ou soluções aproximadas, para um determinado problema (YANG, 2014).

Designa-se otimização o campo de conhecimentos com técnicas orientadas a determinar os extremos (máximos ou mínimos) de uma ou mais funções em domínios específicos (CUNHA; TAKAHASHI; ANTUNES, 2012). De maneira geral, na otimização primeiramente é feita a formulação correta do problema, a partir daí a tarefa principal é encontrar as soluções ótimas através de um procedimento de solução apoiado nas técnicas matemáticas mais adequadas (YANG, 2014). Um bom paralelo ilustrativo é feito por *Xin-She Yang* que compara este processo a uma caça ao tesouro:

”Imagine que estamos tentando caçar um tesouro escondido em uma paisagem montanhosa dentro de um tempo limite. Em um extremo, suponha que estamos com os olhos vendados sem qualquer orientação. Nesse caso, o processo de busca é essencialmente uma busca aleatória pura, que normalmente não é eficiente. Em outro extremo, se nos disserem que o tesouro está colocado no pico mais alto de um região, subiremos diretamente até o penhasco mais íngreme e tentaremos chegar ao ponto mais alto, o pico. Este cenário corresponde às técnicas clássicas de escalada. Na maioria dos casos, nossa busca está entre esses extremos. Não estamos com os olhos vendados e não sabemos onde olhar. O cenário mais provável é que façamos um passeio aleatório enquanto procuramos alguns dicas.”

Assim como outros objetos de estudo, os algoritmos de otimização podem ser classificados em grupos. Uma das forma de dividi-los é baseado na sua natureza do algoritmo, neste contexto temos duas categorias: *algoritmos determinísticos* e *algoritmos estocásticos*. No primeiro grupo temos algoritmos que seguem um procedimento rigoroso, de maneira que seus passos e respostas são passiveis de repetição, um exemplo é o algoritmo de escalada (hill-climbing) no qual para um mesmo ponto de partida, ele seguirá o mesmo caminho toda execução. Por outro lado, o segundo grupo sempre têm alguma aleatoriedade, algoritmos genéticos são um bom exemplo. Embora os resultados finais possam não ter uma grande diferença, a população e os caminhos de cada indivíduo será diferente a cada execução, ou seja, os passos não são exatamente repetíveis (YANG, 2014).

Dentro do grupo dos algoritmos estocásticos também é possível separá-los em dois grupos: *heurísticos* e *meta-heurísticos*. Os métodos heurísticos de otimização tem como principal objetivo resolver problemas complexos de grande escala de maneira rápida (isso quer dizer com baixo custo computacional), para tal é necessário se utilizar de conhecimentos prévios do problema em questão como forma de direcionar o processo por atalhos na procura pela solução (MELO, 2009). Os métodos meta-heurísticos por sua vez têm uma natureza mais ampla, resolvendo os problemas de maneira mais genérica. Esses usam, de maneira complementar e compensatória, técnicas de randomização e busca local (YANG, 2014).

Vale ressaltar que essas definições de heurísticas e meta-heurísticas não são consenso

na literatura. No entanto, há uma tendência em nomear todos os algoritmos estocásticos com randomização e busca local como meta-heurística, uma vez que, a randomização fornece uma boa maneira de se afastar a pesquisa local para pesquisar em escala global.

Os algoritmo meta-heurístico tem em sua concepção dois componentes importantes, a diversificação e a intensificação. O primeiro diz respeito a capacidade de gerar soluções para explorar o espaço de busca em escala global. Por sua vez a intensificação é sobre a habilidade em focar a busca em uma região local, baseado nas informações de que sugiram que um boa solução é encontrada nesta região. Combinados, a seleção das melhores soluções que garante que estas convergirão para o otimalidade, enquanto a diversificação evita que as soluções fiquem estagnadas (paradas) em ótimos locais. A adequada combinação dos dois é geralmente o pilar para tornar a otimalidade global alcançável (YANG, 2014).

Desde seu surgimento, há pouco mais de meio século, as ideias de heurísticas baseadas em mecanismos de adaptação dos seres vivo foi pouco a pouco se firmando. A observação empírica da natureza sugere que tais mecanismos de adaptação, tem a capacidade de produzir respostas satisfatórias para problemas de grande complexidade. Exemplos como processo evolutivo de adaptação das espécies ao ambiente, a busca coordenada de alimentos das colônias de formigas e a resposta do sistema imunológico de mamíferos à invasão de agentes patógenos, são buscas sobre funções de elevada complexidade, que mudam com o passar do tempo, em espaços de alternativas de elevada dimensão, e que apresentam elevado nível de sucesso. Por esse motivo acredita-se que transportar as estruturas destes mecanismos para a construção de soluções computacionais para tratar de problemas de otimização de alta complexidade tem capacidade de obter bastante êxito (CUNHA; TAKAHASHI; ANTUNES, 2012).

2.4 Teoria da evolução

Uma das teorias biológicas mais importantes é a *Teoria da evolução*, conhecida por sua apresentação clássica feita por *Charles Darwin* em seu livro “A Origem das Espécies” (1859) (RIDLEY, 2004), até hoje é base de uma área do conhecimento conhecida como biologia evolutiva. A teoria da evolução, que introduziu o conceito de evolução por seleção natural, foi bastante contestada em princípio por apresentar alguns pontos frágeis, por não esclarecer quais eram os mecanismos responsáveis pelas variações verificadas nas espécies e como tais variações eram transmitidas(RIDLEY, 2004). No início do século XX, ao relacionar as obras de *Darwin* e *Gregor Mendel*, novos conceitos relacionadas à transferência das características dos seres vivos entre as gerações deram origem à *Teoria Sintética da Evolução*. As obras “Genetic Theory of Natural Selection” (1930), de *Ronald Fisher*, e Evolution, the “Modern Synthesis” (1932), de *Julian Huxley*, foram importantes para consolidar a estrutura conceitual da síntese entre variação e herança genética com seleção natural (SANTANA; TIDON; SIMON, 2020). Na segunda metade do século XX, a teoria sintética da evolução passou a ser a explicação científica mais

aceita para o evolucionismo biológico, e que tem como ideias principais: a variabilidade genética e seleção natural.

O processo evolutivo é observado do ponto de vista da população, que pode ser entendido como o conjunto de indivíduos da mesma espécie que vive em um determinado local e ao mesmo tempo. Observa-se que quanto maior é a diversidade em uma população, maior é a probabilidade de ela se adaptar a mudanças que venham a acontecer no meio, pois maior é a probabilidade de que hajam indivíduos portadores de conjuntos genéticos que sejam favorecidos pela seleção natural (FERNANDES, 2009).

Uma das definições de evolução relaciona a mesma como uma mudança no *fundo genético* das populações, entendendo o fundo genético como o conjunto de todos os genes presentes numa população num dado momento (FERNANDES, 2009).

2.5 Algoritmo genético

Algoritmo genético (AG) é um método otimização, bio-inspirado, que toma como base a teoria de seleção natural apresentada por Charles Darwin em seu livro "A Origem das Espécies". O modelo é inspirado nos mecanismos de evolução de populações de seres vivos. Introduzido por John Holland (Holland, 1975) e popularizados por um dos seus alunos, David Goldberg (Goldberg, 1989), o método iterativo de evolução segue a seguinte estrutura base: criação de população inicial, cálculo de aptidão de cada indivíduo, verificação das condições de parada, seleção dos melhores indivíduos, cruzamento das melhores soluções, mutação de alguns indivíduos (veja Figura 1).

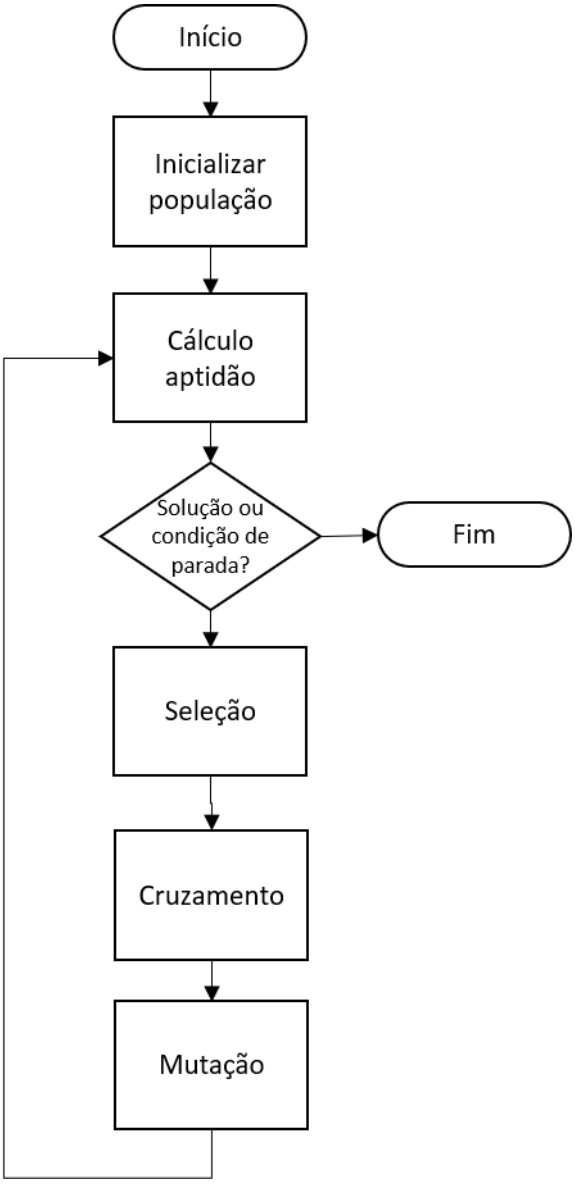
2.6 Outras definições relevantes

Alguns outros termos e conceitos biológicos serão referenciados no presente trabalho. A seguir são apresentadas breves definições que ilustram o ponto de partida para suas utilizações.

2.6.1 Ecossistema

Proposto pela primeira vez pelo ecólogo *Sir Arthur G. Tansley* (1935) na revista científica Ecology, o termo *Ecossistema* pode ser entendido como um conjunto de comunidades que vivem em um determinado local e interagem entre si e com o meio ambiente, ou também, como um conjunto de elementos formado pelos componentes bióticos e abióticos. Sendo os componente bióticos constituídos pelos seres vivos do sistema, tais como plantas e animais. Por sua vez fatores abióticos, são as partes sem vida do ambiente, como o solo, a atmosfera, a luz e a água. Por definição, ecossistemas são sistemas complexos com auto-organização, auto-regulação e sistema de auto-desenvolvimento. A principal característica do ecossistema é a presença de um conjunto relativamente fechado, estável no espaço e no tempo, de matéria e fluxo de energia entre

Figura 1 – Estrutura básica de Algorítimo Genético



Fonte: Elaborada pelo autor.

os fatores bióticos e abióticos. Um ecossistema também é um sistema aberto e caracteriza-se pelos fluxos de entrada e saída de matéria e energia, dadas as características que descrevem os ecossistemas biológicos. A utilização do termo foi expandida para descrever outros sistemas complexos com auto-organização, auto-regulação e sistema de auto-desenvolvimento.

2.6.2 Coevolução

A coevolução foi definida por *Ehrlich e Raven* (1964) como o resultado de um processo evolutivo afetando dois ou mais táxons[1] com relações ecológicas estreitas, mas que não têm

[1] Unidade taxonômica nomeada (p.ex. Homo sapiens, Hominidae ou Mammalia), pela qual indivíduos ou conjuntos de espécies são assinalados. [Oxford Languages]

trocas genéticas entre si, dos quais as pressões seletivas recíprocas agem de forma que a evolução de cada seja parcialmente dependente e impactante na evolução do outro (PINTO-COELHO, 2007). A coevolução pode incluir interações como competição, predação, mutualismo e outras formas de protocooperação.

2.6.3 Dinâmica populacional

Um aspecto muito importante da ecologia populacional é o estudo da variação do conjunto de organismos que compõe a população, ao longo do tempo e do espaço, ao estudo que se ocupa de explicar essas variações é dado o nome de *Dinâmica populacional* (GOMES, 2002). A dinâmica de populações pode ser observada pela variação temporal dos parâmetros populacionais, tais como tamanho de população, taxa de recrutamento, taxa de sobrevivência, taxa de mortalidade, taxa de fecundidade, migração e estrutura etária. A dinâmica populacional relaciona a variação numérica dos parâmetros populacionais a suas causas. Têm-se como causas diretas o nascimentos, as imigrações, mortes e emigrações, por sua vez os fatores que afetam estas são considerados as causas finais.

CAPÍTULO

3

BREVE REVISÃO

3.1 Simulador de ecossistemas

É possível encontrar na literatura uma vasta gama de implementações de simuladores de ecossistemas. Num contexto meio-ambiental estes sistemas ajudam a prever como os ecossistemas reagirão a mudanças ambientais, como a alteração da temperatura ou a introdução de espécies invasoras. Estes sistemas simuladores podem ser baseados em equações matemáticas complexas ou em algoritmos evolutivos e são aplicados a diversos propósitos. Trabalhos como "Modelagem Computacional de Ecossistemas com Competição por Recursos e Evolução em Ambientes Heterogêneos"(DAZA, 2020) "Simulação de Mini-Ecossistemas Vegetais em Tempo Real"(ENDO, 2003) são exemplos de trabalhos que se apoiam nesta metodologia para desenvolver seus estudos.

Alguns exemplos de simuladores de ecossistemas incluem:

- ECOSIM[1]: um simulador de dinâmica de populações de várias espécies em um ecossistema.
- Landis-II[2]: um simulador de dinâmica florestal que modela a evolução de florestas ao longo do tempo.
- NetLogo[3]: um ambiente de simulação que permite programar modelos de ecossistemas e explorar o impacto de ações sobre os seres vivos e o meio ambiente.
- SORTIE[4]: um simulador de dinâmica de populações de árvores em uma floresta.

[1] https://github.com/connor-brooks/ecosim
[2] https://www.landis-ii.org
[3] https://ccl.northwestern.edu/netlogo
[4] http://www.sortie-nd.org

- LANDIS-PRO[5]: um simulador de paisagem que modela a dinâmica de ecossistemas em grande escala.

Estes são apenas alguns exemplos de simuladores de ecossistemas. Há muitos outros simuladores que abrangem diferentes tipos de ecossistemas, como oceanos, rios, savanas, entre outros. O Simulador ECOSIM foi usado como modelo de base para uma nova implementação, revisada e estendida, deste simulador, onde foi incluído o operador genético "elemento estrangeiro". Este novo simulador foi implementado como parte deste trabalho e pode ser acessado em: https://github.com/soutofelipe/ecosim

3.2 Elemento estrangeiro

Após busca na bibliografia especializada, foi possível encontrar diversas estratégia para ampliar a variabilidade genética das populações em AGs com intuito de acelerar e garantir o atingimento da otimalidade neste tipo de implementação. Uma dessas técnicas é a a *Hiper-mutação*, proposto inicialmente por Cobb e Grefenstette (COBB; GREFENSTETTE, 1993). Trata-se de um mecanismo que aumenta as taxas de mutação periodicamente, de acordo com critérios de análise, como por exemplo a analise o fitness médio da população, que pode indicar conformação semelhante, neste caso aumenta-se a taxa de mutação para promover diversidade na população. No mesmo trabalho Cobb e Grefenstette (COBB; GREFENSTETTE, 1993) também propõem o conceito de *Imigrantes Aleatórios* e são tratados também em trabalhos como "A Self-Organizing Random Immigrants Genetic Algorithm for Dynamic Optimization Problems"(TINóS; YANG, 2007) e "Técnicas de Controle da Diversidade de Populações em Algoritmos Genéticos para Determinação de Estruturas de Proteínas"(Ó, 2009). Esse mecanismo consiste na substituição indivíduos da população por novos indivíduos criados aleatoriamente, a cada geração. Neste contexto a escolha dos indivíduos a serem substituídos pode ser feita de maneira aleatória ou através de critérios como o fitness. Algumas discussões (não acadêmicas) propõem a diferenciação entre "Imigrante Aleatório"e "Elemento estrangeiro"baseado no processo de surgimento do indivíduo. Sendo o imigrante aleatório criado randomicamente (como já descrito anteriormente) e o elemento estrangeiro por sua vez sendo o resultado da exportação de indivíduos entre simulações apartadas e concorrentes. Para o desenvolvimento deste trabalho o conceito de "Elemento estrangeiro"será exatamente igual ao de "Imigrante aleatório"já encontrado na literatura.

[5] https://www.fs.usda.gov/research/treesearch/45665

CAPÍTULO

4

METODOLOGIA

4.1 Estratégia

Dado o contexto teórico, estes conceitos foram utilizados para construção de um simulador de ecossistemas. No presente capítulo, primeiramente será estreitado o conceito do principal efeito de interesse, na sequência se apresentam as estruturas e características do simulador. Esse será o ponto de partida para o capítulo seguinte, onde será possível explorar os impactos de alguns parâmetros. E com a escolha de um padrão de parâmetros, será colocada a teste a validade do operador genético *elemento estrangeiro*.

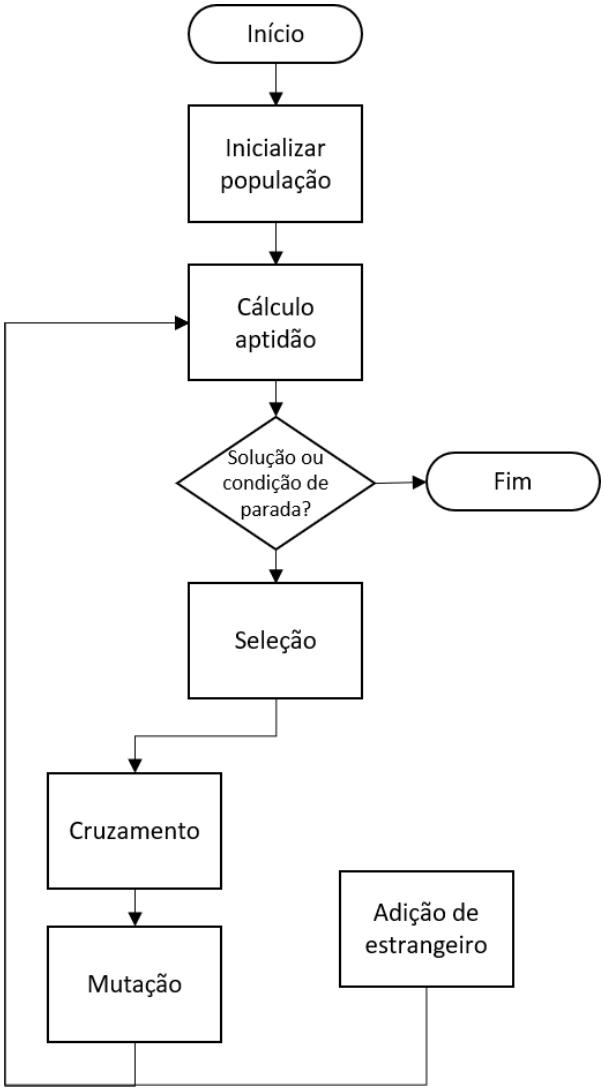
4.2 Primeiro contato com elemento estrangeiro

Apesar de a inspiração para o operador em questão ter surgido de uma implementação prática de um algoritmo genético[1], no presente trabalho não foi implementado um AG ou sistema evolutivo ”convencional”. O trabalho desenvolvido consiste de uma simulação de ecossistema, com populações antagônicas que competem pela sobrevivência, o comportamento de seleção e evolução deverá ser observado dada a pressão de seleção que o ambiente exerce sobre as populações (e indivíduos). Na implementação original, do operador, a cada geração havia uma probabilidade (constante) de que um dos indivíduos novos da próxima geração fosse gerado por um motor de geração aleatória, ao invés de ser resultado da reprodução (com combinação de genes). Em relação ao restante do processo tudo seguia o que pode ser entendido como fluxo padrão de um AG. O indivíduo gerado dessa forma passava pela mesma função de avaliação de aptidão e sofria a mesma pressão de seleção. O diagrama do fluxo base de um AG com a

[1] A implementação em questão era a otimização de problema de alocação de trades para carteiras de ações de fundos de investimentos. Por questões de propriedade intelectual o código não poderá ser compartilhado com este trabalho, então optou-se por um estudo e aplicação de Algoritmos Genéticos em Sistemas Dinâmicos (Ecossistemas)

extensão deste passo pode ser observado na Figura 2.

Figura 2 – Estrutura de Algoritmo Genético com passo probabilístico de inserção de estrangeiro



Fonte: Elaborada pelo autor.

4.3 Variabilidade de algoritmos genéticos

De maneira geral o sucesso de um algoritmo de otimização se traduz na capacidade de encontrar a solução ótima num tempo adequado. Quando se fala de algoritmos genéticos o caminho para o sucesso passa por duas habilidades que precisam ser equilibradas, uma boa convergência, geralmente conquistada a partir de uma adequada pressão de seleção da população e uma boa exploração do espaço de soluções, que ao contrário da primeira é resultado da presença de uma maior variância genética da população.

4.3.1 Mutação

O operador de mutação dentro de um algoritmo genérico tem exatamente a função de gerar diferenciação entre as soluções geradas a partir da reprodução e as soluções genitoras. No experimento conduzido a mutação segue a mesma lógica da implementação clássica de algoritmos genético, ou seja, após a reprodução um número limitado de genes tem um probabilidade Pm de ser alterado aleatoriamente.

4.3.2 Elemento estrangeiro

A mutação para o processo de evolução, dentro de um algoritmos genético, tem papel crucial na exploração do espaço de solução. Porém a mutação tem de ser usada com certa cautela, pois se a mesma tiver grande probabilidade de ocorrência e gerar uma diferença nos indivíduos que pode dificultar que haja convergência da população. Visto que todos os indivíduos nascidos a cada nova geração são sujeitos a mesma. Dito isto, o operador de elemento estrangeiro surge como alternativa complementar para este ponto. Neste etapa do processo, há uma probabilidade Pe de que a cada geração um indivíduo, sem nenhum gene proveniente dos indivíduos presentes, seja introduzido na população. No presente experimento existe um controle de ativação deste operador, afim de comparar os efeitos do mesmo em relação a mutação e também a ausência de operadores de variabilidade.

4.4 Simulador ecosim

4.4.1 Introdução ao Ecosim

O simulador utilizado neste estudo é estendido do `Ecosim` trabalho de Connor Brooks (inspirado em (MORIWAKI *et al.*, 1997)). A ferramenta executa a simulação de um sistema dinâmico que mimetiza um ambiente biológico com seres que disputam a sobrevivência, para analisá-la pode-se partir de três pilares: instância, ambiente e agentes.

4.4.2 Instância

A instância comporta em si um ambiente e seus agentes, dessa forma é possível entende-la como o ecossistema, os parâmetros associados a ela definem o ponto de partida do ecossistema: *população inicial*, *percentual de geração populacional entre carnívoros e herbívoros* e a *quantidade inicial de plantas*. Como saída, além das séries temporais extraídas de cada simulação, a cada execução uma dessas instâncias pode ser observada através da interface gráfica (aqui é possível ver uma simulação <https://www.youtube.com/watch?v=qGwsEklOaGY>). Ressaltando que a cada execução da ferramenta, múltiplas instâncias podem ser executadas em paralelo.

4.4.3 Ambiente

O ambiente é o espaço ocupado pelos agentes, são associados a ele parâmetros que definem as mecânicas do ecossistema, exemplo: *coordenadas do espaço*, *frequência de aparecimento das plantas*, *energia padrão das plantas*, *energia inicial dos agentes* (ativos), *máxima velocidade* dos agentes, *energia limite de vida*, *limiar genético* entre herbívoro e carnívoro e *taxa de mutação*.

4.4.4 Agentes

Todos os elementos que interagem no ambiente são os agentes do ecossistema, existem três tipos de agentes: carnívoros, herbívoros (agentes ativos ou vivos) e plantas (agente passivo ou matéria orgânica que serve de alimento). Cada agente tem suas características individuais e variáveis próprias. As variáveis são as propriedades que variam com o tempo, e são: *posição*, *velocidade*, *energia* e *estado*. Já as características são as propriedades perenes de cada indivíduo, no conceito dos algoritmos genéticos seu cromossomo, são: *dieta* (carnívoros, herbívoros ou próprio), *velocidade metabólica*(impacta no gasto energético e na velocidade), *energia mínima para reprodução*, *comportamento de rebanho*, *alcance da visão* e *oscilação*. Os agentes têm interações de reprodução (entre os da mesma espécie) e de predação (sendo que herbívoros comem planta e carnívoros comem herbívoros). Enquanto a predação e a morte por falta de energia fazem o papel da seleção natural, a reprodução faz o papel da perpetuação dos genes dos mais adaptados, além disso outros operadores genéticos como mutação acrescentam a variabilidade genética. Neste contexto as populações dos agente passam por um processo de coevolução.

4.4.5 Parâmetros

Para melhor entender em detalhes o simulador e como funciona a simulação um ótimo ponto de partida é entender sua classe de configuração (Apêndice A - código fonte 2, e o que ela controla).

Configurações da Engine do Ecosim:

- *ENGINE_FPS* - quantidade de quadros por segundo a renderizar na interface visual;
- *ENGINE_WINDOW_X* e *ENGINE_WINDOW_Y* - configuração da resolução da janela da interface gráfica;

Configurações da Instância:

- *WORLD_MIN_COORD* e *WORLD_MAX_COORD* - configuração do tamanho do ambiente onde estarão os agentes.

- *INSTANCE_INIT_AGENT_COUNT* - quantidade inicial de agentes vivos[2] na simulação;
- *INSTANCE_FOOD_SPAWN_INIT* - quantidade inicial de plantas[3] na simulação;
- *INSTANCE_FOOD_SPAWN_FREQ* - a frequência de tempo na qual plantas são inseridas no ambiente;
- *INSTANCE_FOOD_SPAWN_MIN* e *INSTANCE_FOOD_SPAWN_MAX* - quantidade mínima e máxima de plantas inseridas a cada vez;
- *INSTANCE_FOOD_ENERGY* - valor energético das plantas;
- *INSTANCE_FOOD_METAB* - velocidade metabólica das plantas;
- *INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MIN* e *INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MAX* - intervalo da relação de força entre o predador e a presa na qual é possível que o predador coma a presa, isso impacta na pressão cadeia alimentar tem no sistema;
- *INSTANCE_REPRODUCTION_TIME_INTERVAL* - intervalo de tempo no qual um mesmo agente tem que esperar para se reproduzir (novamente);
- *INSTANCE_CONTROL_MUTATE_START* e *INSTANCE_CONTROL_MUTATE_STOP* - controla dentro da simulação o intervalo de tempo no qual o operador genético estará em funcionamento, muito usado para testar o impacto ao propiciar comparação entre os períodos com e sem sua atuação;
- *INSTANCE_FOREIGNER_START* e *INSTANCE_FOREIGNER_STOP* - da mesma forma que os parâmetros anteriores, controla o intervalo em que o operador de elemento estrangeiro está em funcionamento;
- *INSTANCE_FOREIGNER_SPAWN_PROBABILITY* - define a probabilidade de, a cada intervalo, um estrangeiro entrar no sistema;
- *INSTANCE_FOREIGNER_SPAWN_MIN* e *INSTANCE_FOREIGNER_SPAWN_MAX* - controla a quantidade máxima e mínima de estrangeiros que podem ser inseridos uma vez que sorteado de inseri-los.

