MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Teste de modelos ambientais desenvolvidos via TerraME

RELATÓRIO DE RENOVAÇÃO DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/CNPq/INPE)

Bolsista: Leoni Augusto Romain da Silva
E-mail: augustoromain@gmail.com

Responsável: Dr. Valdivino Alexandre de Santiago Júnior
E-mail: valdivino.santiago@inpe.br

SÃO JOSÉ DOS CAMPOS

Julho/2017

.

RESUMO

A área temática *Modelagem do Sistema Terrestre e Projeção* do Centro de Ciência do Sistema Terrestre (CCST/INPE) objetiva pesquisar a representação do Sistema Terrestre (ST), abrangendo não somente as dimensões físicas e biológicas, como também as dimensões humanas. Existem diversas ações de pesquisa sólidas em relação a essa área temática do CCST/INPE, sendo que uma delas é o TerraME: um ambiente de desenvolvimento para a modelagem dinâmica espacial que apóia o conceito de Autômatos Celulares Aninhados (Nested-CA). Assegurar que os modelos ambientais estejam consistentes/corretos, é uma tarefa bastante desafiadora pois requer o conhecimento no domínio de aplicação, além do conhecimento da linguagem de programação em que o código-fonte do modelo foi escrito. Por outro lado, as metodologias, técnicas e processos da Engenharia de Software podem contribuir para melhorar a qualidade de um produto de software. A área de Verificação e Validação (V&V) da Engenharia de Software almeja contribuir para essa melhoria da qualidade. Teste de software é, muito provavelmente, o processo mais adotado, na prática, entre todos relacionadas à V&V. Os objetivos específicos desse projeto são: a.) investigar diversas técnicas para geração de casos de teste de software para modelos ambientais desenvolvidos via TerraME; b.) realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos TerraME, obtiveram melhor custo e efetividade. Esse relatório apresenta as atividades desenvolvidas no período de 01 de agosto de 2016 a 13 de julho de 2017.

1.) INTRODUÇÃO

O uso de modelagem ambiental é um esforço que cada vez mais é utilizado por pesquisadores para analisar fenômenos e processos voltados ao meio ambiente. Muitos são os fatores que levam a utilizar processos de modelagem, como a facilidade de simulação via esses modelos, mesmo em casos de sistemas complexos.

A área temática *Modelagem do Sistema Terrestre e Projeção* do Centro de Ciência do Sistema Terrestre (CCST/INPE) objetiva pesquisar a representação do Sistema Terrestre (ST), abrangendo não somente as dimensões físicas e biológicas, como também as dimensões humanas. Existem diversas ações de pesquisa sólidas em relação a essa área temática do CCST/INPE, sendo que uma delas é o TerraME: um ambiente de desenvolvimento para a modelagem dinâmica espacial que apóia o conceito de Autômatos Celulares Aninhados (Nested-CA) [Carneiro 2006].

O TerraME é capaz de apoiar três paradigmas de modelagem: modelagem dinâmica do sistema, autômatos celulares e modelagem baseada em agentes. O ambiente de desenvolvimento TerraME é apresentado na Figura 1. Os principais componentes do TerraME são:

- O interpretador TerraME, que é a parte fundamental de todo o ambiente, pois é responsável pela execução do código-fonte do modelo, escrito na linguagem de modelagem TerraME (uma extensão da linguagem Lua), e também para chamar funções do framework TerraME;

- O framework TerraME, um conjunto de módulos escritos na linguagem C ++ que fornece funções e classes para a modelagem dinâmica espacial. Tal framework também se conecta a bancos de dados espaciais TerraLib;

- TerraView, uma aplicação de Sistema de Informação Geográfica (SIG) desenvolvida sobre a biblioteca C++ TerraLib para o gerenciamento de banco de dados espacial [Câmara et al. 2000];

- Um editor de texto ou um ambiente de desenvolvimento integrado, como o Eclipse, que fornece sintaxe de destaque para a linguagem de programação Lua [Ierusalimschy et al. 1996] e, portanto, para o código-fonte de modelo TerraME.

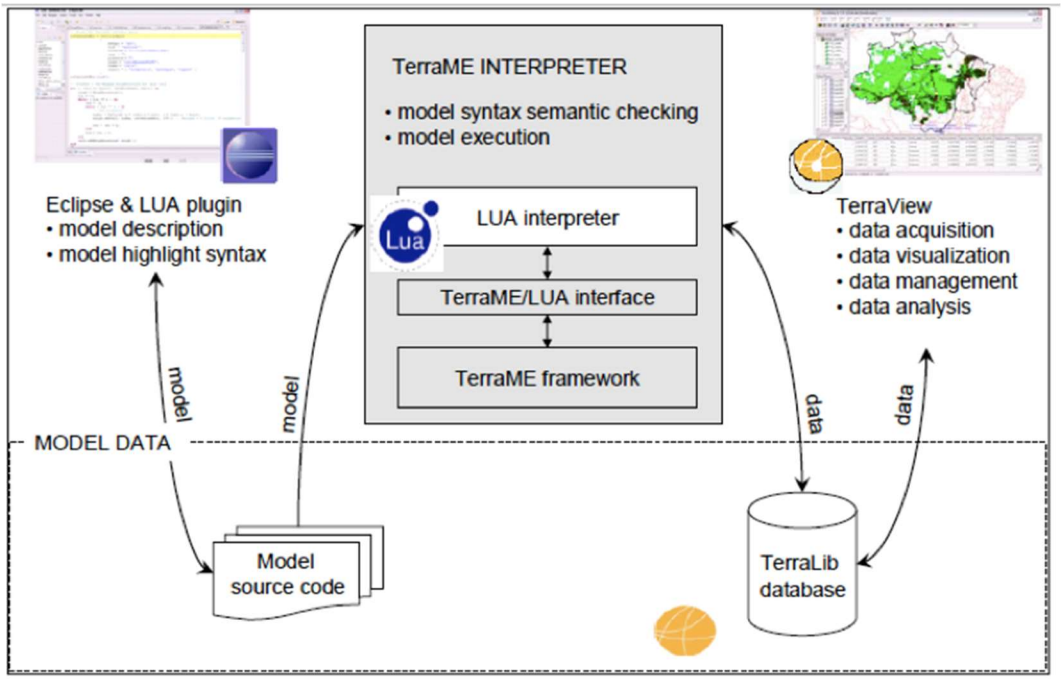

Figura *1*- Ambiente de Desenvolvimento TerraME

Assegurar que os modelos ambientais estejam consistentes/corretos, é uma tarefa bastante desafiadora pois requer o conhecimento no domínio de aplicação, além do conhecimento da linguagem de programação em que o código-fonte do modelo foi escrito. Notar que um modelo incorreto pode resultar em uma análise, pelo especialista, inadequada onde, por exemplo, estimativas incoerentes de áreas desmatadas em florestas brasileiras ou de queimadas ocorrendo no nosso território podem ser divulgadas para a sociedade. Portanto, é de fundamental relevância tentar se certificar que os modelos ambientais desenvolvidos via TerraME estejam coerentes.

Por outro lado, as metodologias, técnicas e processos da Engenharia de Software podem contribuir para melhorar a qualidade de um produto de software. A área de Verificação e Validação (V&V) da Engenharia de Software almeja contribuir para essa melhoria da qualidade. Teste de software é, muito provavelmente, o processo mais adotado, na prática, entre todos relacionadas à V&V. O objetivo de testar um produto de software é encontrar defeitos no código-fonte do mesmo. Inúmeras teorias, metodologias, abordagens têm sido propostas e/ou usadas para as diversas atividades do processo de Teste de software. Uma das atividades do processo de Teste mais estudada, mas que ainda apresenta diversos desafios, é a **geração/seleção de casos de teste**. No fundo, dado que a execução de teste

exaustivo não é viável, a ideia é utilizar de formas para selecionar, de infinitas possibilidades, um conjunto de dados de entrada de teste do domínio de entrada de um programa P, de forma a detectar o maior número possível de defeitos. Existem diversas abordagens para esse propósito, como particionamento por classes de equivalência, testes aleatórios [Anand et al. 2013], designs combinatoriais [Balera e Santiago Júnior 2015][Mathur 2008], Testes Baseados em Modelos (TBM) [Utting e Legeard 2007][Santiago Júnior 2011], teste de mutação [Delamaro et al. 2007], entre outras.

