LEONARDO ANDREATTA DE ALCANTARA

DESENVOLVIMENTO DE UM MODELO DE SIMULAÇÃO SOCIAL

Monografia de especialização apresentada
ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Especialista em Tecnologia Java".

Orientador: Prof. Dr. Gustavo Alberto Giménez Lugo

CURITIBA

2011

TERMO DE APROVAÇÃO

Título de Monografia

Desenvolvimento de um modelo de simulação social

por

Leonardo Andreatta de Alcantara

Esta monografia foi apresentada às 18:30 do dia 01 de Setembro de 2011 como requisito parcial para a obtenção do título de ESPECIALISTA EM TECNOLOGIA JAVA, Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho _____

(aprovado, aprovado com restrições, ou reprovado)

_____
Prof. Gustavo Alberto Giménez Lugo, Dr.
Universidade Tecnológica Federal do Paraná

_____
Profª. Marília Abrahão Amaral, Drª.
Universidade Tecnológica Federal do Paraná

Visto da coordenação:

_____
Prof. João Alberto Fabro, Dr.
Universidade Tecnológica Federal do Paraná

RESUMO

ALCANTARA, Leonardo A. Desenvolvimento de um modelo de simulação social. 2011. 53 f. Monografia (Especialização em Tecnologia Java) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

Simulação social é uma ferramenta poderosa no auxílio à gerência municipal, estadual e federal para análise de crescimento populacional. Esta monografia apresenta o desenvolvimento de um modelo de simulação social, baseada em agentes, utilizando a tecnologia Java. Discute, brevemente, os conceitos envolvidos em simulação social e como a utilização de agentes de software fornece flexibilidade e robustez ao desenvolvimento de modelos. Apresenta ferramentas, baseadas em Java, que possibilitam a criação de modelos utilizando agentes de software e discute as diferenças entre as mesmas.

Palavras-chave: Simulação Social, Agentes de Software, NetLogo, Repast J, Repast Symphony.

ABSTRACT

ALCANTARA, Leonardo A. Development of a Social Simulation model.. 53 p. Monografia (Especialização em Tecnologia Java) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

Social simulation is a powerful tool to assist municipal, state and federal management in overseeing the population growth. This document presents the development of a social simulation model, based on software agents, utilizing Java technology. Discusses briefly the concepts involved in social simulation and how the usage of software agents provide flexibility and sturdiness to the model development. Presents tools, based on Java, which allows the creation of the models and also discusses the differences between the tools.

Keywords: Social Simulation, Software Agents, NetLogo, Repast J, Repast Symphony.

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

MIT – Massachusetts Institute of Technology

API – Application Programming Interface

2D – Two Dimensions

3D – Three Dimensions

IDE - Integrated Development Environment

URL - Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

1.1 Tema

O desenvolvimento urbano de um município é um processo complexo que leva em consideração fatores sociais, políticos e econômicos. O planejamento urbano realizado pela gerência do município é um processo importante, a longo prazo, por definir tanto quantativamente quanto qualitativamente os investimentos a serem feitos. Uma ferramenta capaz de prover estimativas sobre o crescimento do município, a longo prazo, é de extrema valia para o processo de tomada de decisão no planejamento urbano.

A utilização de GeoProcessamento como base de tal ferramenta, de acordo com (Godoy, 2004), é baseada numa visão estatística do mundo e elaboradas sobre suposições pré-estabelecidas. Devido a isto um modelo baseado em GeoProcessamento será intrisicamente limitado a um conjunto restrito de cenários capazes de serem simulados. Uma abordagem dinâmica para modelos de crescimento urbano permite flexibilidade e possibilita novas representações dos dados e cenários.

A adoção de agentes de software no desenvolvimento de modelos de simulação social proporciona dinamismo ao mesmo. O uso dessa abordagem permite interatividade entre as entidades e com o ambiente em que elas se encontram. Com a utilização de parâmetros no modelo é possível incluir e alterar os dados do modelo sem que o mesmo seja alterado.

1.2 Objeto

Criar um modelo de simulação social baseado no munícipio de Mandirituba, Paraná, Brasil. Este modelo será implementado utilizando ferramentas baseadas em Java e utilizará dados falsos para sua execução. Duas implementações serão criadas em ferramentas distintas de maneira a demonstrar as ferramentas.

1.3 Objetivo Geral

Apresentar como a utilização de agentes de software no desenvolvimento de modelos de simulação social introduzem maior flexibilidade aos mesmos. Ao mesmo tempo serão apresentadas ferramentas que utilizam tecnologia Java para permitir a implementação de agentes de software e demonstrar algumas das diferenças entre elas.

1.4 Objetivos Específicos

- Demonstrar a utilização de agentes de software no desenvolvimento de modelos de simulação social.
- Apresentar ferramentas baseadas em tecnologia Java que permitam a implementação de agentes de software.
- Comparar as implementações desenvolvidas entre si de maneira a ser possível visualizar as diferenças entre cada ferramenta.

1.5 Justificativa

Realizar um estudo sobre as ferramentas, baseadas em Java, disponíveis para a implementação de agentes de software.

1.6 Metodologia Empregada

Inicialmente foram definidas algumas diretivas básicas em relação ao modelo a ser desenvolvido. Estas diretivas se referem ao ambiente em que os agentes serão inseridos (no

caso, o município de Mandirituba), os agentes em si (os diferentes tipos) e o comportamento geral dos mesmos. Com base nestas regras foram feitas duas implementações: uma na ferramenta NetLogo e a outra utilizando o framework Repast J. Ambas as implementações usaram da melhor maneira possível as funcionalidades disponíveis em cada ferramenta. Dado isso ambas as implementações do modelo foram executadas e os resultados registrados e comparados.

2 Modelos

Modelagem se refere ao processo da pesquisa e criação de modelos, ou seja, de representações de um dado sistema, segundo Godoy (2004). Este processo se desenvolve pela definição de um conjunto de hipóteses e regras as quais podem ser comparadas ao sistema real. Ao se criar um modelo é possível reduzir o escopo do estudo do mundo real de maneira a se focar especificamente em um aspecto do mesmo. Modelos são de particular interesse quando não se é possível, por qualquer razão, realizar um estudo direto do sistema real. Um modelo somente pode ser aceito, rejeitado ou modificado após a comparação entre o resultado gerado e o observado, para então o modelo ser reavaliado (Soares-filho, 1998).

Neste estudo o sistema a ser modelado é a distribuição populacional de um município ao longo do tempo. Devido a isso é necessário entender o princípio teórico de modelos espaço-temporais e mais profundamente a definição teórica de espaço e tempo, segundo Couclelis.

> "Espaço absoluto, também chamado Cartesiano ou Newtoniano, é um *container* de coisas e eventos, uma estrutura para localizar pontos, trajetórias e objetos. Espaço relativo, ou Leibnitziano, é o espaço constituído pelas relações espaciais entre coisas e eventos" (Couclelis, 1997)

Segundo Santos (1996) espaço absoluto se define como "espaço dos fixos" enquanto espaço relativo é considerado "espaço dos fluxos". Estes conceitos seriam traduzidos para representações computacionais como representações de recobrimentos planares e representações associadas à conectividade (grafos), segundo Godoy (2004).

O tempo, conceitualmente, pode ser representado por diferentes estruturas definidas principalmente por três aspectos de representação temporal. Estes três aspectos são granularidade, variação e ordem no tempo. Segundo Godoy (2004), associado ao conceito de

variação temporal discreta existe o conceito de Chronos. Um *chronon* se definiria então como a menor duração de tempo suportada por um sistema.

A granularidade temporal de um sistema, de acordo com Godoy (2004), está diretamente relacionada com a duração de um *chronon* e as diferentes granularidades de um sistema temporal conduzem a definição de instante e intervalo de tempo. A ordem temporal refere-se ao modo como o tempo flui (Pedrosa e Câmera, 2002).

Considerando os conceitos apresentados temos a seguinte definição de modelos espaço-temporais, segundo Godoy (2004, p. 11).

> O objetivo dos modelos espaço-temporais é a simulação numérica de processos do mundo real em que os estados do modelo se modificam ao longo do tempo e em função de diversas condições de entrada. Os modelos de SIG Dinâmico descrevem a evolução de padrões especiais de um sistema ao longo do tempo (Pedrosa e Câmara, 2002).

2.1 Modelos Dinâmicos

A enfâse dada pela aplicação de tecnologia de GeoProcessamento, em modelos, é a representação de fenômenos espaciais de forma estática, ou seja, através da elaboração de mapas, imagens e outros. Porém, conforme observado por Godoy (2004), fenômenos espaciais como escoamento de água, planejamento urbano e mudanças de solo são dinâmicos e o uso de representações estáticas não os representam adequadamente.

Desta maneira o desenvolvimento de técnicas e abstrações que tenham capacidade de representar, adequadamente, os fenômenos espaço-temporais, são um dos grandes desafios da Ciência da Informação Espacial (Câmara e Monteiro, 2003).

A elaboração de modelos dinâmicos visam prover uma solução a estes desafios. “Tais modelos têm como principal função realizar a simulação matemática de processos identificados no mundo real onde se observa a mudança de variáveis em consequência a variações em suas forças direcionadas” (Godoy, 2004).

De maneira breve serão abordadas nesse estudo a definição de modelos dinâmicos de análise de mudança. Estes modelos ajudam a compreender a organização do espaço geográfica, ou paisagem, sendo chamados de modelo de paisagem.

Modelos de paisagem são usadas quando se demonstra necessário descrever fenômenos espaciais e predizer a evolução temporal de seus padrões associado a escalas temporais e espaciais, como em análises solo-geomorfológicas (Campos e Marques Júnior, 2006). Godoy (2004), também indica que modelos de paisagem são potencialmente favoráveis

ao mapeamento de fluxo de energia, produzir sucessões em 2D e 3D, designar áreas fontes de absorção e recepção entre outros.

De acordo com Baker (1989), modelos de paisagem podem ser classificados seguindo alguns critérios, entre eles os mais importantes são: nível de agregação, estrutura e tipo de matemática utilizada. Considerando isso pode-se classificar os modelos de paisagem como:

- Modelos totais de paisagem;
- Modelos distribucionais;
- Modelos espaciais de paisagem;

O detalhamento de cada classificação dos modelos foi definida por Godoy (2004, p. 15) como:

> Os modelos totais de paisagem calculam os valores de uma variável ou de um grupo de variáveis em um contexto geral de uma determinada paisagem. Os modelos distribucionais, como o nome sugere, trabalham com a distribuição ou proporção de valores de variáveis em uma paisagem. Os modelos espaciais são os mais complexos e detalhados. Eles têm a capacidade de modelar o destino da configuração e divisão da paisagem. É este tipo de modelo que gera os mapas das mudanças espaciais (Soares-Filho, 1998).

Sendo parte do objetivo a geração de mapas das mudanças espaciais é necessário trabalhar com o desenvolvimento do modelo espacial da mudança da paisagem. Este trabalho envolve a escolha do formato dos dados, vetor ou *raster*, variáveis de inclusão, tamanho do pixel ou resolução vetorial, e finalmente o algoritmo de mudança (Soares-Filho, 1998).

Segundo Baker (1989), uma maneira adequada de pensar sobre a modelagem de mudanças consiste em imaginar uma paisagem que tem um grande número de elementos de paisagem sobrepostos em uma grade, ou seja, a representação *raster*. Considerando isso os elementos ou estados presentes dentro de uma paisagem podem ser representados por pixels (células de valores iguais numa cadeia de informações únicas). Segundo Godoy (2004), associado à uma representação gráfica de pixel existem vários parâmetros como a escolha do tamanho da célula ou pixel em função da escala de observação e a cor utilizada para a representação.