Configurações gerais dos agentes:

- *AGENT_MAX_VELOCITY* e *AGENT_MIN_VELOCITY* - escala de deslocamento do agente no espaço nas direções positivas e negativas dos eixos;

[2] nesse simulador as plantas não são tratadas como seres vivos, uma vez que não se reproduzem e não morrem a menos que sejam comidas

[3] também é possível nesse contexto se referir as plantas como *matéria orgânica*

- *AGENT_ENERGY_DEFAULT* - energia padrão que agente recebe ao ser gerado;
- *AGENT_ENERGY_SEXUAL_REPRODUCTION* - energia gasta pelos agentes progenitores durante a reprodução sexuada;
- *AGENT_ENERGY_ASEXUAL_REPRODUCTION* - energia gasta pelo agente progenitor durante a reprodução assexuada;
- *AGENT_MAX_SPEED* - valor máximo da velocidade do agente no espaço;
- *AGENT_MATE_REACH_SPACE* - distância máxima entre os agente para que seja possível a reprodução sexuada;
- *AGENT_EAT_REACH_SPACE* - distância máxima entre os agentes para que seja possível predador comer a presa;
- *AGENT_ENERGY_DEAD* - energia mínima necessária para que um agente vivo não morra naturalmente;
- *AGENT_TIME_FACTOR* - fator que define o impacto da passagem do tempo nos agentes;
- *AGENT_DIET_BOUNDARY* - valor que define a probabilidade de na geração aleatória um agente ser carnívoro ou herbívoro;
- *AGENT_SEXUED_BOUNDARY* - valor que define a probabilidade de na geração aleatória um agente ter reprodução sexuada ou assexuada;
- *AGENT_DNA_SPECIES_RANGE* - intervalo de variância genética na qual os indivíduos podem ser considerados da mesma espécie;
- *AGENT_DNA_MUTATE_PROBABILITY* - probabilidade de um alelo sofrer mutação durante a geração do agente;
- *AGENT_DNA_MAX_MUTATED_GENS* - quantidade máxima de alelos possíveis de sofrerem uma mutação ao mesmo tempo;
- *AGENT_DNA_MUTATE_RATE* - para uma funcionalidade alternativa de mutação, onde a mutação é feita como fator do valor do alelo.

Configurações dos alelos:

- *AGENT_METAB_MAX* e *AGENT_METAB_MIN* - intervalo do alelo de intensidade metabólica, alelo impacta na velocidade do agente, na força de predação e no gasto energético;
- *AGENT_VISION_MAX* e *AGEN_VISION_MIN* - intervalo do alelo de visão, impacta na distância da percepção sobre outros agentes, seja para predação/fuga e para movimento em rebanho;

- *AGENT_REBIRTH_MAX* e *AGENT_REBIRTH_MIN* - intervalo do alelo de paternidade, impacta na energia mínima necessária para o agente poder reproduzir;
- *AGENT_DIET_MIN* e *AGENT_DIET_MAX* - intervalo de sorteio para a definição da dieta;
- *AGENT_SEXUED_MIN* e *AGENT_SEXUED_MAX* - intervalo de sorteio para a definição do modo de reprodução;
- *AGENT_FLOCK_MAX* e *AGENT_FLOCK_MIN* - intervalo do alelo de comportamento de rebanho, define o quanto o agente tem tendência ou não de andar em rebanho (bandos);
- *AGENT_WOBBLE_MAX* e *AGENT_WOBBLE_MIN* - intervalo do alelo de oscilação, fator da função senoidal que incide no movimento dos agentes.

4.4.6 Simulação

No decorrer de cada iteração (passagem de tempo no ambiente), os agentes passam por algumas ações e/ou operadores:

- *Nascer* - Um agente nasce em dois contextos, no início da simulação (de forma espontânea com os genes gerados por funções pseudo aleatórias), ou a partir da reprodução ao longo do tempo;
- *Movimentar* - Para se movimentar os agentes levam em consideração os elementos no seu campo de visão. Há três diferentes relações, para agentes da mesma espécies R_i calcula-se a velocidade de rebanho Vr , para os agente que servem de comida C_i , para os agente que são predadores P_i .

  Considerando:

$$Norm(\overrightarrow{V}) = \frac{\overrightarrow{V}}{\| \overrightarrow{V} \|} \tag{4.1}$$

  Temos $\overrightarrow{Vr}$:

$$\overrightarrow{Vr} = Norm(\frac{\sum_{i=1}^{Nr} \overrightarrow{R_i v}}{Nr}) + Norm(\frac{\sum_{i=1}^{Nr} \overrightarrow{R_i p}}{Nr} - \overrightarrow{Ap}) + Norm(\frac{\sum_{i=1}^{Nr} \overrightarrow{R_i p} - \overrightarrow{Ap}}{-Nr}) \tag{4.2}$$

  E $\overrightarrow{Vp}$:

$$\overrightarrow{Vp} = Norm(\overrightarrow{Av} - \sum_{i=1}^{Nc} Norm(\overrightarrow{Ap_{i-1}} - \overrightarrow{C_i p}) + \sum_{i=1}^{Np} Norm(\overrightarrow{Ap_{i-1}} - \overrightarrow{P_i p})) \tag{4.3}$$

  Assim chega-se a velocidade final do agente $\overrightarrow{Av}$ usando o dado genético de rebanho Tr

$$\overrightarrow{Av} = Norm(Norm(\overrightarrow{Vr} \times Tr) + \overrightarrow{Vp}) \tag{4.4}$$

  E por fim na nova posição do agente $\overrightarrow{Ap}$ levando em consideração a oscilação W

$$\overrightarrow{Ap} = \overrightarrow{Ap} + \overrightarrow{Av} \times 0.5 \times (\sin(W^2 \times t) + 2) \tag{4.5}$$

- *Comer* - Para comer o agente analisa os elementos predáveis C_i, quais estão ao alcance, em seguida para cada um é feito um sorteio, com probabilidade baseada na relação das forças entre predador e presa *Fpp* :

  Para saber se está ao alcance faz-se o cálculo da maior distância entre o predador e a presa *Dpp* com base em suas posições:

$$Dpp(A,C) = \max(Ap_x - Cp_x, Ap_y - Cp_y) \tag{4.6}$$

  Dada as energias de predador e presa (*Ae* e *Ce*), metabolismo de predador e presa (*Am* e *Cm*) e o fator que estabelece peso entre energia e metabolismo na predação (*Fem*).

$$Fpp(A,C) = \frac{Ae \times Fem + Am}{Ce \times Fem + Cm} \tag{4.7}$$

Algoritmo 1 – Algoritmo alimentação

```
1:  procedimento ALIMENTAR(A, C[])   ▷ Agente A e lista de comida no campo de visão C[]
2:      para todo C faça
3:          se Dpp(A, C_i) < P(EAT_REACH_SPACE) então   ▷ Função P() busca parâmetro
4:              se Fpp(A, C_i) > Soteiro(PREDATOR_PREY_STRENGTH_RATIO) então
5:                  Ae ← Ae + C_i e                ▷ Predador A agrega energia da presa C_i
6:                  Expurgar(C_i)
7:              fim se
8:          fim se
9:      fim para
10: fim procedimento
```

- *Reproduzir* - Dada a característica do agente, esse pode se reproduzir de maneira sexuada ou assexuada. Quando a reprodução é assexuada, o cromossomo do descendente é gerado como clone do progenitor, cabendo exclusivamente ao operador de mutação gerar a variação. Já quando a reprodução é sexuada o código genético do descendente é construído como uma combinação dos cromossomos dos seu progenitores, havendo um sorteio alelo a alelo. Para ambas formas de reprodução as condições de energia mínima para reprodução e intervalo de tempo da última reprodução, têm que ser atendidos, para a reprodução sexuada uma condição a mais é necessária, a de que os progenitores estejam próximos (a uma distância máxima menor que o do parâmetro de distância para acasalamento). Após a reprodução, independente do modo, o(s) progenitor(es) perde um valor energético (necessário para transmitir ao herdeiro), além disso o operador de mutação também é acionado antes durante o processo de criação do novo agente.

  Existem também duas modalidades de mutação programadas, mas que por hora só uma é acionada por vez. Na primeira, o valor de um ou mais alelos (a depender do parâmetro AGENT_DNA_MAX_MUTATED_GENS) tem uma probabilidade *P* (AGENT_DNA_-MUTATE_PROBABILITY) de sofrer uma mutação e ter seu valor gerado aleatoriamente

(dentro dos valores possíveis para aquele alelo). Na outra, cada alelo tem uma probabilidade de ser alterado dado um fator que resulta do sorteiro multiplicado pelo parâmetro de taxa de mutação (AGENT_DNA_MUTATE_RATE).

- *Morrer* - A cada passo do tempo, os agente tem seu valor de reserva de energia descontado, como resultado do seu funcionamento metabólico. Quando o agente atinge um valor energético abaixo do parâmetro que define a linha limite de vida (AGENT_ENERGY_-DEAD), o agente morre. Isso faz como que ele passe a ser considerado matéria orgânica, que é equivalente às plantas, passando então a valer e se comportar como tal (o que quer dizer que: não se movimenta, tem o metabolismo muito diminuido, não se reproduz e não se alimenta de outros).

- *Expurgar* - Quando um agente serve de alimento para outro agente, ele é eliminado do ambiente e deixa de existir.

É importante destacar que muitos destes parâmetros e operações/ações que ocorrem na simulação foram inicialmente definidos no simulador original do Ecosim, onde adaptações e remodelagens foram feitas pelo autor deste trabalho, de modo a ampliar os controles de pressão de seleção e, principalmente, incluir o operador elemento estrangeiro. O simulador como um todo foi re-escrito e re-estruturado em .NetCore.

4.4.7 Saídas

Durante a simulação, para cada instância executada e gerada uma pasta com o *timestamp* e número da instância (`../logs/aaaaMMddhhmmss/n`), dentro desta pasta são gerados 4 arquivos, `config.json`, `register_data.csv`, `species_data.csv` e `logger_data.csv`.

- Em `config.json` há o registro do conjunto de parâmetros utilizados naquela execução.

- Em `register_data.csv` são gravados registros com todos os operadores (exceto movimentar).

- Em `species_data.csv` há o registro da quantidade de agentes de cada espécie.

- Em `logger_data.csv` há o registro de diversas séries temporais para análise.

A partir das saídas, principalmente das séries temporais, é que é possível se analisar as dinâmicas populacionais.

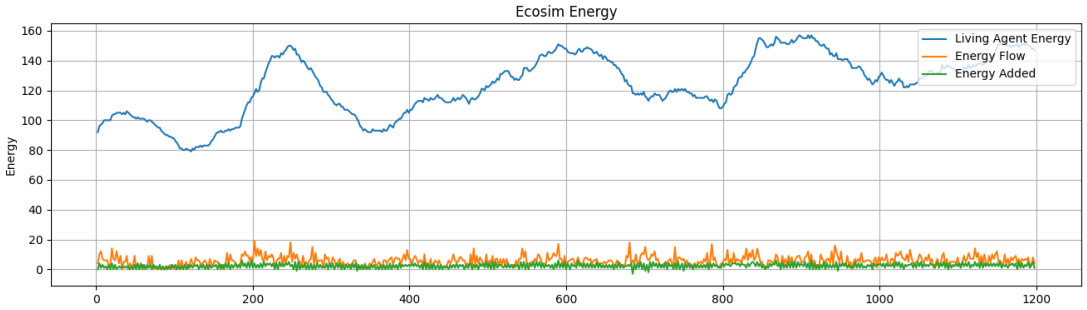


Figura 3 – Dados de Energia

Fonte: Elaborada pelo autor.

Na Figura 3 é possível observar as informações de *energia*, no gráfico estão as quantidades de energia total do sistema, de energia transmitida por operadores e de energia adicionada ao sistema. Nas séries temporais também estão presentes as informações de energia específica das populações de herbívoros e carnívoros.

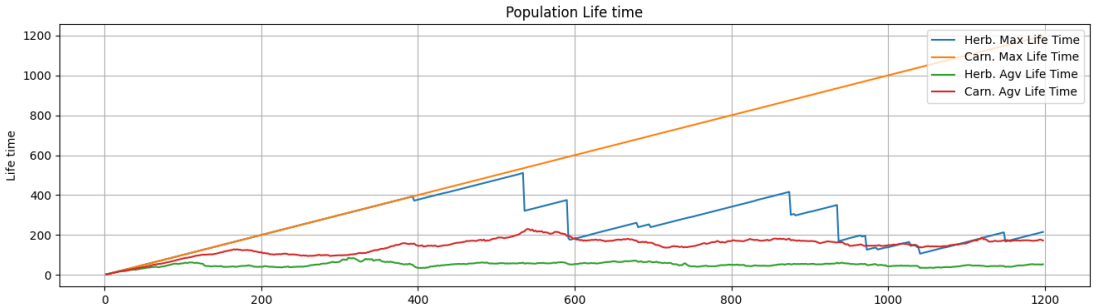


Figura 4 – Dados de Tempo de Vida

Fonte: Elaborada pelo autor.

No gráfico da Figura 4, se encontram os dados de *tempo de vida* médio e máximo dos agentes, expresso separado por classe (herbívoros e carnívoros).

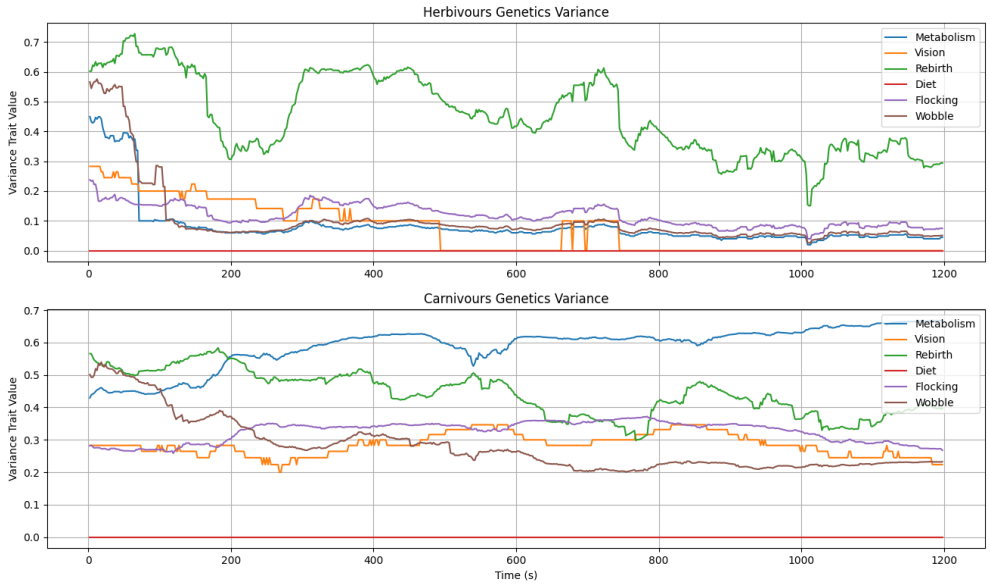


Figura 5 – Dados de Genética

Fonte: Elaborada pelo autor.

Acima, na Figura 5, estão os dados de *genética*, mais especificamente o de variabilidade. Esta informação é expressa pela variância calculada para cada um dos alelos do cromossomo (metabolismo, visão, energia de reprodução, dieta, comportamento de rebanho, oscilação), no arquivo de dados também é possível observar os valores médios dos mesmos.

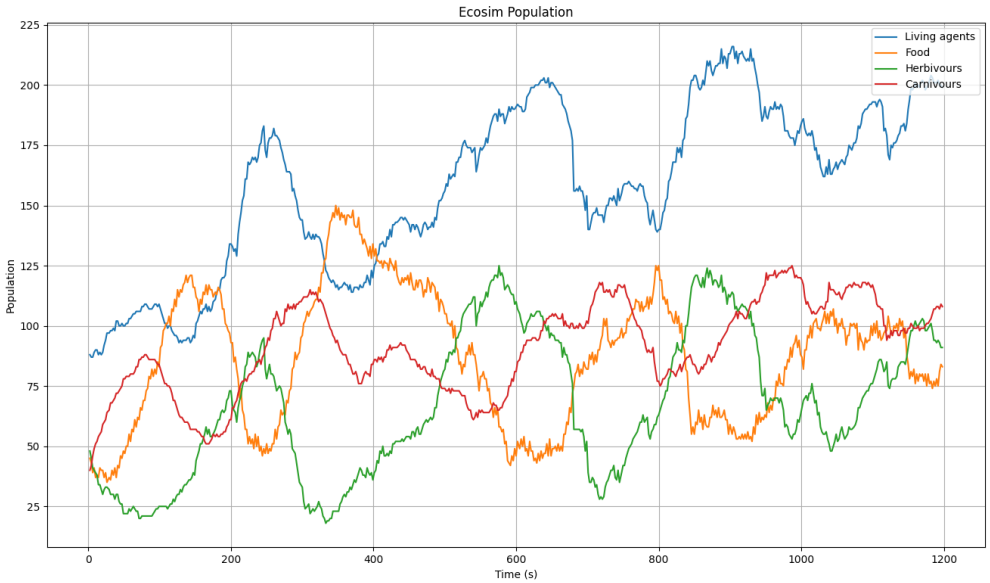


Figura 6 – Dados de População

Fonte: Elaborada pelo autor.

Por fim, o conjunto de dados com impacto de análise mais direto, que são os dados de *população* (ou densidade populacional), que são apresentados na Figura 6. Aqui se encontram as informações de quantidade de plantas, quantidade total de agentes vivos (herbívoros e carnívoros) e as quantidades específicas de agentes herbívoros e carnívoros;

4.4.8 Outras considerações

Em relação à energia é importante salientar que o sistema tem mecanismos de geração de energia através da inserção de matéria orgânica (plantas). Há também a transferência de energia através da alimentação (onde um agente se alimenta de outro) e transferência energéticas pela reprodução, uma vez que para gerar um novo agente o(s) agente(s) progenitor(es) fornece(m) energia para o agente criado. Por fim, há um gasto energético ao longo do tempo para manter o agente vivo, consequentemente um consumo de energia do sistema ao longo do tempo.

Sobre a interface gráfica (exemplificada na Figura 7) é válido destacar, é baseada em OpenGL e se utiliza da biblioteca GLFW para se integrar com .NetCore. Através de *shadders* são renderizados os agentes, com o seguinte particularidade de cores: os agentes carnívoros são representados por círculos com cores vermelhas adicionadas de tonalidades de azul a depender de espécie, analogamente os herbívoros tem sua cor base verde com variâncias de azul e por fim as plantas são representadas pela cor cinza.

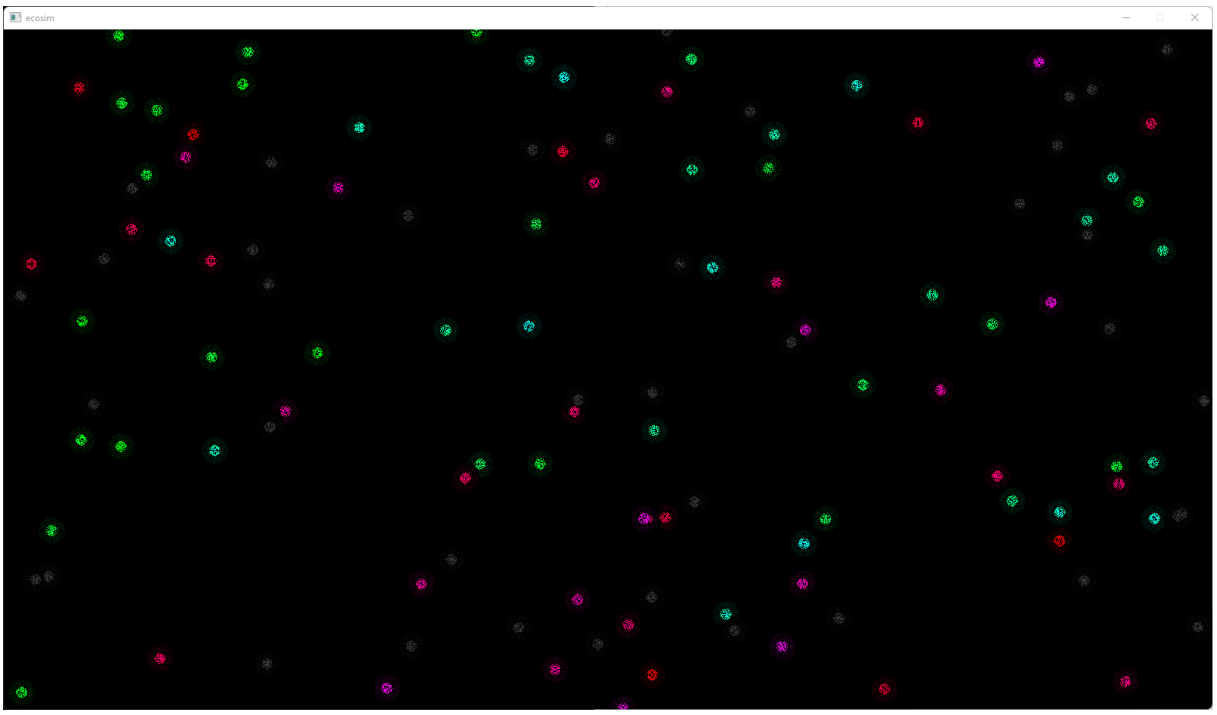


Figura 7 – Interface Gráfica

Fonte: Elaborada pelo autor.

4.4.9 Considerações finais deste capítulo

É importante destacar que esse modelo de simulação foi criado a partir de estudos de ecossistemas, visando implementar a simulação de sistemas dinâmicos em um ecossistema natural com co-evolução. Tais sistemas naturais e biológicos possuem propriedades interessantes que podem ser exploradas em diferentes aplicações computacionais, seja na área da biologia, da medicina, na agricultura, e até mesmo em mercados financeiros. Um sistema dinâmico regido por regras de predação, competição por recursos, e com populações de indivíduos que buscam "sobreviver"ou "dominar"o ambiente onde se encontram, possui inúmeras aplicações e ramificações de estudo, tanto de cunho teórico quanto de cunho aplicado e prático.

CAPÍTULO

5

EXPERIMENTOS

5.1 Simulação de controle

O processo de simulação é sensível às parametrizações, para análise dos diversos comportamentos foi selecionado, de maneira empírica, um conjunto de parâmetros que foi definido como representação do comportamento base (ou simulação de controle).

5.1.1 Parâmetros usados

Os parâmetros definidos

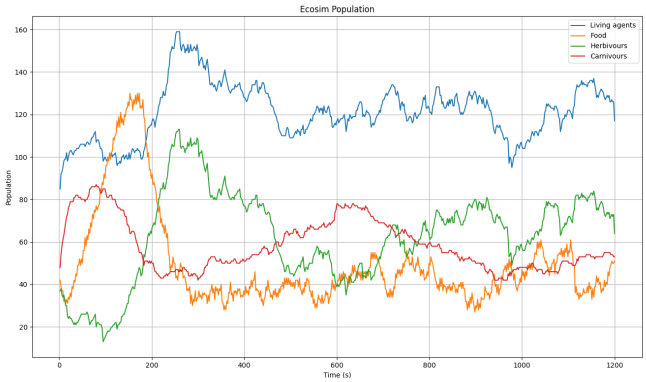
```
{
    "ecosim_dotnetcore.Configuration": {
        "ENGINE_FPS": 60,
        "WORLD_MIN_COORD": -1,
        "WORLD_MAX_COORD": 1,
        "INSTANCE_INIT_AGENT_COUNT": 90,
        "INSTANCE_FOOD_SPAWN_FREQ": 4,
        "INSTANCE_FOOD_SPAWN_INIT": 50,
        "INSTANCE_FOOD_SPAWN_MIN": 5,
        "INSTANCE_FOOD_SPAWN_MAX": 7,
        "INSTANCE_FOOD_ENERGY": 0.5,
        "INSTANCE_FOOD_METAB": 0.0001,
        "INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MIN": 0.1,
        "INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MAX": 1,
        "INSTANCE_REPRODUCTION_TIME_INTERVAL": 4,
        "INSTANCE_CONTROL_MUTATE_START": 6000,
        "INSTANCE_CONTROL_MUTATE_STOP": 12000,
```

```
18:         "INSTANCE_FOREIGNER_START": 18000,
19:         "INSTANCE_FOREIGNER_STOP": 24000,
20:         "INSTANCE_FOREIGNER_SPAWN_MIN": 0,
21:         "INSTANCE_FOREIGNER_SPAWN_MAX": 3,
22:         "INSTANCE_FOREIGNER_SPAWN_PROBABILITY": 0.01,
23:         "AGENT_MAX_VELOCITY": 1,
24:         "AGENT_MIN_VELOCITY": -1,
25:         "AGENT_ENERGY_DEFAULT": 1,
26:         "AGENT_ENERGY_SEXUAL_REPRODUCTION": 0.25,
27:         "AGENT_ENERGY_ASEXUAL_REPRODUCTION": 0.5,
28:         "AGENT_MAX_SPEED": 0.0015,
29:         "AGENT_MATE_REACH_SPACE": 0.08,
30:         "AGENT_EAT_REACH_SPACE": 0.02,
31:         "AGENT_ENERGY_DEAD": 0.3,
32:         "AGENT_TIME_FACTOR": 0.3,
33:         "AGENT_DIET_BOUNDARY": 0.5,
34:         "AGENT_SEXUED_BOUNDARY": 0,
35:         "AGENT_DNA_SPECIES_RANGE": 0.5,
36:         "AGENT_DNA_MUTATE_PROBABILITY": 0.01,
37:         "AGENT_DNA_MAX_MUTATED_GENS": 2,
38:         "AGENT_DNA_MUTATE_RATE": 0.1,
39:         "AGENT_METAB_MAX": 0.8,
40:         "AGENT_METAB_MIN": 0.1,
41:         "AGENT_VISION_MAX": 0.2,
42:         "AGENT_VISION_MIN": 0.1,
43:         "AGENT_REBIRTH_MAX": 3,
44:         "AGENT_REBIRTH_MIN": 1,
45:         "AGENT_DIET_MIN": 0,
46:         "AGENT_DIET_MAX": 1,
47:         "AGENT_SEXUED_MIN": 0,
48:         "AGENT_SEXUED_MAX": 1,
49:         "AGENT_FLOCK_MAX": 0.5,
50:         "AGENT_FLOCK_MIN": 0,
51:         "AGENT_WOBBLE_MAX": 3,
52:         "AGENT_WOBBLE_MIN": 1,
53:         "LOGGER_ENABLE": 1,
54:         "LOGGER_FREQ": 2
55:     }
56: }
```

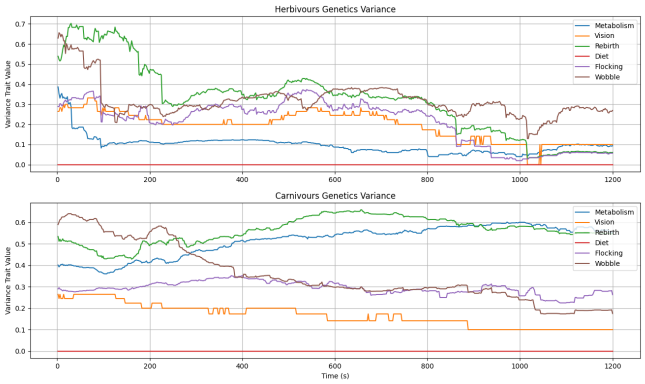
5.1.2 Dados das simulações

Esse conjunto de parâmetros foi escolhido como grupo de controle por ter demonstrado ao longo de uma série de simulações (para tal ajuste foram executadas pouco mais de 100 simulações[1]) grande probabilidade de apresentar regimes de equilíbrio[2], além de apresentar pressões seletivas capazes de produzir convergência genética. Na Figura 8a observa-se um regime de equilíbrio com coexistência entre os diferentes agentes do ecossistema com uma oscilação quantitativa entre barreiras sem tendências claras de extinção. Adicionado a isso, na Figura 8b é possível observar a diminuição da variância genética, principalmente entre os herbívoros, o que indica uma convergência das características genéticas.

Figura 8 – Simulação Controle 01



(a) População



(b) Genética

Fonte: Elaborada pelo autor.