Portanto, é muito interessante investigar como essas técnicas para geração de casos de teste de software, usualmente aplicadas a produtos desenvolvidos no escopo da Engenharia de Software, podem ajudar na detecção de problemas/defeitos de modelos ambientais. Por exemplo, designs combinatoriais [Mathur 2008] são um conjunto de técnicas de geração de casos de teste de software que buscam a seleção de um pequeno número de casos de teste, uma vez que o domínio de entrada, e o número de subdomínios em suas partições, é largo e complexo. Essa técnica tem-se mostrado eficiente na revelação de defeitos por meio da interação de várias variáveis de entrada. Desse modo, algoritmos para gerar casos de teste de software via designs combinatoriais, por exemplo via a técnica de Matriz de Cobertura com Níveis Variados (MCNV) [Balera e Santiago Júnior 2015], podem ser usados onde as variáveis do modelo ambiental seriam os fatores que são entrada para o algoritmo. Em TBM, pode-se pensar em um meta-modelo, ou seja, um modelo comportamental de um modelo ambiental, para gerar casos de teste de software. Tal meta-modelo pode ser elaborado em diversas linguagens/modelos formais tais como Statecharts [Santiago Júnior 2011] [Santiago Júnior e Vijaykumar 2012][Santiago Júnior et al. 2012], Máquinas de Estados Finitos [Endo e Simão 2013], e Sistemas de Transição que apóiam o método de Verificação Formal chamado Model Checking [Baier e Katoen 2008][Clarke e Emerson 2008][Queille e Sifakis 2008][Fraser et al. 2009]. Portanto, tal investigação pode ser muito frutífera para diminuir os defeitos dos modelos ambientais desenvolvidos via TerraME, e agregar valor ao produto como um todo.

Dado que diversas técnicas para geração de casos de teste serão consideradas nesse projeto, é também muito relevante realizar uma comparação estatística rigorosa, via experimento controlado ou quasiexperimento [Zannier et al. 2006], para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e eficiência. A ausência de estudos de avaliação mais rigorosos na área de Teste de software é um fato comprovado por recentes publicações, não somente na comunidade brasileira, mas na comunidade internacional de Engenharia de Software [Lemos et al. 2013]. Desse modo, um dos resultados importantes do projeto é determinar, entre as técnicas selecionadas para geração de casos de teste, que tiveram melhor custo (geração de um menor conjunto de casos de

teste) e melhor eficiência (habilidade de detectar defeitos no código-fonte que representa o modelo ambiental TerraME).

Como uma contribuição secundária desse projeto, também é valioso realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes). Análise estática de código-fonte é um processo automatizado de revisão de código para avaliação do mesmo, ou outra representação do sistema, sem executá-lo [Chess e West 2007]. Portanto, isso difere de Teste onde é necessária a execução do programa. Semelhante a inspeções manuais, análise estática automática é uma técnica em que o código-fonte do produto (modelo TerraME, nesse caso) é verificado a procura de padrões especiais que são automaticamente classificados como potencialmente defeituosos. Como exemplo, partes do código-fonte que não possuem verificação dos valores de dados de entrada, onde é possível que ocorra uma exceção devido a divisão por zero ou overflow numérico, são potenciais causadores de defeitos quando do uso do software.

Os objetivos específicos desse projeto são:

a.) investigar diversas técnicas para geração de casos de teste de software para modelos ambientais desenvolvidos via TerraME;

b.) realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos TerraME, obtiveram melhor custo e efetividade.

Esse relatório apresenta as atividades desenvolvidas no período de **01 de agosto de 2016 a 13 de julho de 2017.**

2.) CRONOGRAMA DE ATIVIDADES E ETAPAS CONCLUÍDAS

Conforme mostrado no “Formulário para Solicitação de Bolsa PIBIC”, a metodologia a ser empregada para atender aos objetivos do projeto está descrita a seguir.

1. Estudar a fundamentação teórica relativa ao projeto. Especificamente, se familiarizar com os conceitos relacionados à Teste de software, linguagem de programação Lua, modelagem ambiental, TerraME, e análise estática de código-fonte;

2. Investigar diversas técnicas para geração de casos de teste de software (teste aleatório, designs combinatoriais, Testes Baseados em Modelos) para modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);

3. Executar os casos de teste gerados pelas diversas técnicas para geração de casos de teste e analisar os resultados das execuções. Para isso, será ampliado o uso do Jenkins, um servidor de automação para construção, entrega e automação de projetos de software que tem sido adotado para o desenvolvimento do TerraME, para outros repositórios do projeto tais como Modelos de Autômatos Celulares e Modelos Espaciais baseados em Agentes;

4. Realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e efetividade. Para isso, será planejado e conduzido um experimento controlado ou quasi-experimento;

5. Realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);

6. Submeter artigo para conferência e/ou workshop e/ou simpósio na área de Engenharia de Software e/ou Ciência do Sistema Terrestre, e elaborar relatório final de atividades.

O cronograma de atividades pode ser visualizado na Figura 1, cada atividade está de acordo com os números e acima, e cada uma das colunas de Ano I e II representa um mês.

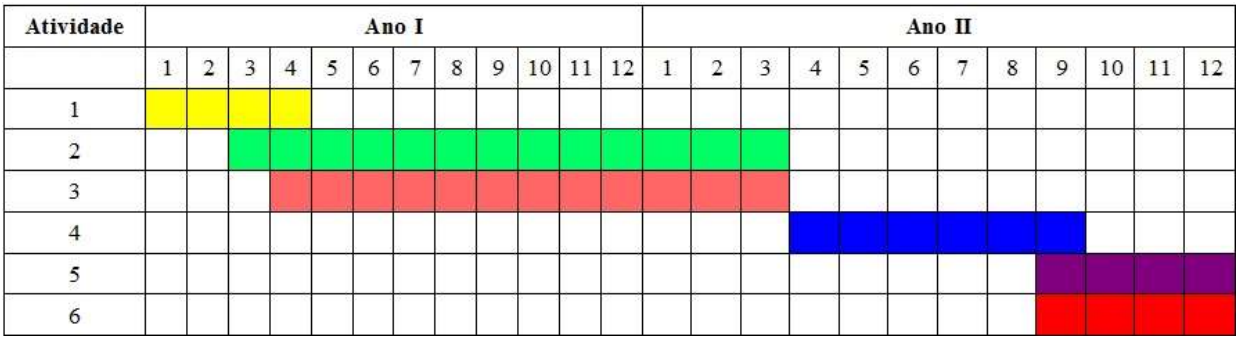

Figura *2* - Cronograma de Atividades

Dessa forma, esse relatório compreende o mês 1 (agosto/2016) ao mês 12 (julho/2017) referente ao primeiro ano. Considerando as atividades previstas para serem desenvolvidas, conforme a Figura 2, a Tabela 1 exibe as atividades concluídas e a concluir considerando o período referente a este relatório.

Tabela 1 – Etapas Concluídas e a Concluir

	Atividades da Metodologia	Previsão	Realização
1	Estudar a fundamentação teórica relativa ao projeto. Especificamente, se familiarizar com os conceitos relacionados à Teste de software, linguagem de programação Lua, modelagem ambiental, TerraME, e análise estática de código-fonte;	100%	100%
2	Investigar diversas técnicas para geração de casos de teste de software (teste aleatório, designs combinatoriais, Testes Baseados em Modelos) para modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);	77%	72%
3	Executar os casos de teste gerados pelas diversas técnicas para geração de casos de teste e analisar os resultados das execuções. Para isso, será ampliado o uso do Jenkins, um servidor de automação para construção, entrega e automação de projetos de software que tem sido adotado para o desenvolvimento do TerraME, para outros repositórios do projeto tais como Modelos de Autômatos Celulares e Modelos Espaciais baseados em Agentes;	75%	65%
4	Realizar uma comparação estatística rigorosa para identificar quais das técnicas, usadas para geração de casos de teste para os modelos ambientais, obtiveram melhor custo e efetividade. Para isso, será planejado e conduzido um experimento controlado ou quasi-experimento;	0%	0%
5	Realizar análise estática do código-fonte dos modelos ambientais desenvolvidos via TerraME (por exemplo, repositórios do projeto Modelos de Autômatos Celulares, Modelos Espaciais baseados em Agentes);	0%	100%
6	Submeter artigo para conferência e/ou workshop e/ou simpósio na área de Engenharia de Software e/ou Ciência do Sistema Terrestre, e elaborar relatório final de atividades.	0%	0%

Na Tabela 1 acima, a coluna **Previsão** mostra a porcentagem prevista para a realização da atividade, e a coluna **Realização** mostra a porcentagem realmente realizada da atividade, considerando o período a que se refere esse relatório (01 de agosto de 2016 a 13 de julho de 2017). Desse modo, pode-se dizer que todas as atividades previstas para esse período da bolsa foram realizadas de forma satisfatória. A atividade 1 foi totalmente concluída (100%), assim como a atividade 2 foi bastante evoluída até o momento. A atividade 3 está sendo desenvolvida de forma concomitante a atividade 2, onde os casos de testes estão sendo executados conforme estão sendo gerados. Por outro lado, a atividade 5, que não estava prevista para ser realizada no período a que se refere esse relatório, foi totalmente (100%) concluída.

Na atividade 1, foram realizados estudos sobre a fundamentação teórica relacionada ao projeto, onde ocorreu uma familiarização com conceitos relacionados a teste de software, modelagem ambiental, TerraME, linguagem lua e análise estática de código-fonte. Foi, inclusive, cursada uma disciplina na Pós-Graduação do INPE, CST-

317: Introduction to Earth System Modelling, com o objetivo de se aprofundar em realização de modelagem ambiental usando o TerraME.