A partir dos mapas gerados pela mudança pode-se então elaborar as simulações. Uma simulação consiste em modelar a dinâmica de um sistema e assim reproduzir, em ambiente computacional, a complexidade do mecanismo em desenvolvimento, que funciona através da troca de materiais, energia, informações e espécies/estados entre os elementos ou componentes do sistema, segundo Godoy (2004 apud Soares-Filho et al., 2003). Neste estudo o processo de simulação a ser utilizado será a Simulação Social Baseada em Agentes (*Agent*

Based Social Simulation) e mais especificamente o sistema de Modelo Baseado em Agentes (*Agent Based Modeling*).

3 Simulação Social Baseada em Agentes e Modelo Baseado em Agentes

Um agente, em um contexto de ciências de computação, é um *software* o qual age em nome de uma entidade numa relação de ação, ou seja, em um "acordo" para agir em nome da entidade. Este comportamento de um agente implica que o mesmo possui a capacidade de decidir qual ação é apropriada dado o contexto e situação onde o agente está inserido (Wikipedia, 2011).

Não obstante o termo "agente" é uma abstração (idéia ou conceito) em termos de *software* como outras abstrações existentes, por exemplo Programação Orientada a Objetos (*Object Oriented Programming*). A utilização do conceito de que um componente, ou parte, de *software* pode realizar ações baseadas em suas próprias regras e assim reagir a uma outra entidade, ou o meio onde está inserido, permite diferentes visões sobre um problema. Este mesmo conceito também permite a descrição de uma entidade de *software* complexa que pode, com certa autonomia, realizar ações representando uma outra entidade, ou seja, simulando-a.

Várias definições podem ser atribuídas a agentes, sendo que Davidsson (2000), provê um resumo sobre as características gerais que podem ser atribuídas aos agentes como:

- pró-atividade: desde ações puramente reativas até entidades completamente autônomas para decidir seu comportamento
- linguagem de comunicação: desde a completa falta de comunicação entre agentes, sinalização simples entre agentes até uma comunicação completa
- clareza espacial: desde a falta total de noção espacial até a especificação precisa da sua locação em um espaço geométrico simulado.
- mobilidade: desde a uma completa inabilidade de movimentação por parte do agente até a capacidade de locomoção dentro do espaço geométrico simulado
- adaptatividade: desde entidades estáticas, neste caso significando entidades que não apresentam evolução em seu comportamento, até capacidade evolutiva autônoma

- conceitos de modelagem: desde conceitos tradicionais de modelagem até conceitos mentalistas como crenças, desejos e intenções.

De acordo com Davidsson (2000), a simulação baseada em agentes não é um paradigma de simulação completamente novo e original e é influenciada, e em certa parte extende, outros paradigmas existentes. Entre estes paradigmas temos a simulação paralela e distribuída de eventos discretos, simulação orientada a objetos e micro simulação dinâmica.

Ao considerar os atributos de um agentes temos que em um extremo se assemelha com a idéia de uma modelagem estática em que não existe, ou quase não existe, a possibilidade de mudança do que é especificado. No outro extremo, no entanto, é possível visualizar semelhanças com modelos dinâmicos onde, por consequência dos atributos em si, a mudança é inerente e intrínsica ao modelo em si e as entidades que os permeiam.

A exploração e escolha de quais atributos são especificados, e também o quão desenvolvidos eles são, para um agente depende diretamente de quais entidades serão simuladas e em qual cenário ou contexto (Davidsson, 2000). A Simulação Social está inclusa nesse contexto por utilizar a computação baseada em agentes para desenvolver e teorizar fenômenos sociais.

O modelo neste caso visa tratar sobre estes fenômenos sociais de um ponto de vista computacional de maneira a prover a simulação dos mesmos. Tais fenômenos, e suas consequências, podem não ser tão facilmente observados no mundo real. A possibilidade de sua simulação em um ambiente computacional, assim como repetidas execuções de maneira a se poder estudar vários cenários, provê informações valiosas sobre estes fenômenos.

4 Crescimento urbano e segregação espacial

Conforme mencionado a Simulação Social é utilizada como uma ferramenta computacional que facilita o estudo de fenômenos sociais. Uma das aplicações possíveis dessa ferramenta é a simulação de ocupação e crescimento de uma cidade pela sua população ao longo do tempo.

A medida que uma cidade é ocupada é possível perceber certos padrões na organização espacial e uma das causas para tal evento é a segregação espacial. A segregação espacial ocorre em áreas onde existe uma forte homegenidade social interna assim como uma

forte disparidade social, de acordo com (Maretto et al, 2010). As diferenças sociais entre estas áreas são decorrência, principalmente, da habilidade de cada grupo de pagar por sua residência e, segundo (Castells, 1983), esta segregação residencial acaba por ser um produto da existência de classes sociais diferentes.

De acordo com (Mareto et al, 2010) a segregação social é um fenômeno espacial relacionado com a renda e diferenças sociais entre as grupos, ou classes, sociais existentes. A existência destas classes acaba por refletir na distribuição espacial dos grupos encontrados na população. Áreas uniformes são reflexo da distribuição de renda da população assim como o tipo da residência e sua localização são decorrência da acessibilidade a recursos públicos (por exemplo: hospitais, escolas, comércio e outros) de acordo com (Harvey, 1975) e (Sabatini, Cáceres e Gerda, 2001).

Esta separação social, conforme mencionado, não ocorre de maneira aleatória e sim seguindo um padrão. Esta separação, de acordo com (Maretto et al, 2010), exibe várias características de sistemas adaptativos complexos, particulamente de sistemas de emergência e sistemas não lineares. Apesar da ocupação urbana ser um fenômeno social de larga escala ela é resultante das interações entre indivíduos em um nível local, de acordo com (Epstein e Axtell, 1996) e (Schelling, 1978).

O crescimento e ocupação urbana possui uma dinâmica espaço-temporal onde uma área é ocupada, a partir de um ponto no tempo, por um grupo social e pode, posteriormente, ser ocupada por outro grupo. Isto pode ser considerado como um processo de renovação urbana, de acordo com (Castells, 1974).

Modelos esquemáticos de estrutura urbana interna são, em estudos urbanos, tentativas de sumarizar em diagramas as características morfológicas e sociais de uma cidade, inclusive da segregação urbana, de acordo com (Barros, 2004). Existem três padrões, também referenciados como modelos, de ocupação urbana nomeados de acordo com seus elaboraradores. Estes modelos são chamados Kohl, Burgess e Hoyt. De acordo com (Maretto et al, 2010) apesar destes modelos clássicos serem criticados eles são considerados como possibilidades teóricas e não como situações totalmente reais.

O modelo de Kohl, criado em 1841 pelo geógrafo alemão J. G. Kohl, é possivelmente o primeiro modelo de segregação urbana desenvolvido, de acordo com (Sjoberg, 1960). Este modelo foi desenvolvido para descrever as sociedades pré-industriais européias do século 19. De acordo com Kohl, o centro urbano é ocupado pela grupo social mais rico já que, devido a dificuldades de locomoção da época, se encontra mais perto das instituições urbanas como serviços governamentais, a igreja, instituições financeiras e outras.

A periferia ficava então destinada a grupos sociais menos privilegiados por estes não terem condições de adquirir residências privelegiadas no centro.

Existem vários exemplos reais de tal alocação dos grupos sociais como cidades africanas em períodos coloniais, Moscou no final do século 19 e mais recentemente cidades americanas antes da guerra civil e algumas cidades na américa latina hoje em dia, segundo (Barros, 2004) e (Corrêa, 1995).

O modelo de Burgess, criado em 1924 pelo socialista Ernest Burgess, foi baseado nas cidades americanas de 1920. Este modelo determina que o centro da cidade é ocupada por grupos sociais menos favorecidos financeiramente enquanto que a periferia, ou subúrbio, é ocupada por grupos sociais favorecidos financeiramente. O modelo de Burgess se preocupa com o crescimento da cidade, no que é referenciado como "metobolismo urbano", sugerindo que o processo de expansão da população pode ser represtado por 5 círculos concêntricos.

Cada círculo possui uma idade, ou data de ocupação, e características diferentes. De acordo com (Maretto et al, 2010) os cincos círculos são: uma zona central, uma zona de transição com diferentes utilizações da terra onde predominam residências deterioradas; uma zona residencial da classe trabalhadora que conseguiu emergir da zona deteriorada; uma zona residencial com habitações melhores com residências exclusivas e apartamentos de alta classe; e a zona suburbana e comunidades satélites.

Para Burgess, a partir do começo da industrialização o grupo social mais favorecido gradualmente abandona as suas residências localizadas no centro devido ao acréscimo da poluição e a violência. Operários, imigrantes e outras pessoas pertencentes a grupos sociais menos favorecidos então migram para o centro devido a baixa nos preços das moradias e para ficarem mais perto de oportunidades de emprego.

Esta tendência de crescimento da cidade acontecia das zonas internas em direção às zonas externas do modelo de Burgess é referenciada como 'sucessão – invasão', de acordo com (Castells, 1974). Isto demonstra como Burgess concebeu seu modelo como sendo um modelo de crescimento, levando em consideração um processo dinâmico que se desenvolve temporalmente.

Conforme demonstrado por Kohl em seu modelo, as sociedades européias pré-industriais tem o centro como a zona mais importante enquanto que Burgess dita o contrário. Neste quesito pode-se considerar que o modelo de Kohl é o inverso do modelo de Burgess, conforme demostrado na Figura 1.

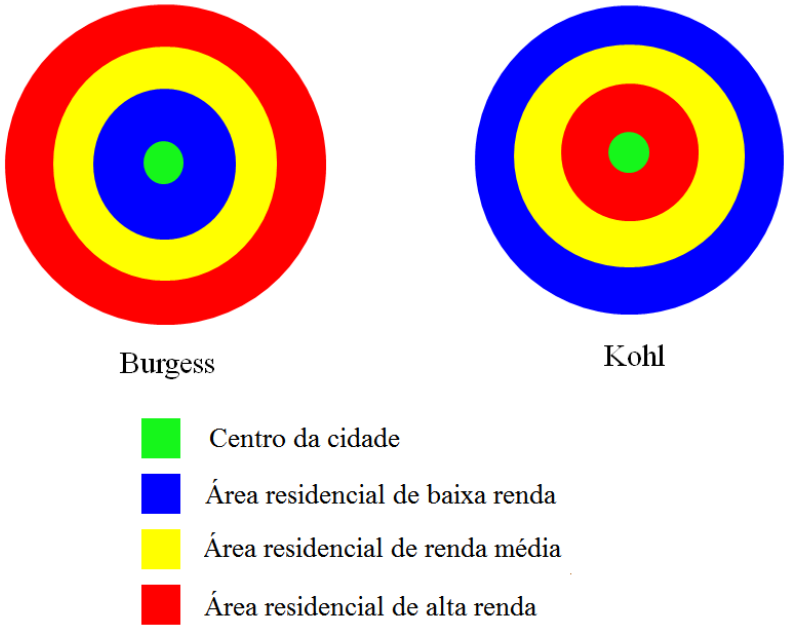

Figura 1 - Diagrama dos modelos de Kohl e Burgess

O modelo de Hoyt, desenvolvido pelo economista americano Homer Hoyt em 1939, é uma variação do modelo de zonas concêntricas. Apesar de manter como base algumas características do modelo de zonas concêntricas, como por exemplo a existência de uma zona central de negócios, Hoyt sugere algumas mudanças. Uma delas é que o crescimento da cidade a partir do centro é feito em torno de certas amenidades como estradas, avenidas e outras artérias de transporte.

Dessa maneira a ordem de ocupação dos grupos sociais mais favorecidos economicamente segue uma lógica previsível porque normalmente ela ocorrerá nas áreas com as maiores rotas de tráfego. Estas áreas, de acordo com Hoyt, invariavelmente terão a melhor infra-estrutura e também amenidades sociais, sejam naturais ou criadas. Os grupos sociais que poderiam ser considerado média classe se agrupam em torno das áreas onde os grupos mais favorecidos se estabelecem sendo destinadas aos grupos menos favoráveis o restante das áreas.