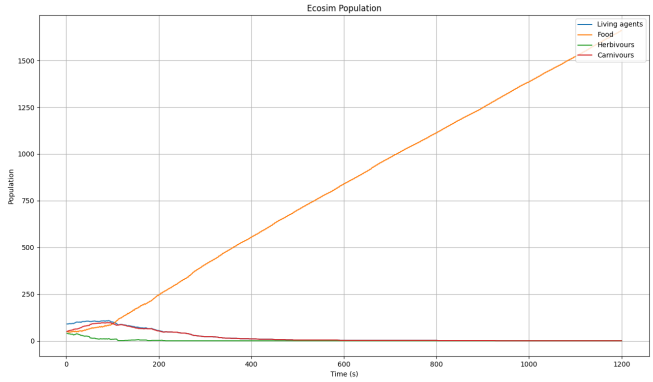
No ponto anterior fala-se em *probabilidade* e *capacidade* pois mesmo com este conjunto

[1] Alguns exemplares de simulações podem ser encontrados no Apêndice C

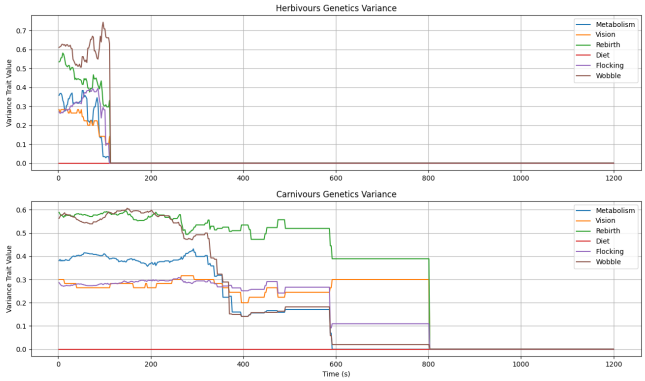
[2] Regime de equilíbrio: em um sistema dinâmico, considera-se um regime de equilíbrio quando o sistema atinge um regime contínuo e estável, ou conforme considerado neste trabalho, quando entra em um regime dinâmico com ciclos de relativa estabilidade.

de parâmetros ocorrem simulações que levam ao regime estável de extinção dos agentes vivos, com crescimento linear das plantas. Como pode-se observar na Figura 9a. Para cada um dos efeitos examinados foram executadas 20 simulações com o intuito de poder observar tendências.

Figura 9 – Simulação Controle 02



(a) População



(b) Genética

Fonte: Elaborada pelo autor.

5.2 Efeito dos parâmetros

Nesta sessão serão explorados alguns resultados[3] obtidos a partir de variações de alguns parâmetros a partir do conjunto de controle.

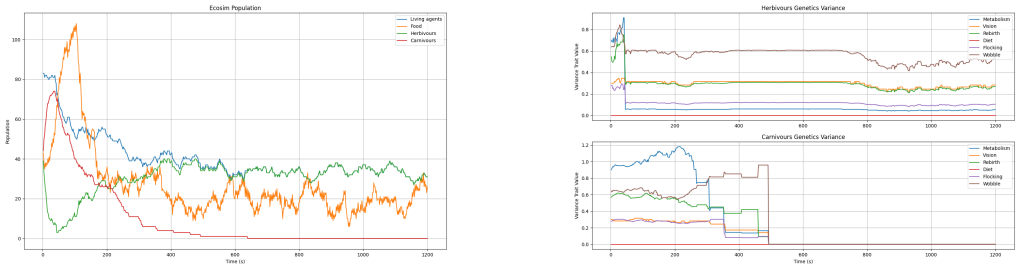
5.2.1 Efeito do metabolismo

Nos reultados apresentados nos gráficos na Figura 10a vê-se uma tendência de extinção dos carnívoros, que se repete com constância ao elevar os parâmetos de metabolismos máximo e

[3] Assim como na sessão anterior, mais resultados podem ser encontrados no Apêndice C

mínimo (*AGENT_METAB_MAX* e *AGENT_METAB_MIN*) respectivamente para os valores de 1,6 e 0,2.

Figura 10 – Simulação Metabolismo Elevado

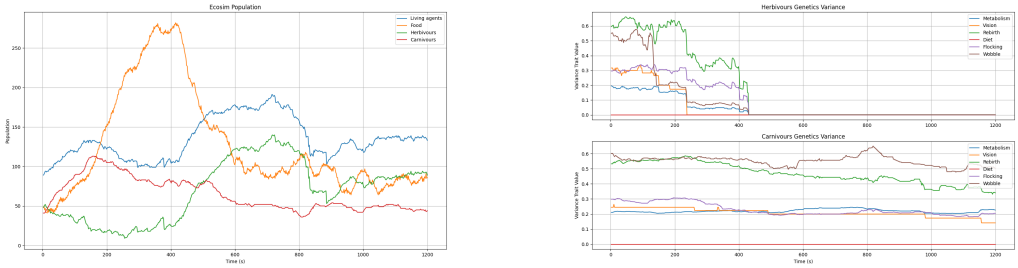


(a) População (b) Genética

Fonte: Elaborada pelo autor.

Por outro, ao variar os mesmos parâmetros em um sentido de redução (para os valores 0,4 e 0,05) o resultado nas populações aponta para regimes de estabilidade com convivência. Em relação a seleção, a tendência foi diminuição da variância genética dos herbívoros, como visto na Figura 11b.

Figura 11 – Simulação Metabolismo Reduzido



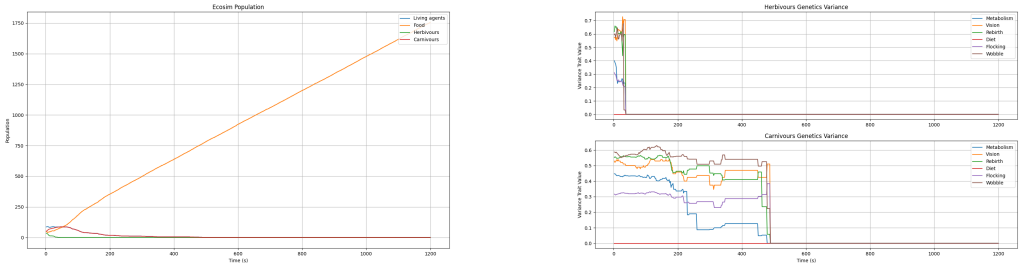
(a) População (b) Genética

Fonte: Elaborada pelo autor.

5.2.2 Efeito da visão

Nos experimentos com os valores dos parâmetros de visão elevados a 0.4 e 0.2 (máximo e mínimo), os resultados demonstraram uma tendência a clara de extinção precoce dos agentes vivos (exemplo Figura 12a).

Figura 12 – Simulação Visão Elevada

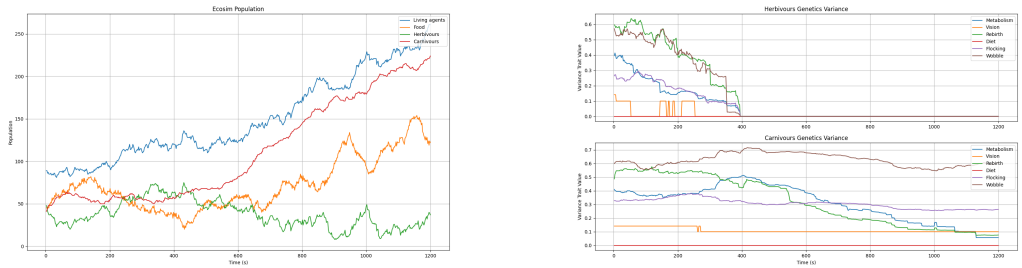


(a) População (b) Genética

Fonte: Elaborada pelo autor.

Em contraponto ao reduzir os valores para 0.1 e 0.05 (máximo e mínimo), a tendência não se apresentou clara, como pode se observar nos gráficos da Figura 13 e Figura 14.

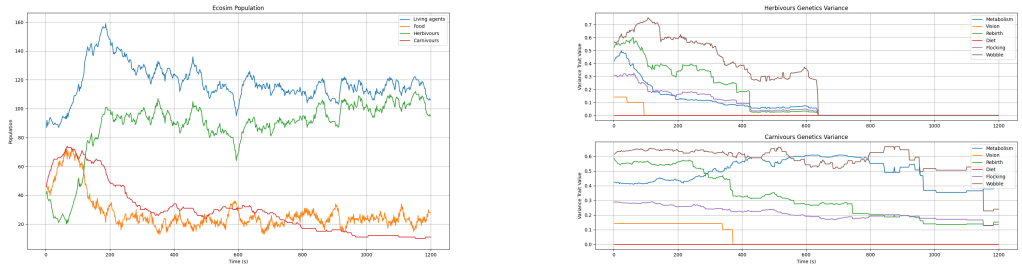
Figura 13 – Simulação Visão Reduzida



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 14 – Simulação Visão Reduzida



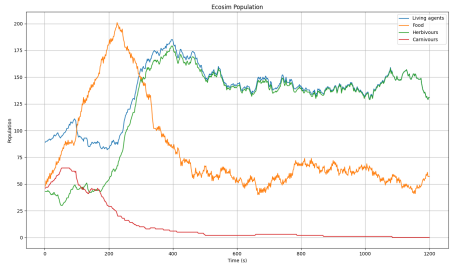
(a) População (b) Genética

Fonte: Elaborada pelo autor.

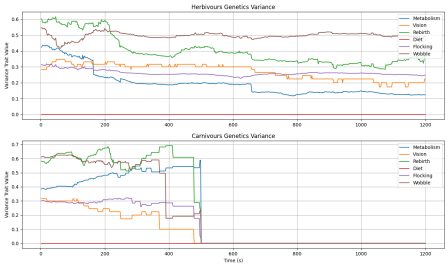
5.2.3 Pressão do espaço

Ao se dobrar o tamanho das coordenadas no espaço (quadruplicar a área) de 4 unidades quadradas para 16. Os carnívoros tem mais dificuldade em perseguir os herbírovos, isso leva a uma tendência de extinção dos carnívoros e após isso, o sistema entra um regime com plantas e herbívoros, além disso a baixa pressão seletiva nos herbívoros faz com que haja menor impacto na convergência genética, como o exemplo da Figura 15.

Figura 15 – Simulação Espaço Ampliado



(a) População

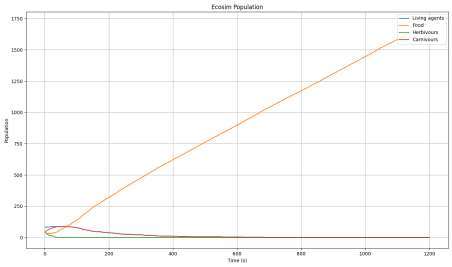


(b) Genética

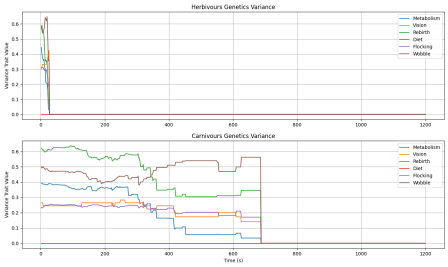
Fonte: Elaborada pelo autor.

Com efeito exatamente oposto, ao reduzir pela metade as coordenadas do ambiente, os herbívoros são rapidamente exterminados, o que leva após um tempo a extinção dos carnívoros por falta de comida, observável na Figura 16a.

Figura 16 – Simulação Espaço Reduzido



(a) População



(b) Genética

Fonte: Elaborada pelo autor.

5.2.4 Pressão de reprodução

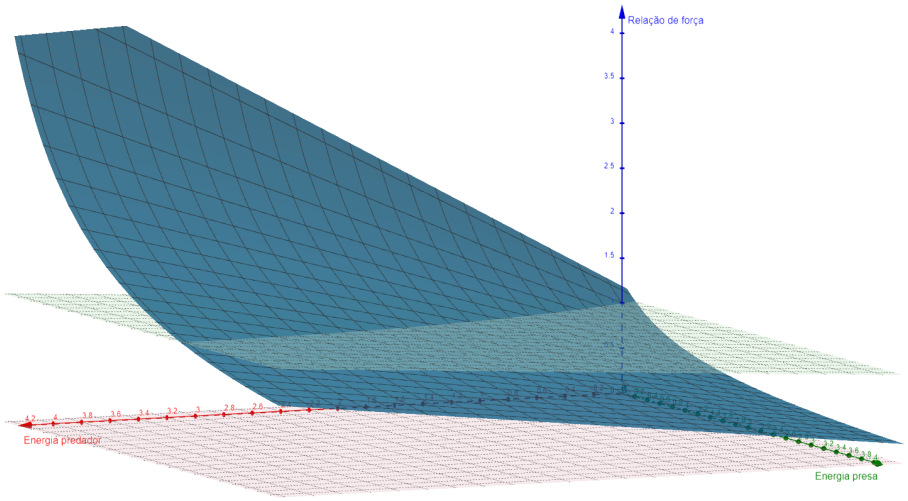
Experimentos[4] realizados alterando o intervalo entre as reproduções de um mesmo indivíduo não demonstraram tendências de dinâmicas populacionais claras, tanto ao se aumentar em dobro o intervalo quanto ao se reduzir pela metade.

[4] Dados destes experimentos presentes no Apêndice C

5.2.5 Pressão de predação

A pressão de predação pode ser compreendida pelos parâmetros que impactam nas probabilidades que definem o êxito ou não de um predador quando esse encontra (ao seu alcance) uma presa, como visto em Algoritmo 1. A paritr da Equação 4.7, fixando os metabolismos nos valores medianos (0,45) é possível obaservar o comportamento a partir da Figura 17, onde os platôs são os limites do sorteio de predação e a superfície curva é o resultado do fator de predação (Fpp) variando-se as energias de predador e presa. Toda vez que a função retornar valor maior que o sorteado o predador tem sucesso.

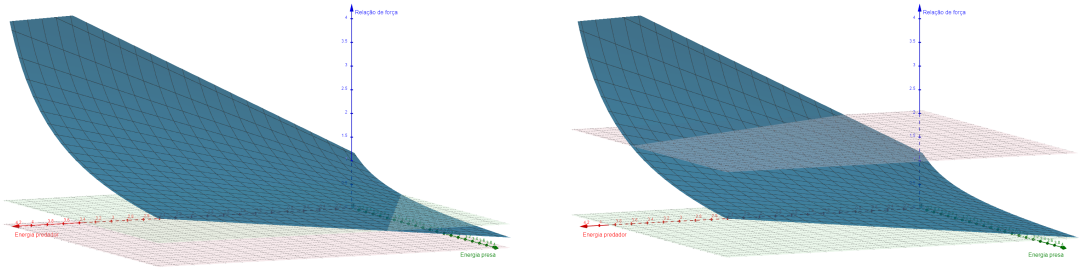
Figura 17 – Curva de Predação



Fonte: Elaborada pelo autor.

Alterando-se os parâmetros que controlam a pressão pelo sorteio (INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MIN e INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MAX), alteramos a probabilidade de sucesso da caçada, como se observa pelos volumes nos gráficos da Figura 17.

Figura 18 – Curva de Predação

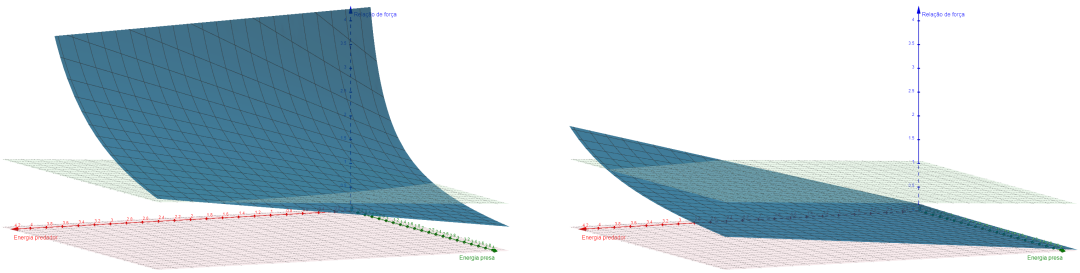


(a) Reduzindo o sorteio (b) Aumentando o sorteio

Fonte: Elaborada pelo autor.

Nos gráficos na Figura 19 também é possível ver os efeitos das diferenças metabólicas entre predador e presa.

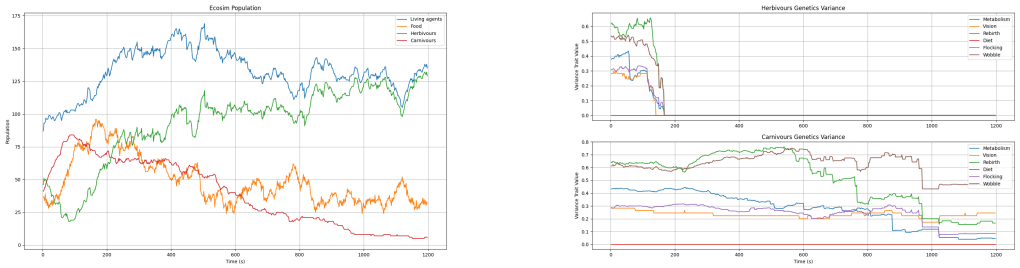
Figura 19 – Curva de Predação



(a) Predador com metabolimos maior (b) Presa com metabolimos maior

Fonte: Elaborada pelo autor.

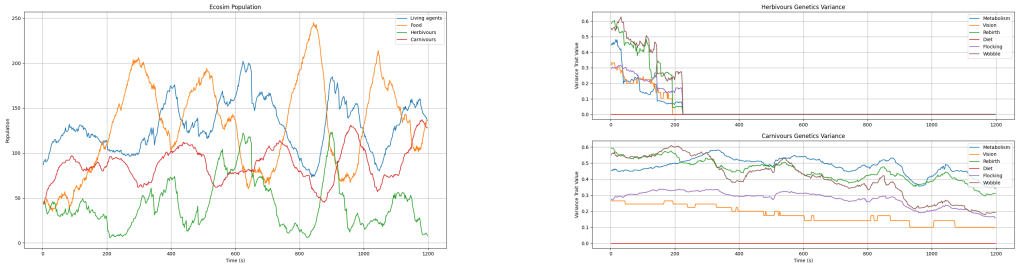
Figura 20 – Simulação mais pressão na predação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 21 – Simulação menos pressão na predação



(a) População (b) Genética

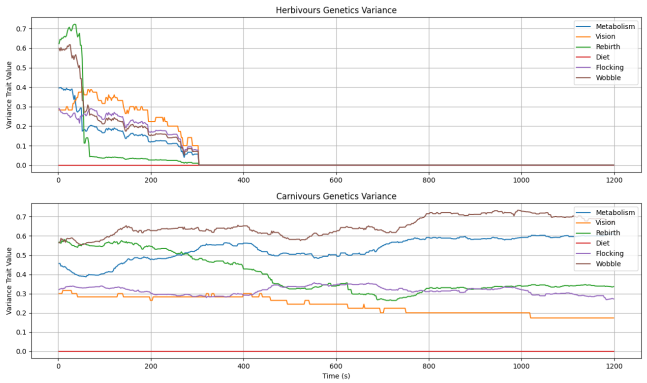
Fonte: Elaborada pelo autor.

5.3 Simulação dos efeitos dos operadores

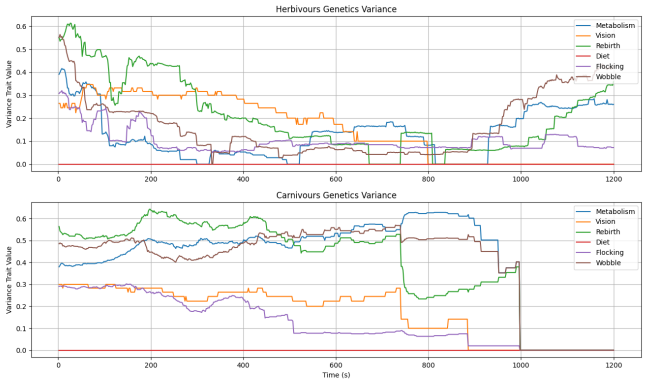
5.3.1 Operador de mutação

As simulações com mutação não demonstraram nenhuma tendência populacional específica, mas suas amostras apresentaram uma dinâmica que tarda a convergência genética se comparada à que ocorre no grupo controle. Exemplificado na Figura 22. Nestas simulações os parâmetros definiam uma probabilidade de 1% de mutação para cada indivíduo gerado, com uma variação de até 10% nos valores de cada alelo que sofreu mutação e com um número máximo de 2 alelos.

Figura 22 – Convergência genética



(a) Simulação controle



(b) Simulação com mutação

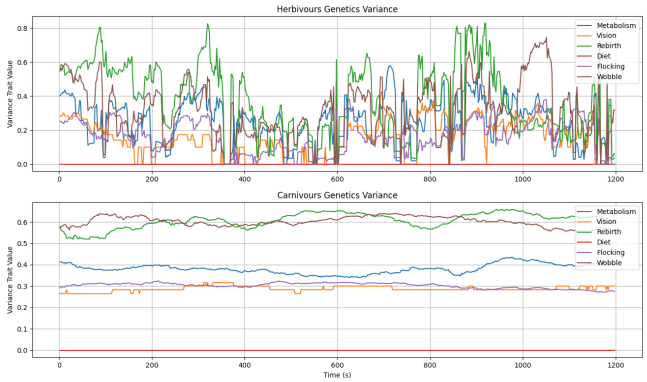
Fonte: Elaborada pelo autor.

5.3.2 Operador de elemento estrangeiro

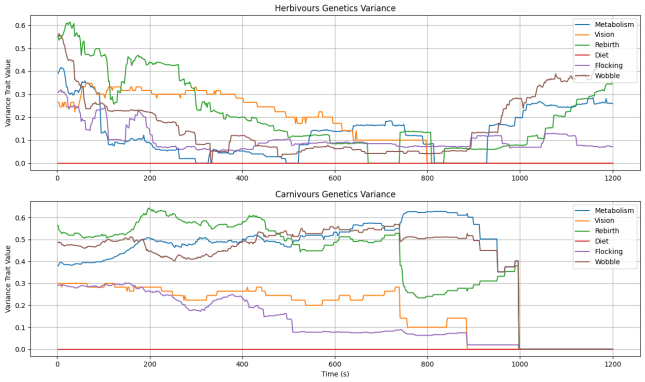
Os experimentos com presença de elementos estrangeiros apresentaram uma tendência à predominância das populações carnívoras. Mas como ponto mais relevante, as simulações foram capazes de produzir picos de variância genética mais elevados de maneira geral, o que pode ser

observado na Figura 23. Nestas simulações os parâmetros definiam a probabilidade de 1% de surgimento de estrangeiros a cada iteração, sendo que a quantidade de estrangeiros gerados em caso de sorteio positivo poderia variar de 0 a 3 indivíduos.

Figura 23 – Convergência genética



(a) Simulação com estrangeiro



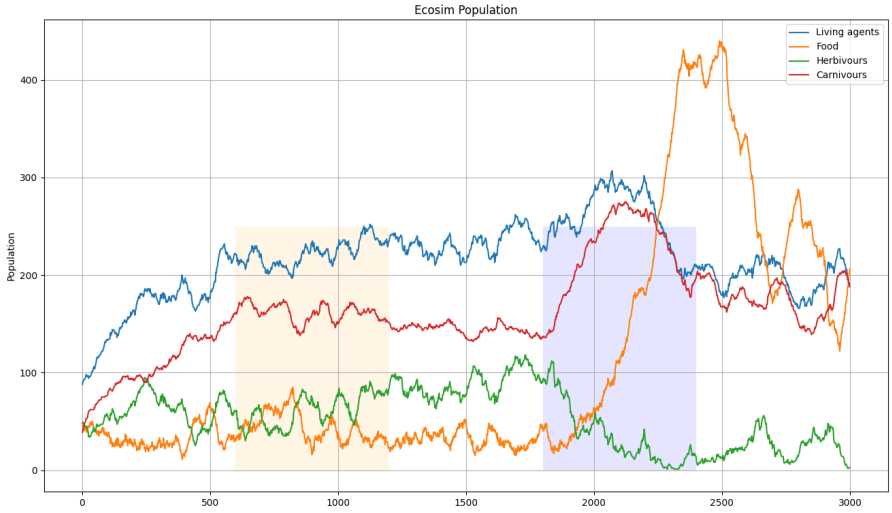
(b) Simulação com mutação

Fonte: Elaborada pelo autor.

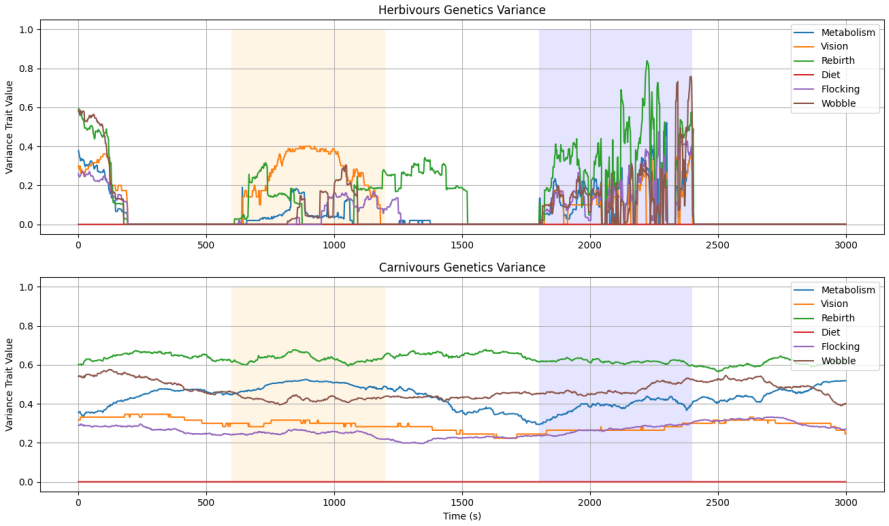
5.3.3 Simulação com intervalos intercalados

Para conseguir melhor observar a diferença do impacto entre os operadores (mutação e elemento estrangeiro), foram realizadas simulações com janelas de funcionamento de cada operador intercaladas por janelas sem esses operadores de variabilidade. A intenção principal é conseguir observar o impacto na variabilidade genética. Para tal os intervalos de tempo onde os operadores estão ativos são identificados por faixas no plano de fundo dos gráficos separando por cores amarelo e azul, para mutação e elemento estrangeiro respectivamente, para os intervalos sem cor nenhum dos operadores está ativo. Para tal experimento foram executadas 50 simulações, com os mesmos parâmetros de mutação e de inserção de estrangeiros vistos nos experimentos anteriores.

Figura 24 – Simulação com Mutação e Elemento Estrangeiro



(a) População



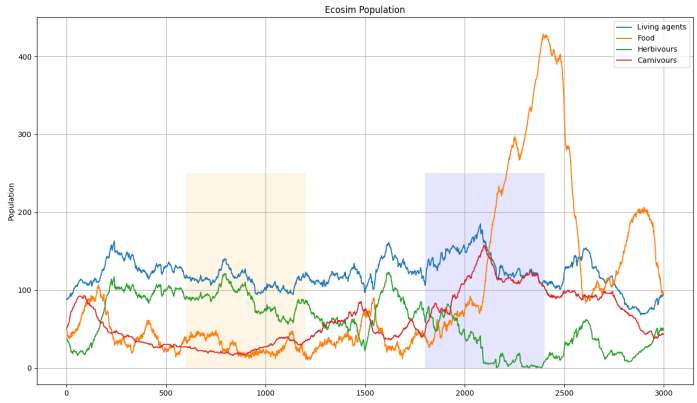
(b) Genética

Fonte: Elaborada pelo autor.