Na atividade 2, foi feita uma investigação sobre as técnicas de geração de casos de teste propostas (teste aleatório, designs combinatoriais, Testes Baseados em Modelos), para considerar quais as melhores formas de se aplicar essas técnicas, bem como quais ferramentas e frameworks devem ser utilizados no projeto. Essa atividade é uma das mais desafiadoras, principalmente para se gerar os resultados esperados dos casos de teste para um determinado conjunto de entradas (parâmetros dos modelos ambientais). A solução adotada, e que está sendo investigada, envolve a geração de dados de entrada de teste utilizando teste aleatório, designs combinatoriais, Testes Baseados em Modelos para modelos ambientais desenvolvidos via TerraME. Uma nova metodologia, denominada "*Test **Da**ta Generation and **O**racle via Knowledge **B**ase and **M**achine **L**earning*" (DaOBML – Geração de Dados e Oráculo de Teste via Base de Conhecimento e Aprendizado de Máquina), foi proposta e está sendo desenvolvida. A metodologia DaOBML usa as técnicas acima para gerar os dados de entrada de teste e, uma vez gerados os dados de entrada, um oráculo de teste (que inclui a execução de testes) toma parte desses dados para gerar uma base de conhecimento. Então, técnicas de detecção de características em imagens (maps do TerraME), e algoritmos de aprendizado de máquina são usados pelo oráculo para comparar as saídas de acordo com outros dados de entrada de teste com a base de conhecimento. Com isso, é possível dar um veredito se há ou não defeitos nos modelos ambientais.

A atividade 3 (execução) ocorre depois da atividade 2 (geração de casos de teste) em um processo tradicional de teste de software. Mas, com a metodologia DaOBML, a atividade 3 (relacionada ao oráculo de teste) vem sendo realizada de forma concomitante a atividade 2. Investigações de ferramentas/frameworks para a linguagem Lua, tais como o Lunit e Lunatest, foram feitas. Mas, decidiu-se que não serão usados.

A atividade 5, que não estava prevista para ser realizada no período a que se refere esse relatório, foi totalmente concluída. Assim, foi feita a análise estática de código-fonte de todos os modelos dos pacotes de Autômatos Celulares e modelos Espaciais baseados em Agentes que se encontram dentro do TerraME. A conclusão foi que alguns warnings, descobertos por meio da ferramenta de análise estática LuaCheck, representam conflitos entre a linguagem Lua e o TerraME, que apesar de serem apontados como warnings, não precisam ser modificados pois este conflito ocorre devido as modificações que o TerraME possui em relação a linguagem lua. Os resultados também revelaram warnings considerados válidos e que ajudaram a melhorar os modelos dos pacotes de Autômatos Celulares e Baseados em Agentes, reduzindo linhas de código desnecessárias para o funcionamento dos modelos.

O detalhamento do desenvolvimento das atividades está apresentado a seguir.

3.) ATIVIDADE 1: FUNDAMENTAÇÃO TEÓRICA

3.1) Teste de Software

As atividades de teste são fundamentais para assegurar a qualidade do produto de software a ser entregue e mantido para os clientes. Testes devem ser planejados, especificados, projetados, construídos e documentados de forma que seja possível repetir ciclos de execução e aumentar sua capacidade em revelar falhas no software. (Lima,2013)

A Figura 3 apresenta o ciclo de vida da fase de teste:

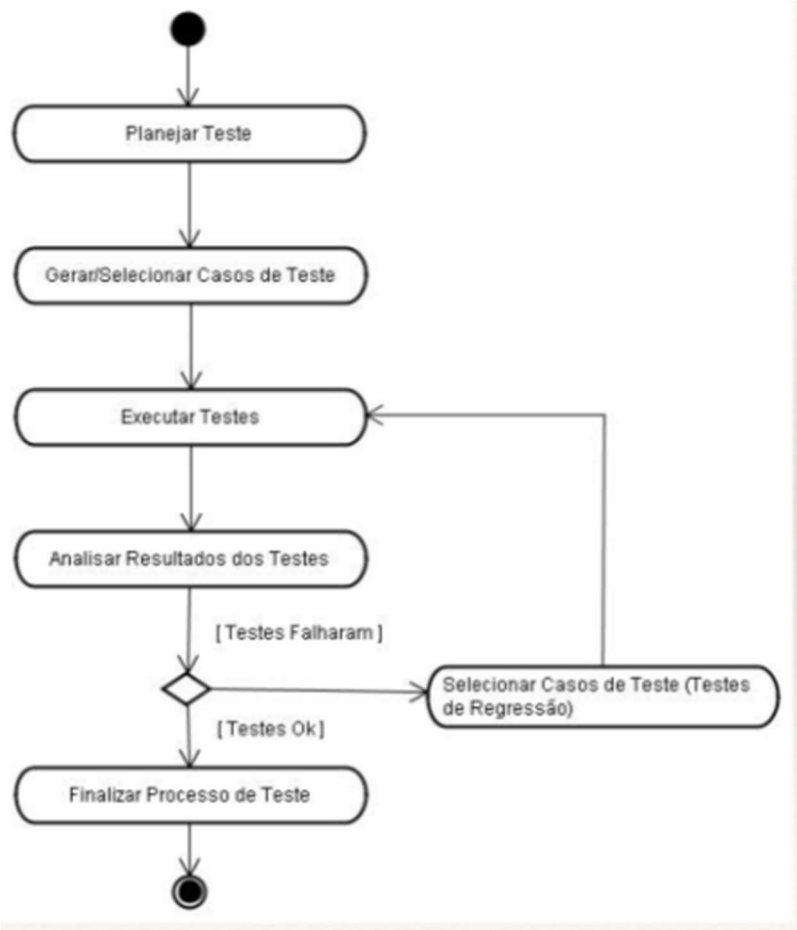

Figura *3* - Ciclo de vida do teste de software. Fonte (Santiago, 2016)

A etapa inicial consiste em planejar os casos de testes, de acordo com objetivos e requisitos presentes na documentação do projeto, cada tipo de aplicação contém características distintas que precisam ser analisadas durante a realização do planejamento de teste.

O passo seguinte é gerar/selecionar os casos de teste e preparar o ambiente de teste que será utilizado para executar os testes posteriormente. Após todos os testes estarem definidos é dado a execução dos testes e seu resultado e armazenado para ser analisado.

A análise consiste em verificar os resultados obtidos dos testes, e compara-los com os resultados esperados e requisitos que foram definidos no planejamento do teste, afim de encontram possíveis falhas e possam ter sido geradas. Caso o teste seja dado como falho é aplicado teste de regressão, que consiste em garantir que não surjam novos defeitos em códigos que já tenham sido analisados.

Após a finalização dos testes e correção das falhas, e feito a finalização do projeto, liberando-o para ser utilizado para seu fim especifico.

3.2) Modelagem Ambiental via TerraME e linguagem Lua

Afim de aperfeiçoar os conhecimentos em linguagem lua, e modelagem ambiental via TerraME, foi realizada a disciplina de introdução à Modelagem do Sistema Terrestre, disponibilizada pelo curso de Pós-graduação de Ciência do Sistema Terrestre (CST-INPE). Dentro do escopo da disciplina, foi desenvolvido como projeto final, um modelo ambiental de autômatos celulares, que aborta o tema sobre vírus influenza e simula sua manifestação dentro de um espaço celular.

Utilizando o ambiente computacional TerraME, foi desenvolvido, em linguagem Lua, uma versão do modelo CA para o vírus Influenza A, onde é apresentado uma nova perspectiva de como o vírus da gripe se manifesta entre as células do corpo humano, e como o organismo reage a essa infecção.

Para realizar a implementação do modelo CA, foram considerados alguns parâmetros, descritos em [Beauchemin et al. 2005], bem como foi definido um diagrama de transição de estados das células. Porém, as mudanças de transição definidas nesse diagrama não são exatamente como descritos no artigo, para que se pudesse chegar a alguns resultados em tempo hábil.

A Tabela 1 mostra os parâmetros usados para o desenvolvimento do modelo CA para o vírus Influenza A, e a Figura 1 mostra o diagrama de transição de estados das células.

Tabela 2 – Parâmetros usados no modelo CA.