De acordo com (Maretto et al, 2010) a lógica do modelo de Hoyt se baseia numa tendência auto-segregatória dos grupos mais favorecidos em torno de um eixo de deslocamento, normalmente relacionado a um meio de transporte. Este deslocamento é realizado a partir da zona central e então atravessa as melhores áreas da cidade. Dado isso os grupos mais favorecidos podem exercer controle sobre estas áreas privilegiadas e a partir dessa ação os outros grupos realizarão a ocupação do restante dos setores da cidade.

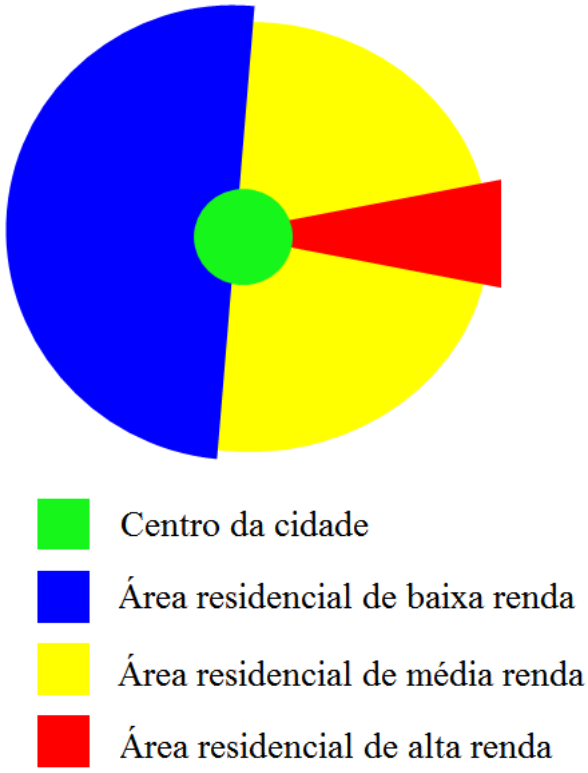

Figura 2 - Diagrama do modelo de Hoyt

5 Descrição do município utilizado no modelo

O modelo desenvolvido neste estudo se baseará, em parte, no município de Mandirituba, Paraná. Este município foi escolhido devido ao acesso a algumas informações sobre ele que foram disponibilizadas pela gerência municipal. Apesar de nem todas as informações disponibilizadas terem sido utilizadas no modelo elas foram utilizadas para se entender o perfil da população do município.

Dados do último censo do município apresentam uma população total de 22.235 habitantes, divididos em 11.338 homens e 10.897 mulheres, que residem em uma área de 379.000 km^2. A quantidade de habitantes demonstra que este é um município pequeno. Do total da população somente 7.419 habitantes efetivamente mora no centro urbano da cidade, ou seja, 67% da população reside na zona rural do município.

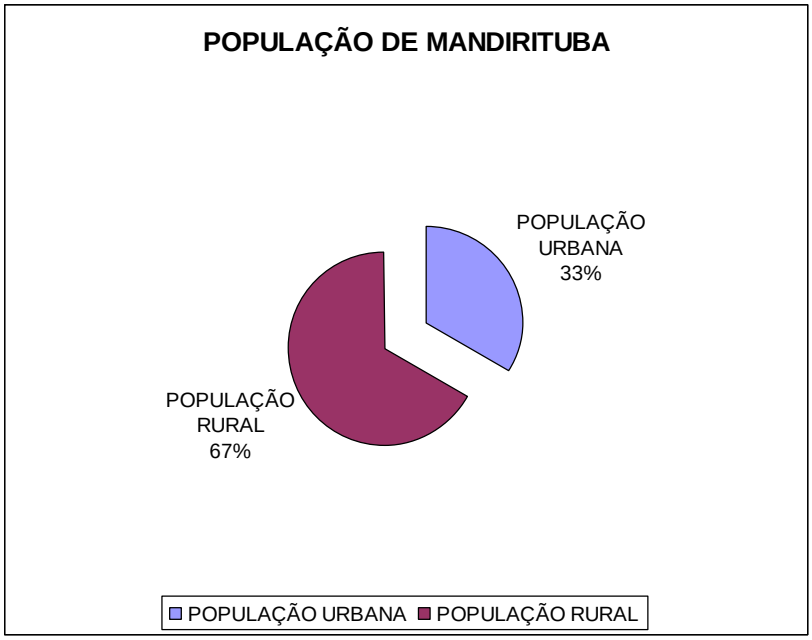

Figura 3 - Distribuição da população de Mandirituba

O centro urbano da cidade é uma área pequena relativa ao tamanho total do município. Apesar do município possuir cerca de 61 bairros o centro da cidade em si consiste de 5 bairros somente. Esta zona central é atrevessada por uma rodovia que liga a cidade à capital do estado, Curitiba.

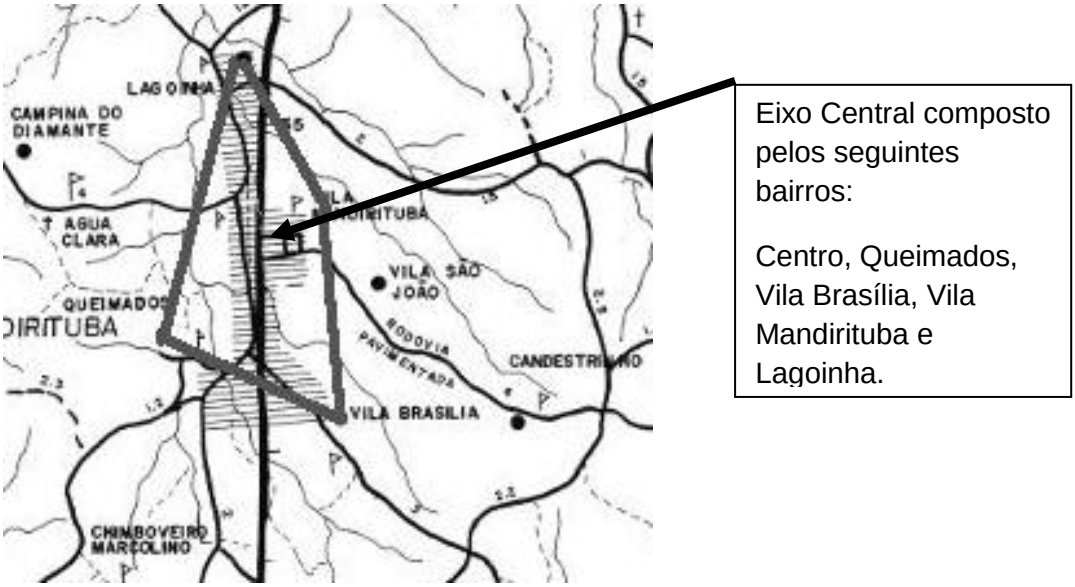

Figura 4 - Zona central do município de Mandirituba

O restante dos bairros, assim como da população em geral, se encontra dispersa pelo território do município que é, conforme mencionado, de 379.000 km^2. A renda média do trabalhador residente de Mandirituba é de 2,2 salários mínimos, de acordo com o censo.

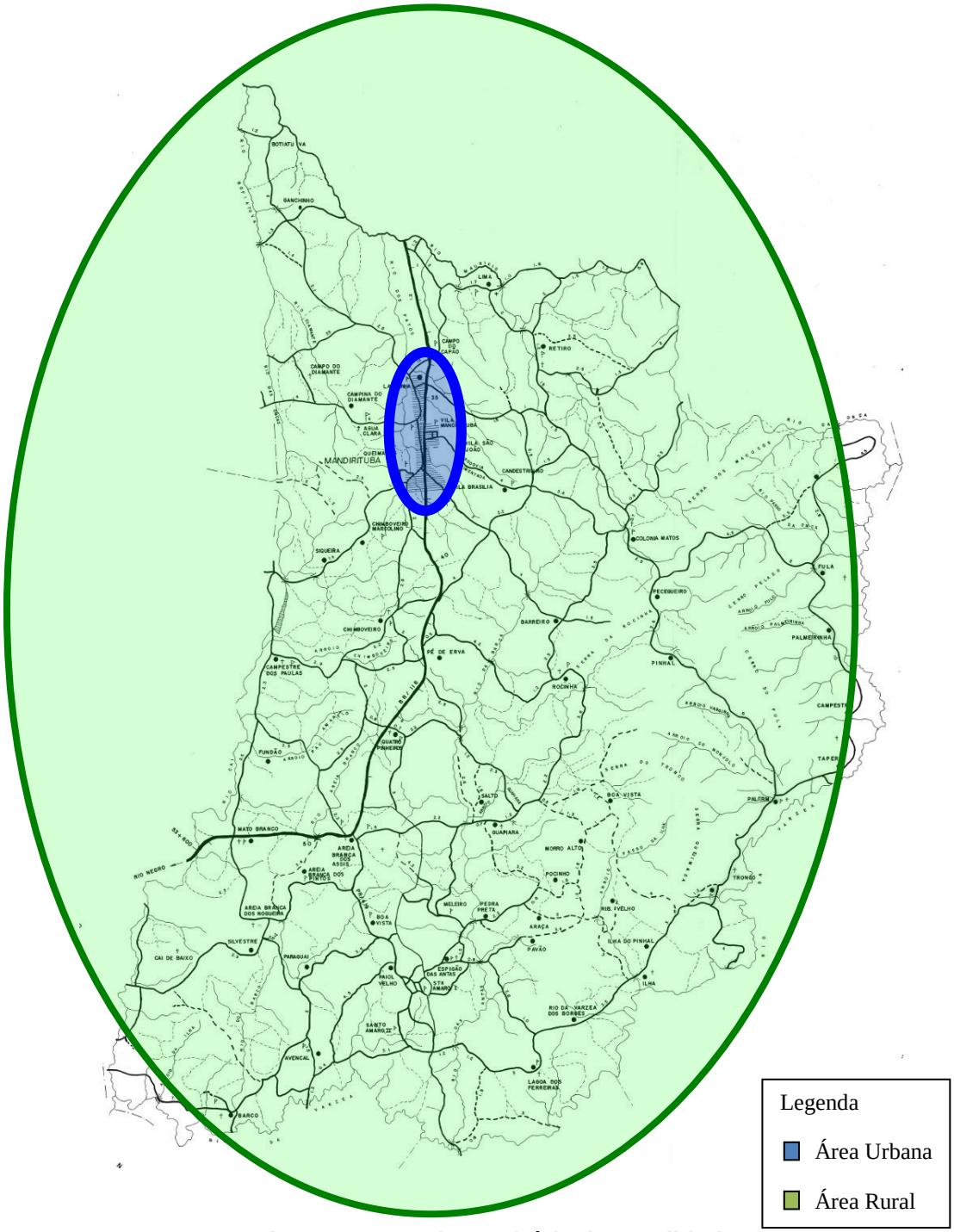

Figura 5 - Mapa do município de Mandirituba

6 Metodologia

As implementações foram criadas em um ambiente Windows 7 Professional 64 bits e um hardware contendo um processador i5 M450 de 2,40 GHz com 2 núcleos capazes de simular 4 núcleos e 4 GB de memória RAM. Todas as medições

apresentadas em porcentagem se referem a estes valores. Em relação ao processador o uso dos 4 núcleos é simbolizado pelo valor 100%. Isto significa que 25% de uso se refere a uma *thread* ocupando por completo um núcleo.

Os únicos processos em atividade durante a execução das implementações eram as próprias ferramentas e a ferramenta nativa do Windows "Monitor de Desempenho" utilizada para obter os gráficos de recursos. A ferramenta responsável por executar as implementações era reinicializada a cada execução. A ferramenta *Memory Cleaner* também era executada antes de cada execução para liberar o máximo de memória possível.

6.1 Descrição do Modelo

O modelo a ser implementado em ambas as ferramentas apresentadas neste estudo seguirá um conjunto de regras genéricas que deverão ser respeitadas por ambas as implementações. Desta maneira foi necessário realizar uma simplificação do modelo de maneira a minizar a influência das limitações de cada ferramenta. Posteriormente estas limitações serão discutidas em mais detalhes.