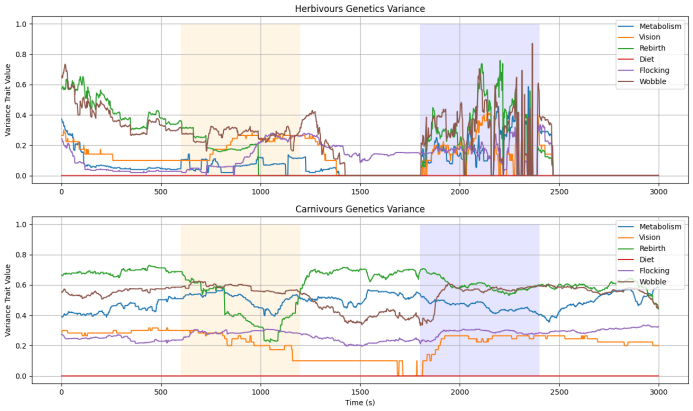
Analisando os resultados da simulação apresentada na Figura 24, é possível observar no primeiro intervalo uma expressiva convergência genética entre os herbívoros, acompanhado de um regime populacional com crescimento entre os carnívoros. Durante o período com mutação ativa, a dinâmica populacional em relação a quantidade de indivíduos não demonstra sofrer muito impacto, todavia, a variância genética dos herbívoros apresenta considerável impacto. Após essa fase, na simulação o regime populacional segue inicialmente com a mesma dinâmica, e novamente a população de herbívoros mostra tendência de convergência genética. No intervalo

seguinte, com a inserção de elementos estrangeiros, é observável um grande impacto nas métricas populacionais e genéticas, a população de herbívoros apresenta uma variância genética grande, maior em relação a todos os períodos anteriores. Quanto à quantidade de indivíduos há uma alteração na dinâmica vigente, o que na sequência leva a uma tendência de extinção dos herbívoros.

Figura 25 – Simulação com Mutação e Elemento Estrangeiro



(a) População



(b) Genética

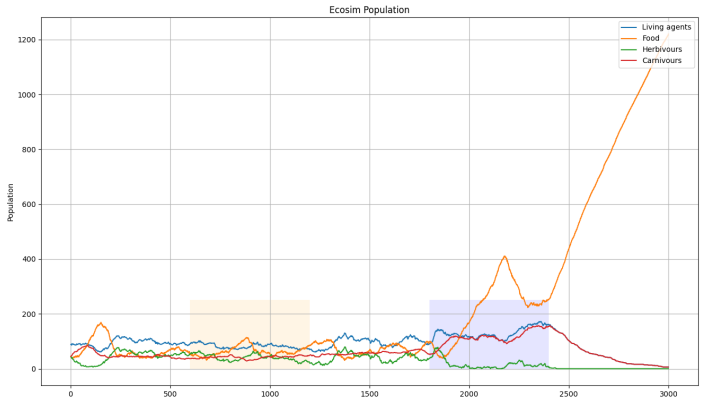
Fonte: Elaborada pelo autor.

No segundo exemplo, ilustrado na Figura 25, o regime das populações é diferente, porém compartilham a semelhança de seguir uma dinâmica dentro de certos limites até o início do quarto intervalo (início da inserção do elemento estrangeiro), após este período é observável um maior impacto nas quantidades de todos os agentes, também se repete o evento de aumento marcante da variância genética entre os herbívoros.

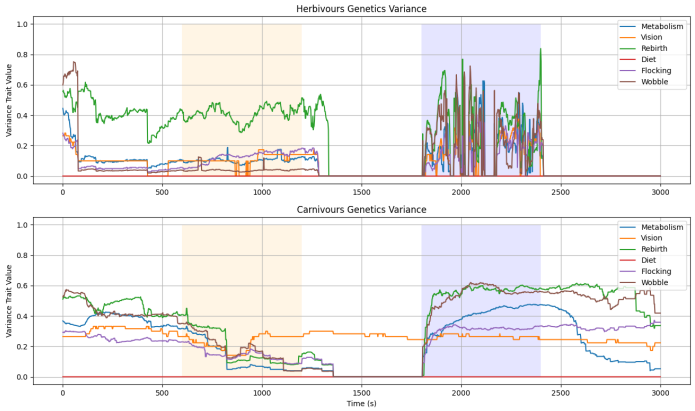
Em mais uma simulação observada, as características anteriores se repetem, com um adicional de que neste experimento há também a convergência genética entre os carnívoros. Da

mesma maneira, a ocorrência de elementos estrangeiros também impacta na variância genética dessa população.

Figura 26 – Simulação com Mutação e Elemento Estrangeiro



(a) População



(b) Genética

Fonte: Elaborada pelo autor.

5.4 Considerações finais deste capítulo

Neste capítulo foi possível observar que um sistema dinâmico pode alcançar estados de equilíbrio dinâmico, ou, até mesmo, levar a uma tendência de extinção, dentro de um ecossistema como o que foi proposto. Foi então apresentado um sistema de referência (sistema de controle), que permitiu fazer uma análise de impacto separadamente, de cada parâmetro e de cada operador, na dinâmica do sistema. Com isso, foi possível então analisar os operadores de mutação e de elemento estrangeiro, assim como o seu impacto na dinâmica populacional do ecossistema. Conseguiu-se demonstrar que o operador de mutação e o operador elemento estrangeiro se comportam de maneiras diferentes e possuem impactos diferentes na dinâmica populacional,

e portanto, o operador estrangeiro vem a ser uma contribuição interessante e diferenciada na simulação de ecossistemas e de sistemas dinâmicos como os estudados.

A ideia de inserir um elemento estrangeiro, permite causar impactos significativos nos ecossistemas, e assim alterar a dinâmica e o regime estável/equilibrado de um determinado sistema, sendo portanto, um elemento relevante para estudos em sistemas dinâmicos. O estudo de sistemas dinâmicos que possuem uma convergência para um regime de equilíbrio, pode ser fortemente impactado, quando da inserção de um elemento estrangeiro que perturba a dinâmica atual. Considerar este tipo de elementos em ecossistemas, ou em aplicações de simulação variadas (p.ex. em biologia, agricultura, aplicações financeiras), pode levar a novas descobertas e novos modelos computacionais e de simulação.

CAPÍTULO

6

CONCLUSÃO

6.1 Efeito do elemento estrangeiro

Se por um lado a análise dos resultados obtidos das simulações não são suficiente para comprovar a eficácia do operador *Elemento Estrangeiro* no contexto geral de algoritmos genéticos, há evidência claras de seu impacto na variância genética. Pode-se observar também a diferença que há neste quesito entre o operador *Elemento Estrangeiro* e o operador *Mutação*. Assim como explorado no processo de experimentação, o operador pode ser utilizado como alternativa para incremento de variância em intervalos específicos, ou combinados com estratégias de probabilidade regressiva entre gerações (análogo ao que ocorre nos algoritmos de *Recozimento Simulado*[1].

6.2 Resultados do simulador

Tão importante para este trabalho quanto o tema motivador, é a ferramenta de simulação com sua modelagem projetada para incorporar diversos aspectos de um ecossistema biológico. O simulador demonstrou grande potencial para explorar dinâmicas populacionais, capacidade de gerar variações do ambiente a partir das parametrizações e observabilidade, tanto em tempo real pela interface gráfica quando para análises posteriores partir da extração dos dados. Este simulador se constitui em uma importante melhoria dos sistemas anteriores, no qual foi baseado, sendo também uma importante contribuição deste trabalho.

[1] Também conhecido como *Simulated Annealing*

6.3 Principais contribuições e trabalhos futuros

Falando do *Elemento Estrangeiro*, a principal contribuição foi apresentar o tema já trazendo indícios de que o operador demonstra aderência a seu propósito. Para estudos futuros, afim de estreitar o funcionamento do operador em casos de otimização, sugere-se a implementação do operador em soluções de algoritmos genéticos aplicadas a problemas que têm necessidade de grande exploração do espaço de solução.

Já em relação ao simulador, o trabalho gerou uma ferramenta com código fonte que está aberto. A ferramenta pode ser obtida, a partir de repositório na internet[2], para utilização em outros estudos baseados em dinâmicas populacionais, além de poder ser expandida e estendida. Sobre a estrutura o próximo passo deve ser a melhoria no processo de registro dos dados, a intenção é torna-lo assíncrono para evitar efeito de gargalo com a gravação em disco. Em relação a simulação, ideias possíveis são: a alteração da função determinística de movimento por algum modelo evolutivo (mimetizando um processo de aprendizagem), outra é a implementação de estrutura geral de controle de parâmetros no tempo (que simularia impactos no ambiente), ou também habilitar a capacidade de intervir de maneira interativa com o ambiente, através da interface gráfica.

[2] https://github.com/soutofelipe/ecosim

REFERÊNCIAS

Ó, V. T. do. *Técnicas de controle da diversidade de populações em algoritmos genéticos para determinação de estruturas de proteínas*. Tese (Doutorado), Ribeirão Preto, 2009. Citado na página 40.

CAPRA, F. **Teia da vida - Uma nova compreensão científica dos sistemas vivos**. São Paulo: Editora Cultrix, 1996. Citado na página 31.

COBB, H. G.; GREFENSTETTE, J. J. Genetic algorithms for tracking changing environments. **S. Forrest (ed.), 5th International Conference on Genetic Algorithms, 523-530, Morgan Kaufmann**, 1993. Citado na página 40.

CUNHA, A. G.; TAKAHASHI, R.; ANTUNES, C. H. **Manual de computação evolutiva e metaheurística**. Coimbra: Imprensa da Universidade de Coimbra, 2012. Citado nas páginas 33 e 34.

DAZA, S. L. C. **Modelagem Computacional de Ecossistemas com Competição por Recursos e Evolução em Ambientes Heterogêneos**. Tese (Doutorado), Recife, 2020. Citado na página 39.

ENDO, L. C. Y. **Simulação de Mini-Ecossistemas Vegetais em Tempo Real**. Tese (Doutorado), São Paulo, 2003. Citado na página 39.

FERNANDES, J. B. **Neodarwinismo - Teoria sintética da evolução**. 2009. Disponível em: <http://bionaturalife.blogspot.com/2009/01/neodarwinismo-teoria-sinttica-da-evoluo.html2009>. Acesso em: 20/01/2022. Citado na página 35.

GOMES, M. C. Introdução à dinâmica populacional. **Secção de Genética e Dinâmica Populacional do Departamento de Biologia Vegetal da Faculdade de Ciências de Lisboa**, 2002. Citado na página 37.

HOEKSTRA, A. G.; KROC, J.; SLOOT, P. M. **Simulating Complex Systems by Cellular Automata**. Berlin: Springer, 2010. Citado na página 32.

KLEE, H.; ALLEN, R. **Simulation of Dynamic Systems, 2a. edição**. New York: CRC Press, 2011. Citado na página 32.

MELO, V. V. de. **Técnica de aumento de eficiência para metaheurística aplicadas a otimização global contínua e discreta**. Tese (Doutorado), São Carlos, 2009. Citado na página 33.

MONTEIRO, L. H. A. **Sistemas dinâmicos, 2a. edição**. São Paulo: Editora Livraria da Física, 2006. Citado nas páginas 31 e 32.

MORIWAKI, K.; INUZUKA, N.; YAMADA, M.; SEKI, H. I. H. Method for evolutionary agents in a competitive environment. **On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2)**, II, 1997. Citado nas páginas 28 e 43.

PINTO-COELHO, R. M. **Fundamentos em ecologia**. Porto Alegre: Artmed, 2007. Citado na página 37.

RIDLEY, M. **Evolução. 3ª edição**. São Paulo: Artmed, 2004. Citado na página 34.

ROSA, J.; SOUZA, M. T.; RECHA, L. O.; MAGNABOSCOA, L. Q.; LUNG, L. C. Co-evolution of antagonistic intelligent agents using genetic algorithms. **International Conference on Computational Science, ICCS**, 2013. Citado na página 28.

SANTANA, D. N.; TIDON, R.; SIMON, S. J. As bases fisicalistas do evolucionismo na teoria sisntética da evolução. UNB, Brasília, 2020. Citado na página 34.

SILVEIRA, T.; OLIVEIRA, H. C. B.; SILVA, L. E. Controle de inércia para fuga de mínimos locais de funções não-lineares na otimização por enxame de partículas. **Departamento de Ciências Exatas - Universidade Federal de Alfenas - MG - Brasil**, 2007. Citado na página 28.

TINóS, R.; YANG, S. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. FFCLRP - USP, Ribeirão Preto, 2007. Citado na página 40.

VILLATE, J. E. **Introdução aos sistemas dinâmicos: uma abordagem prática com Maxima**. Porto/PT: Universidade do Porto, 2007. Citado na página 31.

YANG, X.-S. **Nature-Inspired Optimization Algorithms**. London: Elsevier, 2014. Citado nas páginas 33 e 34.

APÊNDICE

A

CÓDIGO FONTE ECOSIM

Código-fonte 1 – Ecosim

```
1:
2: using System;
3: using System.Collections.Generic;
4: using System.Text;
5:
6: namespace ecosim_dotnetcore
7: {
8:     class Ecosim
9:     {
10:         static Simulator simulator;
11:         static void Main(string[] args)
12:         {
13:             simulator = new Simulator();
14:             for (int i = 0; i < 1; i++)
15:                 simulator.RunSimulation(true, 1, 3000);
16:         }
17:     }
18: }
```

Código-fonte 2 – Classe de configurações do Ecosim

```
1: using System;
2: using System.Collections.Generic;
3: using System.Text;
4:
5: namespace ecosim_dotnetcore
6: {
7:     class Congiguration
8:     {
9:         public Dictionary<string, dynamic> parameters;
10:
11:         public Congiguration(double x)
12:         {
13:             parameters = new Dictionary<string, dynamic>();
14:
15:             /* Engine config: */
16:             parameters.Add("ENGINE_FPS", 60);
```

```
17:             parameters.Add("ENGINE_WINDOW_X", 1600);
18:             parameters.Add("ENGINE_WINDOW_Y", 900);
19:             /* End engine config */
20:
21:             /* Main World config */
22:             parameters.Add("WORLD_MIN_COORD", -1.0);
23:             parameters.Add("WORLD_MAX_COORD", 1.0);
24:             /* End world config */
25:
26:             /* Instance config */
27:             parameters.Add("INSTANCE_INIT_AGENT_COUNT",90);
28:
29:             parameters.Add("INSTANCE_FOOD_SPAWN_FREQ",4);
30:             parameters.Add("INSTANCE_FOOD_SPAWN_INIT",50);
31:             parameters.Add("INSTANCE_FOOD_SPAWN_MIN",5);
32:             parameters.Add("INSTANCE_FOOD_SPAWN_MAX",7);
33:             parameters.Add("INSTANCE_FOOD_ENERGY",0.5);
34:             parameters.Add("INSTANCE_FOOD_METAB", 0.0001);
35:
36:             parameters.Add("INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MIN", 0.2);
37:             parameters.Add("INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MAX", 1.0);
38:
39:             parameters.Add("INSTANCE_REPRODUCTION_TIME_INTERVAL", 5.0);
40:
41:
42:             parameters.Add("INSTANCE_CONTROL_MUTATE_START",600.0);
43:             parameters.Add("INSTANCE_CONTROL_MUTATE_STOP",1200.0);
44:
45:             parameters.Add("INSTANCE_FOREIGNER_START", 1800.0);
46:             parameters.Add("INSTANCE_FOREIGNER_STOP", 2400.0);
47:             parameters.Add("INSTANCE_FOREIGNER_SPAWN_MIN",0);
48:             parameters.Add("INSTANCE_FOREIGNER_SPAWN_MAX",3);
49:             parameters.Add("INSTANCE_FOREIGNER_SPAWN_PROBABILITY",0.01);
50:             /* Instance config */
51:
52:             /* Agent general config */
53:             /* Agent transparency */
54:             //parameters.Add("AGENT_RGB_ALPHA",0.9);
55:             /* Agent vision field transparency */
56:             //parameters.Add("AGENT_VIS_ALPHA",0.2);
57:             /* Maximum agent velocity */
58:             parameters.Add("AGENT_MAX_VELOCITY",1.0);
59:             parameters.Add("AGENT_MIN_VELOCITY",-1.0);
60:             /* Default energy for new-spawned agents */
61:             parameters.Add("AGENT_ENERGY_DEFAULT",1.0);
62:             parameters.Add("AGENT_ENERGY_SEXUAL_REPRODUCTION", 0.25);
63:             parameters.Add("AGENT_ENERGY_ASEXUAL_REPRODUCTION", 0.5);
64:             /* The maximum speed any agents can move at */
65:             parameters.Add("AGENT_MAX_SPEED",0.0015);
66:             parameters.Add("AGENT_MATE_REACH_SPACE", 0.08);
67:             parameters.Add("AGENT_EAT_REACH_SPACE", 0.02);
68:             /* The energy level at which an agent dies */
69:             parameters.Add("AGENT_ENERGY_DEAD",0.3);
70:             /* How quickly ageing effects the agents */
71:             parameters.Add("AGENT_TIME_FACTOR",0.3);
72:
73:             parameters.Add("AGENT_DIET_BOUNDARY",0.5);
74:             parameters.Add("AGENT_SEXUED_BOUNDARY",0.0);
75:             parameters.Add("AGENT_DNA_SPECIES_RANGE", 0.5);
76:
```

```
77:             /* The amount a DNA trait changes if mutation occurs */
78:             parameters.Add("AGENT_DNA_MUTATE_PROBABILITY", 0.01);
79:             parameters.Add("AGENT_DNA_MAX_MUTATED_GENS", 2);
80:             parameters.Add("AGENT_DNA_MUTATE_RATE", 0.1);
81:
82:             /* End agent general config */
83:
84:             /* Agent DNA config */
85:             /* Metabolism trait max/min
86:              * How quickly an agent can move. Faster moving agents burn energy a lot
87:              * quicker */
88:             parameters.Add("AGENT_METAB_MAX",0.8);
89:             parameters.Add("AGENT_METAB_MIN",0.1);
90:             /* Vision trait max/min
91:              * How wide the agents field of view is */
92:             parameters.Add("AGENT_VISION_MAX",0.2);
93:             parameters.Add("AGENT_VISION_MIN",0.1);
94:             /* Rebirth trait max/min
95:              * How much energy is stored within an agent until it splits, creating a
96:              * possibly mutated clone of itself, halving it's energy */
97:             parameters.Add("AGENT_REBIRTH_MAX",3.00);
98:             parameters.Add("AGENT_REBIRTH_MIN",1.00);
99:             /* Diet trait max/min
100:             * If greater or equal to zero, the agent eats other agents, if less than
101:             * zero, the agent eats only dead agents */
102:            parameters.Add("AGENT_DIET_MIN",0.00);
103:            parameters.Add("AGENT_DIET_MAX",1.00);
104:            /* Diet trait max/min
105:             * If greater or equal to zero, the agent eats other agents, if less than
106:             * zero, the agent eats only dead agents */
107:            parameters.Add("AGENT_SEXUED_MIN",0.00);
108:            parameters.Add("AGENT_SEXUED_MAX",1.00);
109:            /* Flock max/min
110:             * How strong flocking behaviours influence the movement of an agent */
111:            parameters.Add("AGENT_FLOCK_MAX",0.50);
112:            parameters.Add("AGENT_FLOCK_MIN",0.00);
113:            /* Wobble max/min
114:             * How many times per second the agent "wobbles" This is a movement based
        on
115:             * a sin wav, which gives a temporary boost of speed, followed by a equal
116:             * period of slower movement */
117:            parameters.Add("AGENT_WOBBLE_MAX",3.0);
118:            parameters.Add("AGENT_WOBBLE_MIN",1.0);
119:            /* End agent DNA config */
120:
121:            /*Log config */
122:            parameters.Add("LOGGER_ENABLE",1);
123:            parameters.Add("LOGGER_FREQ",2);
124:            /*End log config */
125:
126:        }
127:
128:    }
129: }
```

Código-fonte 3 – Classe de controle do ecossistema

```
1: using System;
2: using System.Collections.Generic;
```

```
3: using System.Text;
4:
5: namespace ecosim_dotnetcore
6: {
7:     class Ecosystem
8:     {
9:         int ecosystemId;
10:        public Ecosystem(int paramId)
11:        {
12:            ecosystemId = paramId;
13:            Cache.ecossytemData[ecosystemId].agentNextId = 0;
14:            Cache.ecossytemData[ecosystemId].agents = new Dictionary<int, Agent>();
15:            Cache.ecossytemData[ecosystemId].speciestNextId = 0;
16:            Cache.ecossytemData[ecosystemId].species = new Dictionary<int, Species>()
    ;
17:
18:            Cache.ecossytemData[ecosystemId].clock = 0.0;
19:            Cache.ecossytemData[ecosystemId].congiguration = new Congiguration((
    double)ecosystemId);
20:
21:            Cache.ecossytemData[ecosystemId].lastLogTime = 0.0;
22:            Cache.ecossytemData[ecosystemId].metrics = new Metrics(ecosystemId);
23:            Cache.ecossytemData[ecosystemId].logger = new Logger(ecosystemId);
24:
25:            Cache.ecossytemData[ecosystemId].logger.LogConfiguration(Cache.
    ecossytemData[ecosystemId].congiguration);
26:
27:            Cache.ecossytemData[ecosystemId].lastUpdateTime = 0.0;
28:            IncludeAgents(GetConfiguration("INSTANCE_INIT_AGENT_COUNT"), Agent.
    AgentStates.LIVING, "START_LIVING", GetConfiguration("AGENT_ENERGY_DEFAULT"));
29:            Cache.ecossytemData[ecosystemId].lastFoodTime = 0.0;
30:            IncludeAgents(GetConfiguration("INSTANCE_FOOD_SPAWN_INIT"), Agent.
    AgentStates.DEAD, "START_DEAD", GetConfiguration("INSTANCE_FOOD_ENERGY"));
31:            IncludeBetaAgents();
32:        }
33:        dynamic GetConfiguration(string configurationParam)
34:        {
35:            return Cache.ecossytemData[ecosystemId].congiguration.parameters[
    configurationParam];
36:        }
37:        void IncludeBetaAgents()
38:        {
39:            //agents.Add(agentNextId, new Agent(Agent.AgentStates.AGENT_STATE_LIVING,
     "BETA", agentNextId, true, Agent.AgentDiet.AGENT_DIET_LIVING));
40:            //agentNextId++;
41:            //agents.Add(agentNextId, new Agent(Agent.AgentStates.AGENT_STATE_LIVING,
     "BETA", agentNextId, true, Agent.AgentDiet.AGENT_DIET_DEAD));
42:            //agentNextId++;
43:            //agents.Add(agentNextId, new Agent(Agent.AgentStates.AGENT_STATE_DEAD, "
    BETA", agentNextId, true, Agent.AgentDiet.AGENT_DIET_SELF));
44:            //agentNextId++;
45:        }
46:        void IncludeAgents(int amount, Agent.AgentStates state, string origin, double
     originEnergy)
47:        {
48:            Agent temporaryAgent;
49:            for (int i = 0; i < amount; i++)
50:            {
51:                temporaryAgent = new Agent(ecosystemId, state, origin, originEnergy);
52:                Cache.ecossytemData[ecosystemId].agents.Add(temporaryAgent.id,
```

```
temporaryAgent);
53:             }
54:         }
55:         public void RunInteration(double time)
56:         {
57:             Console.WriteLine(time);
58:             Cache.ecossytemData[ecosystemId].metrics.Clear();
59:             Cache.ecossytemData[ecosystemId].pruneList = new List<int>();
60:             Cache.ecossytemData[ecosystemId].lastUpdateTime = time;
61:             UpdateAgents();
62:             FoodDrop();
63:             InsertForeigner();
64:             Cache.ecossytemData[ecosystemId].metrics.GenerateLog(time);
65:         }
66:         public void UpdateAgents()
67:         {
68:             Cache.ecossytemData[ecosystemId].clock += 0.05;
69:             Agent temporaryAgent;
70:             List<Agent> childAgents = new List<Agent>();
71:             foreach (KeyValuePair<int, Agent> currentAgent in Cache.ecossytemData[
ecosystemId].agents)
72:             {
73:                 if (currentAgent.Value.state == Agent.AgentStates.LIVING)
74:                 {
75:                     temporaryAgent = currentAgent.Value.Update();
76:                     if (temporaryAgent != null)
77:                         childAgents.Add(temporaryAgent);
78:                 }
79:                 else
80:                 {
81:                     currentAgent.Value.UpdateEnergy();
82:                     if (!currentAgent.Value.HasEnergy())
83:                     {
84:                         currentAgent.Value.Prune(currentAgent.Key);
85:                         Cache.ecossytemData[ecosystemId].pruneList.Add(currentAgent.
Key);
86:                     }
87:                 }
88:             }
89:             for (int i = 0; i < childAgents.Count; i++)
90:                 Cache.ecossytemData[ecosystemId].agents.Add(childAgents[i].id,
childAgents[i]);
91:             for (int i = 0; i < Cache.ecossytemData[ecosystemId].pruneList.Count; i
++)
92:                 Cache.ecossytemData[ecosystemId].agents.Remove(Cache.ecossytemData[
ecosystemId].pruneList[i]);
93:         }
94:         public void FoodDrop()
95:         {
96:             int foodAmount;
97:             if (Cache.ecossytemData[ecosystemId].lastUpdateTime > Cache.ecossytemData
[ecosystemId].lastFoodTime + GetConfiguration("INSTANCE_FOOD_SPAWN_FREQ"))
98:             {
99:                 foodAmount = (int)Cache.RANDF_MIN(GetConfiguration( "
INSTANCE_FOOD_SPAWN_MIN"), GetConfiguration("INSTANCE_FOOD_SPAWN_MAX"));
100:                IncludeAgents(foodAmount, Agent.AgentStates.DEAD, "FOOD_DROP", 0.0);
101:                Cache.ecossytemData[ecosystemId].lastFoodTime = Cache.ecossytemData[
ecosystemId].lastUpdateTime;
102:            }
103:        }
```

```
104:         public void InsertForeigner()
105:         {
106:             int foreignerAmount;
107:
108:             if (Cache.ecossytemData[ecosystemId].lastUpdateTime > GetConfiguration("
        INSTANCE_FOREIGNER_START") && Cache.ecossytemData[ecosystemId].lastUpdateTime <
        GetConfiguration("INSTANCE_FOREIGNER_STOP") && GetConfiguration("
        INSTANCE_FOREIGNER_SPAWN_PROBABILITY") > Cache.RANDF(1))
109:             {
110:                 foreignerAmount = (int)Cache.RANDF_MIN(GetConfiguration( "
        INSTANCE_FOREIGNER_SPAWN_MIN"), GetConfiguration( "INSTANCE_FOREIGNER_SPAWN_MAX")
        );
111:                 IncludeAgents(foreignerAmount, Agent.AgentStates.LIVING, "FOREIGNER",
         0.0);
112:             }
113:         }
114:     }
115: }
```