Parâmetro	Descrição
qHealthy	Quantidade de células em tempo de execução
qDead	
qImmune	
qInfected	
qInfectious	
healthyLifespan	Tempo maximo de vida útil das células
immLifespan	
infectedLifespan	
infectiousLifespan	
infectiousTime	Tempo maximo para célula *infected* se tornar *infectious*
reviveTime	Tempo maximo para célula *dead* se tornar *healthy*

ageOfHealthy	Idade atual da célula em tempo de execução
ageOfImmune	
ageOfInfected	
ageOfInfectious	
timeToRevive	Mede tempo que uma célula esta no estado *dead*
turnInfectious	Mede quanto tempo falta para célula passar do estado *infected*
infectNeigh	Quantidade de vizinhos no estado *infectious* em tempo de execução

Nesse modelo, as células não foram separadas em grupos e*pithelial* e *immune* contendo cada um seus próprios estados. As células foram tratadas como um único grupo, e podem assumir os seguintes estados: *healthy*, *infectious*, *infected*, *immune* e *dead.* As transições entre os estados ocorrem da seguinte forma:

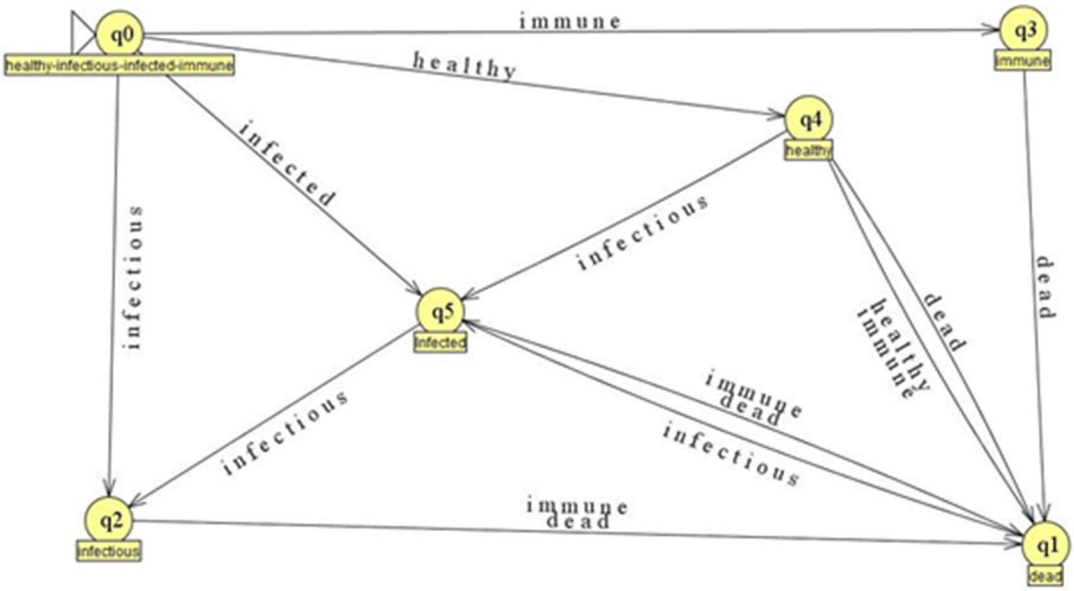

Figura *4* - Diagrama de transição de estados do model CA.

- As células começam aleatoriamente nos estados *healthy*, *infected*, *infectious eimmune*
- As células *healthy* se tornarão *dead q*uando atingirem uma idade maior que a healthyLifespan.
- As células *healthy* se tornarão *infected q*uando sua célula vizinha tiver o estado *infectious* e quando *infectNeigh* tiver valor maior do que 3.
- As células *infected* se tornarão *dead q*uando atingirem uma idade maior que a *infectedLifespan*.
- As células *infected* se tornarão *infectious* quando *turnInfectious* for maior que *infectiousTime.*
- As células *infected* se tornarão *dead* se tiverem uma célula vizinha com o estado *immune.*
- As células *immune* se tornarão *dead* quando atingirem uma idade maior que a *immLifespan*.

- As células *dead* voltarão ao estado *healthy* quando *timeToRevive* for maior que *reviveTime* e seu vizinho tiver o estato *healthy* ou *immune*.
- As células *dead* se tornarão *infectious* quando sua célula vizinha tiver o estado *infected.*

Para analisar a forma como o vírus se comporta, foram criados dois cenários diferentes com uma alternância de alguns atributos como *healthyLifespan*, *immLifespan*,*infectedLifespan ereviveTime.* Esses dois cenários são descritos a seguir.

Caso I

healthyLifespan= 3

immLifespan = 1

infectedLifespan = 3

reviveTime= 5

As Figuras 5 e 6 mostram os resultados para esse cenário. Conforme pode ser visualizado, nesse Caso I, foram considerados valores extremamente baixos para o tempo de vida útil das células e para o tempo que uma célula morta demora para reviver, devido as condições préestabelecidas no modelo. Isso provocou, rapidamente, a morte das células saudáveis e imunes, o que provocou, momentaneamente, o aumento de células mortas.

Porém, as células infetadas (infected) passaram ao estado infeccioso (infectious) na mesma proporção. Devido à condição de o modelo tornar, quando possível, as células mortas em células infecciosas, isso ocasionou, ao final da execução do modelo, a reprodução em massa das células infecciosas. Em outra visão, pode-se deduzir que se um organismo possui imunidade baixa ele se torna mais propenso a adquirir o vírus Influenza A.

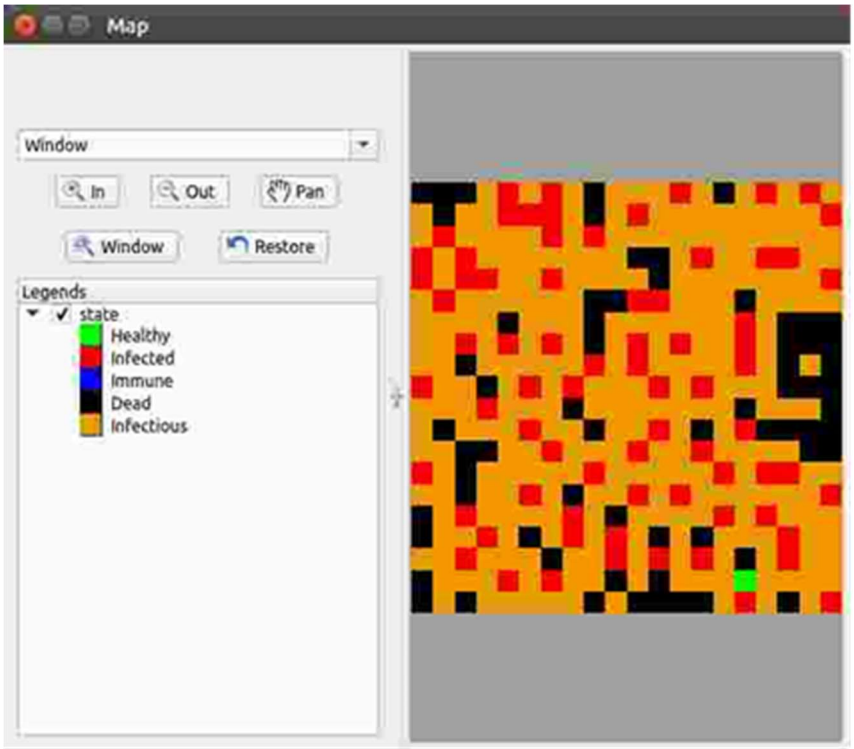

Figura *5* - Mapa do modelo CA utilizado no Caso I

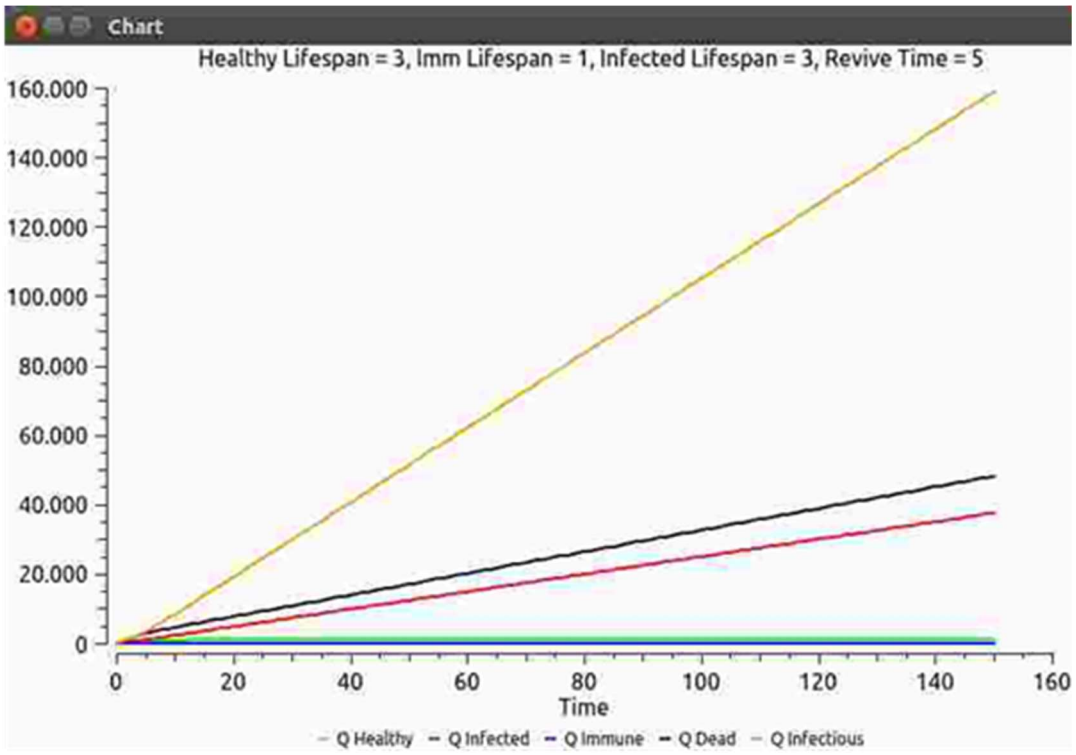

Figura *6* - Resultados do modelo CA utilizado no Caso I

Caso II

healthyLifespan = 70
immLifespan = 95
infectedLifespan = 82
reviveTime = 20

As Figuras 7 e 8 mostram os resultados para esse cenário. O Caso II apresenta resultado divergente ao Caso I. Nesse cenário, o valor das variáveis foi elevado, o que possibilitou as células imunes e saudáveis a se manifestarem por mais tempo, impedindo que as células infecciosas se multiplicarem, já que as células imunes, agora em grande quantidade, conseguem matar essas células infecciosas. Pode-se concluir que o organismo possui uma imunidade maior, e, assim, comprova-se que o vírus da gripe não se manifesta com eficiência no organismo.