Devido a estrutura da população, sua renda, e do município em si foi decidida a utilização do modelo de Kohl como base do modelo desenvolvido no estudo. Conforme já mencionado, o modelo de Kohl descreve sociedades pré-industriais onde a população pertencente ao grupo social menos favorecido financeiramente se encontra na periferia. Dado o tamanho da população de Mandirituba, e sua distribuição predominantemente rural, em conjunto com a baixa renda média da população este município possui uma similaridade com as sociedades descritas pelo modelo de Kohl.

De maneira a simplificar a implementação do modelo não serão utilizados parametrizações no modelo. Desta maneira os dados sobre os quais o modelo será construído serão pré-determinados e mantidos em ambas as implementações.

Dois conjuntos de dados simulados serão utilizados. Estes dados são relativos ao saneamento básico de Mandirituba e não são baseados em dados reais do município. A utilização de dados de saneamento básico se deve ao fato de que apesar de pessoas poderem habitar regiões sem iluminição elétrica, ou mesmo outras comodidades, o mesmo não se aplica a água.

O primeiro conjunto de dados contém uma distribuição normal do saneamento a partir do centro urbano do município, demonstrado na Figura 5. Dessa maneira o centro urbano possuirá maior saneamento básico e o mesmo declinará a medida que se afaste do centro. Assim regiões mais afastadas do centro possuem menos atratividade para os seus habitantes. Apesar de simplificado, isso visa simular a atratividade que o centro urbano proporciona em relação a periferia. Este aspecto será explorado mais adiante na descrição do modelo.

O segundo conjunto de dados também possui uma distribuição normal de saneamento básico mas contém duas zonas de foco ao invés de uma. Neste conjunto de dados tanto o centro urbano do município quanto a região localizada entre o centro e Curitiba possuem mais saneamento básico que o restante do município. Este conjunto de dados traça um parelelo com o modelo de Hoyt, mas foi criado somente como uma diferenciação da primeira de maneira a termos dois conjuntos de dados.

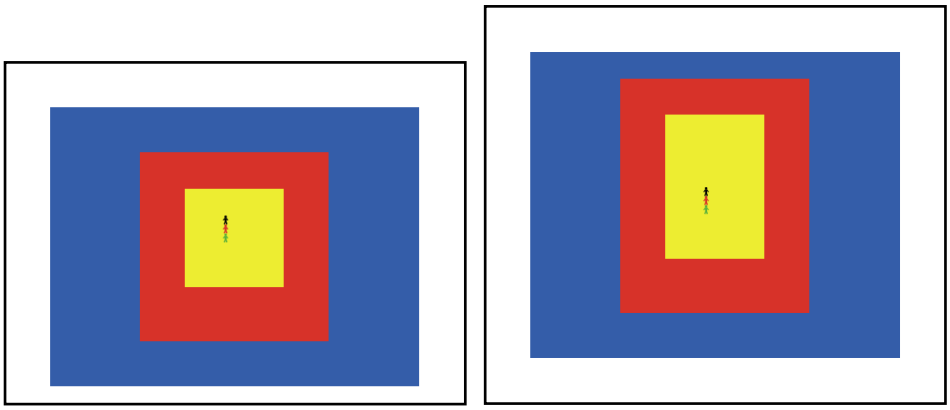

Figura 6 - Distribuição do saneamento básico em ambos os conjuntos de dados

A área em branco na Figura 6 representa uma região com 50% de saneamento básico; a área em azul contém 75% de saneamento; a área em vermelho contém 85% e a área em amarelo contém 90% de saneamento básico.

O modelo comtempla a criação de três tipos de agentes de software representando grupos sociais distintos. Estas três classes são:

- *Upper class* (representado graficamente como)
- *Middle class* (representado graficamente como)
- *Lower class* (representado graficamente como)

A criação de somente três classes visa abstrair a existências dos vários graus existentes na classificação brasileira de classes (classe A, B, C e outras). Dessa maneira podemos ter uma simplificação na implementação deste modelo. A utilização de somente três

classes também remete ao modelo de Kohl, que prevê três classes, ou grupos, sociais, de acordo com (Maretto et al, 2010).

Outra razão da utilização de somente três classes é relativa a distribuição da quantidade de habitantes pertencentes a cada classe. Conforme o obtido no censo de 2011, o município de Mandirituba contém 22.235 habitantes e devido a isto este modelo conterá no máximo 22.200 agentes de software. Considerando a renda média dos trabalhadores do município a seguinte divisão da proporção dos habitantes de cada classe foi feita:

- 5% de agentes *Upper class*
- 10% de agentes *Middle class*
- 85% de agentes *Lower class*

6.2 Lógica de comportamento dos agentes

O modelo sempre se iniciará com a criação de três agentes, um de cada classe já especificada. Durante a execução do modelo novos agentes serão criados de maneira aleatória. Estes agentes terão suas classes escolhidas de maneira arbitrária obedecendo sempre a proporção estabelecida anteriormente.

Os agentes irão então procurar uma residência para alocar dentro dos limites do modelo. Os agentes procurando uma residência irão utilizar a porcentagem de saneamento básico disponível na área onde se encontram para decidir se irão ou não ocupa-lá. A descrição do fluxo da lógica básica dos agentes é demonstrada na Figura 7.

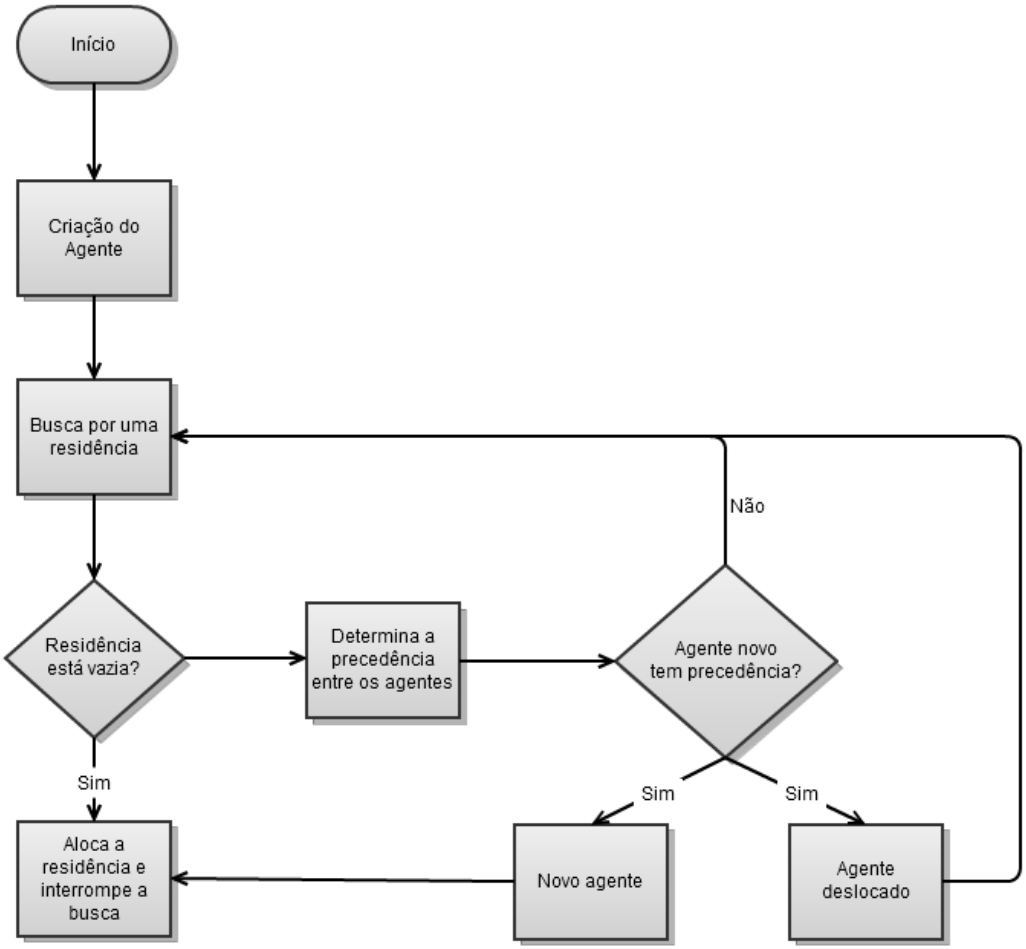

Figura 7 - Lógica comportamental dos agentes

Agentes do tipo *Upper class* somente se estabelecem em áreas que tenham 90% ou mais de saneamento básico; os agentes do tipo *Middle class* alocam áreas com 60% ou mais de saneamento básico; já os agentes do tipo *Lower class* somente alocam áreas com 30% ou mais de saneamento.

A partir do momento que um agente encontra uma área que satisfaça o seu requerimento mínimo de saneamento básico ele irá tentar se estabelecer. Caso a área encontrada esteja vazia a sua alocação pode ser realizada imediatamente. No entanto caso a área já esteja ocupada por outro agente é necessário observar a regra de precedência para determinar qual dos agentes deverá permanecer na área. A regra da precedência é baseada nas classes dos agentes envolvidos na disputa pela residência.

Agentes do tipo *Upper class* sempre terão prioridade sobre as outras classes sendo a única exceção a essa regra um outro agente *Upper class*. Isso significa que caso um agente *Upper* resida numa área a qual outro agente tente alocar o novo agente será obrigado a

continuar sua busca. Agentes do tipo *Middle class* tem precedência sobre Agentes do tipo *Lower class* e não podem deslocar um Agente da mesma classe. Os Agentes do tipo *Lower class* não possuem precedência sobre nenhuma outra classe, ou seja, nunca conseguirão deslocar outro Agente nem da mesma classe.

Quando um agente não tem precedência, seja um agente previamente alocado ou um agente que esteja tentando habitar uma residência, ele será obrigado a procurar uma nova residência. Esta busca sempre é realizada numa vizinhança de Moore da área onde o agente atualmente se encontra. Este comportamento de busca se repetirá até que seja possível para um agente alocar uma residência, seja ela estando vazia ou através do deslocamento de um agente.

Independentemente da ferramenta a ser realizada a implementação para ambos os cenários deve sempre ser executada por 10 anos simulados. Isso determina que independentemente da unidade temporal utilizada para cada ferramenta sempre deve ser calculadas quantas unidades temporais devem ser executadas de maneira a simular uma década.

7 Ferramenta Netlogo

A primeira ferramenta utilizada para a implementação do modelo foi o NetLogo. Esta ferramenta foi inicialmente criada por Uri Wilensky e atualmente mantida pelo Centro de Aprendizado Conectado (*Center for Connected Learning*) da Universidade NorthWestern em Chigago, USA.

O NetLogo é um ambiente de modelagem programável que é utilizado para construir simulações naturais e de fenômenos sociais. A ferramenta é construída utilizando a tecnologia Java de maneira a ser multi-plataforma. Apesar de ser baseada em Java o NetLogo possui uma linguagem estruturada própria de programação chamada Logo. Esta linguagem é uma extensão da linguagem existente na ferramenta StarLogo, originalmente criada em 1989-1990 no MIT (*Massachusetts Institute of Technology*) Media Labs.