Código-fonte 4 – Classe dos agentes

```
1:
2: using System;
3: using System.Collections.Generic;
4: using System.Text;
5:
6: namespace ecosim_dotnetcore
7: {
8:     class Agent
9:     {
10:         public enum AgentDiet
11:         {
12:             NONE = -1,
13:             MEAT = 0,
14:             PLANT = 1,
15:             SELF = 2
16:         }
17:         public enum AgentReprpoduction
18:         {
19:             ASSEXUED = 0,
20:             SEXUED = 1
21:         }
22:         public enum AgentStates
23:         {
24:             PRUNE = -1,
25:             DEAD = 0,
26:             LIVING = 1
27:         }
28:
29:         public enum AgentAttraction
30:         {
31:             NONE = -1,
32:             FLOCK = 0,
33:             SEEK = 1,
34:             AVOID = 2
35:         }
36:
37:         public struct AgentDNA
38:         {
```

```
39:             public AgentDiet diet;
40:             public AgentReprpoduction reprpoduction;
41:             public double metabolism;
42:             public double vision;
43:             public double rebirth;
44:             public double flock;
45:             public double wobble;
46:         }
47:         public struct AgentColor
48:         {
49:             public double r;
50:             public double g;
51:             public double b;
52:         }
53:
54:         public struct VisionQuad
55:         {
56:             public SpaceCoordinates upBoyndary;
57:             public SpaceCoordinates lowBoyndary;
58:         }
59:
60:         public AgentDNA DNA;
61:         public AgentStates state;
62:         public double energy;
63:         public AgentColor color;
64:         public SpaceCoordinates position;
65:         public SpaceCoordinates velocity;
66:         public VisionQuad visionQuad;
67:
68:         public int id;
69:         public string origin;
70:         public bool betaAgente;
71:
72:         public int interactions;
73:         public List<int>[] nearAgents;
74:
75:         public int analised;
76:         public int inVision;
77:
78:         public double birthDate;
79:         public int idSpecies;
80:         public double lastParenting;
81:         //public double deathDate;
82:
83:         int ecosystemId;
84:
85:         public Agent(int paramEcossytemID, AgentStates stateParam, string originParam
    , double originEnergy, int parentID = -1, AgentDNA paranDNA = new AgentDNA(),
    bool beta = false, AgentDiet betaType = AgentDiet.NONE)
86:         {
87:             ecosystemId = paramEcossytemID;
88:             betaAgente = beta;
89:             id = NextAgentId(true);
90:             birthDate = Cache.ecossytemData[ecosystemId].lastUpdateTime;
91:             lastParenting = Cache.ecossytemData[ecosystemId].lastUpdateTime;
92:             state = stateParam;
93:             origin = originParam;
94:             idSpecies = -1;
95:
96:             nearAgents = new List<int>[3];
```

```
97:             interactions = 0;
98:
99:             if (stateParam == AgentStates.LIVING)
100:            {
101:                energy = GetConfiguration("AGENT_ENERGY_DEFAULT");
102:                if (parentID != -1)
103:                {
104:                    SpaceCoordinates positionIncremet = new SpaceCoordinates(0.03);
105:                    DNA = paranDNA;
106:                    Mutate(GetConfiguration("AGENT_DNA_MUTATE_PROBABILITY"));
107:                    if (Cache.RANDF(1) > 0.1)
108:                    {
109:                        idSpecies = GetAgent(parentID).idSpecies;
110:                        Cache.ecossytemData[ecosystemId].species[idSpecies].
    IncludeIndividual();
111:                    }
112:                    else
113:                    {
114:                        SetupSpecies();
115:                    }
116:                    position = new SpaceCoordinates(GetAgent(parentID).position);
117:                    if (Cache.RANDF(1) < 0.5)
118:                        positionIncremet.x *= -1;
119:                    if (Cache.RANDF(1) < 0.5)
120:                        positionIncremet.y *= -1;
121:                    position.AddCoordinates(positionIncremet);
122:                    velocity = new SpaceCoordinates(GetAgent(parentID).velocity);
123:                    SetupVisionQuad();
124:                    Cache.ecossytemData[ecosystemId].metrics.RegisterEvent(Cache.
    ecossytemData[ecosystemId].lastUpdateTime, GetAgent(parentID), this, originParam)
    ;
125:                }
126:                else
127:                {
128:                    DNA = RadomDNA();
129:                    if (beta)
130:                        DNA.diet = betaType;
131:                    position = new SpaceCoordinates(Cache.RANDF_MIN(GetConfiguration(
    "WORLD_MIN_COORD"), GetConfiguration("WORLD_MAX_COORD")), Cache.RANDF_MIN(
    GetConfiguration("WORLD_MIN_COORD"), GetConfiguration("WORLD_MAX_COORD")));
132:                    velocity = new SpaceCoordinates(Cache.RANDF_MIN(GetConfiguration(
    "AGENT_MIN_VELOCITY"), GetConfiguration("AGENT_MAX_VELOCITY")), Cache.RANDF_MIN(
    GetConfiguration("AGENT_MIN_VELOCITY"), GetConfiguration("AGENT_MAX_VELOCITY")));
133:                    SetupVisionQuad();
134:                    SetupSpecies();
135:                    Cache.ecossytemData[ecosystemId].metrics.RegisterEvent(Cache.
    ecossytemData[ecosystemId].lastUpdateTime, this, this, originParam);
136:                }
137:                for (int i = 0; i < nearAgents.Length; i++)
138:                    nearAgents[i] = new List<int>();
139:            }
140:            else
141:            {
142:                //state = AgentStates.DEAD;
143:                energy = GetConfiguration("INSTANCE_FOOD_ENERGY");
144:                DNA = new AgentDNA();
145:                DNA.diet = AgentDiet.SELF;
146:                DNA.metabolism = GetConfiguration("INSTANCE_FOOD_METAB");
147:                position = new SpaceCoordinates(Cache.RANDF_MIN(GetConfiguration("
    WORLD_MIN_COORD"), GetConfiguration("WORLD_MAX_COORD")), Cache.RANDF_MIN(
```

```
        GetConfiguration("WORLD_MIN_COORD"), GetConfiguration("WORLD_MAX_COORD")));
148:                velocity = new SpaceCoordinates();
149:            }
150:            Cache.ecossytemData[ecosystemId].metrics.AddEnergyAdded(energy -
        originEnergy);
151:            SetupColors();
152:        }
153:        Agent GetAgent(int agentID)
154:        {
155:            return Cache.ecossytemData[ecosystemId].agents[agentID];
156:        }
157:        int NextAgentId()
158:        {
159:            return Cache.ecossytemData[ecosystemId].agentNextId;
160:        }
161:        int NextAgentId(bool add)
162:        {
163:            int returnID = Cache.ecossytemData[ecosystemId].agentNextId;
164:            if (add)
165:                Cache.ecossytemData[ecosystemId].agentNextId++;
166:            return returnID;
167:        }
168:        dynamic GetConfiguration(string configurationParam)
169:        {
170:            return Cache.ecossytemData[ecosystemId].congiguration.parameters[
        configurationParam];
171:        }
172:        double GetClock()
173:        {
174:            return Cache.ecossytemData[ecosystemId].clock;
175:        }
176:        AgentDNA RadomDNA()
177:        {
178:            AgentDNA tempDNA = new AgentDNA();
179:            double tempRand;
180:            tempDNA.metabolism = Cache.RANDF_MIN(GetConfiguration("AGENT_METAB_MIN"),
         GetConfiguration("AGENT_METAB_MAX"));
181:            tempDNA.vision = Cache.RANDF_MIN(GetConfiguration("AGENT_VISION_MIN"),
        GetConfiguration("AGENT_VISION_MAX"));
182:            tempDNA.rebirth = Cache.RANDF_MIN(GetConfiguration("AGENT_REBIRTH_MIN"),
        GetConfiguration("AGENT_REBIRTH_MAX"));
183:            tempDNA.flock = Cache.RANDF_MIN(GetConfiguration("AGENT_FLOCK_MIN"),
        GetConfiguration("AGENT_FLOCK_MAX"));
184:            tempDNA.wobble = Cache.RANDF_MIN(GetConfiguration("AGENT_WOBBLE_MIN"),
        GetConfiguration("AGENT_WOBBLE_MAX"));
185:            tempRand = Cache.RANDF_MIN(GetConfiguration("AGENT_DIET_MIN"),
        GetConfiguration("AGENT_DIET_MAX"));
186:            if (tempRand < GetConfiguration("AGENT_DIET_BOUNDARY"))
187:                tempDNA.diet = AgentDiet.MEAT;
188:            else
189:                tempDNA.diet = AgentDiet.PLANT;
190:
191:            tempRand = Cache.RANDF_MIN(GetConfiguration("AGENT_SEXUED_MIN"),
        GetConfiguration("AGENT_SEXUED_MAX"));
192:            if (tempRand < GetConfiguration("AGENT_SEXUED_BOUNDARY"))
193:                tempDNA.reprpoduction = AgentReprpoduction.ASSEXUED;
194:            else
195:                tempDNA.reprpoduction = AgentReprpoduction.SEXUED;
196:            return tempDNA;
197:        }
```

```
198:         public void SetupSpecies()
199:         {
200:             foreach (KeyValuePair<int, Species> specieWithKey in Cache.ecossytemData[
    ecosystemId].species)
201:             {
202:                 if (idSpecies == -1 && specieWithKey.Value.SameSpecies(DNA))
203:                 {
204:                     idSpecies = specieWithKey.Key;
205:                     Cache.ecossytemData[ecosystemId].species[idSpecies].
    IncludeIndividual();
206:                 }
207:             }
208:             if (idSpecies == -1)
209:             {
210:                 Cache.ecossytemData[ecosystemId].species.Add(Cache.ecossytemData[
    ecosystemId].speciestNextId, new Species(DNA, ecosystemId));
211:                 idSpecies = Cache.ecossytemData[ecosystemId].speciestNextId;
212:                 Cache.ecossytemData[ecosystemId].speciestNextId++;
213:             }
214:         }
215:         public void SetupColors()
216:         {
217:             if (DNA.diet == AgentDiet.SELF)
218:             {
219:                 color.r = 0.2;
220:                 color.g = 0.2;
221:                 color.b = 0.2;
222:             }
223:             else if (origin == "FOREIGNER")
224:             {
225:                 if (DNA.diet == AgentDiet.MEAT)
226:                 {
227:                     color.r = 1;
228:                     color.g = 0.7;
229:                 }
230:                 else if (DNA.diet == AgentDiet.PLANT)
231:                 {
232:                     color.r = 0.7;
233:                     color.g = 1;
234:                 }
235:                 color.b = 0.7;
236:             }
237:             else
238:             {
239:                 if (DNA.diet == AgentDiet.MEAT)
240:                 {
241:                     color.r = 1;
242:                     color.g = 0;
243:                 }
244:                 else if (DNA.diet == AgentDiet.PLANT)
245:                 {
246:                     color.r = 0;
247:                     color.g = 1;
248:                 }
249:                 if (betaAgente)
250:                     color.b = 1;
251:                 else
252:                     color.b = idSpecies / (double)Cache.ecossytemData[ecosystemId].
    species.Count;
253:             }
```

```
254:         }
255:         void UpdateColors()
256:         {
257:             color.b = idSpecies / (double)Cache.ecossytemData[ecosystemId].species.
    Count;
258:         }
259:         void SetupVisionQuad()
260:         {
261:             double halfRad = DNA.vision * 0.5;
262:             visionQuad.upBoyndary = new SpaceCoordinates(position);
263:             visionQuad.upBoyndary.AddCoordinates(halfRad);
264:             visionQuad.lowBoyndary = new SpaceCoordinates(position);
265:             visionQuad.lowBoyndary.SubtractCoordinates(halfRad);
266:         }
267:         public AgentAttraction GetAttraction(AgentDiet targetDiet, int
    targetIdSpecies)
268:         {
269:             if (targetDiet == AgentDiet.NONE) return AgentAttraction.NONE;
270:
271:             /* meat eater vs other meat eater */
272:             else if (DNA.diet == AgentDiet.MEAT && targetDiet == AgentDiet.MEAT &&
    idSpecies == targetIdSpecies)
273:                 return AgentAttraction.FLOCK;
274:
275:             /* meat eater vs any living */
276:             else if (DNA.diet == AgentDiet.MEAT && targetDiet == AgentDiet.PLANT)
277:                 return AgentAttraction.SEEK;
278:
279:             /* meat eater vs dead */
280:             else if (DNA.diet == AgentDiet.MEAT && targetDiet == AgentDiet.SELF)
281:                 return AgentAttraction.NONE;
282:
283:             /* plant eater vs dead */
284:             else if (DNA.diet == AgentDiet.PLANT && targetDiet == AgentDiet.SELF)
285:                 return AgentAttraction.SEEK;
286:
287:             /* plant eater vs platn eater */
288:             else if (DNA.diet == AgentDiet.PLANT && targetDiet == AgentDiet.PLANT &&
    idSpecies == targetIdSpecies)
289:                 return AgentAttraction.FLOCK;
290:
291:             /* plant eater vs meat eater*/
292:             else if (DNA.diet == AgentDiet.PLANT && targetDiet == AgentDiet.MEAT)
293:                 return AgentAttraction.AVOID;
294:
295:             /* plant eater vs living */
296:             else if (DNA.diet == AgentDiet.PLANT)
297:                 return AgentAttraction.NONE;
298:
299:             return AgentAttraction.NONE;
300:         }
301:         public Agent Update()
302:         {
303:             Agent childAgents = null;
304:             FindNearAgents();
305:             MoveFlock();
306:             MoveSeekOrAvoid();
307:             Collision();
308:             UpdateLocation(GetClock());
309:             UpdateEnergy();
```

```
310:             UpdateColors();
311:             if (!IsAlive())
312:             {
313:                 Die();
314:             }
315:             if (CanReproduct())
316:             {
317:                 if (DNA.reprpoduction == Agent.AgentReprpoduction.SEXUED)
318:                 {
319:                     childAgents = Mate();
320:                 }
321:                 else
322:                 {
323:                     childAgents = Reproduct();
324:                 }
325:             }
326:             return childAgents;
327:         }
328:         public void FindNearAgents()
329:         {
330:             Agent.AgentAttraction attraction;
331:             inVision = 0;
332:             analised = 0;
333:             for (int i = 0; i < nearAgents.Length; i++)
334:                 nearAgents[i] = new List<int>();
335:             foreach (KeyValuePair<int, Agent> currentAgent in Cache.ecossytemData[
    ecosystemId].agents)
336:             {
337:                 if (id != currentAgent.Key)
338:                     analised++;
339:                 if (CanSee(currentAgent.Value))
340:                 {
341:                     inVision++;
342:                     attraction = GetAttraction(currentAgent.Value.DNA.diet,
    currentAgent.Value.idSpecies);
343:                     if (attraction != Agent.AgentAttraction.NONE)
344:                         nearAgents[(int)attraction].Add(currentAgent.Key);
345:                 }
346:
347:             }
348:         }
349:         public void MoveFlock()
350:         {
351:             SpaceCoordinates finalVelocity = new SpaceCoordinates();
352:             SpaceCoordinates alignVelocity = FlockAlign();
353:             SpaceCoordinates cohesionVelocity = FlockCohesion();
354:             SpaceCoordinates seperationVelocity = FlockSeperation();
355:             /* Velocity alignment */
356:             finalVelocity.AddCoordinates(alignVelocity);
357:             /* Cohesion: Go to center of mass */
358:             finalVelocity.AddCoordinates(cohesionVelocity);
359:             /* Seperation: Avoid otherss */
360:             finalVelocity.AddCoordinates(seperationVelocity);
361:
362:             finalVelocity.MultiplyCoordinates(DNA.flock);
363:
364:             velocity.AddCoordinates(finalVelocity);
365:
366:             velocity.Normalize();
367:         }
```

```
368:         public SpaceCoordinates FlockAlign()
369:         {
370:             SpaceCoordinates returnVelocity;
371:             int nearAgentsCount = nearAgents[(int)Agent.AgentAttraction.FLOCK].Count;
372:             returnVelocity = new SpaceCoordinates();
373:
374:             if (nearAgentsCount == 0) return returnVelocity;
375:
376:             for (int i = 0; i < nearAgentsCount; i++)
377:             {
378:                 returnVelocity.AddCoordinates(GetAgent(nearAgents[(int)Agent.
    AgentAttraction.FLOCK][i]).velocity);
379:             }
380:
381:             returnVelocity.DevideCoordinates(nearAgentsCount);
382:             returnVelocity.Normalize();
383:
384:             return returnVelocity;
385:         }
386:         public SpaceCoordinates FlockCohesion()
387:         {
388:             SpaceCoordinates returnVelocity;
389:             int nearAgentsCount = nearAgents[(int)Agent.AgentAttraction.FLOCK].Count;
390:             returnVelocity = new SpaceCoordinates();
391:
392:             if (nearAgentsCount == 0) return returnVelocity;
393:
394:             for (int i = 0; i < nearAgentsCount; i++)
395:             {
396:                 returnVelocity.AddCoordinates(GetAgent(nearAgents[(int)Agent.
    AgentAttraction.FLOCK][i]).position);
397:             }
398:
399:             returnVelocity.DevideCoordinates(nearAgentsCount);
400:             returnVelocity.SubtractCoordinates(position);
401:             returnVelocity.Normalize();
402:
403:             return returnVelocity;
404:         }
405:         public SpaceCoordinates FlockSeperation()
406:         {
407:             SpaceCoordinates returnVelocity;
408:             int nearAgentsCount = nearAgents[(int)Agent.AgentAttraction.FLOCK].Count;
409:             returnVelocity = new SpaceCoordinates();
410:
411:             if (nearAgentsCount == 0) return returnVelocity;
412:
413:             for (int i = 0; i < nearAgentsCount; i++)
414:             {
415:                 returnVelocity.AddCoordinates(GetAgent(nearAgents[(int)Agent.
    AgentAttraction.FLOCK][i]).position);
416:                 returnVelocity.SubtractCoordinates(position);
417:             }
418:
419:             returnVelocity.DevideCoordinates(-1 * nearAgentsCount);
420:             returnVelocity.Normalize();
421:
422:             return returnVelocity;
423:         }
424:         public void MoveSeekOrAvoid()
```

```
425:         {
426:             SpaceCoordinates newVelocity = new SpaceCoordinates(position);
427:             for (int i = 0; i < nearAgents[(int)Agent.AgentAttraction.SEEK].Count; i
    ++)
428:             {
429:                 newVelocity.SubtractCoordinates(GetAgent(nearAgents[(int)Agent.
    AgentAttraction.SEEK][i]).position);
430:                 newVelocity.Normalize();
431:                 velocity.SubtractCoordinates(newVelocity);
432:             }
433:             newVelocity = new SpaceCoordinates(position);
434:             for (int i = 0; i < nearAgents[(int)Agent.AgentAttraction.AVOID].Count; i
    ++)
435:             {
436:                 newVelocity.SubtractCoordinates(GetAgent(nearAgents[(int)Agent.
    AgentAttraction.AVOID][i]).position);
437:                 newVelocity.Normalize();
438:                 velocity.AddCoordinates(newVelocity);
439:             }
440:             velocity.Normalize();
441:         }
442:         public void Collision()
443:         {
444:             double reachSpace = GetConfiguration("AGENT_EAT_REACH_SPACE"); ;
445:             double minPressure = GetConfiguration("
    INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MIN");
446:             double maxPressure = GetConfiguration("
    INSTANCE_PREDATOR_PREY_STRENGTH_RATIO_MAX");
447:             for (int i = 0; i < nearAgents[(int)Agent.AgentAttraction.SEEK].Count; i
    ++)
448:             {
449:                 if (!GetAgent(nearAgents[(int)Agent.AgentAttraction.SEEK][i]).
    betaAgente)
450:                 {
451:                     if (((position.x - reachSpace < GetAgent(nearAgents[(int)Agent.
    AgentAttraction.SEEK][i]).position.x) &
452:                     (position.x + reachSpace > GetAgent(nearAgents[(int)Agent.
    AgentAttraction.SEEK][i]).position.x)) &&
453:                     ((position.y - reachSpace < GetAgent(nearAgents[(int)Agent.
    AgentAttraction.SEEK][i]).position.y) &
454:                     (position.y + reachSpace > GetAgent(nearAgents[(int)Agent.
    AgentAttraction.SEEK][i]).position.y)))
455:                     {
456:                         if(Cache.RANDF_MIN(minPressure, maxPressure) < ((energy / 3)
    + DNA.metabolism ) / ((GetAgent(nearAgents[(int)Agent.AgentAttraction.SEEK][i]).
    energy / 3) + GetAgent(nearAgents[(int)Agent.AgentAttraction.SEEK][i]).DNA.
    metabolism)) {
457:                             energy += GetAgent(nearAgents[(int)Agent.AgentAttraction.
    SEEK][i]).energy;
458:                             Cache.ecossytemData[ecosystemId].metrics.AddEnergyFlow(
    GetAgent(nearAgents[(int)Agent.AgentAttraction.SEEK][i]).energy);
459:                             interactions++;
460:                             GetAgent(nearAgents[(int)Agent.AgentAttraction.SEEK][i]).
    Prune(id);
461:                             Cache.ecossytemData[ecosystemId].pruneList.Add(nearAgents
    [(int)Agent.AgentAttraction.SEEK][i]);
462:                         }
463:                     }
464:                 }
465:             }
```

```
466:         }
467:         public void UpdateLocation(double clock)
468:         {
469:             double maxSpeed = AgentMaxSpeed();
470:             double wobbleFactor = (2.0 + Math.Sin(DNA.wobble + DNA.wobble * clock)) *
     0.5;
471:             SpaceCoordinates wobble = new SpaceCoordinates(velocity);
472:             wobble.MultiplyCoordinates(maxSpeed * Math.Abs(wobbleFactor));
473:             position.AddCoordinates(wobble);
474:             WarpLocation();
475:             SetupVisionQuad();
476:         }
477:         public void UpdateEnergy()
478:         {
479:             energy -= AGENT_METAB_ENERGY_SCALE(DNA.metabolism) * GetConfiguration("
    AGENT_TIME_FACTOR");
480:         }
481:         public bool IsAlive()
482:         {
483:             return betaAgente || (energy >= GetConfiguration("AGENT_ENERGY_DEAD"));
484:         }
485:         public bool HasEnergy()
486:         {
487:             return betaAgente || (energy >= 0);
488:         }
489:         public double AgentMaxSpeed()
490:         {
491:             return DNA.metabolism * GetConfiguration("AGENT_MAX_SPEED");
492:         }
493:         public void WarpLocation()
494:         {
495:             if (position.x < GetConfiguration("WORLD_MIN_COORD"))
496:                 position.x = GetConfiguration("WORLD_MAX_COORD");
497:             else if (position.x > GetConfiguration("WORLD_MAX_COORD"))
498:                 position.x = GetConfiguration("WORLD_MIN_COORD");
499:
500:             if (position.y < GetConfiguration("WORLD_MIN_COORD"))
501:                 position.y = GetConfiguration("WORLD_MAX_COORD");
502:             else if (position.y > GetConfiguration("WORLD_MAX_COORD"))
503:                 position.y = GetConfiguration("WORLD_MIN_COORD");
504:         }
505:         public void Die()
506:         {
507:             Cache.ecossytemData[ecosystemId].metrics.RegisterEvent(Cache.
    ecossytemData[ecosystemId].lastUpdateTime, this, this, "DIE");
508:             state = AgentStates.DEAD;
509:             DNA.diet = AgentDiet.SELF;
510:             DNA.metabolism = GetConfiguration("INSTANCE_FOOD_METAB");
511:             velocity.Set(0.0);
512:             color.r = 0.2;
513:             color.g = 0.2;
514:             color.b = 0.2;
515:             Cache.ecossytemData[ecosystemId].species[idSpecies].RemoveIndividual();
516:         }
517:         public void Prune(int eaterId)
518:         {
519:             Cache.ecossytemData[ecosystemId].metrics.RegisterEvent(Cache.
    ecossytemData[ecosystemId].lastUpdateTime, GetAgent(eaterId), this, "EAT");
520:             if (state == AgentStates.LIVING)
521:                 Cache.ecossytemData[ecosystemId].species[idSpecies].RemoveIndividual
```

```
();
522:            state = AgentStates.PRUNE;
523:            DNA.diet = AgentDiet.NONE;
524:        }
525:        public bool CanReproduct()
526:        {
527:            return ((energy > DNA.rebirth) && (Cache.ecossytemData[ecosystemId].
lastUpdateTime > lastParenting + GetConfiguration("
INSTANCE_REPRODUCTION_TIME_INTERVAL"))) ;
528:        }
529:        public Agent Reproduct(int agentID = 0)
530:        {
531:            double transferedEnergy;
532:            interactions++;
533:            if(DNA.reprpoduction == AgentReprpoduction.SEXUED)
534:             {
535:                transferedEnergy = energy * GetConfiguration("
AGENT_ENERGY_SEXUAL_REPRODUCTION");
536:                energy *= (1 -  GetConfiguration("AGENT_ENERGY_SEXUAL_REPRODUCTION"))
;
537:                lastParenting = Cache.ecossytemData[ecosystemId].lastUpdateTime;
538:                transferedEnergy += GetAgent(agentID).energy * GetConfiguration("
AGENT_ENERGY_SEXUAL_REPRODUCTION");
539:                GetAgent(agentID).energy *=  (1 -  GetConfiguration("
AGENT_ENERGY_SEXUAL_REPRODUCTION"));
540:                GetAgent(agentID).lastParenting = Cache.ecossytemData[ecosystemId].
lastUpdateTime;
541:                Cache.ecossytemData[ecosystemId].metrics.AddEnergyFlow(
transferedEnergy);
542:                return new Agent(ecosystemId, AgentStates.LIVING, "REPRODUCT",
transferedEnergy, id, MixDNA(GetAgent(agentID).DNA));
543:            }
544:            else
545:            {
546:                transferedEnergy = energy * GetConfiguration("
AGENT_ENERGY_ASEXUAL_REPRODUCTION");
547:                energy *= (1 - GetConfiguration("AGENT_ENERGY_ASEXUAL_REPRODUCTION"))
;
548:                lastParenting = Cache.ecossytemData[ecosystemId].lastUpdateTime;
549:                Cache.ecossytemData[ecosystemId].metrics.AddEnergyFlow(
transferedEnergy);
550:                return new Agent(ecosystemId, AgentStates.LIVING, "SPLIT",
transferedEnergy, id, DNA);
551:            }
552:        }
553:        public Agent Mate()
554:        {
555:            Agent childAgents = null;
556:            double reachSpace = GetConfiguration("AGENT_MATE_REACH_SPACE");
557:            for (int i = 0; i < nearAgents[(int)Agent.AgentAttraction.FLOCK].Count; i
++)
558:            {
559:                if (GetAgent(nearAgents[(int)Agent.AgentAttraction.FLOCK][i]).
CanReproduct() &&
560:                    (((position.x - reachSpace < GetAgent(nearAgents[(int)Agent.
AgentAttraction.FLOCK][i]).position.x) &
561:                    (position.x + reachSpace > GetAgent(nearAgents[(int)Agent.
AgentAttraction.FLOCK][i]).position.x)) &&
562:                    ((position.y - reachSpace < GetAgent(nearAgents[(int)Agent.
AgentAttraction.FLOCK][i]).position.y) &
```

```
563:                     (position.y + reachSpace > GetAgent(nearAgents[(int)Agent.
    AgentAttraction.FLOCK][i]).position.y))))
564:                 {
565:                     childAgents = Reproduct(nearAgents[(int)Agent.AgentAttraction.
    FLOCK][i]);
566:                     continue;
567:                 }
568:             }
569:             bool varStop = false;
570:             if (childAgents == null)
571:                 varStop = true;
572:             return childAgents;
573:         }
574:         public AgentDNA MixDNA(AgentDNA otherDNA)
575:         {
576:             AgentDNA tempDNA = new AgentDNA();
577:             tempDNA = new AgentDNA();
578:             tempDNA.diet = DNA.diet;
579:             tempDNA.reprpoduction = DNA.reprpoduction;
580:             if (Cache.RANDF(1) > 0.5)
581:                 tempDNA.metabolism = DNA.metabolism;
582:             else
583:                 tempDNA.metabolism = otherDNA.metabolism;
584:
585:             if (Cache.RANDF(1) > 0.5)
586:                 tempDNA.vision = DNA.vision;
587:             else
588:                 tempDNA.vision = otherDNA.vision;
589:
590:             if (Cache.RANDF(1) > 0.5)
591:                 tempDNA.rebirth = DNA.rebirth;
592:             else
593:                 tempDNA.rebirth = otherDNA.rebirth;
594:
595:             if (Cache.RANDF(1) > 0.5)
596:                 tempDNA.flock = DNA.flock;
597:             else
598:                 tempDNA.flock = otherDNA.flock;
599:
600:             if (Cache.RANDF(1) > 0.5)
601:                 tempDNA.wobble = DNA.wobble;
602:             else
603:                 tempDNA.wobble = otherDNA.wobble;
604:             return tempDNA;
605:         }
606:         public void Mutate(double probability)
607:         {
608:             if(birthDate > GetConfiguration("INSTANCE_CONTROL_MUTATE_START") &&
    birthDate < GetConfiguration("INSTANCE_CONTROL_MUTATE_STOP"))
609:             {
610:                 int mutatedGens = 0;
611:                 if (Cache.RANDF(1) < probability && mutatedGens <= GetConfiguration("
    AGENT_DNA_MAX_MUTATED_GENS"))
612:                 {
613:                     DNA.metabolism = Cache.RANDF_MIN(GetConfiguration("
    AGENT_METAB_MIN"), GetConfiguration("AGENT_METAB_MAX"));
614:                     mutatedGens++;
615:                 }
616:                 if (Cache.RANDF(1) < probability && mutatedGens <= GetConfiguration("
    AGENT_DNA_MAX_MUTATED_GENS"))
```

```
617:                 {
618:                     DNA.vision = Cache.RANDF_MIN(GetConfiguration("AGENT_VISION_MIN")
    , GetConfiguration("AGENT_VISION_MAX"));
619:                     mutatedGens++;
620:                 }
621:                 if (Cache.RANDF(1) < probability && mutatedGens <= GetConfiguration("
    AGENT_DNA_MAX_MUTATED_GENS"))
622:                 {
623:                     DNA.rebirth = Cache.RANDF_MIN(GetConfiguration("AGENT_REBIRTH_MIN
    "), GetConfiguration("AGENT_REBIRTH_MAX"));
624:                     mutatedGens++;
625:                 }
626:                 if (Cache.RANDF(1) < probability && mutatedGens <= GetConfiguration("
    AGENT_DNA_MAX_MUTATED_GENS"))
627:                 {
628:                     DNA.flock = Cache.RANDF_MIN(GetConfiguration("AGENT_FLOCK_MIN"),
    GetConfiguration("AGENT_FLOCK_MAX"));
629:                     mutatedGens++;
630:                 }
631:                 if (Cache.RANDF(1) < probability && mutatedGens <= GetConfiguration("
    AGENT_DNA_MAX_MUTATED_GENS"))
632:                 {
633:                     DNA.wobble = Cache.RANDF_MIN(GetConfiguration("AGENT_WOBBLE_MIN")
    , GetConfiguration("AGENT_WOBBLE_MAX"));
634:                     mutatedGens++;
635:                 }
636:                 //if (Cache.RANDF(1) <= probability && mutatedGens <=
    GetConfiguration("AGENT_DNA_MAX_MUTATED_GENS"))
637:                 //{
638:                 //    if (Cache.RANDF_MIN(GetConfiguration("AGENT_DIET_MIN"),
    GetConfiguration("AGENT_DIET_MAX")) < GetConfiguration("AGENT_DIET_BOUNDARY"))
639:                 //        DNA.diet = AgentDiet.MEAT;
640:                 //    else
641:                 //        DNA.diet = AgentDiet.PLANT;
642:                 //    mutatedGens++;
643:                 //}
644:                 //if (Cache.RANDF(1) <= probability && mutatedGens <=
    GetConfiguration("AGENT_DNA_MAX_MUTATED_GENS"))
645:                 //{
646:                 //    if (Cache.RANDF_MIN(GetConfiguration("AGENT_SEXUED_MIN"),
    GetConfiguration("AGENT_SEXUED_MAX")) < GetConfiguration("AGENT_SEXUED_BOUNDARY")
    )
647:                 //        DNA.reprpoduction = AgentReprpoduction.ASSEXUED;
648:                 //    else
649:                 //        DNA.reprpoduction = AgentReprpoduction.SEXUED;
650:                 //    mutatedGens++;
651:                 //}
652:             }
653:         }
654:         public void RangeMutate(double probability)
655:         {
656:             double temporaryGen;
657:             temporaryGen = DNA.metabolism + (Cache.RANDF(1) > probability ? Cache.
    RANDF_MIN(-1.0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0);
658:             if (temporaryGen > GetConfiguration("AGENT_METAB_MAX"))
659:                 DNA.metabolism = GetConfiguration("AGENT_METAB_MAX");
660:             else if (temporaryGen < GetConfiguration("AGENT_METAB_MIN"))
661:                 DNA.metabolism = GetConfiguration("AGENT_METAB_MIN");
662:             else
663:                 DNA.metabolism = temporaryGen;
```

```
664:
665:            temporaryGen = DNA.vision + (Cache.RANDF(1) > probability ? Cache.
        RANDF_MIN(-1.0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0);
666:            if (temporaryGen > GetConfiguration("AGENT_VISION_MAX"))
667:                DNA.vision = GetConfiguration("AGENT_VISION_MAX");
668:            else if (temporaryGen < GetConfiguration("AGENT_VISION_MIN"))
669:                DNA.vision = GetConfiguration("AGENT_VISION_MIN");
670:            else
671:                DNA.vision = temporaryGen;
672:
673:            temporaryGen = DNA.rebirth + (Cache.RANDF(1) > probability ? Cache.
        RANDF_MIN(-1.0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0);
674:            if (temporaryGen > GetConfiguration("AGENT_REBIRTH_MAX"))
675:                DNA.rebirth = GetConfiguration("AGENT_REBIRTH_MAX");
676:            else if (temporaryGen < GetConfiguration("AGENT_REBIRTH_MIN"))
677:                DNA.rebirth = GetConfiguration("AGENT_REBIRTH_MIN");
678:            else
679:                DNA.rebirth = temporaryGen;
680:
681:            temporaryGen = DNA.flock + (Cache.RANDF(1) > probability ? Cache.
        RANDF_MIN(-1.0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0);
682:            if (temporaryGen > GetConfiguration("AGENT_FLOCK_MAX"))
683:                DNA.flock = GetConfiguration("AGENT_FLOCK_MAX");
684:            else if (temporaryGen < GetConfiguration("AGENT_FLOCK_MIN"))
685:                DNA.flock = GetConfiguration("AGENT_FLOCK_MIN");
686:            else
687:                DNA.flock = temporaryGen;
688:
689:            temporaryGen = DNA.wobble + (Cache.RANDF(1) > probability ? Cache.
        RANDF_MIN(-1.0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0);
690:            if (temporaryGen > GetConfiguration("AGENT_WOBBLE_MAX"))
691:                DNA.wobble = GetConfiguration("AGENT_WOBBLE_MAX");
692:            else if (temporaryGen < GetConfiguration("AGENT_WOBBLE_MIN"))
693:                DNA.wobble = GetConfiguration("AGENT_WOBBLE_MIN");
694:            else
695:                DNA.wobble = temporaryGen;
696:
697:            temporaryGen = Math.Abs((double)DNA.diet - (Cache.RANDF(1) > probability
        ? Cache.RANDF_MIN(0, 1.0) * GetConfiguration("AGENT_DNA_MUTATE_RATE") : 0));
698:            if (temporaryGen < GetConfiguration("AGENT_DIET_BOUNDARY"))
699:                DNA.diet = AgentDiet.MEAT;
700:            else
701:                DNA.diet = AgentDiet.PLANT;
702:
703:        }
704:        public bool CanSee(Agent targetAgent)
705:        {
706:            return ((visionQuad.upBoyndary.x >= targetAgent.position.x) &
707:                (visionQuad.upBoyndary.y >= targetAgent.position.y)) &&
708:                ((visionQuad.lowBoyndary.x <= targetAgent.position.x) &
709:                (visionQuad.lowBoyndary.y <= targetAgent.position.y));
710:        }
711:        public List<float> Vertex()
712:        {
713:            List<float>  agentVertex = new List<float>();
714:            agentVertex.Add((float)position.x);
715:            agentVertex.Add((float)position.y);
716:            agentVertex.Add((float)energy);
717:            agentVertex.Add(0.5f);
718:            agentVertex.Add((float)color.r);
```

```
719:                agentVertex.Add((float)color.g);
720:                agentVertex.Add((float)color.b);
721:                agentVertex.Add(energy < 1.0 ? (float)energy : 1.0f);
722:                return agentVertex;
723:            }
724:            /* The rate energy is burned over time, with respect to metabolism rates = (
        where x is agents metabolism */
725:            public double AGENT_METAB_ENERGY_SCALE(double x) { return (0.001 * x); }
726:            /* How large agents are, with respect to their energy = (Where x is energy);
        */
727:            public double AGENT_ENERGY_SIZE_SCALE(double x) { return ((10 * x) + 4); }
728:
729:        }
730: }
```