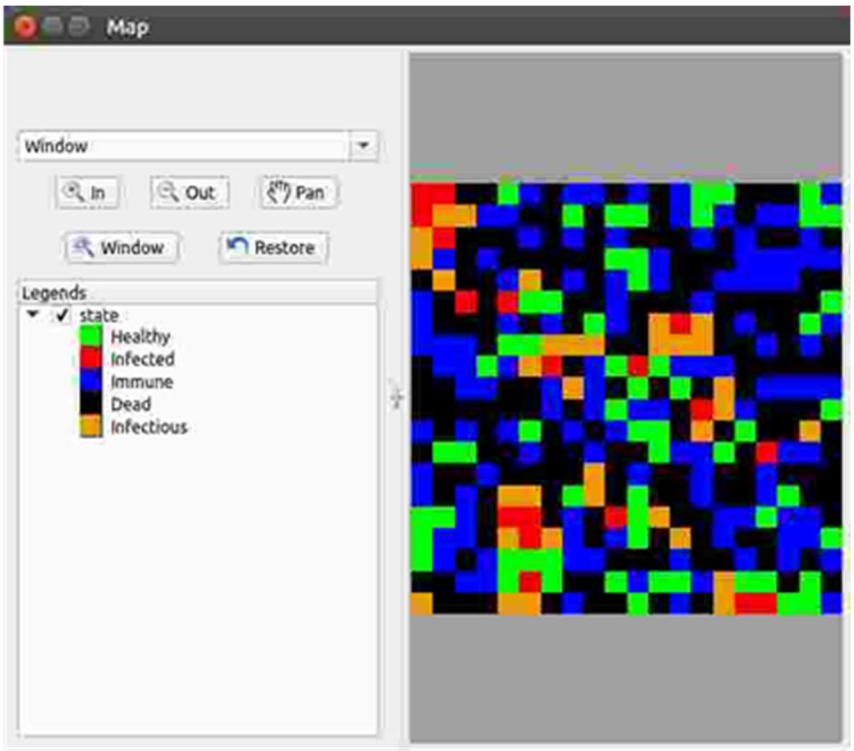

Figura 7 - Mapa do modelo CA utilizado no Caso II

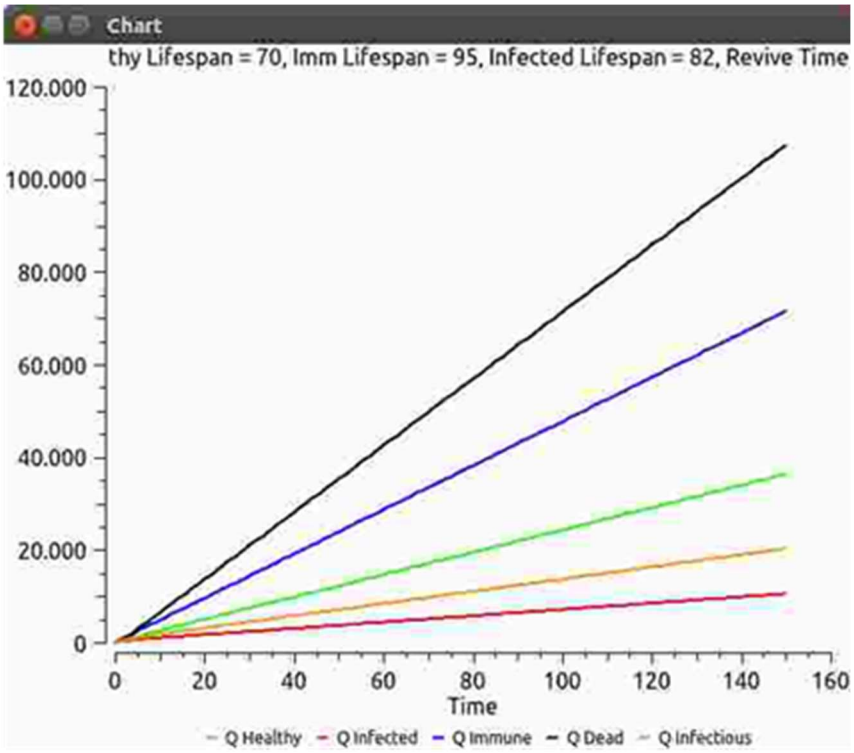

Figura *8* - Gráfico do modelo CA utilizado no Caso II

No decorrer desta atividade, foi possível concluir que o vírus da gripe (Influenza A) se manifesta de forma distinta em cada organismo. No Caso I, o vírus se manifestou em maior escala devido ao organismo possuir uma imunidade muito baixa em relação a quantidade de células que estavam infectadas. No Caso II, ocorreu um efeito contrário, pois o organismo estava com a imunidade consideravelmente alta, o que inibe propagação do vírus.

Para realizar esses casos, foi preciso alterar os valores das variáveis, para ajustar o modelo e encontrar um cenário com valores equilibrados. Sendo assim, o modelo só torna o resultado eficiente quando os valores são manipulados de acordo com o que se espera do fenômeno modelado.

4.) ATIVIDADES 2 e 3: GERAÇÃO DE CASOS DE TESTE E EXECUÇÃO DE CASOS DE TESTE

Teste de software é um processo essencial para garantir que um projeto funcione corretamente de acordo com suas especificações. Existem muitas formas de se aplicar técnicas de teste de software. Para esse projeto de pesquisa, a solução adotada, e que está sendo investigada, envolve a geração de dados de entrada de teste utilizando teste aleatório, designs combinatoriais, Testes Baseados em Modelos para modelos ambientais desenvolvidos via TerraME. Uma nova metodologia, denominada "*Test **Da**ta Generation and **O**racle via Knowledge **B**ase and **M**achine **L**earning*" (DaOBML – Geração de Dados e Oráculo de Teste via Base de Conhecimento e Aprendizado de Máquina), foi proposta e está sendo desenvolvida. A metodologia DaOBML usa as técnicas acima para gerar os dados de entrada de teste

e, uma vez gerados os dados de entrada, um oráculo de teste (que inclui a execução de testes) toma parte desses dados para gerar uma base de conhecimento. Então, técnicas de detecção de características em imagens (maps do TerraME), e algoritmos de aprendizado de máquina são usados pelo oráculo para comparar as saídas de acordo com outros dados de entrada de teste com a base de conhecimento. Com isso, é possível dar um veredito se há ou não defeitos nos modelos ambientais.

Com a nova abordagem DaOBML, a geração e execução de casos de teste está sendo feito de forma simultânea. As seções 4.1 e 4.2 abordam as atividades de geração de dados de teste. A seção 4.3 aborda a parte de oráculo de teste (que inclui a execução de casos de teste).

4.1) Teste Aleatório

Teste aleatório também conhecido como RT (Random Testing), é uma técnica onde os valores do teste são gerados aleatoriamente, possui uma fácil implementação e é muito usado em testes de segurança para detectar vulnerabilidades que possam existir no sistema, o que torna essa técnica útil também para testes de robustez avaliando como o sistema se comporta com diversas entradas aleatórias inesperadas. (Unicamp, 2014)

Inicialmente, para aplicar o teste aleatório, foi definido utilizar 4 variáveis do modelo Influenza, pois são as que mais influenciam no comportamento do algoritmo, como testar com valores aleatórios apresenta possibilidades infinitas, foi definido como critério de mínimo e máximo valores entre (0,100) , e utilizado a função Random, para alternar entre esses valores, os mesmos são passados por parâmetro dentro de um environment através de uma classe de tese do modelo influenza, que é representado abaixo:

```lua
require "influenza"

local healthyRandom = math.random(0, 100)
local immLifeRandom = math.random(0, 100)
local InfectRandom = math.random(0, 100)
local reviverandom = math.random(0, 100)

env = Environment{

    teste = Influenza{
        healthyLifespan =  healthyRandom,
        immLifespan = immLifeRandom,
        infectedLifespan = InfectRandom,
        reviveTime = reviverandom}
}

env:run()
```