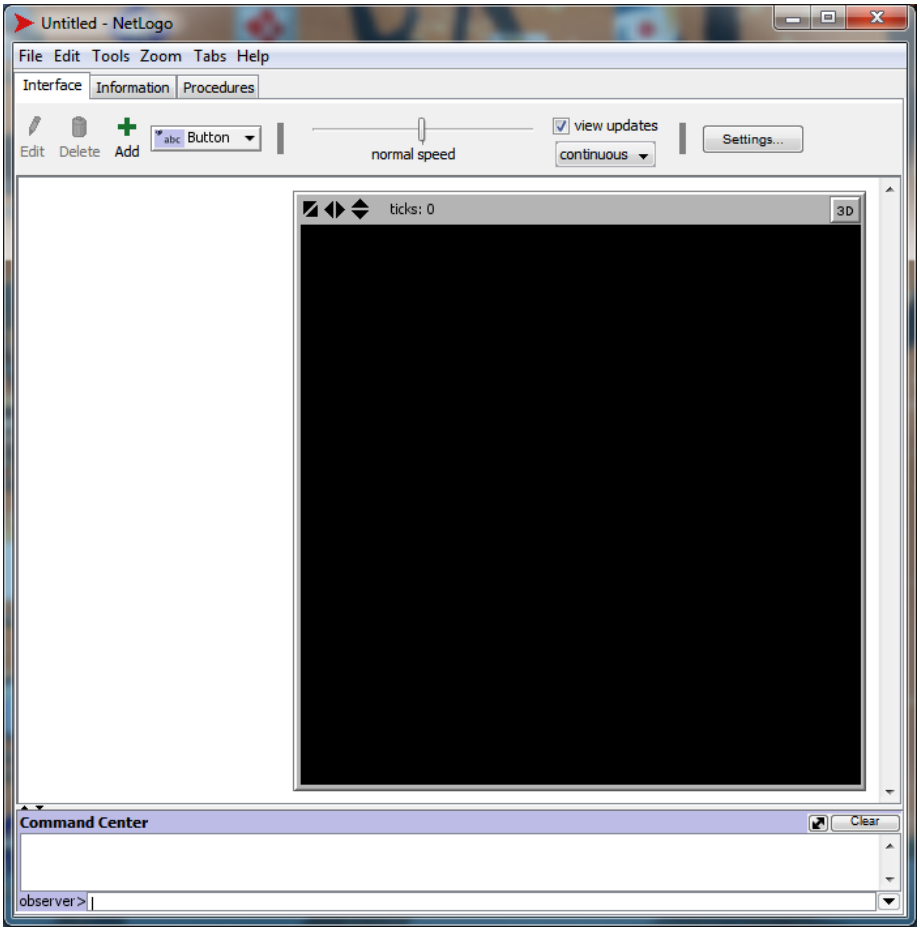

Figura 8 - Ferramenta NetLogo

O NetLogo, demonstrado na Figura 8, é utilizado para modelar sistemas complexos que evoluem ao longo do tempo. A ferramenta permite a criação de diversos agentes que operam de maneira independente. Dessa maneira é possível explorar a conexão entre os agentes em um nível micro enquanto se observa, ao mesmo tempo, os padrões que estas conexões, ou interações, geram em nível macro.

A linguagem Logo (Figura 9) utilizada na ferramenta possui uma grande similaridade com linguagens de scripts e possui uma curva de aprendizagem moderamente baixa. A linguagem também possui uma API (*Application Programming Interface*) contendo diversas funções para operar sobre os agentes e o mundo em que são criadas assim como métodos utilitários.

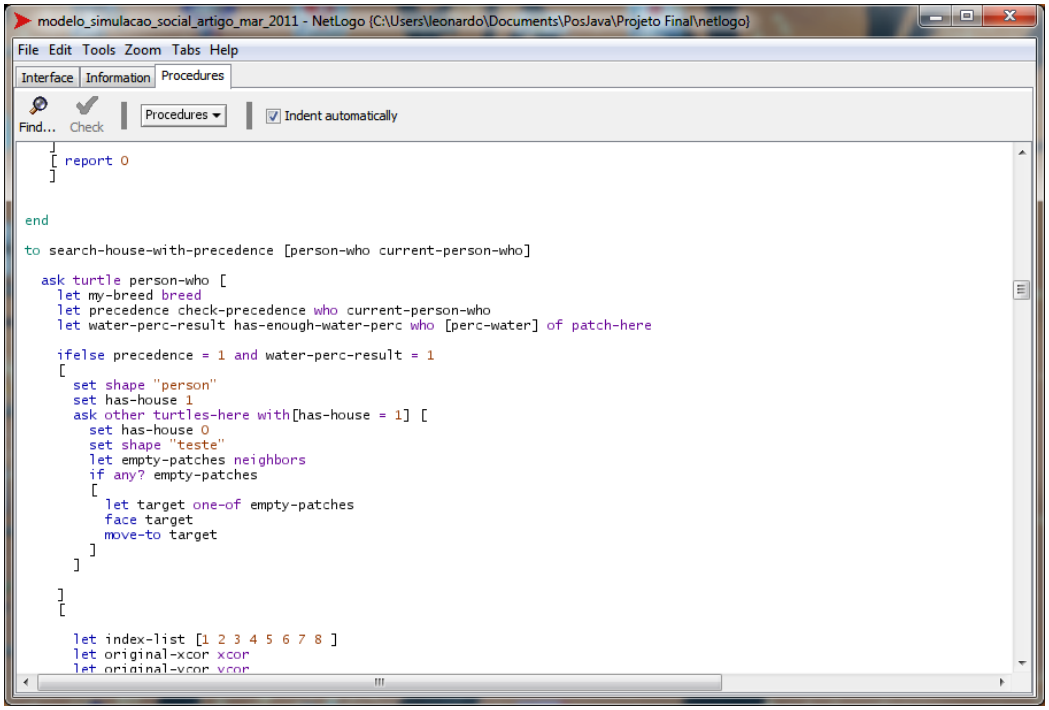

Figura 9 - Linguagem Logo

Os agentes criados na ferramenta são nomeados *turtles* (tartarugas) enquanto cada posição da matriz onde eles existem é nomeada *patch* (terreno). Tanto agentes quanto terrenos possuem atributos básicos como o seu identificador, forma e cor mas também podem ser extendidos de maneira a conter outros atributos definidos pelo implementador/modelador. Estas atributos criados pelo implementador podem ser utilizados para modelar objetos reais dentro da ferramenta.

O NetLogo possui uma unidade temporal chamada *tick* (período). Um período compreende a unidade de tempo real necessária para que todos os agentes possam executar o seu comportamento ou conjunto de instruções. Os períodos podem ser utilizados pelo implementador para definir, por exemplo, quando um novo agente deve ser introduzido no modelo assim como quaisquer outras condições que dependam da passagem do tempo para serem executadas.

A ferramenta suporta a criação de componentes gráficos embutidos na implementação, demonstrados na Figura 10, que permitem a utilização de parâmetros. Estes componentes também permitem a criação de gráficos e contadores em tempo real da situação do modelo.

Apesar de haver suporte a utilização de parâmetros o mesmo é bem simplificado, em termos gráficos, basicamente se resumindo a utilização de botões para indicar o estado de alguma configuração e *sliders* para definir quantidades. Existe a possibilidade da leitura de dados via arquivo texto mas, no entanto, não é suportado acesso a bancos de dados

diretamente. Isto acaba por limitar em grande parte a utilização da ferramenta para simular modelos de larga escala.

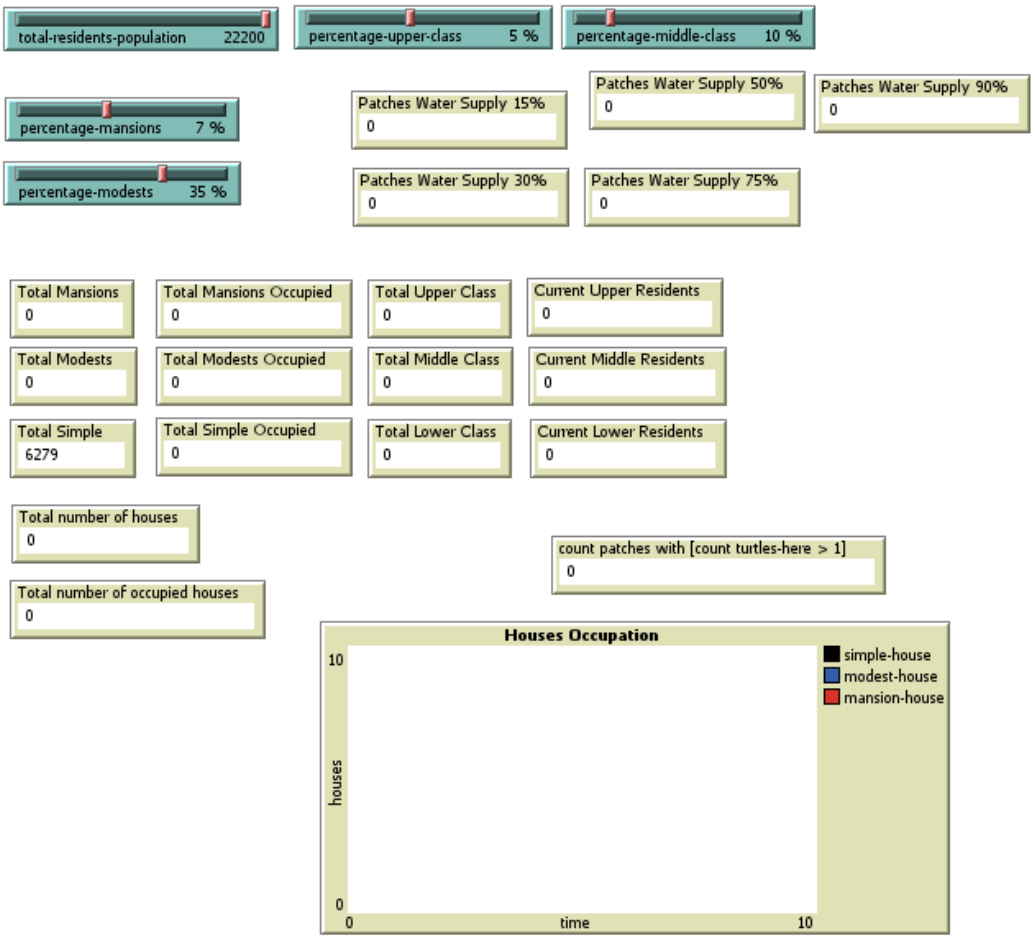

Figura 10 - Exemplo de sliders, contadores e gráficos do NetLogo

O NetLogo também disponibiliza uma grande biblioteca de exemplos em que é possível executar grande uma variedade de modelos baseados em agentes. Estes exemplos pertencem a vários tipos de simulações indo desde a simulação de penetração de óleo no solo até a simulação de contaminação de uma população por zumbis. Todos os exemplos tem o seu código fonte disponibilizado permitindo assim o seu estudo.