Código-fonte 5 – Classe de apuração das métricas

```
1: using System;
2: using System.Collections.Generic;
3: using System.Text;
4:
5: namespace ecosim_dotnetcore
6: {
7:     class Metrics
8:     {
9:         enum BasicMetricsOrder
10:        {
11:            //GENERAL
12:            LOG_TIME,
13:            LOG_POPULATION,
14:            LOG_FOOD,
15:            LOG_HERBIVOUR,
16:            LOG_CARNIVOUR,
17:            LOG_HERB_ENERGY_AVG,
18:            LOG_CARN_ENERGY_AVG,
19:            LOG_TOTAL_ENERGY,
20:            LOG_ENERGY_FLOW,
21:            LOG_ENERGY_ADDED,
22:            //HERB
23:            LOG_METABOLISM_AVG_H,
24:            LOG_VISION_AVG_H,
25:            LOG_REBIRTH_AVG_H,
26:            LOG_DIET_AVG_H,
27:            LOG_FLOCK_AVG_H,
28:            LOG_WOBBLE_AVG_H,
29:            LOG_LIFE_TIME_MAX_H,
30:            LOG_LIFE_TIME_AVG_H,
31:
32:            LOG_METABOLISM_VAR_H,
33:            LOG_VISION_VAR_H,
34:            LOG_REBIRTH_VAR_H,
35:            LOG_DIET_VAR_H,
36:            LOG_FLOCK_VAR_H,
37:            LOG_WOBBLE_VAR_H,
38:
39:            //CARNI
40:            LOG_METABOLISM_AVG_C,
41:            LOG_VISION_AVG_C,
42:            LOG_REBIRTH_AVG_C,
```

```
43:             LOG_DIET_AVG_C,
44:             LOG_FLOCK_AVG_C,
45:             LOG_WOBBLE_AVG_C,
46:             LOG_LIFE_TIME_MAX_C,
47:             LOG_LIFE_TIME_AVG_C,
48:
49:             LOG_METABOLISM_VAR_C,
50:             LOG_VISION_VAR_C,
51:             LOG_REBIRTH_VAR_C,
52:             LOG_DIET_VAR_C,
53:             LOG_FLOCK_VAR_C,
54:             LOG_WOBBLE_VAR_C
55:
56:         };
57:
58:         enum BasicMetricsAnalyticsOrder
59:         {
60:             LOG_METABOLISM,
61:             LOG_VISION,
62:             LOG_REBIRTH,
63:             LOG_DIET,
64:             LOG_FLOCK,
65:             LOG_WOBBLE
66:         };
67:
68:         public int ecosystemId;
69:
70:         public const int BASIC_METRICS_QUANT = 38;
71:         public const int GENETIC_METRICS_QUANT = 8;
72:         public const int ANALYTICS_METRICS_QUANT = 6;
73:         public const int HERB_METRICS_START = 10;
74:         public const int CARN_METRICS_START = 24;
75:
76:         public double[] basicMetrics;
77:         public string[] basicMetricsLable;
78:
79:         public List<double>[] herbMetricsAnalytics;
80:         public List<double>[] carnMetricsAnalytics;
81:         public StringBuilder registerBook;
82:         public StringBuilder speciesLog;
83:
84:         public double energyFlow;
85:         public double energyAdded;
86:
87:         List<float> agentVertex;
88:         public Metrics(int paramEcosystemId)
89:         {
90:             ecosystemId = paramEcosystemId;
91:             energyFlow = 0.0;
92:             energyAdded = 0.0;
93:             basicMetrics = new double[BASIC_METRICS_QUANT];
94:             herbMetricsAnalytics = new List<double>[ANALYTICS_METRICS_QUANT];
95:             carnMetricsAnalytics = new List<double>[ANALYTICS_METRICS_QUANT];
96:             basicMetricsLable = new string[] { "x_time", "y_pop", "y_food", "y_herb",
      "y_meat", "y_energ_h_a", "y_energ_c_a", "y_total_energ", "y_energ_flow", "
    y_energ_added", "y_metab_a_h", "y_vision_a_h", "y_rebirth_a_h", "y_diet_a_h", "
    y_flock_a_h", "y_wobble_a_h", "y_life_time_m_h", "y_life_time_a_h", "y_metab_v_h"
    , "y_vision_v_h", "y_rebirth_v_h", "y_diet_v_h", "y_flock_v_h", "y_wobble_v_h", "
    y_metab_a_c", "y_vision_a_c", "y_rebirth_a_c", "y_diet_a_c", "y_flock_a_c", "
    y_wobble_a_c", "y_life_time_m_c", "y_life_time_a_c", "y_metab_v_c", "y_vision_v_c
```

```
", "y_rebirth_v_c", "y_diet_v_c", "y_flock_v_c", "y_wobble_v_c" };
97:             Clear();
98:         }
99:         dynamic GetConfiguration(string configurationParam)
100:        {
101:            return Cache.ecossytemData[ecosystemId].congiguration.parameters[
configurationParam];
102:        }
103:        public void Clear()
104:        {
105:            agentVertex = new List<float>();
106:            for (int i = 0; i < basicMetrics.Length; i++)
107:                basicMetrics[i] = 0.0;
108:            for (int i = 0; i < ANALYTICS_METRICS_QUANT; i++)
109:            {
110:                herbMetricsAnalytics[i] = new List<double>();
111:                carnMetricsAnalytics[i] = new List<double>();
112:            }
113:            registerBook = new StringBuilder();
114:            speciesLog = new StringBuilder();
115:        }
116:        public float[] GetVertex()
117:        {
118:            return agentVertex.ToArray();
119:        }
120:        public void GenerateLog(double time)
121:        {
122:            bool log = (Cache.ecossytemData[ecosystemId].lastLogTime +
GetConfiguration("LOGGER_FREQ")) < time;
123:            foreach (KeyValuePair<int, Agent> currentAgent in Cache.ecossytemData[
ecosystemId].agents)
124:            {
125:                agentVertex.AddRange(currentAgent.Value.Vertex());
126:                if (log)
127:                    AddToMetrics(currentAgent.Value, time);
128:            }
129:            if (log)
130:            {
131:                Summarize(time);
132:                //Cache.ecossytemData[ecosystemId].logger.LogMetrics(Cache.
ecossytemData[ecosystemId].metrics);
133:                Cache.ecossytemData[ecosystemId].logger.LogMetricsCSV(Cache.
ecossytemData[ecosystemId].metrics);
134:                //LogSpecies(Cache.ecossytemData[ecosystemId].species, time);
135:                Cache.ecossytemData[ecosystemId].logger.LogSpecies(Cache.
ecossytemData[ecosystemId].metrics.speciesLog);
136:                Cache.ecossytemData[ecosystemId].lastLogTime = time;
137:            }
138:            //Cache.ecossytemData[ecosystemId].logger.LogRegisterBook(Cache.
ecossytemData[ecosystemId].metrics.registerBook);
139:
140:        }
141:        public void AddEnergyFlow(double energy)
142:        {
143:            energyFlow += energy;
144:        }
145:        public void AddEnergyAdded(double energy)
146:        {
147:            energyAdded += energy;
148:        }
```

```
149:         public void AddToMetrics(Agent agent, double time)
150:         {
151:             if (agent.state == Agent.AgentStates.LIVING)
152:             {
153:                 basicMetrics[(int)BasicMetricsOrder.LOG_POPULATION] += 1;
154:                 basicMetrics[(int)BasicMetricsOrder.LOG_TOTAL_ENERGY] += agent.energy
    ;
155: 
156:                 if (agent.DNA.diet == Agent.AgentDiet.MEAT)
157:                 {
158:                     basicMetrics[(int)BasicMetricsOrder.LOG_CARNIVOUR] += 1;
159:                     basicMetrics[(int)BasicMetricsOrder.LOG_CARN_ENERGY_AVG] += agent
    .energy;
160:                     basicMetrics[(int)BasicMetricsOrder.LOG_METABOLISM_AVG_C] +=
    agent.DNA.metabolism * 100;
161:                     basicMetrics[(int)BasicMetricsOrder.LOG_VISION_AVG_C] += agent.
    DNA.vision * 100;
162:                     basicMetrics[(int)BasicMetricsOrder.LOG_REBIRTH_AVG_C] += agent.
    DNA.rebirth * 100;
163:                     basicMetrics[(int)BasicMetricsOrder.LOG_DIET_AVG_C] += (double)
    agent.DNA.diet * 100;
164:                     basicMetrics[(int)BasicMetricsOrder.LOG_FLOCK_AVG_C] += agent.DNA
    .flock * 100;
165:                     basicMetrics[(int)BasicMetricsOrder.LOG_WOBBLE_AVG_C] += agent.
    DNA.wobble * 100;
166:                     if (basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_MAX_C] < (
    time - agent.birthDate))
167:                         basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_MAX_C] = (
    time - agent.birthDate);
168:                     basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_AVG_C] += (time
     - agent.birthDate);
169:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.
    LOG_METABOLISM].Add(agent.DNA.metabolism * 100);
170:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_VISION].
    Add(agent.DNA.vision * 100);
171:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_REBIRTH
    ].Add(agent.DNA.rebirth * 100);
172:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_DIET].
    Add((double)agent.DNA.diet * 100);
173:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_FLOCK].
    Add(agent.DNA.flock * 100);
174:                     carnMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_WOBBLE].
    Add(agent.DNA.wobble * 100);
175:                 }
176:                 else
177:                 {
178:                     basicMetrics[(int)BasicMetricsOrder.LOG_HERBIVOUR] += 1;
179:                     basicMetrics[(int)BasicMetricsOrder.LOG_HERB_ENERGY_AVG] += agent
    .energy;
180:                     basicMetrics[(int)BasicMetricsOrder.LOG_METABOLISM_AVG_H] +=
    agent.DNA.metabolism * 100;
181:                     basicMetrics[(int)BasicMetricsOrder.LOG_VISION_AVG_H] += agent.
    DNA.vision * 100;
182:                     basicMetrics[(int)BasicMetricsOrder.LOG_REBIRTH_AVG_H] += agent.
    DNA.rebirth * 100;
183:                     basicMetrics[(int)BasicMetricsOrder.LOG_DIET_AVG_H] += (double)
    agent.DNA.diet * 100;
184:                     basicMetrics[(int)BasicMetricsOrder.LOG_FLOCK_AVG_H] += agent.DNA
    .flock * 100;
185:                     basicMetrics[(int)BasicMetricsOrder.LOG_WOBBLE_AVG_H] += agent.
```

```
        DNA.wobble * 100;
186:                    if (basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_MAX_H] < (
        time - agent.birthDate))
187:                        basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_MAX_H] = (
        time - agent.birthDate);
188:                    basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_AVG_H] += (time
         - agent.birthDate);
189:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.
        LOG_METABOLISM].Add(agent.DNA.metabolism * 100);
190:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_VISION].
        Add(agent.DNA.vision * 100);
191:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_REBIRTH
        ].Add(agent.DNA.rebirth * 100);
192:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_DIET].
        Add((double)agent.DNA.diet * 100);
193:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_FLOCK].
        Add(agent.DNA.flock * 100);
194:                    herbMetricsAnalytics[(int)BasicMetricsAnalyticsOrder.LOG_WOBBLE].
        Add(agent.DNA.wobble * 100);
195:                }
196:            }
197:            else if (agent.state == Agent.AgentStates.DEAD)
198:            {
199:                basicMetrics[(int)BasicMetricsOrder.LOG_FOOD] += 1;
200:            }
201:        }
202:        public void Summarize(double time)
203:        {
204:            basicMetrics[(int)BasicMetricsOrder.LOG_TIME] = time;
205:            basicMetrics[(int)BasicMetricsOrder.LOG_METABOLISM_AVG_H] /= basicMetrics
        [(int)BasicMetricsOrder.LOG_HERBIVOUR];
206:            basicMetrics[(int)BasicMetricsOrder.LOG_VISION_AVG_H] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_HERBIVOUR];
207:            basicMetrics[(int)BasicMetricsOrder.LOG_REBIRTH_AVG_H] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_HERBIVOUR];
208:            basicMetrics[(int)BasicMetricsOrder.LOG_DIET_AVG_H] /= basicMetrics[(int)
        BasicMetricsOrder.LOG_HERBIVOUR];
209:            basicMetrics[(int)BasicMetricsOrder.LOG_FLOCK_AVG_H] /= basicMetrics[(int
        )BasicMetricsOrder.LOG_HERBIVOUR];
210:            basicMetrics[(int)BasicMetricsOrder.LOG_WOBBLE_AVG_H] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_HERBIVOUR];
211:            basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_AVG_H] /= basicMetrics
        [(int)BasicMetricsOrder.LOG_HERBIVOUR];
212:
213:            basicMetrics[(int)BasicMetricsOrder.LOG_METABOLISM_AVG_C] /= basicMetrics
        [(int)BasicMetricsOrder.LOG_CARNIVOUR];
214:            basicMetrics[(int)BasicMetricsOrder.LOG_VISION_AVG_C] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_CARNIVOUR];
215:            basicMetrics[(int)BasicMetricsOrder.LOG_REBIRTH_AVG_C] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_CARNIVOUR];
216:            basicMetrics[(int)BasicMetricsOrder.LOG_DIET_AVG_C] /= basicMetrics[(int)
        BasicMetricsOrder.LOG_CARNIVOUR];
217:            basicMetrics[(int)BasicMetricsOrder.LOG_FLOCK_AVG_C] /= basicMetrics[(int
        )BasicMetricsOrder.LOG_CARNIVOUR];
218:            basicMetrics[(int)BasicMetricsOrder.LOG_WOBBLE_AVG_C] /= basicMetrics[(
        int)BasicMetricsOrder.LOG_CARNIVOUR];
219:            basicMetrics[(int)BasicMetricsOrder.LOG_LIFE_TIME_AVG_C] /= basicMetrics
        [(int)BasicMetricsOrder.LOG_CARNIVOUR];
220:
221:            //basicMetrics[(int)BasicMetricsOrder.LOG_HERB_ENERGY_AVG] /=
```

```
        basicMetrics[(int)BasicMetricsOrder.LOG_HERBIVOUR];
222:            //basicMetrics[(int)BasicMetricsOrder.LOG_CARN_ENERGY_AVG] /=
        basicMetrics[(int)BasicMetricsOrder.LOG_CARNIVOUR];
223:            basicMetrics[(int)BasicMetricsOrder.LOG_ENERGY_FLOW] = energyFlow;
224:            basicMetrics[(int)BasicMetricsOrder.LOG_ENERGY_ADDED] = energyAdded;
225:
226:            for (int i = 0; i < ANALYTICS_METRICS_QUANT; i++)
227:            {
228:                for(int j = 0; j < herbMetricsAnalytics[i].Count; j++)
229:                    basicMetrics[i + HERB_METRICS_START + GENETIC_METRICS_QUANT] +=
        Math.Pow(herbMetricsAnalytics[i][j] - basicMetrics[i + HERB_METRICS_START], 2);
230:                basicMetrics[i + HERB_METRICS_START + GENETIC_METRICS_QUANT] /=
        herbMetricsAnalytics[i].Count;
231:
232:                for (int j = 0; j < carnMetricsAnalytics[i].Count; j++)
233:                    basicMetrics[i + CARN_METRICS_START + GENETIC_METRICS_QUANT] +=
        Math.Pow(carnMetricsAnalytics[i][j] - basicMetrics[i + CARN_METRICS_START], 2);
234:                basicMetrics[i + CARN_METRICS_START + GENETIC_METRICS_QUANT] /=
        carnMetricsAnalytics[i].Count;
235:            }
236:            for (int i = 0; i < basicMetrics.Length; i++)
237:            {
238:                if (double.IsNaN(basicMetrics[i]))
239:                    basicMetrics[i] = 0;
240:                else
241:                    basicMetrics[i] = Math.Round(basicMetrics[i]);
242:            }
243:            energyFlow = 0.0;
244:            energyAdded = 0.0;
245:        }
246:        public void RegisterEvent(double time, Agent actAgent, Agent targetAgent,
        string eventType)
247:        {
248:            //registerBook.Append(time);
249:            //registerBook.Append(";");
250:            //registerBook.Append(eventType);
251:            //registerBook.Append(";");
252:            //registerBook.Append(actAgent.id);
253:            //registerBook.Append(";");
254:            //registerBook.Append(actAgent.birthDate);
255:            //registerBook.Append(";");
256:            //registerBook.Append(actAgent.idSpecies);
257:            //registerBook.Append(";");
258:            //registerBook.Append(actAgent.DNA.diet);
259:            //registerBook.Append(";");
260:            //registerBook.Append(actAgent.state);
261:            //registerBook.Append(";");
262:            //registerBook.Append(actAgent.origin);
263:            //registerBook.Append(";");
264:            //registerBook.Append(actAgent.position.x);
265:            //registerBook.Append(";");
266:            //registerBook.Append(actAgent.position.y);
267:            //registerBook.Append(";");
268:            //registerBook.Append(targetAgent.id);
269:            //registerBook.Append(";");
270:            //registerBook.Append(targetAgent.birthDate);
271:            //registerBook.Append(";");
272:            //registerBook.Append(targetAgent.idSpecies);
273:            //registerBook.Append(";");
274:            //registerBook.Append(targetAgent.DNA.diet);
```

```
275:                 //registerBook.Append(";");
276:                 //registerBook.Append(targetAgent.state);
277:                 //registerBook.Append(";");
278:                 //registerBook.Append(targetAgent.origin);
279:                 //registerBook.Append(";");
280:                 //registerBook.Append(targetAgent.position.x);
281:                 //registerBook.Append(";");
282:                 //registerBook.Append(targetAgent.position.y);
283:                 //registerBook.AppendLine();
284:             }
285:             public void LogSpecies(Dictionary<int, Species> species, double time)
286:             {
287:                 foreach (KeyValuePair<int, Species> specieWithKey in species)
288:                 {
289:                     speciesLog.Append(time);
290:                     speciesLog.Append(";");
291:                     speciesLog.Append(specieWithKey.Value.referenceDNA.diet);
292:                     speciesLog.Append(";");
293:                     speciesLog.Append(specieWithKey.Key);
294:                     speciesLog.Append(";");
295:                     speciesLog.Append(specieWithKey.Value.individuals);
296:                     speciesLog.AppendLine();
297:                 }
298:             }
299:         }
300: }
```

APÊNDICE

B

CÓDIGO PLOT DAS MÉTRICAS

Código-fonte 6 – Plot das métricas de população

```
1:
2: #!/usr/bin/env python3
3: import matplotlib
4: import matplotlib.pyplot as plt
5: import numpy as np
6: import sys
7: import importlib
8: import logger_data
9: sys.tracebacklimit = 0
10:
11: # Try to import the datafile ecosim generated
12: importlib.reload(logger_data)
13: try:
14:     from logger_data import *
15: except ImportError:
16:     print("No data, please ensure LOGGER_ENABLED is set to (1)!")
17:     exit()
18:
19: # Grab int values for population
20: x_time = np.asarray(x_time)
21: y_pop = np.asarray(y_pop)
22: y_food = np.asarray(y_food)
23: y_herb = np.asarray(y_herb)
24: y_meat = np.asarray(y_meat)
25:
26: f = plt.figure(figsize=(16,9))
27:
28: # Plot population
29: ax1 = plt.subplot(111)
30: #plt.fill_between((600,1200), 0, 250, facecolor='orange', alpha = 0.1)
31: #plt.fill_between((1800,2400), 0, 250, facecolor='blue', alpha = 0.1)
32: plt.plot(x_time, y_pop, label='Living agents')
33: plt.plot(x_time, y_food, label='Food')
34: plt.plot(x_time, y_herb, label='Herbivours')
35: plt.plot(x_time, y_meat, label='Carnivours')
36: plt.title("Ecosim Population")
37: plt.ylabel("Population")
38: plt.legend(loc='upper right')
```

```
39:
40: plt.grid()
41:
42: plt.savefig('pop_fx.png')
```