Figura *9* - classe de teste aleatório do modelo influenza

Posteriormente, o teste aleatório foi aplicado para os modelos do pacote de autômatos celulares do TerraME, assim no modelo Influenza, foram utilizadas as variáveis que mais afetam o comportamento dos modelos, os valores de mínimo e máximo foram definidos baseando-se nos valores padrões de cada modelo.

```lua
import "ca"

math.randomseed(os.time())
ft = math.random(10,90)
d = math.random(60,80)

scene = Anneal{

            finalTime = ft,
            dim = d,
}
print(ft)
print(d)
scene:run()
```

Figura 10 - Classe de teste aleatorio do modelo Anneal

```
import "ca"
math.randomseed(os.time())

pc = math.random() + math.random(0, 1)
if(pc > 1) then
    pc = pc/2
end

dc = math.random() + math.random(1, 4)
if (dc > 3.5) then
    dc = dc /2
end

wc = math.random() + math.random(0, 1)
if ( wc > 1.2) then
    wc = wc/2
end

rps = math.random(100,500)
r = math.random(100,500)
ft = math.random(20,90)

scene = BandedVegetation{
    plantCover =pc,
    dryCoeff = dc,
    wetCoeff = wc,
    rainfallPlantSurvival = rps,
    rainfall = r,
    finalTime = ft,
}

scene:run()
```

Figura 11 - classe de teste aleatorio do modelo Banded Vegetation

4.2) Teste Combinatorial

Devido à complexidade dos produtos de software, testar as combinações possíveis de valores de parâmetros de entrada é impraticável. Mathur (Mathur, 2008) define o conceito de designs combinatoriais como um conjunto de técnicas para a geração de casos de teste através da seleção de um pequeno conjunto de casos de teste, mesmo quando o domínio de entrada e os subdomínios de sua partição são grandes e complexos. Ainda, essa técnica tem sido considerada eficaz na detecção de defeitos devido à interação de várias variáveis. (BALERA, 2014)

Inicialmente, para aplicar o teste de designs combinatoriais será utilizado o método de Pairwise, que é muito utilizado por conseguir minimizar a quantidade de casos de testes em situações onde muitas variáveis são consideradas. Para realizar o teste combinatório foi utilizado assim como no teste anterior (teste aleatório) as 4 variáveis mais relevantes, porem dessa vez, cada variável recebeu apenas 4 possíveis valores entre (0,100), que podem ser visualizados abaixo:

Tabela 3 - Tabela com níveis e fatores do modelo

HealthyLifeSpan	immLifespan	infectedLifespan	reviveTime
1	3	2	5
15	25	35	45
7	18	78	90
13	21	99	0

Em seguida foi utilizado a técnica *pairwise test* para gerar menos casos de teste e abranger todos as combinações possíveis, para realizar esse fato, foi utilizado a ferramenta AllPairs, que dado os valores, executa o pairwase test e retorna o resultado, que pode ser visualizado na tabela abaixo:

Tabela 4 - Resultados do teste de Pairwise

Caso	healthyLifespan	immLifespan	infectedLifespan	reviveTime
1	1	3	2	5
2	1	25	35	45
3	1	18	78	90
4	1	21	99	0
5	15	3	35	90
6	15	25	2	0
7	15	18	99	5
8	15	21	78	45
9	7	3	78	0
10	7	25	99	90
11	7	18	2	45
12	7	21	35	5
13	13	3	99	45
14	13	25	78	5
15	13	18	35	0
16	13	21	2	90

É possível aplicar este teste de diferentes formas, uma delas é utilizando uma environment, que possibilita criar vários cenários e executá-los, a grande desvantagem se dá ao fato de que nem todos os modelos possuem a opção de utilizar environments o que torna essa técnica útil apenas para casos específicos.

```
require "influenza"
env = Environment{

    Caso1 = Influenza{ healthyLifespan = 1, immLifespan = 3, infectedLifespan = 2, reviveTime= 5},
    Caso2 = Influenza{ healthyLifespan = 1, immLifespan = 25, infectedLifespan = 35, reviveTime= 45},
    Caso3 = Influenza{ healthyLifespan = 1, immLifespan = 18, infectedLifespan = 78, reviveTime= 90},
    Caso4 = Influenza{ healthyLifespan = 1, immLifespan = 21, infectedLifespan = 99, reviveTime= 0},
    Caso5 = Influenza{ healthyLifespan = 15, immLifespan = 3, infectedLifespan = 35, reviveTime= 90},
    Caso6 = Influenza{ healthyLifespan = 15, immLifespan = 25, infectedLifespan = 2, reviveTime= 0},
    Caso7 = Influenza{ healthyLifespan = 15, immLifespan = 18, infectedLifespan = 99, reviveTime= 5},
    Caso8 = Influenza{ healthyLifespan = 15, immLifespan = 21, infectedLifespan = 78, reviveTime= 45},
    Caso9 = Influenza{ healthyLifespan = 7, immLifespan = 3, infectedLifespan = 78, reviveTime= 0},
    Caso10 = Influenza{ healthyLifespan = 7, immLifespan = 25, infectedLifespan = 99, reviveTime= 90},
    Caso11 = Influenza{ healthyLifespan = 7, immLifespan = 18, infectedLifespan = 2, reviveTime= 45},
    Caso12 = Influenza{ healthyLifespan = 7, immLifespan = 21, infectedLifespan = 35, reviveTime= 5},
    Caso13 = Influenza{ healthyLifespan = 13, immLifespan = 3, infectedLifespan = 99, reviveTime= 45},
    Caso14 = Influenza{ healthyLifespan = 13, immLifespan = 25, infectedLifespan = 78, reviveTime= 5},
    Caso15 = Influenza{ healthyLifespan = 13, immLifespan = 18, infectedLifespan = 35, reviveTime= 0},
    Caso16 = Influenza{ healthyLifespan = 13, immLifespan = 21, infectedLifespan = 2, reviveTime= 90}
}

env:run()
```

Figura *12* - Modelo de teste utilizando Environment

Outra forma de aplicar o teste e possuir acesso a todos os parâmetros do modelo é utilizar um cenário que armazenara todos os parâmetros do modelo de forma direta sem utilizar uma environment, essa técnica pode ser utilizada em qualquer modelo, o que a torna mais eficiente para ser aplicar de forma geral.

```
require "influenza"

scene1 = Influenza{
    qHealthy = 0,
    qDead = 0,
    qImmune = 0,
    qInfected = 0,
    qInfectious = 0,
    dim = 20,
    finalTime = 50,
    healthyLifespan = 55,
    immLifespan = 55,
    infectedLifespan = 55,
    infectiousLifespan = 64,
    infectiousTime = 20,
    reviveTime = 3,

}
scene1:run()
```

Figura *13* - Modelo de teste utilizando atribuição direta

Posteriormente, para aplicar o teste de design combinatoriais, decidiu-se utilizar o algoritmo T-Tuple Reallocation (TTR) nos modelos de autômato celulares do TerraMe, assim como no teste anterior (teste aleatório), foram utilizadas as variáveis que mais influenciam no comportamento do modelo.

4.2.1) TTR

O T-Tuple Reallocation (TTR) é um algoritmo para a geração de designs combinatoriais através da técnica t-way testing. A construção de cada linha do MCV é feita através da realocação de tuplas afim de que cada caso de teste cubra a maior quantidade de tuplas ainda não cobertas possíveis, através do comprimento de “metas”, que são calculadas através da combinação simples entre o valor do strenght e a quantidade de fatores cobertos pelo caso de teste naquele momento (BALERA,2014).

```
Digite o valor do Strenght:
2
Digite em cada linha o número de níveis de cada fator:

4
4
4
0
1       |         1   1   1   meta: 2
2       |         1   2   2   meta: 2
3       |         1   3   3   meta: 2
4       |         1   4   4   meta: 2
5       |         2   1   2   meta: 2
6       |         2   2   1   meta: 2
7       |         2   3   4   meta: 2
8       |         2   4   3   meta: 2
9       |         3   1   3   meta: 2
10      |         3   2   4   meta: 2
11      |         3   3   1   meta: 2
12      |         3   4   2   meta: 2
13      |         4   1   4   meta: 2
14      |         4   2   3   meta: 2
15      |         4   3   2   meta: 2
16      |         4   4   1   meta: 2
Combinação: [1, 2]
Combinação: [1, 3]
Combinação: [2, 3]
Quantidade de casos de teste final: 16
```

Figura 14- Saída do algoritmo TTR

4.3) Oráculo de Teste

Após o entendimento das técnicas de teste aleatório e combinatorial para gerar os dados de entrada de teste, o mesmo processo utilizado no modelo influenza foi aplicado para os modelos do pacote de autômatos celulares (CA) do TerraME, afim de gerar uma base de conhecimento com os maps e charts obtidos nos testes, para serem tomados como referência afim de se obter um veredito do caso de teste sem necessitar explicitar os resultados esperados.

Para realizar a extração de características dos modelos optou-se por desenvolver um algoritmo em Java que permite extrair os valores RGB de cada pixel presente em um map do Terrame, através do auxilio de um framework chamado javaCV (utilizado como interface para a biblioteca OpenCV). Cada pixel é verificado e caso possua uma tonalidade de cor semelhante com um dos modelos da base de dados, esse pixel é considerado pelo algoritmo como válido.