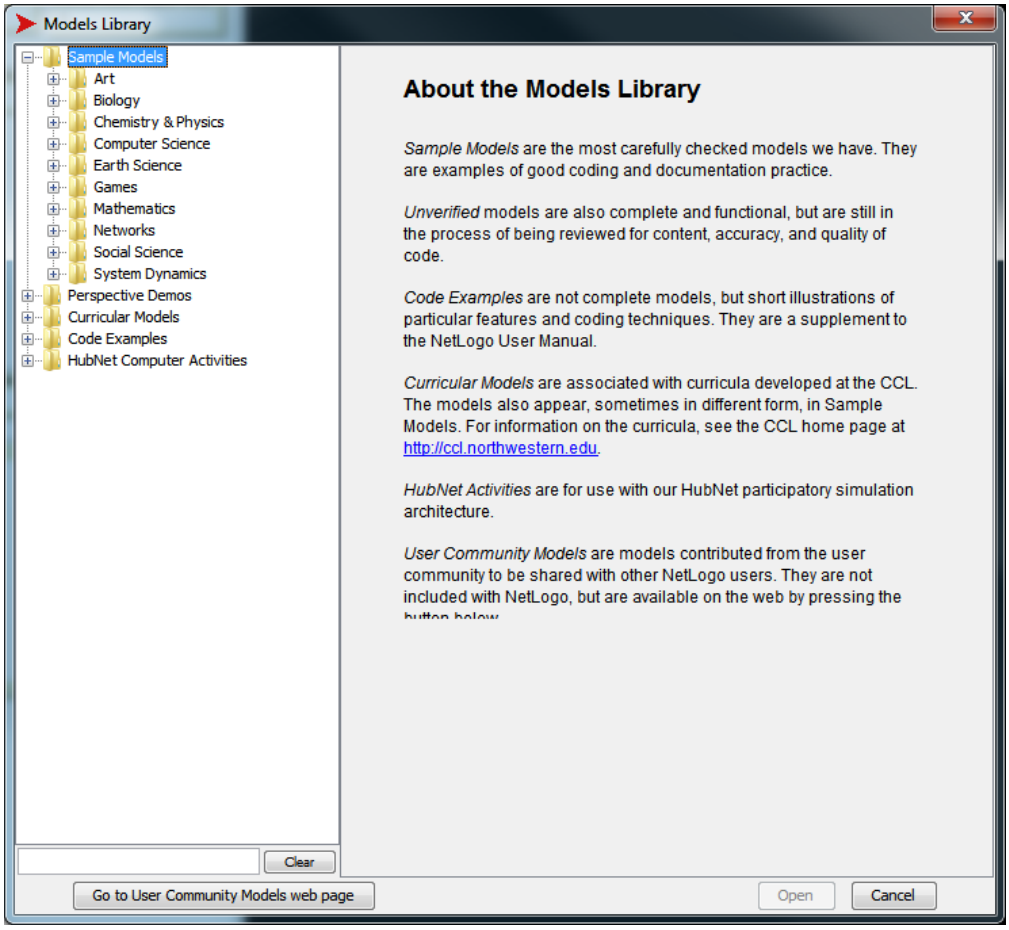

Figura 11 - Biblioteca de exemplos do NetLogo

A ferramenta permite que agentes e terrenos sejam manipulados em termos de forma e de cor de maneira a facilitar a visualização da execução do modelo. A medida que o modelo é executado a representação visual do modelo é atualizada mostrando o movimento dos agentes e suas ações no modelo. Esta visualização é normalmente realizada em 2D mas é possível visualizar em 3D.

Figura 12 - Visualização do modelo

Durante a execução do modelo é possível optar por ver a atualização da visualização do modelo continuamente ou período a período. Apesar da visualização contínua ser interessante por mostrar cada mudança no instante em que a mesma acontece ela também degrada a performance de execução do modelo.

Existe também a possibilidade de controlar a velocidade com a qual o modelo é executado. Isto permite executar o modelo mais devagar de maneira a ser possível ver, no modo contínuo, a evolução de cada agente em separado. Ao executar o modelo mais rápido também é possível adiantar o modelo até uma certa fase do seu desenvolvimento.

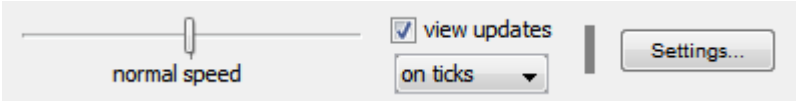

Figura 13 - Controle de visualização e velocidade do modelo

7.1 Implementação do modelo

As regras do modelo são conforme as explicadas na seção 6.2. Foi utilizada, quando possível, a API da ferramenta para minimizar a quantidade de código criada. Apesar de não serem utilizados parâmetros no modelo como um todo foi escolhido usar os componentes sliders para determinar a quantidade de agentes a serem criados e a porcentagem de distribuição entre as classes. A utilização destes componentes é realizada meramente para testes com diferentes quantidades e distribuições de classes sendo que para as execuções oficiais os parâmetros são pré-fixados conforme o mencionado na seção 6.1.

O código-fonte da implementação totalizou 757 linhas considerando comentários no código. Sem os comentários existem no total 691 linhas de código. O tempo total do estudo da ferramenta e do desenvolvimento do modelo podem ser encontrados na Tabela 1:

Tabela 1 - Tempo de desenvolvimento do modelo no NetLogo

Atividade	Tempo (em horas)
Estudo da ferramenta	12
Desenvolvimento da implementação	36
Testes	16
Total	64

Durante o desenvolvimento do modelo foi notado um detalhe importante sobre a ferramenta em termos de programação que é a não existência de depuração do código-fonte. Nas ocasiões em que o comportamento da implementação não condizia com o esperado ou quando havia a existência de um *bug* na implementação era necessário realizar testes de mesa para encontrar o problema. Apesar de isso ser aceitável para modelos pequenos e pouco complexos, como o desenvolvido neste estudo, a existência dessa funcionalidade é importante para modelos complexos.

A documentação disponível da API é bastante extensa apesar de haverem funções mal documentadas. Existe uma lista de discussão da ferramenta onde é possível obter respostas diretamente da equipe de desenvolvimento assim como de outros desenvolvedores. Outra importante fonte de informação é a biblioteca de exemplos que contém códigos variados e normalmente bem comentados.

Foi definido, arbitrariamente, que cada dia real seria equivalente a 14 períodos (ticks) da ferramenta NetLogo. No total isso corresponde a 5110 periodos por ano (considerando um ano como sendo 365 dias) e 51110 períodos em 10 anos. A cada 2 anos executados uma screenshot do desenvolvimento foi tirada de ambos os cenários.

Figura 14 – Execução do primeiro cenário

Durante a execução dos cenários foram feitas medições de consumo de CPU e memória do NetLogo de maneira a determinar a quantidade de recursos gastos para rodar a simulação. O tempo necessário para rodar cada simulação também foi registrado.

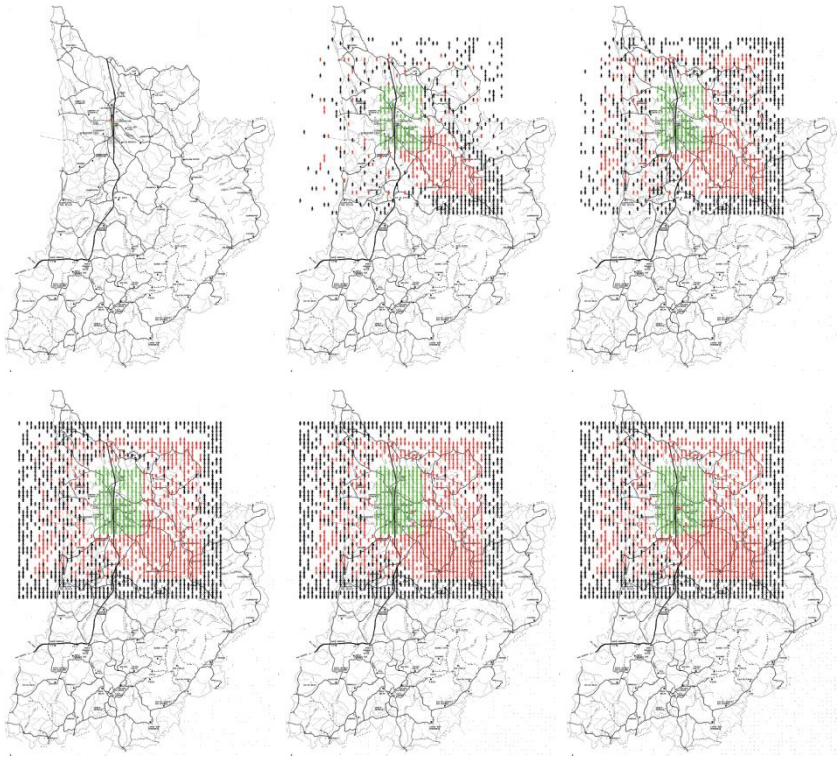

Figura 15 – Execução do segundo cenário

Os gráficos abaixo representam as medições realizadas tanto para o primeiro cenário quanto para o segundo.

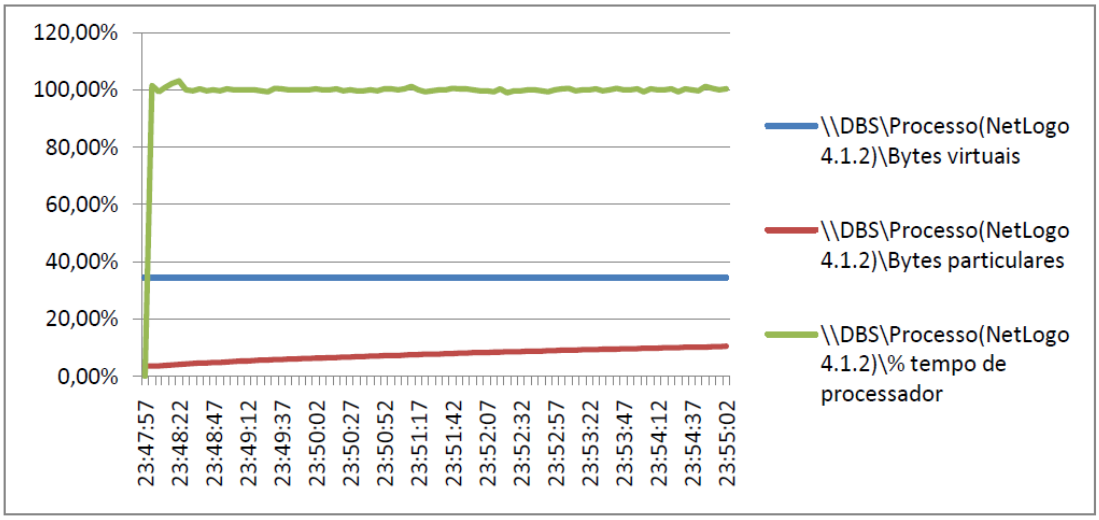

Figura 16 – Gráfico de desempenho da primeira execução do NetLogo

O tempo total de execução do primeiro modelo foi de 7 minutos e 5 segundos. Durante esse período o consumo de CPU foi virutalmente constante a 100% o que se justifica

pela grande quantidade de agentes co-existindo ao mesmo tempo. Considerando o consumo do processador é perceptível que a utilização de um hardware que pudesse oferecer mais possibilidades de execução paralela proveria melhorias na performance. O consumo de memória mostra um aumento gradual desde o ínicio da execução o que é condízel com o aumento do número de agentes durante a execução.

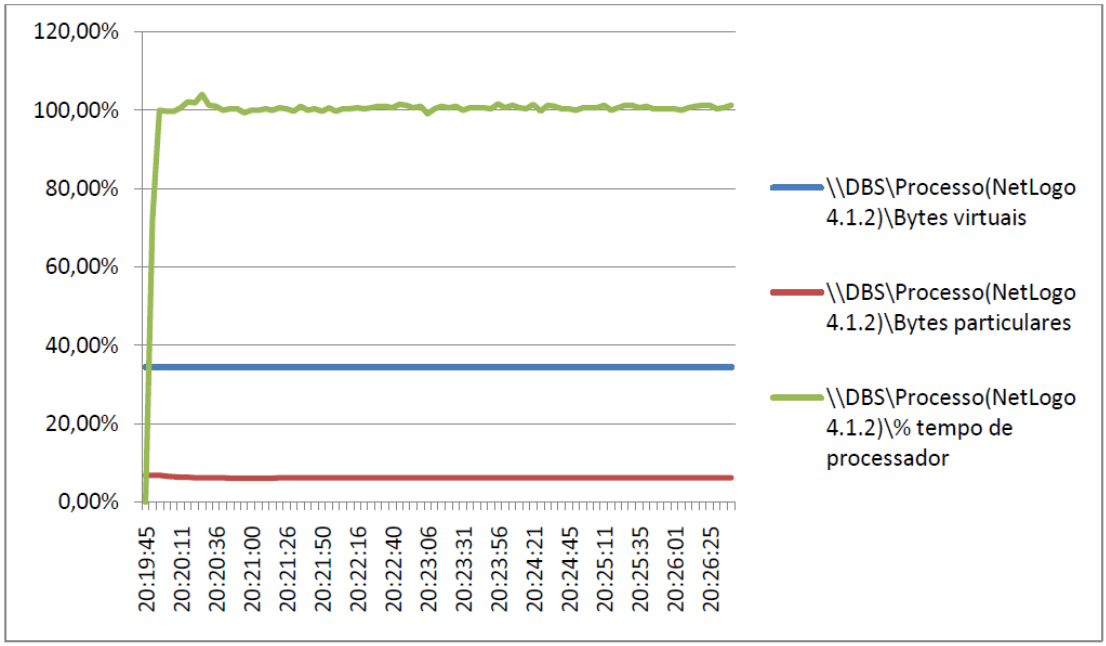

Figura 17 - Gráfico de desempenho da segunda execução do NetLogo

O tempo total de execução da segunda implementação foi de 6 minutos e 56 segundos. De maneira similar à primeira execução o consumo da CPU foi constante em 100% pelas mesmas razões. A mesma consideração em relação ao hardware da primeira implementação se aplica aqui.