Código-fonte 7 – Plot das métricas genéticas

```
1:
2: #!/usr/bin/env python3
3: import matplotlib
4: import matplotlib.pyplot as plt
5: import numpy as np
6: import sys
7: import importlib
8: import logger_data
9: sys.tracebacklimit = 0
10:
11: # Try to import the datafile ecosim generated
12: importlib.reload(logger_data)
13: try:
14:     from logger_data import *
15: except ImportError:
16:     print("No data, please ensure LOGGER_ENABLED is set to (1)!")
17:     exit()
18:
19:
20: # Grab values for traits, scale from int to float
21: y_metab_h = np.sqrt(np.asarray(y_metab_v_h)) / 50
22: y_vision_h = np.sqrt(np.asarray(y_vision_v_h)) / 10
23: y_rebirth_h = np.sqrt(np.asarray(y_rebirth_v_h)) / 100
24: y_diet_h = np.sqrt(np.asarray(y_diet_v_h)) / 1
25: y_flock_h = np.sqrt(np.asarray(y_flock_v_h)) / 50
26: y_wobble_h = np.sqrt(np.asarray(y_wobble_v_h)) / 100
27:
28: y_metab_c = np.sqrt(np.asarray(y_metab_v_c)) / 50
29: y_vision_c = np.sqrt(np.asarray(y_vision_v_c)) / 10
30: y_rebirth_c = np.sqrt(np.asarray(y_rebirth_v_c)) / 100
31: y_diet_c = np.sqrt(np.asarray(y_diet_v_c)) / 1
32: y_flock_c = np.sqrt(np.asarray(y_flock_v_c)) / 50
33: y_wobble_c = np.sqrt(np.asarray(y_wobble_v_c)) / 100
34:
35: f = plt.figure(figsize=(16,9))
36:
37: # Plot traits
38: ax2 = plt.subplot(211)
39: #plt.fill_between((600,1200), 0, 1, facecolor='orange', alpha = 0.1)
40: #plt.fill_between((1800,2400), 0, 1, facecolor='blue', alpha = 0.1)
41: plt.plot(x_time, y_metab_h, label='Metabolism')
42: plt.plot(x_time, y_vision_h, label='Vision')
43: plt.plot(x_time, y_rebirth_h, label='Rebirth')
44: plt.plot(x_time, y_diet_h, label='Diet')
45: plt.plot(x_time, y_flock_h, label='Flocking')
46: plt.plot(x_time, y_wobble_h, label='Wobble')
47: plt.title("Herbivours Genetics Variance")
48: plt.ylabel("Variance Trait Value")
49: plt.legend(loc='upper right')
50:
51: plt.grid()
52:
```

```
53: # Plot traits
54: ax3 = plt.subplot(212)
55: #plt.fill_between((600,1200), 0, 1, facecolor='orange', alpha = 0.1)
56: #plt.fill_between((1800,2400), 0, 1, facecolor='blue', alpha = 0.1)
57: plt.plot(x_time, y_metab_c, label='Metabolism')
58: plt.plot(x_time, y_vision_c, label='Vision')
59: plt.plot(x_time, y_rebirth_c, label='Rebirth')
60: plt.plot(x_time, y_diet_c, label='Diet')
61: plt.plot(x_time, y_flock_c, label='Flocking')
62: plt.plot(x_time, y_wobble_c, label='Wobble')
63: plt.title("Carnivours Genetics Variance")
64: plt.xlabel("Time (s)")
65: plt.ylabel("Variance Trait Value")
66: plt.legend(loc='upper right')
67:
68: plt.grid()
69:
70: plt.savefig('gen_fx.png')
```

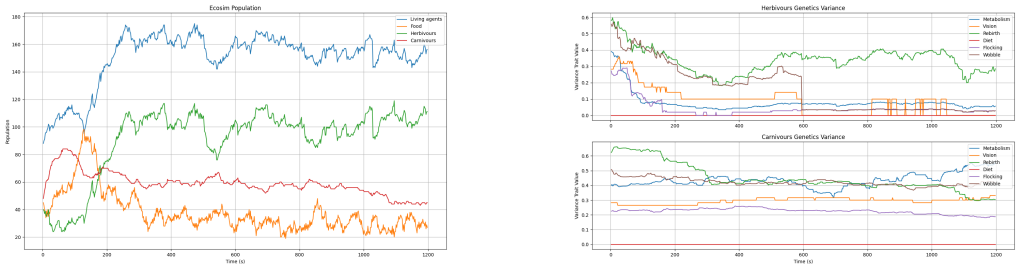
APÊNDICE

C

AMOSTRAS DE DADOS

C.1 Dados de controle

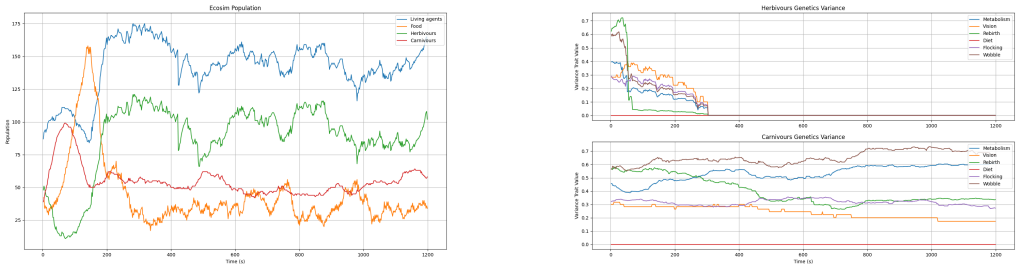
Figura 27 – Simulação Controle



(a) População (b) Genética

Fonte: Elaborada pelo autor.

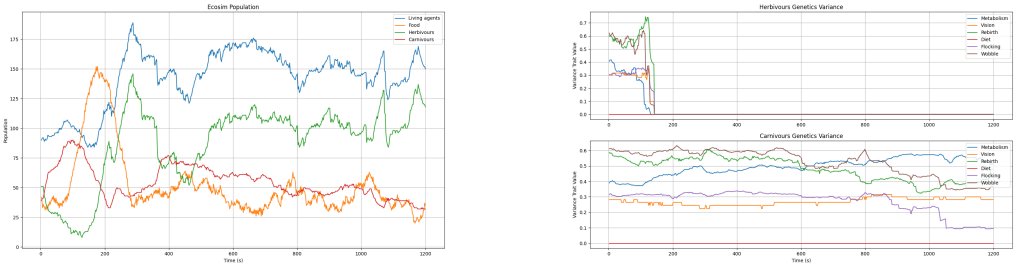
Figura 28 – Simulação Controle



(a) População (b) Genética

Fonte: Elaborada pelo autor.

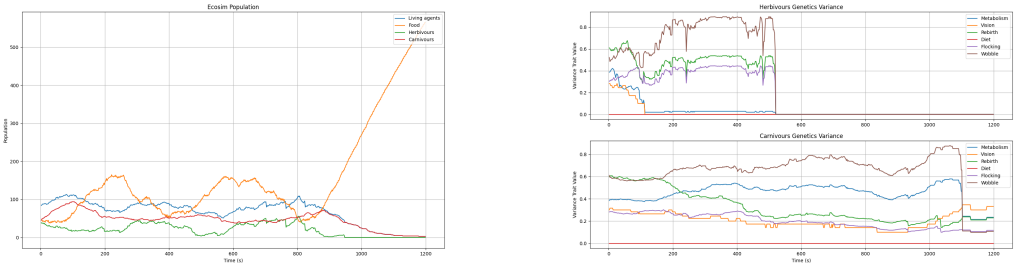
Figura 29 – Simulação Controle



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 30 – Simulação Controle

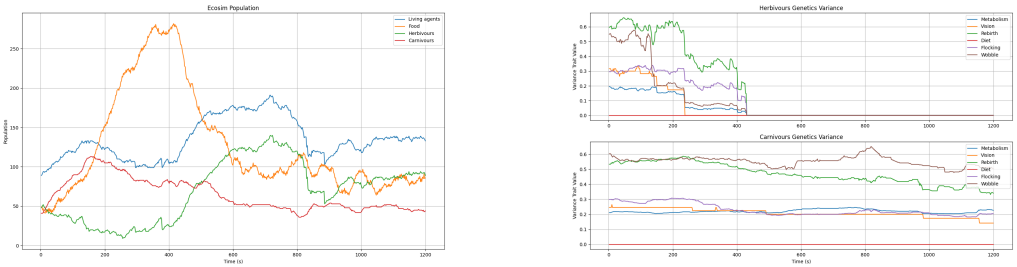


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.2 Experimentos metabolismo

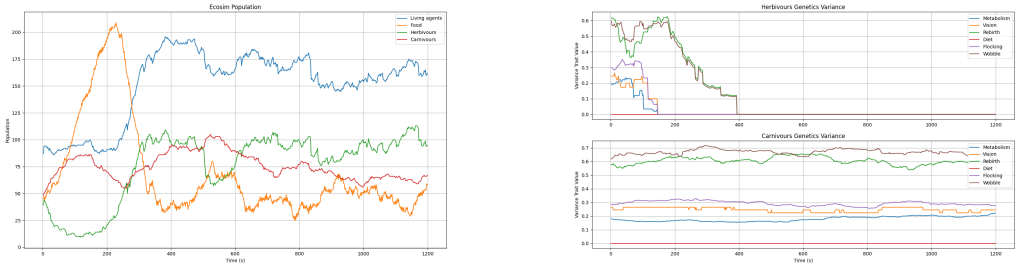
Figura 31 – Simulação Metabolismo - min=0,05 max=0,4



(a) População (b) Genética

Fonte: Elaborada pelo autor.

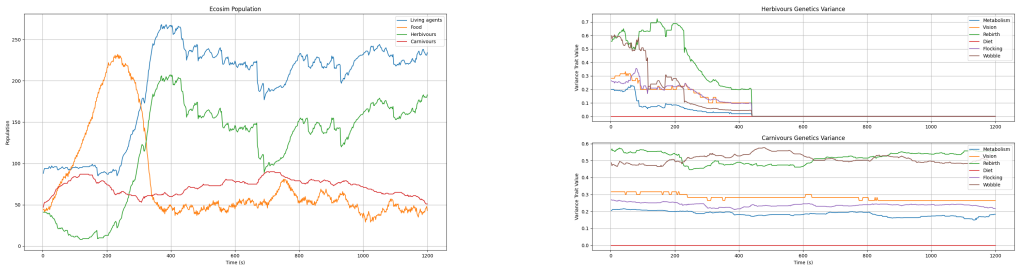
Figura 32 – Simulação Metabolismo - min=0,05 max=0,4



(a) População (b) Genética

Fonte: Elaborada pelo autor.

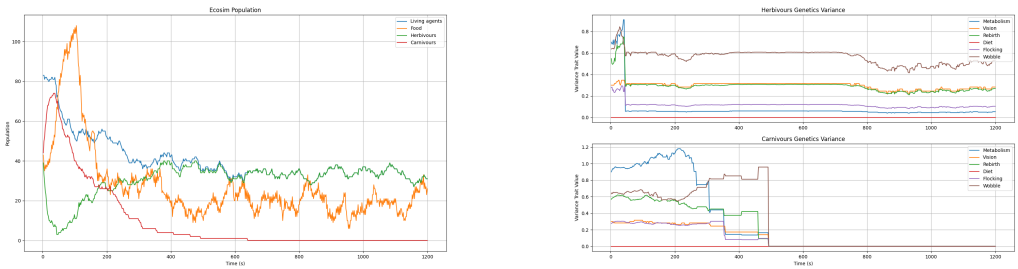
Figura 33 – Simulação Metabolismo - min=0,05 max=0,4



(a) População (b) Genética

Fonte: Elaborada pelo autor.

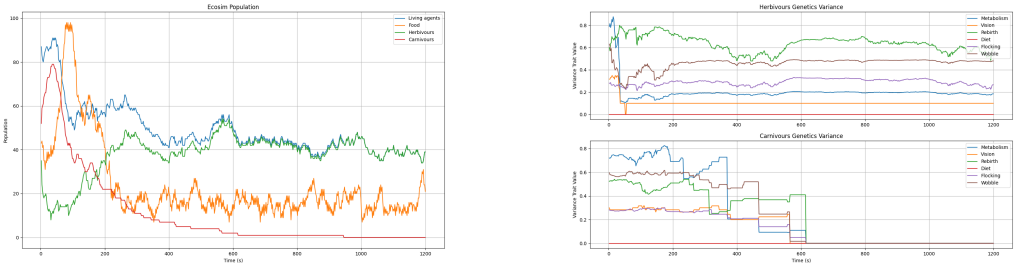
Figura 34 – Simulação Metabolismo - min=0,2 max=1,6



(a) População (b) Genética

Fonte: Elaborada pelo autor.

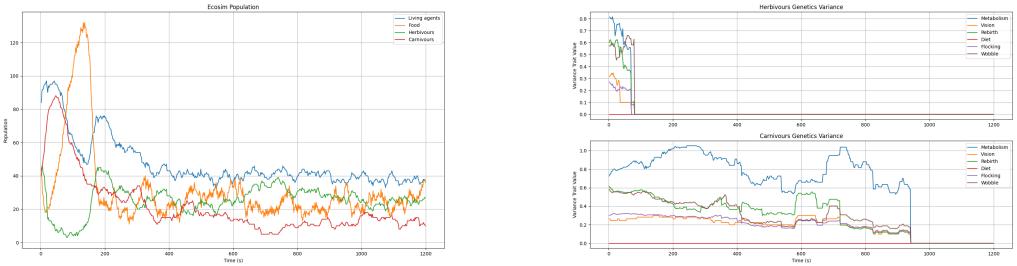
Figura 35 – Simulação Metabolismo - min=0,2 max=1,6



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 36 – Simulação Visão - min=0,2 max=1,6

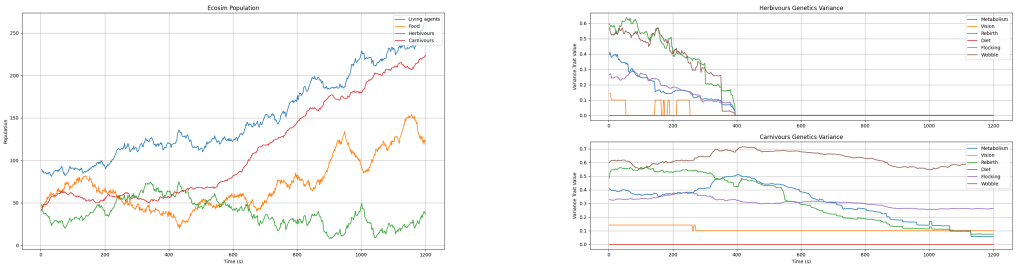


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.3 Experimentos visão

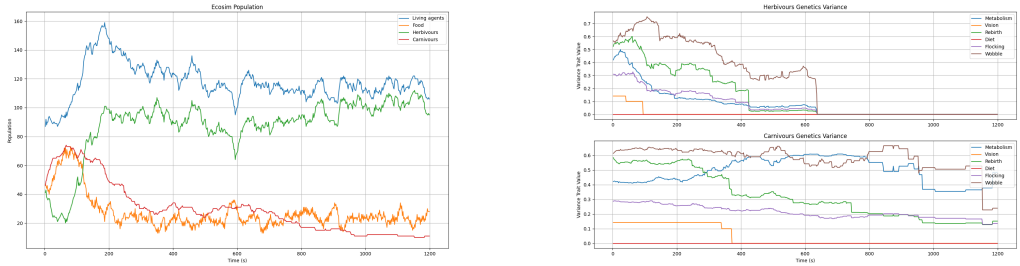
Figura 37 – Simulação Visão - min=0,05 max=0,1



(a) População (b) Genética

Fonte: Elaborada pelo autor.

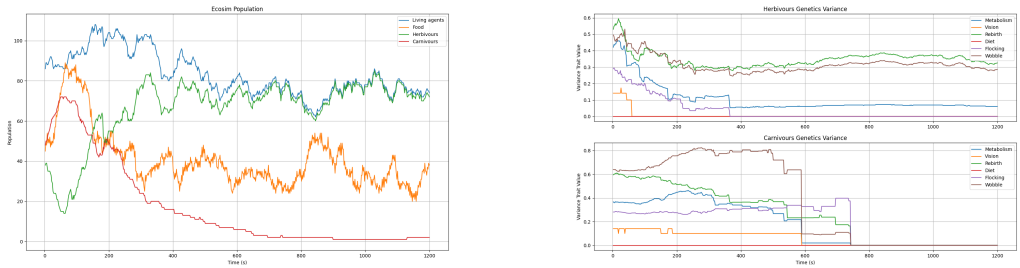
Figura 38 – Simulação Visão min=0,05 max=0,1



(a) População (b) Genética

Fonte: Elaborada pelo autor.

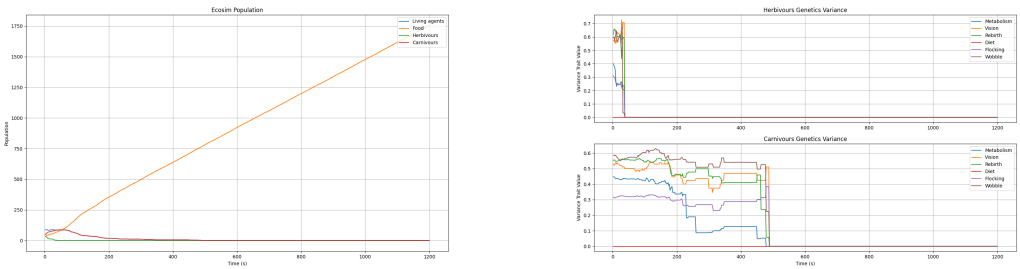
Figura 39 – Simulação Visão - min=0,05 max=0,1



(a) População (b) Genética

Fonte: Elaborada pelo autor.

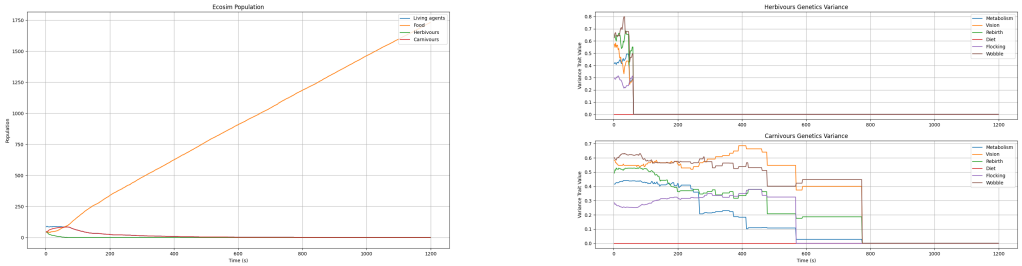
Figura 40 – Simulação Visão - min=0,2 max=0,4



(a) População (b) Genética

Fonte: Elaborada pelo autor.

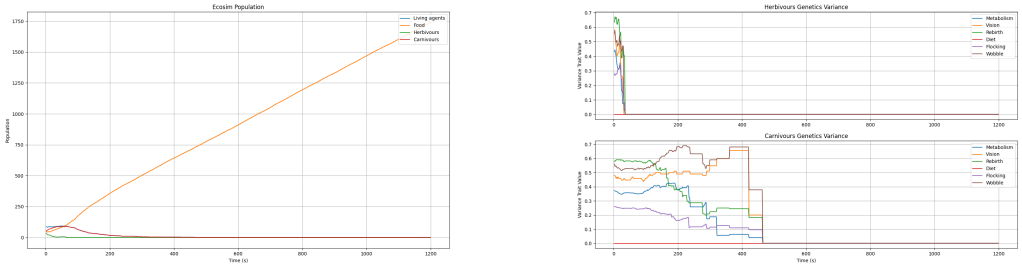
Figura 41 – Simulação Visão - min=0,2 max=0,4



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 42 – Simulação Visão - min=0,2 max=0,4

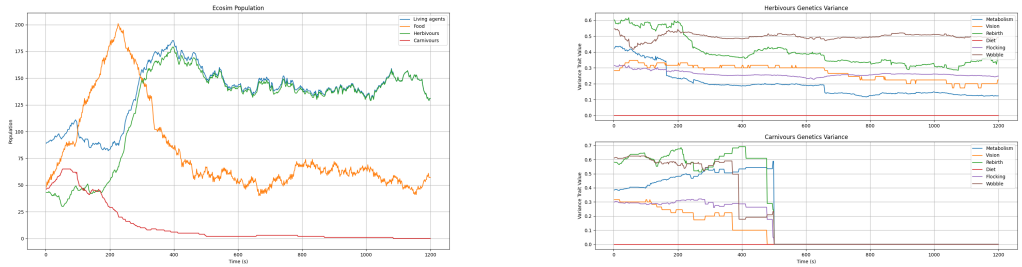


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.4 Experimentos pressão de espaço

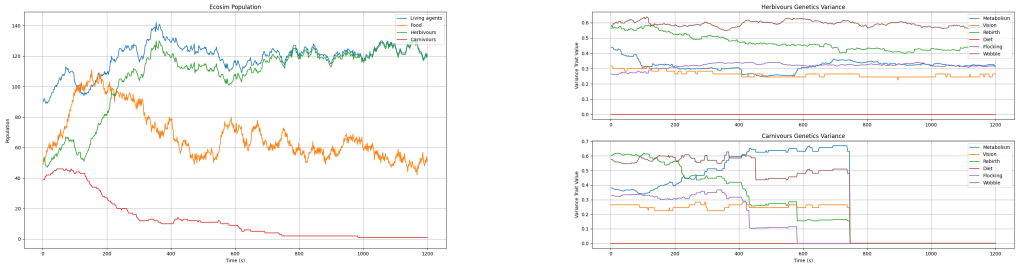
Figura 43 – Simulação espaço - min=-2 max=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

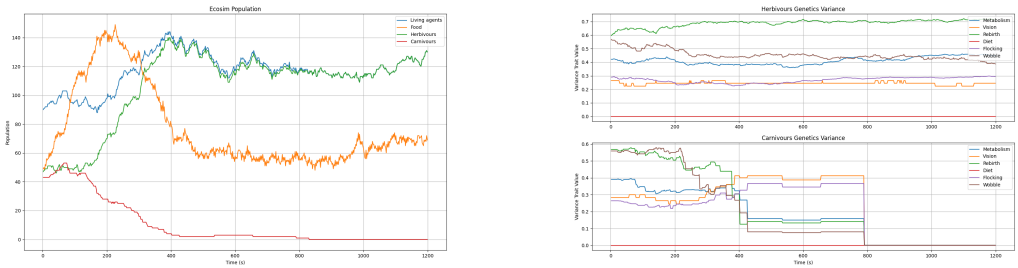
Figura 44 – Simulação espaço - min=-2 max=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

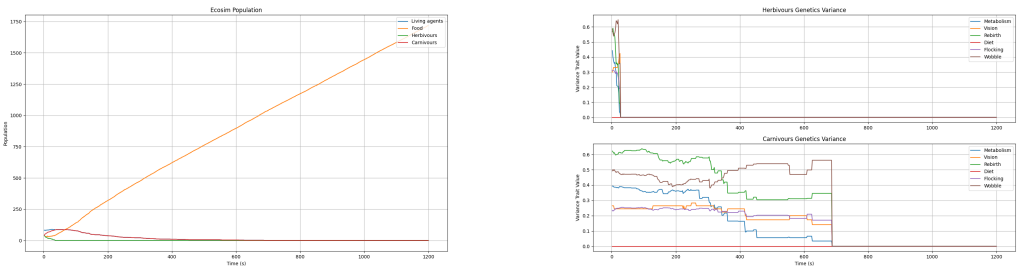
Figura 45 – Simulação espaço - min=-2 max=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

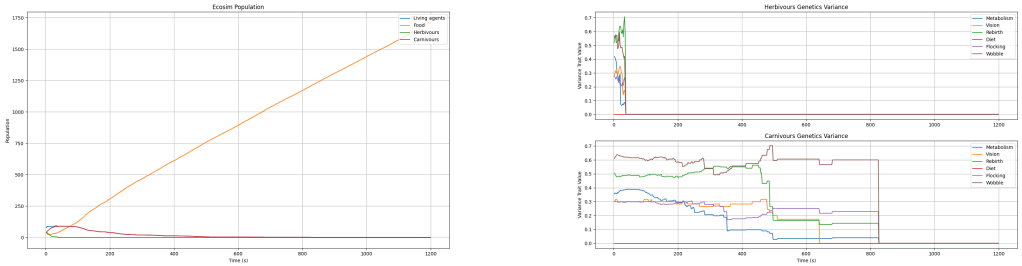
Figura 46 – Simulação espaço - min=-0,5 max=0,5



(a) População (b) Genética

Fonte: Elaborada pelo autor.

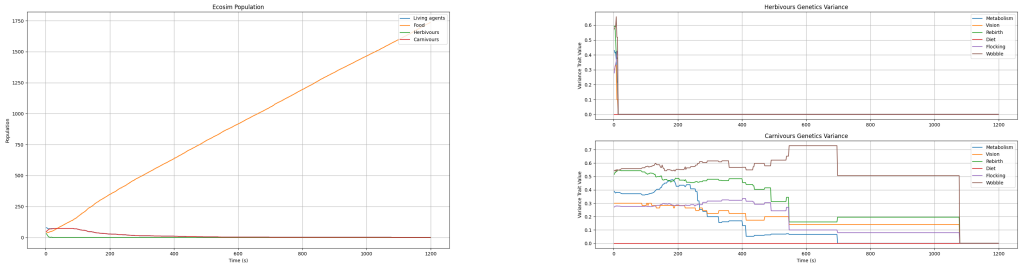
Figura 47 – Simulação espaço - min=-0,5 max=0,5



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 48 – Simulação espaço - min=-0,5 max=0,5

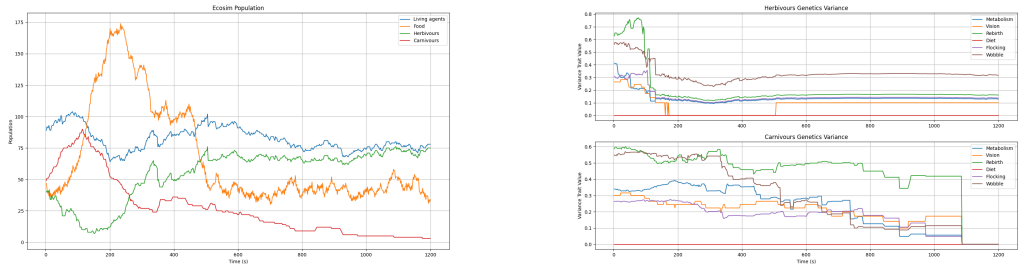


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.5 Experimentos pressão de reprodução

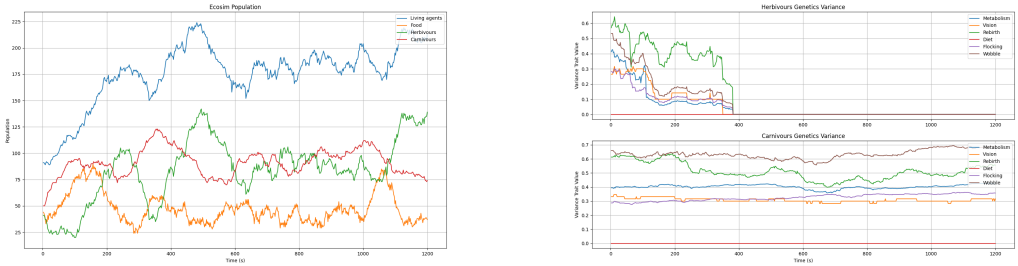
Figura 49 – Simulação reprodução - intervalo de tempo=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

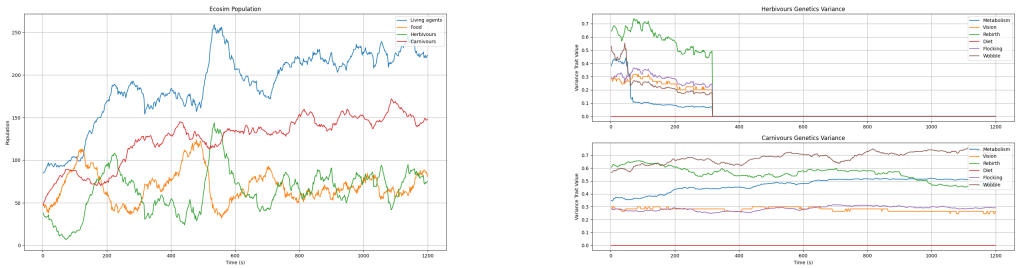
Figura 50 – Simulação reprodução - intervalo de tempo=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

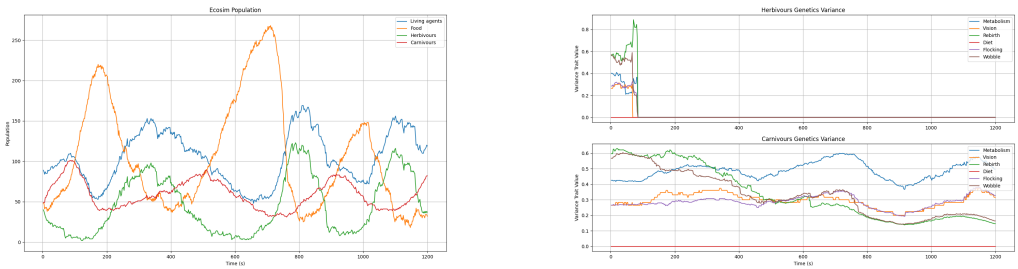
Figura 51 – Simulação reprodução - intervalo de tempo=2



(a) População (b) Genética

Fonte: Elaborada pelo autor.