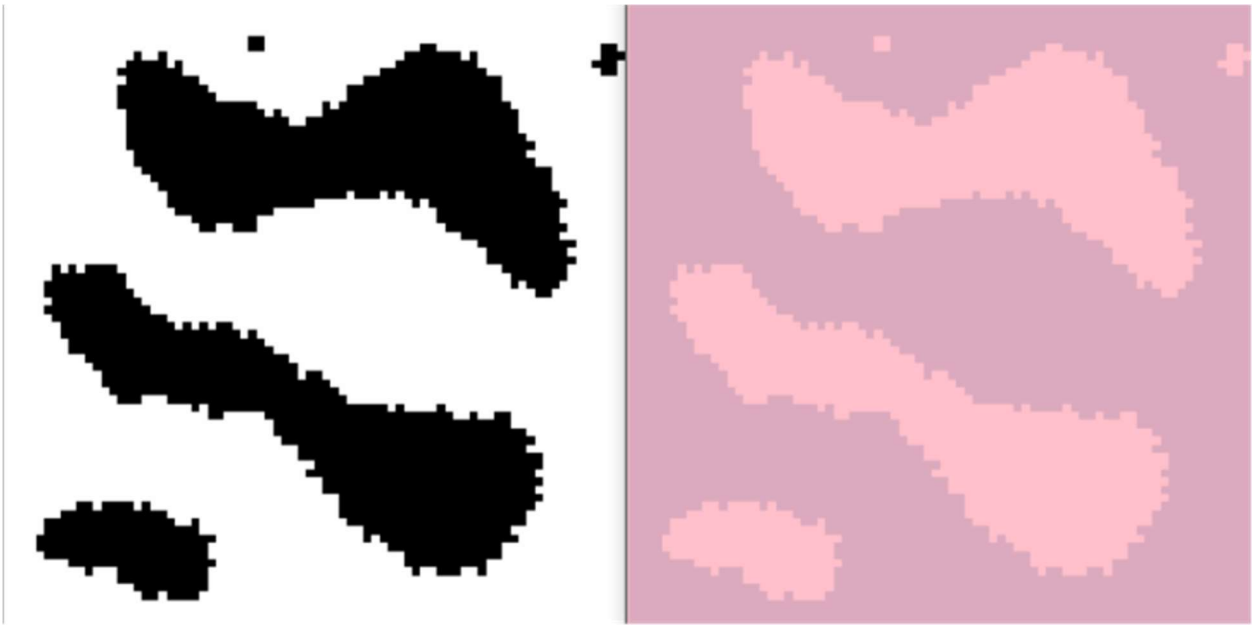

Figura 15 - Modelo Anneal original / modelo extraído

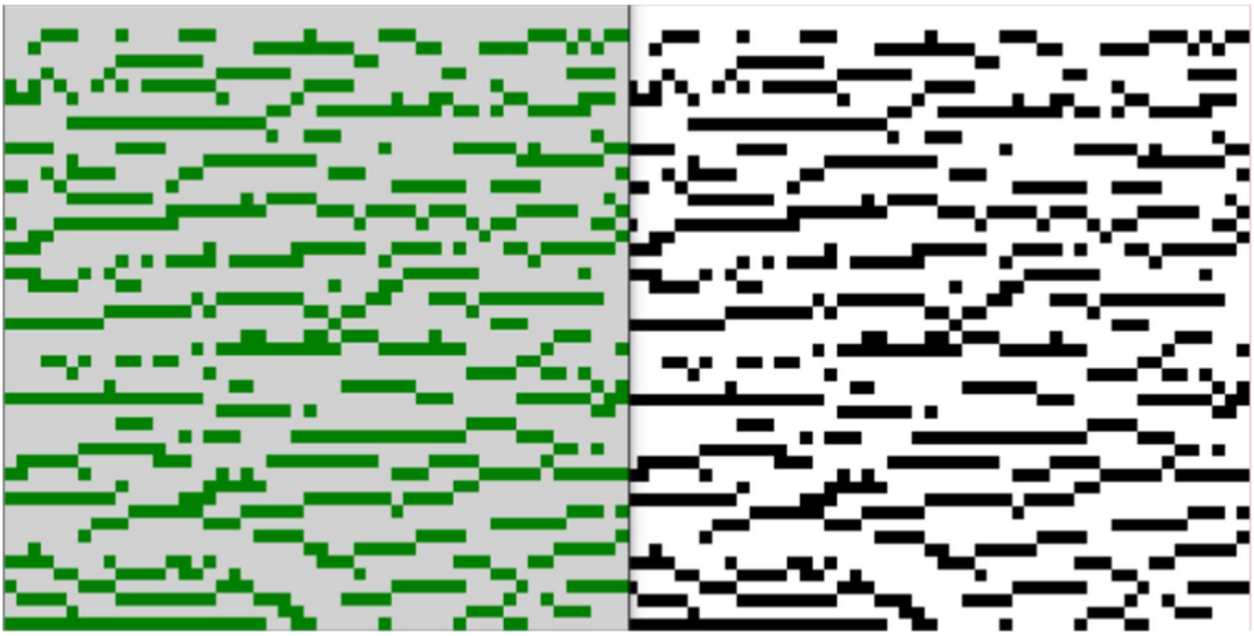

Figura 16 - Modelo Banded Vegetation original / Modelo extraído

As Figuras 15 e 16 mostram respectivamente a esquerda o map do modelo original, e a direita o modelo após a extração dos valores RGB onde cada pixel valido foi pintado de outra cor para que fosse visivel o processo que o algoritmo realiza

Ao realizar o procedimento em todos os maps, os valores RGBs que foram extraidos são enviados para um arquivo .arff afim de gerar um nova base de dados já com valores numericos, para realizar a execução dos casos de testes na atividade 3.

Para obter um possível veredito do caso de teste, foi utilizada a ferramenta WEKA de mineração de dados, para auxiliar no processo de classificação das imagens, utilizando o arquivo .arff que foi criado. Para a manipulação do WEKA foi desenvolvido um algoritmo em java que recebe um nova imagem e realiza a extração RGB para comparar com os dados que já estão na base de dados, através de tecnicas como Naive Bayes e arvore de decisão.

4.3.1) Naive Bayes

O algoritmo Naive Bayes é um classificador probabilístico, que estima a probabilidade de um dado, atraves de sua frequencia em que aparece em um determinado conjunto de caracteristicas dentro de um ambiente de treino, quando um novo dado é inserido, o classificador calcula qual a probabilidade dele pertencer a uma classe na base de dados, segundo as caracteristicas de cada classe.

4.3.2) Arvore de decisão

O algoritmo J48 é utilizado para criar arvores de decisão, onde a raiz da arvore ou seja o topo, é onde se localiza o atributo que possui maior significancia em relação aos outros, em seguida os outros nós da arvore recebem os proximos atributos que contem a maior significancia sucessivamente.

Após os testes realizados com essas tecnicas verificou-se que somente a extração de cores não demonstra um resultado robusto, visto que a cor não é um fator determinante de um modelo, já que um mesmo modelo pode possuir varias cores para seus atributos. Assim, é necessario investigar novas tecnicas de extração de caracteristicas de imagens afim de se obter um resultado esperado de forma eficiente.

6.)ATIVIDADE 5: ANÁLISE ESTÁTICA DE CÓDIGO-FONTE

Para realizar a análise estática foi utilizado a ferramenta LuaCheck (versão 0.18.0), onde todos os modelos dos pacotes de autômatos celulares (CA) e modelos baseados em agentes, foram executados individualmente baseados na versão 2.0-BETA-5 no TerraMe. Optou-se por utilizar uma versão separada do luacheck, pois a versão que se encontra integrada ao TerraMe, não conseguiu executar a análise dos modelos.

Ao realizar uma análise estática utilizando o luacheck em um modelo do TerraME é provável que seja gerado warnings que possivelmente não devem ser considerados validos, isso ocorre devido ao luacheck analisar o código do modelo baseando-se nos padrões de um algoritmo de linguagem lua normal, e não utilizando as nomenclaturas e padrões do TerraME que são desconhecidas pelo analisador, porém ao executar os modelos utilizando o compilador do TerraMe, os mesmos funcionaram corretamente.

- **setting non-standard global variable**

  Por padrão uma variável que não é explicitamente declarada como uma variável local, é considerada uma variável global, o luacheck guarda todas essas variáveis globais em uma lista e ao encontrar uma variável fora da listagem produz um warning.

  Em modelos do TerraMe, este warning ocorre principalmente quando é utilizado a nomenclatura Model para se inicializar um modelo TerraMe, onde o luacheck considera que o nome do modelo não foi declarado anteriormente, portanto, não é definida como local ou global, o mesmo também ocorre em funções como *countNeighbors, forEachNeighbor,* Porém sua utilização se encontra correta dentro dos padrões de um modelo do TerraME.

  Ex: Influenza = Model {}

  luacheck considera ‘influenza’ uma variável global não inicializada por padrão.