O consumo de memória da segunda implementação, no entanto, difere substancialmente do da primeira implementação. Não foi possível elucidar a razão pela qual o consumo de memória se manteve constante durante toda a execução do modelo ao invés de gradualmente aumentar com o tempo e o número de agentes. Foi levantada a hipótese de que como na primeira execução a memória da máquina virtual já havia sido expandida a segunda execução não necessitou de mais memória. De maneira a testar esta hipótese o NetLogo foi reiniciado e o teste executado novamente mas os mesmos resultados foram obtidos.

8 Ferramenta Repast J

A segunda ferramenta utilizada para a implementação do modelo foi o Repast Symphony sendo mais especificamente a implementação em Java conhecida como Repast J. Esta ferramenta possui código aberto e foi inicialmente criada por Sallach, Collier, Howe e outros (Collier, N., Howe, T., e North, M., 2003). Originalmente o seu desenvolvimento aconteceu na Universidade de Chigago em Chigago, USA. Atualmente a ferramenta é suportada e desenvolvida por uma comunidade aberta de desenvolvedores.

O Repast Symphony é um conjunto de plataformas de código aberto para a criação de modelos baseados em agentes e simulação. Conforme mencionado anteriormente este estudo utilizará a versão desenvolvida para a plataforma Java. No entanto é válido mencionar que existem várias outras plataformas como a ReLogo (que permite um desenvolvimento baseado no NetLogo), Groovy, Python e outras. Existe inclusive uma implementação para computadores de alto desempenho que permite a utilização de plataforma de hardware que se utilizem de paralelismo distribuído. O Repast Symphony se encontra em desenvolvimento há quase 10 anos e já foi utilizado para construir simulações sociais, cadeias de consumo e outras utilidades.

O Repast J basicamente provê um framework para desenvolvimento de modelos baseados em agentes. Este framework disponibiliza uma série de classes que permitem a criação de um agente assim como a criação do ambiente onde ele será inserido. Assim como o NetLogo os detalhes específicos de como criar um agente ficam a encargo do implementador.

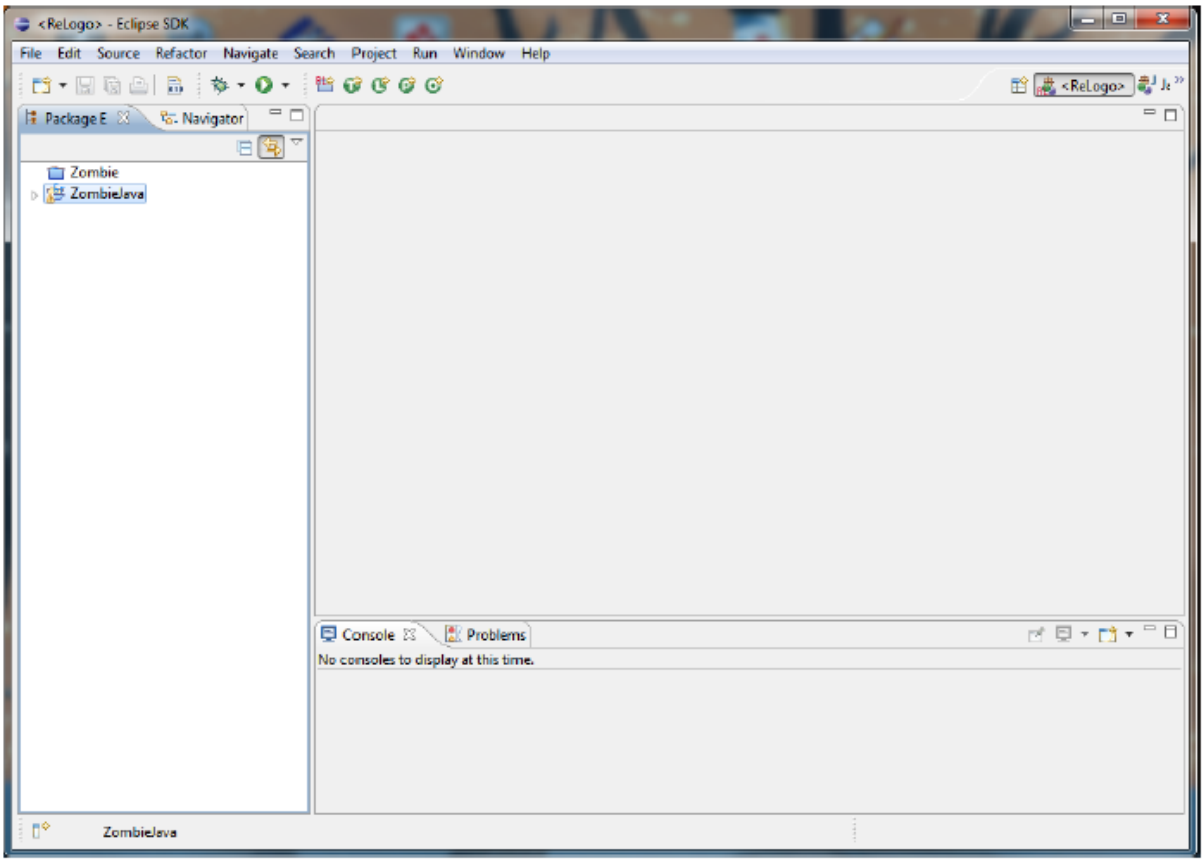

Figura 18 - Ferramenta Repast J

A ferramenta disponibiliza uma IDE (*Integrated Development Environment*) baseada no Eclipse (http://www.eclipse.org/), uma IDE Java de código aberto. A utilização de uma das IDEs mais conhecidas para desenvolvimento Java facilita a utilização da ferramenta por diminuir a sua curva de aprendizagem. Todas as funcionalidades básicas do Eclipse encontram-se presente sendo que são adicionadas novas configurações e recursos para suportar o desenvolvimento baseado em agentes.

```
37
38    @ScheduledMethod(interval = 1, start = 1)
39    public void step() {
40        GridPoint pt = grid.getLocation(this);
41
42        GridCellNgh<Human> nghCreator = new GridCellNgh<Human>(grid, pt,
43                Human.class, 1, 1);
44        List<GridCell<Human>> gridCells = nghCreator.getNeighborhood(true);
45
46        SimUtilities.shuffle(gridCells, RandomHelper.getUniform());
47
48        GridPoint pointWithMostHumans = null;
49
50        int maxCount = -1;
51
52        for (GridCell<Human> cell : gridCells) {
53            if (cell.size() > maxCount) {
54                pointWithMostHumans = cell.getPoint();
55                maxCount = cell.size();
56            }
57        }
58
59        moveTowards(pointWithMostHumans);
60        infect();
```

Figura 19 - Exemplo de código desenvolvido com o framework Repast J

Um detalhe particular do *framework* é a ausência de uma classe Agente a partir da qual seriam derivados os agentes criados pelo implementador. Todos os agentes criados são

classes normais Java sem ser necessário extender uma determinada classe ou implementar uma determinada interface. A partir de um momento que seja necessário que um novo tipo de agente faça parte do modelo é necessário adicionar um agente desse tipo ao contexto do projeto.

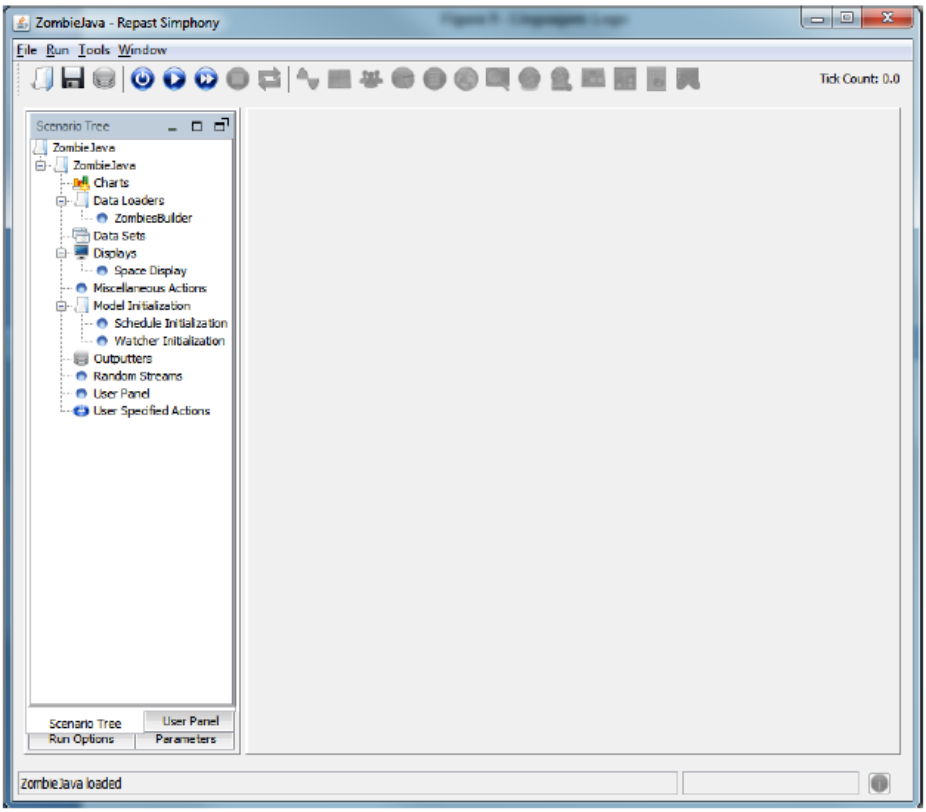

Figura 20 - Ferramenta de execução do Repast J

Um contexto, dentro do *framework*, pode ser entendido tanto como um repositório central das configurações do modelo como o modelo em si. Dentro do contexto serão contidos os agentes, o ambiente (que será discutido em maiores detalhes futuramente) e as configurações necessárias. Conforme já mencionado a partir do momento que um agente necessita ser criado é preciso não somente instanciar um objeto correspondente mas também inseri-lo no contexto.

Dentro do contexto várias outras configurações podem ser adicionadas. Por exemplo, variáveis visíveis a todos os agentes contidos no contexto. Outra configuração inserida no contexto se refere a visualização da execução do modelo. O Repast J se utiliza do JOGL para renderizar as visualizações do modelo tanto em 2D quanto em 3D.