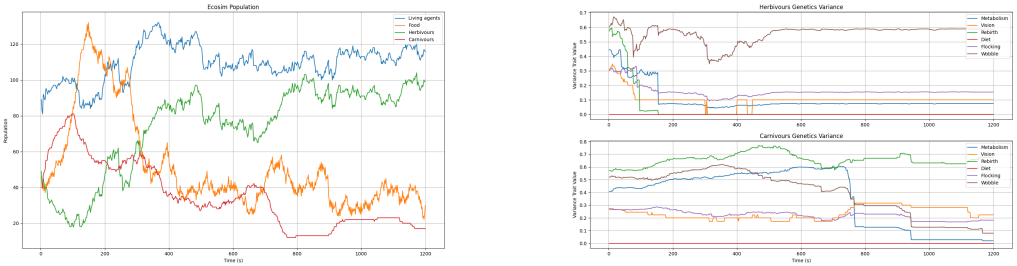
Figura 52 – Simulação reprodução - intervalo de tempo=8



(a) População (b) Genética

Fonte: Elaborada pelo autor.

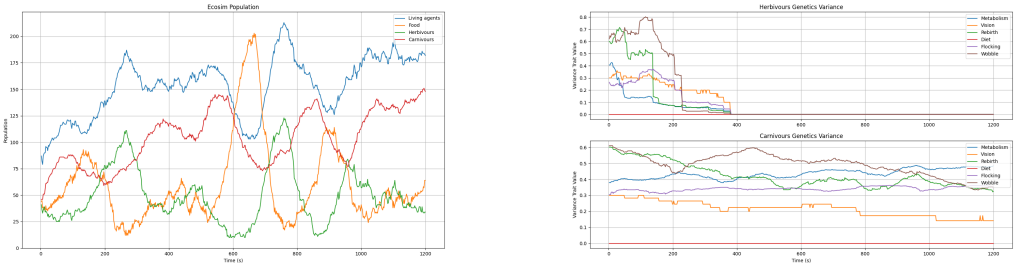
Figura 53 – Simulação reprodução - intervalo de tempo=8



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 54 – Simulação reprodução - intervalo de tempo=8

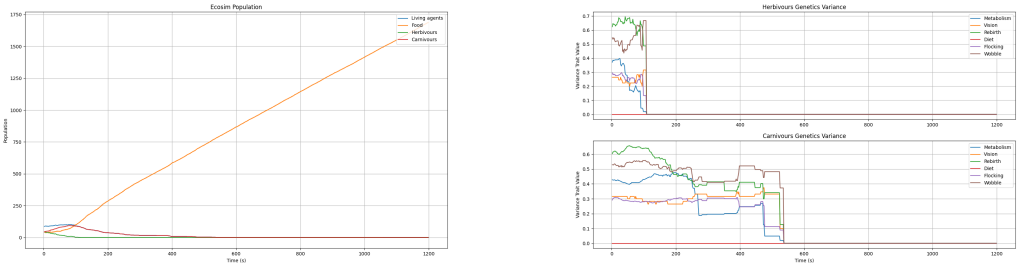


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.6 Experimentos pressão de predação

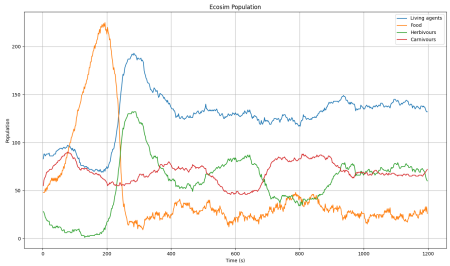
Figura 55 – Simulação pressão de predaçao - intervalo=0 à 0,5



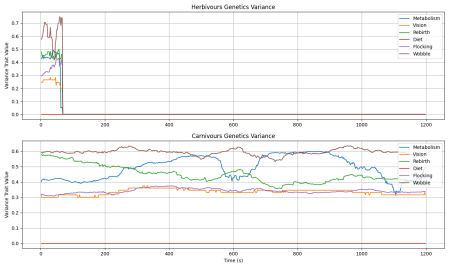
(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 56 – Simulação pressão de predaçao - intervalo=0 à 0,5



(a) População

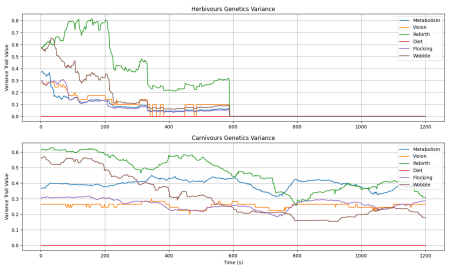


(b) Genética

Fonte: Elaborada pelo autor.

Figura 57 – Simulação pressão de predaçao - intervalo=0 à 0,5

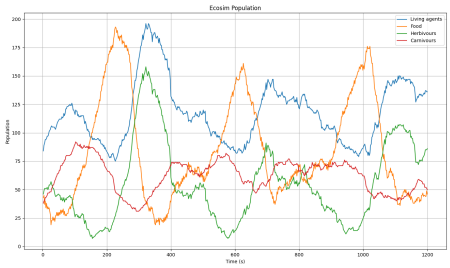
(a) População



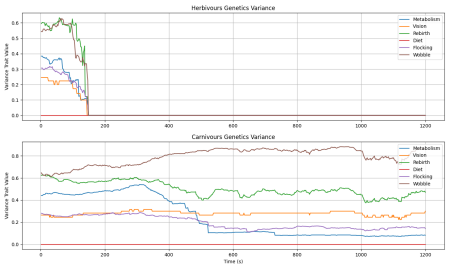
(b) Genética

Fonte: Elaborada pelo autor.

Figura 58 – Simulação pressão de predaçao - intervalo=0,5 à 2,0



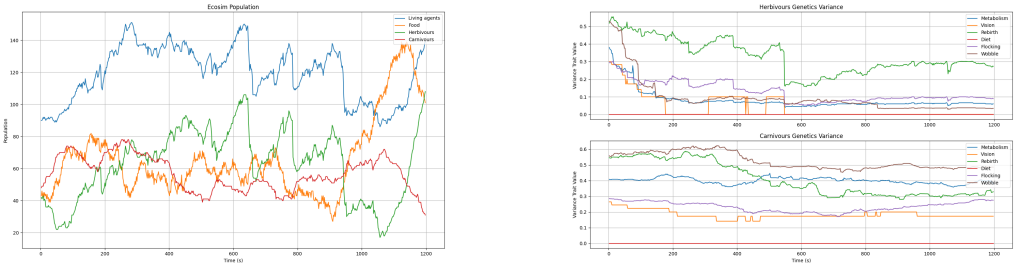
(a) População



(b) Genética

Fonte: Elaborada pelo autor.

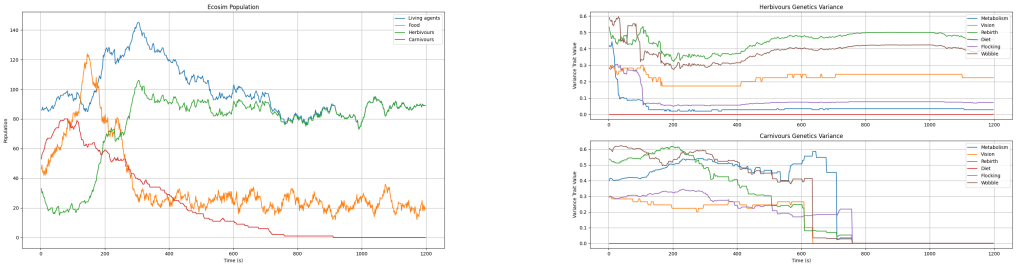
Figura 59 – Simulação pressão de predaçao - intervalo=0,5 à 2,0



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 60 – Simulação pressão de predaçao - intervalo=0,5 à 2,0

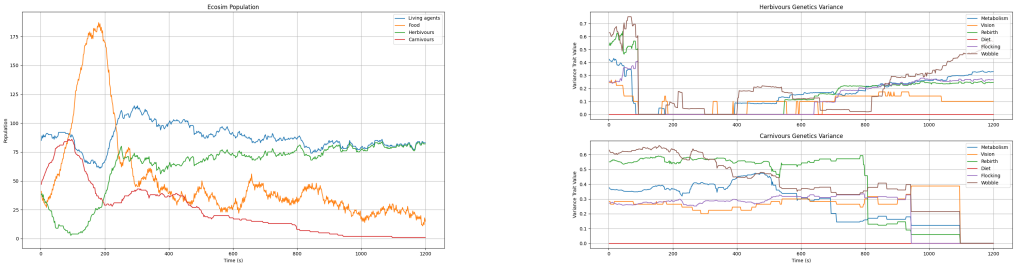


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.7 Experimentos mutação

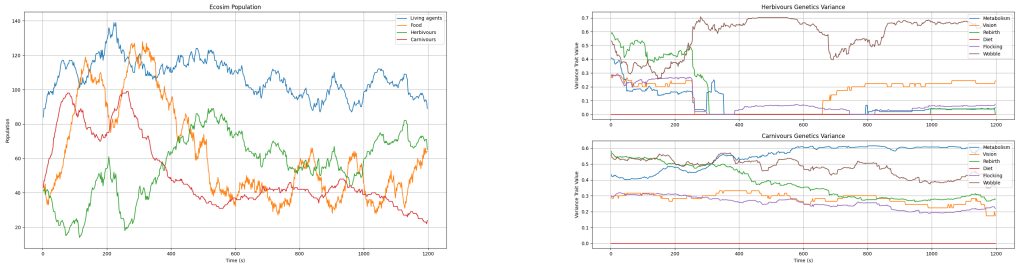
Figura 61 – Simulação com mutação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

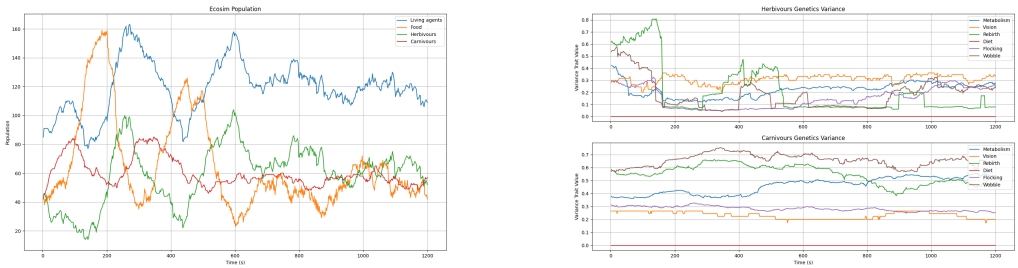
Figura 62 – Simulação com mutação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

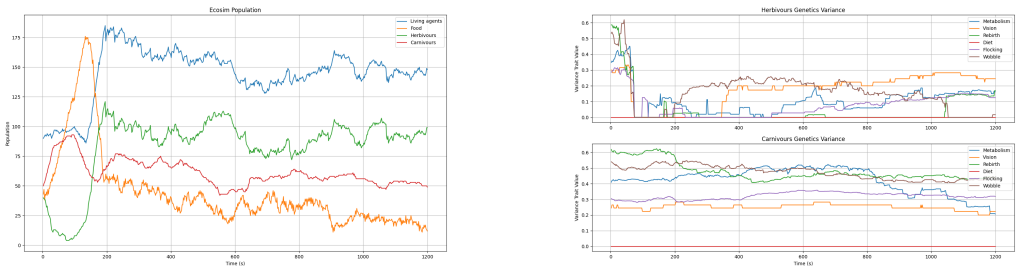
Figura 63 – Simulação com mutação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

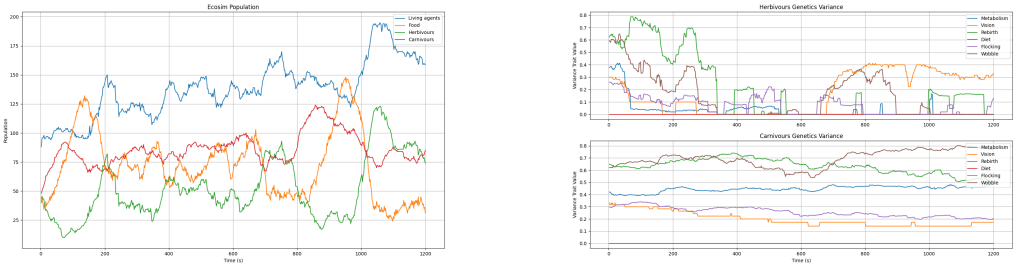
Figura 64 – Simulação com mutação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

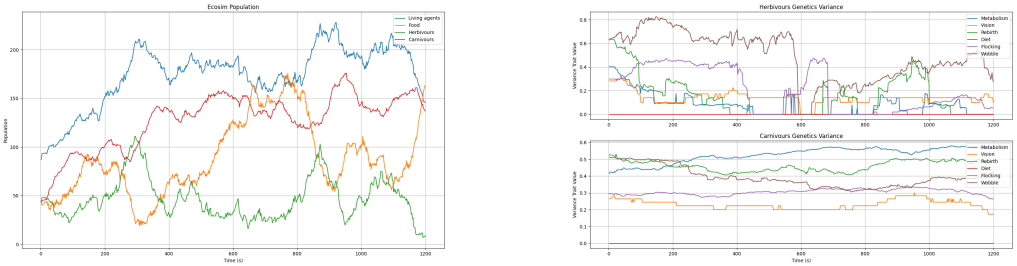
Figura 65 – Simulação com mutação



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 66 – Simulação com mutação

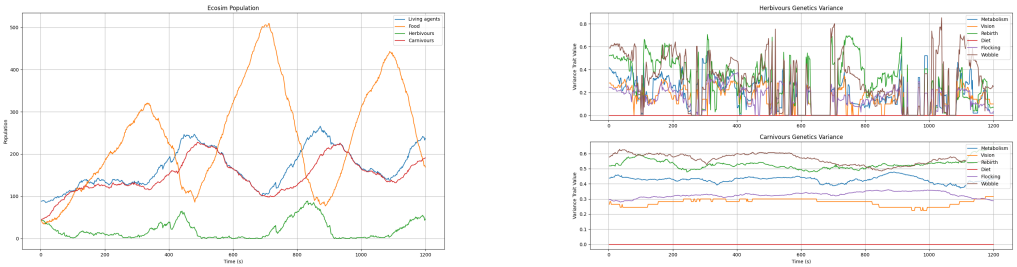


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.8 Experimentos elemento estrangeiro

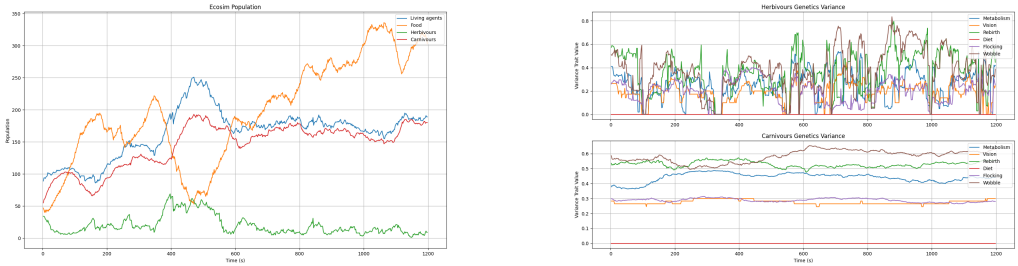
Figura 67 – Simulação com elemento estrangeiro



(a) População (b) Genética

Fonte: Elaborada pelo autor.

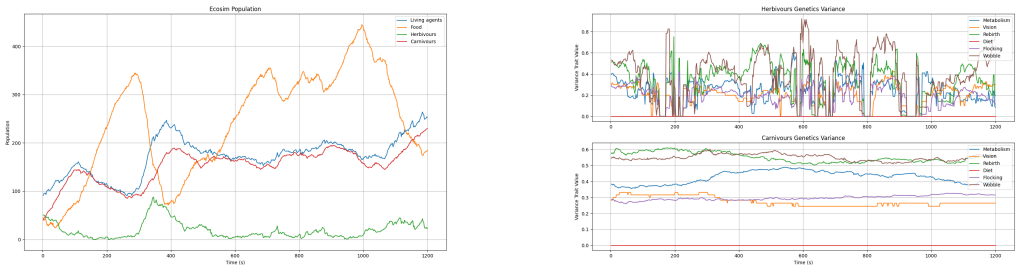
Figura 68 – Simulação com elemento estrangeiro



(a) População (b) Genética

Fonte: Elaborada pelo autor.

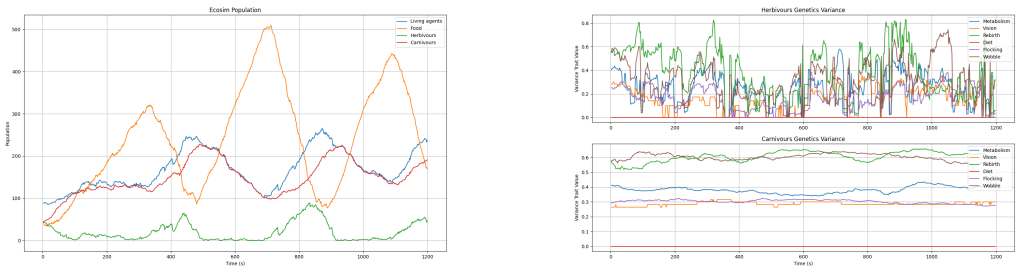
Figura 69 – Simulação com elemento estrangeiro



(a) População (b) Genética

Fonte: Elaborada pelo autor.

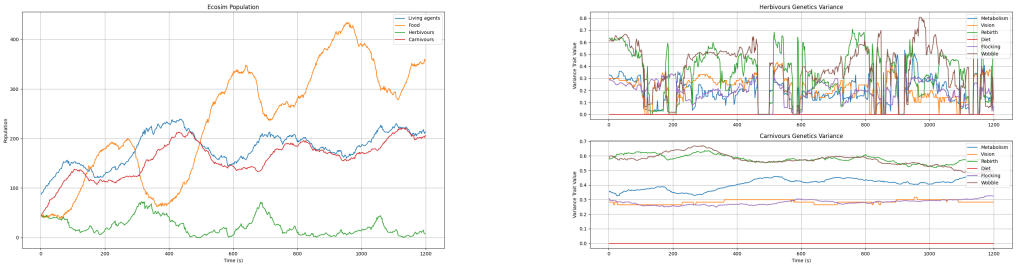
Figura 70 – Simulação com elemento estrangeiro



(a) População (b) Genética

Fonte: Elaborada pelo autor.

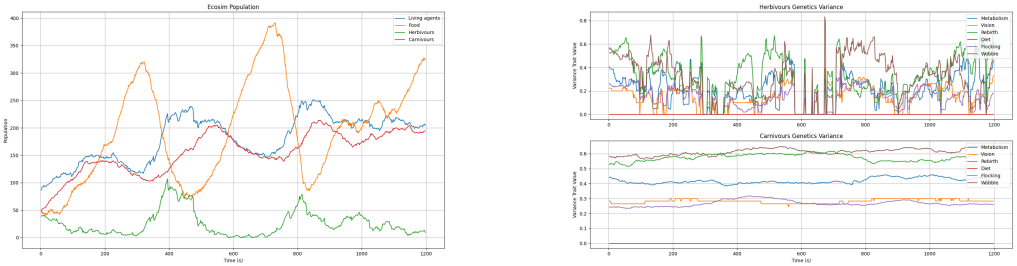
Figura 71 – Simulação com elemento estrangeiro



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 72 – Simulação com elemento estrangeiro

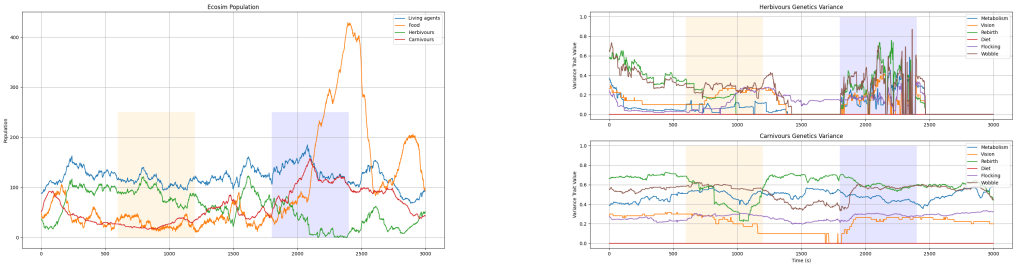


(a) População (b) Genética

Fonte: Elaborada pelo autor.

C.9 Experimentos intervalos intercalados

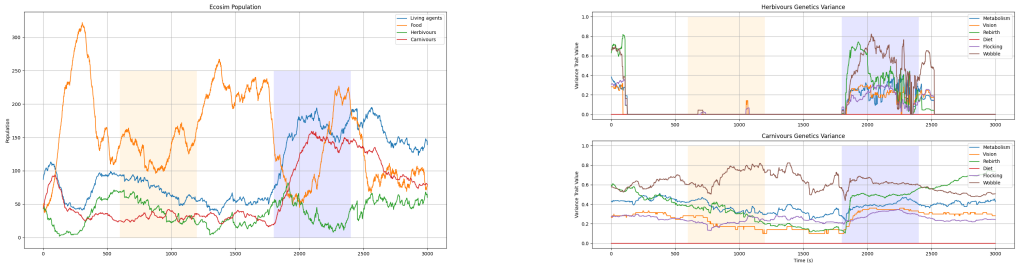
Figura 73 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

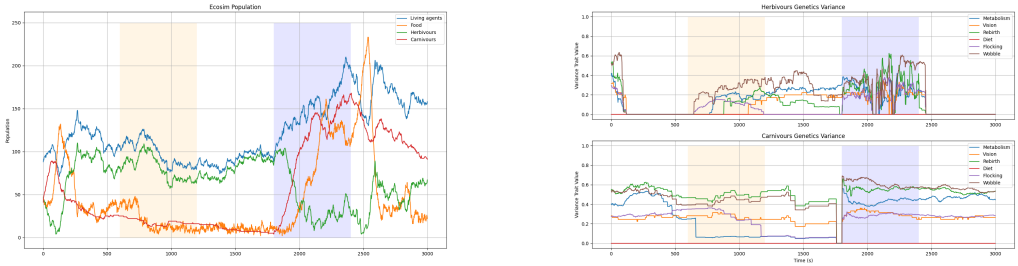
Figura 74 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

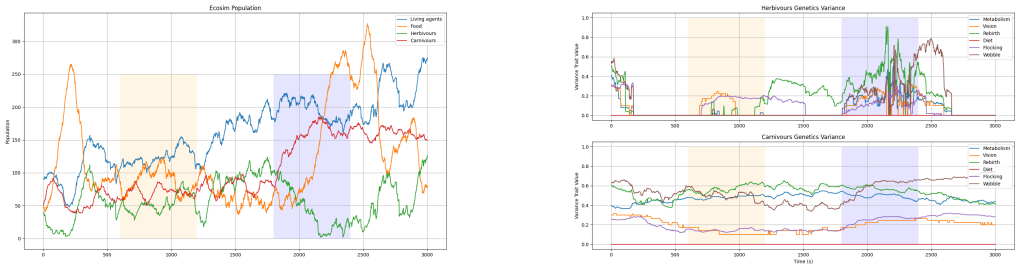
Figura 75 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

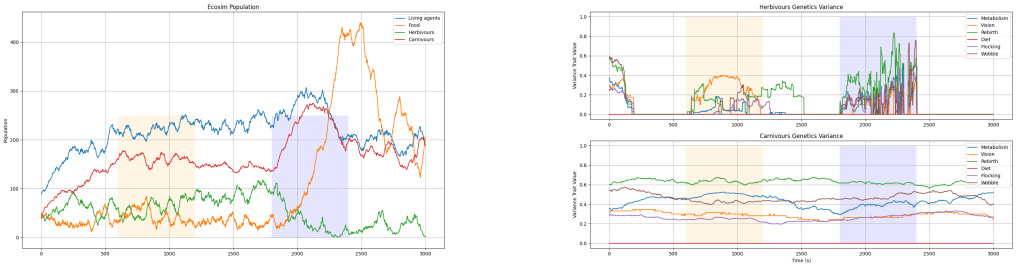
Figura 76 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

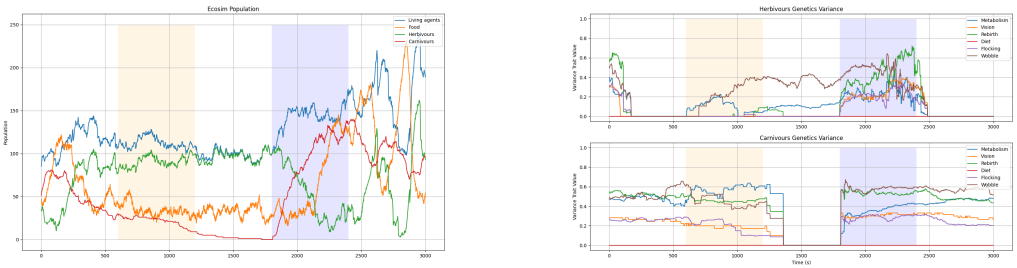
Figura 77 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

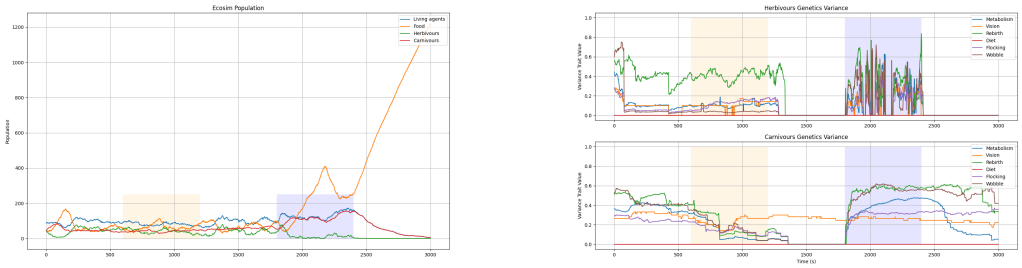
Figura 78 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

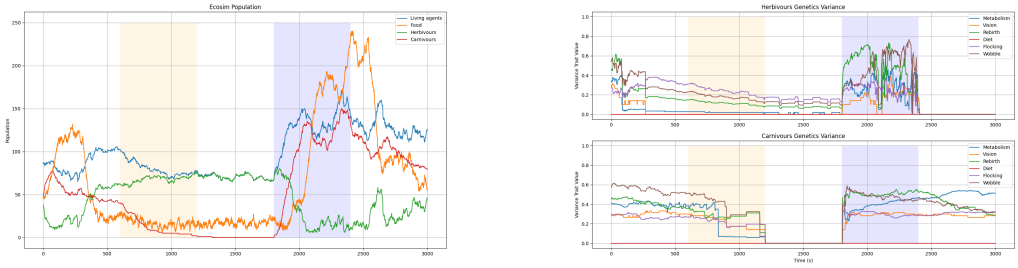
Figura 79 – Simulação com intervalos



(a) População (b) Genética

Fonte: Elaborada pelo autor.

Figura 80 – Simulação com intervalos



(a) População

(b) Genética

Fonte: Elaborada pelo autor.


ICMC
USP
SÃO CARLOS