- **accessing undefined variable**

  Ao utilizar os diversos componentes do TerraMe como Model, Map, Timer, CellularSpace, Chart, Choice, etc, o luacheck ocasiona um warning pois considera que essas nomenclaturas são variáveis que estão sendo utilizadas sem terem sido definidas e inicializadas, espera-se que as mesmas sejam inicializas no escopo do modelo, porém esse warning pode ser considerado invalido, já que as mesmas não são variáveis comuns, mas tipos de funções do TerraME, e sua utilização esta correta.

  Ex: Influenza = Model {}

  Luacheck considera ‘Model’ como uma variável que não foi definida anteriormente e nem inicializada com um valor, por padrão variáveis são criadas com o valor nil.

Warings corretos:

Alguns warnings são considerados validos e podem ser corrigidos, como problemas com espaçamento de código e de tabulação desnecessária, mas considerasse muitas vezes que estes espaços são utilizados para tornar o código mais visível e organizado aos olhos do programador.

- **line contains trailing whitespace**

  Este warning pode ser tratado, removendo espaços desnecessário que forma criados em um linha de código utilizando a tecla ‘espaço’.

- **line contains only whitespace**

  Este waring pode ser tratado, removendo as quebras de linhas desnecessárias usadas dentro do código do modelo.

- **inconsistent indentation (SPACE followed by TAB)**

  Este warning ocorre quando é pressionado a tecla TAB no final de uma linha, e pode ser tratado removendo essa tabulação desnecessária.

A analise estática mostrou-se eficiente para detectar erros nos modelos de Autômatos celulares e baseados em agentes do TerraME, melhorando as sintaxe dos modelos, bem como reduzindo linhas de código com espaçamento e tabulação desnecessárias, alem de revelar supostos warnings que na verdade são trechos de códigos que utilizam a sintaxe do TerraMe, e que podem ser desconsiderados, não são realmente warnings considerados validos.

7.) CONCLUSÃO

Esse relatório apresentou as atividades desenvolvidas no período de 01 de agosto de 2016 a 13 de julho de 2017 relacionadas ao projeto “Teste de modelos ambientais desenvolvidos via TerraME”. Os objetivos previstos para o período a que se refere esse relatório foram alcançados satisfatoriamente, inclusive concluindo totalmente a atividade 5, que não estava prevista para tal período. Uma nova metodologia foi proposta, DaOBML, para identificar defeitos em modelos ambientais desenvolvidos via TerraME. A metodologia está sendo implementada com apoio do framework javaCV e do ambiente de mineração de dados WEKA.

8.) REFERÊNCIAS

[Carneiro 2006] CARNEIRO, T. G. S. Nested-ca: a foundation for multiscale modelling of land use and land cover change. 2006. 114 p. (INPE-14702-TDI/1227). Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2006. Available from: <http://urlib.net/sid.inpe.br/mtc-m17@80/2007/01.03.11.57>. Access in: 2016, June 21.

[Câmara et al. 2000] CÂMARA, G.; SOUZA, R. C. M.; PEDROSA, B. M.; VINHAS, L.; MONTEIRO, A. M. V.; PAIVA, J. A.; CARVALHO, M. T.; GATTASS, M. TerraLib: Technology in support of GIS innovation. In: WORKSHOP BRASILEIRO DE GEO-INFORMÁTICA, 2., 2000, São Paulo. Anais... São José dos Campos: INPE, 2000. p. 126-133.

[Ierusalimschy et al. 1996] IERUSALIMSCHY, R.;FIGUEIREDO, L. H.; FILHO, W. C. Lua — An Extensible Extension Language. Software: Practice and Experience, v. 26, n. 6, p. 635-652, 1996.

[Balera e Santiago Júnior 2015] BALERA, J. M.; SANTIAGO JÚNIOR, V. A. T-tuple reallocation: An algorithm to create mixed-level covering arrays to support software test case generation. In O. Gervasi,
B. Murgante, S. Misra, M. L. Gavrilova, A. M. A. Rocha, C. Torre, D. Taniar, and B. O. Apduhan, editors, Computational Science and Its Applications - ICCSA 2015, volume 9158 of Lecture Notes in Computer Science (LNCS), p. 503-517. Springer International Publishing, 2015.

[Mathur 2008] MATHUR, A. P. Foundations of software testing. Dorling Kindersley (India), Pearson
Education in South Asia, Delhi, India, 2008. 689 p.

[Utting e Legeard 2007] UTTING, M.; LEGEARD, B. Practical Model-Based Testing: A tools Approach. Waltham, MA, USA: Morgan Kaufmann Publishers, 2007.

[Santiago Júnior 2011] SANTIAGO JÚNIOR, V. A. SOLIMVA: A methodology for generating model-based test cases from natural language requirements and detecting incompleteness in software specifications. 2011. 264 p. Thesis (Doctorate at Post Graduation Course in Applied Computing) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brazil, 2011. Avaliable from: <http://urlib.net/8JMKD3MGP7W/3AP764B>. Access in: 2016, June 21.

[Santiago Júnior e Vijaykumar 2012] SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L. Generating model-based test cases from natural language requirements for space application software. Software Quality Journal, v. 20, n. 1, p. 77-143, 2012. DOI: 10.1007/s11219-011-9155-6.

[Santiago Júnior et al. 2012] SANTIAGO JÚNIOR, V. A.; VIJAYKUMAR, N. L.; FERREIRA, E.; GUIMARÃES, D.; COSTA, R. C. GTSC: Automated Model-Based Test Case Generation from Statecharts and Finite State Machines. In: Sessão de Ferramentas do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012, Natal-RN. Anais do III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2012. p. 25-30.

[Anand et al. 2013] ANAND, S.; BURKE, E. K.; CHEN, T. Y.; CLARK, J.; COHEN, M. B.; GRIESKAMP, W.; HARMAN, M.; HARROLD, M. J.; McMINN, P.; BERTOLINO, A.;

LI, J. J.; ZHU, H. An orchestrated survey of methodologies for automated software test case generation, The Journal of Systems and Software, Volume 86, Issue 8, 2013, p. 1978-2001.

[Delamaro et al. 2007] DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Introdução ao teste de software. Campus-Elsevier, 2007. 408 p.

[Endo e Simão 2013] ENDO, A. T.; SIMAO, A. S. . Evaluating Test Suite Characteristics, Cost, and Effectiveness of FSM-based Testing Methods. Information and Software Technology, v. 55, p. 1045-1062, 2013.

[Baier e Katoen 2008] BAIER, C.; KATOEN, J.-P. Principles of model checking. Cambridge, MA,
USA: The MIT Press, 2008.

[Clarke e Emerson 2008] CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization
skeletons using branching time temporal logic. In: GRUMBERG, O.; VEITH, H. (Ed.). 25 years of model checking. Berlin/Heidelberg, Germany: Springer Berlin/Heidelberg, 2008. v. 5000, p. 196-215. Lecture Notes in Computer Science (LNCS).

[Fraser et al. 2009] FRASER, G.; WOTAWA, F.; AMMANN, P. E. Testing with model checkers: a
survey. Software Testing, Verification and Reliability, v. 19, p. 215–261, 2009.

[Zannier et al. 2006] ZANNIER, C.; MELNIK, G.; MAURER, F. On the success of empirical studies in the international conference on software engineering. In Proceedings of the 28th International
Conference on Software Engineering, ICSE '06, p. 341-350, New York, NY, USA, 2006. ACM.

[Lemos et al. 2013] LEMOS, O. A. L.; FERRARI, F. C.; ELER, M. M.; MALDONADO, J. C.; MASIERO, P. C. Evaluation studies of software testing research in Brazil and in the World: A survey of two premier software engineering conferences. J. Syst. Softw., 86(4):951-969, April 2013.

[Chess e West 2007] CHESS, B.; WEST, J. Secure Programming with Static Analysis. Addison-Wesley Professional, 2007, 624 p.

[BALERA, 2017] BALERA, Juliana Marino. T-TUPLE REALLOCATION: UM ALGORITMO PARA CRIAR MATRIZES DE COBERTURA COM NÍVEIS VARIADOS PARA APOIAR A GERAÇÃO DE CASOS DE TESTE DE SOFTWARE. 2014. 50f. Trabalho de Graduação FATEC de São José dos Campos: Professor Jessen Vidal.

[Beauchemin et al. 2005] Catherine Beauchemin; John Samuel; Jack Tuszynski. A simple cellular automaton model for influenza A viral infections. Journal of Theoretical Biology, v. 1, n. 232, p. 223–234, 2005.

[Lima 2017] Lima, Ciro Grippi Barbosa, 2013. PROCEDIMENTO PARA INTRODUÇÃO DE AGILIDADE EM TESTES DE
SOFTWARE.

[RIOS 2017] RIOS, Emerson, 2013. Clinica sobre conceitos básicos de teste de software.

[Unicamp 2014]Unicamp, 2014. Testes caixa preta. Disponível em: <http://www.ic.unicamp.br/~meidanis/courses/mc626/2014s1/materiais/slides/Aula13-Testes-caixa-preta-combinatoria.pdf> acessado em: 02 jan. 2017.