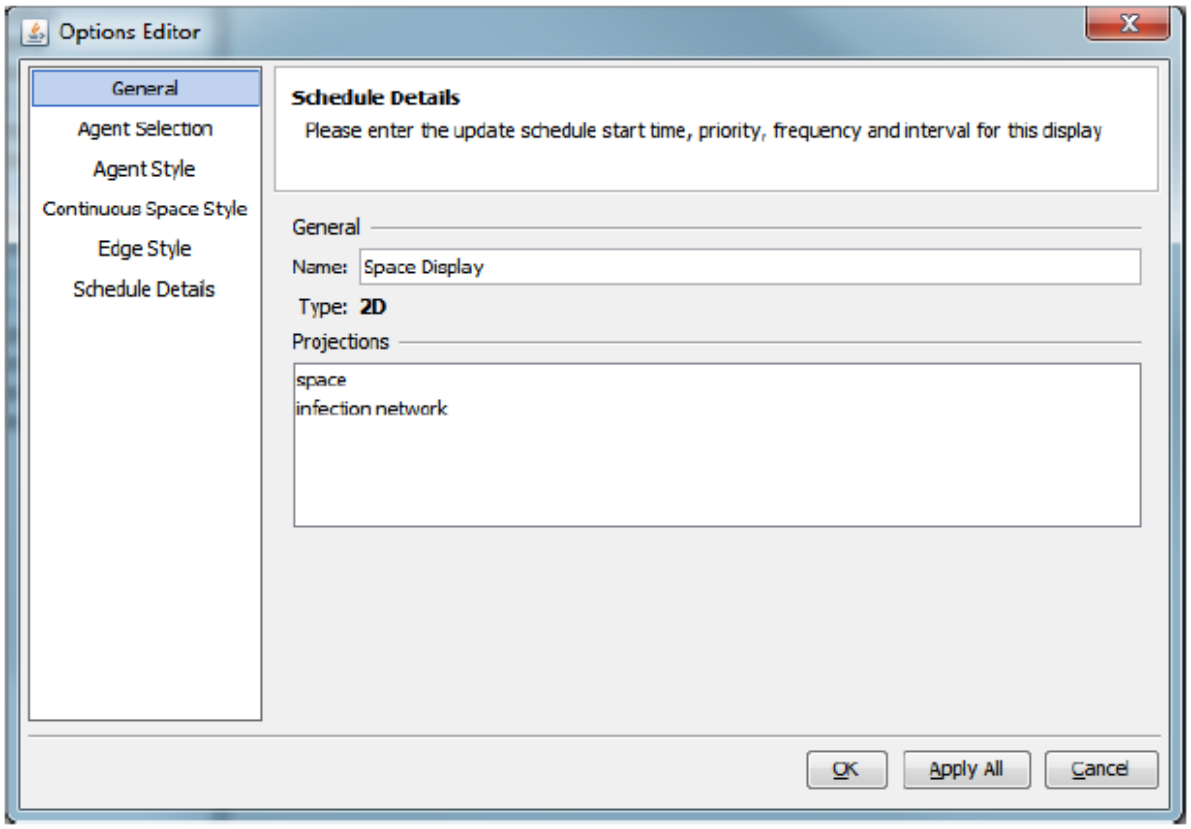

Figura 21 - Configuração de visualização do modelo

A configuração da visualização (*display*) do modelo contém não somente diretivas para a exibição gráfica mas também do ambiente (*grid*) onde os agentes estarão inseridos. As configurações do ambiente como tamanho da matriz, possibilidade de dois agentes coexistirem na mesma célula e outras são definidas em uma instância da classe do framework chamada *Space*. Esta instância é então adicionada no contexto para ser utilizada posteriormente.

Além da configuração presente na instância da classe Space existem outras configurações, pertinentes a representação gráfica do modelo, que são realizadas separadamente. Por exemplo, a configuração de quais agentes serão representados, assim como a sua forma e cor, são realizados em uma ferramenta de configuração (Figura 21). Esta ferramenta também permite a configuração do tamanho, em *pixels*, de cada célula do ambiente assim como da frequência de atualização da representação gráfica.

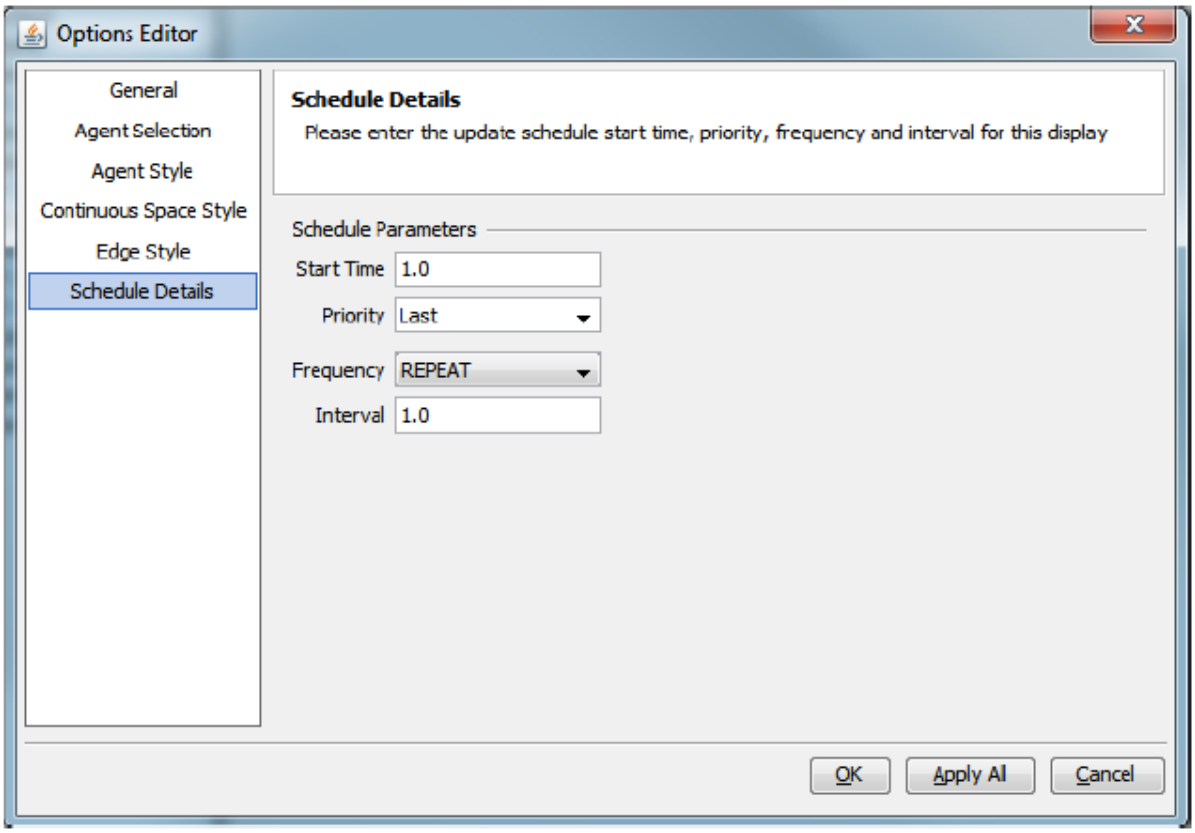

Figura 22 - Configuração da taxa de atualização gráfica do modelo

Apesar da utilização de JOGL, uma biblioteca gráfica muito poderosa, para renderizar o modelo o seu uso ocasionou um problema durante a instalação e utilização da ferramenta. Por padrão a biblioteca não é disponibilizada junto com o Repast J ficando a encargo do implementador obtê-la. Infelizmente não existe documentação nenhuma em relação a versão da bilbioteca a ser utilizada. As últimas versões da biblioteca sofreram modificações nas chamadas ao código nativo que ocasionam falhas ao serem utilizadas pelo Repast J. De maneira a fazer funcionar a ferramenta foi necessário realizar diversas tentativas com versões diferentes da JOGL até ser encontrada uma versão compatível.

O Repast J possui uma unidade temporal chamada *tick* (período), de maneira similar ao NetLogo. Da mesma maneira cada período é o tempo necessário para que o conjunto de instruções de cada agente existente seja executado e a mesma utilidade pode ser feita para os períodos do Repast J.

Conforme demonstrado na Figura 20 o Repast J possui uma ferramenta de execução do modelo. Esta ferramenta referencia todas as configurações já mencionadas para poder executar o modelo.

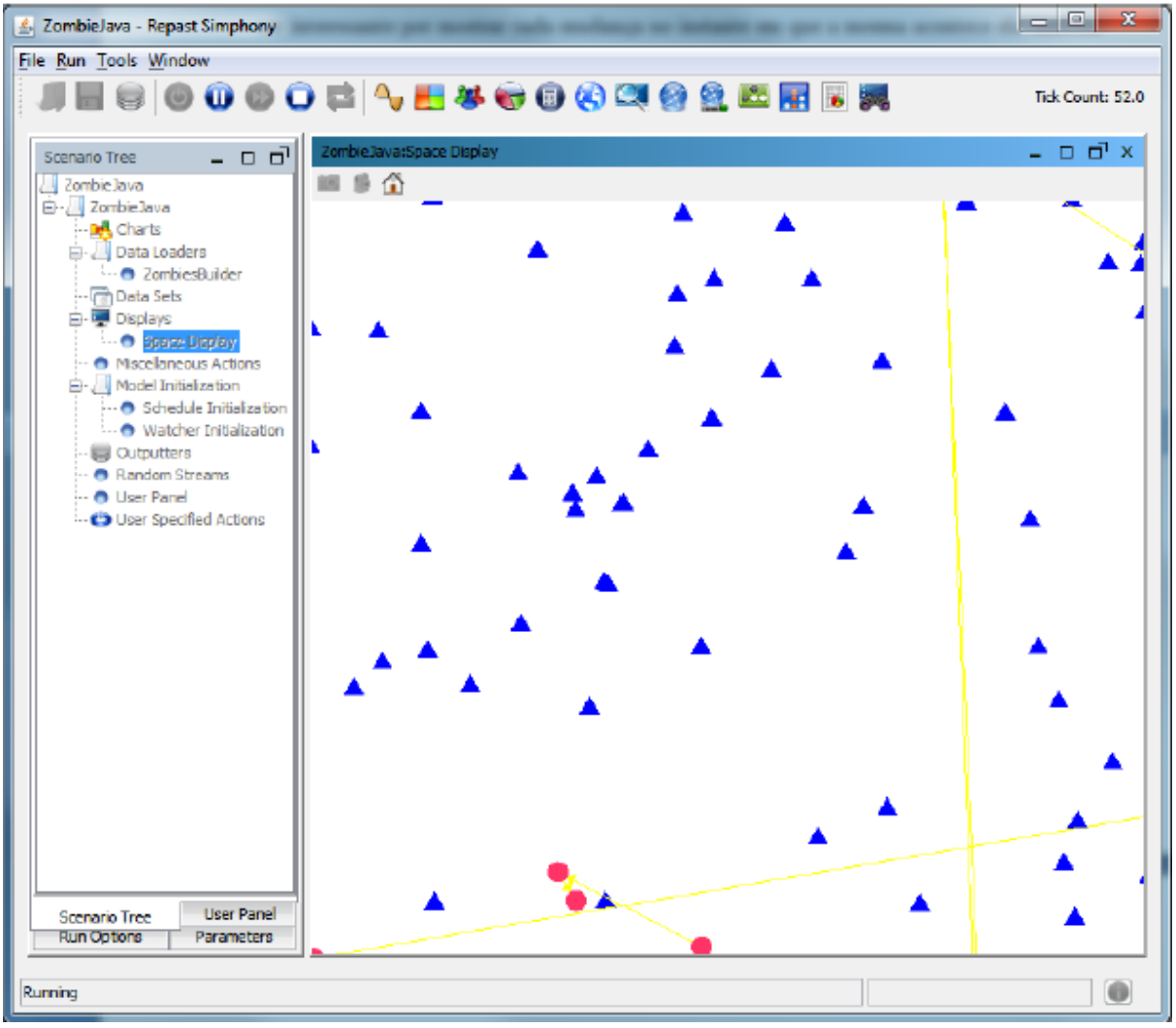

Figura 23 - Execução de um modelo no Repast J

8.1 Implementação do modelo

O *framework* disponibilizado pela ferramenta foi utilizado de maneira a construir o modelo. Apesar de um estudo mais profundo ser necessário de maneira a otimizar o uso do *framework* foi seguido os exemplos disponíveis na documentação para a criação da estrutura do código.

Conforme mencionado anteriormente não foi utilizado nenhum parâmetro para a construção do modelo. A quantidade de agentes e a distribuição deles entre as três classes criadas foi feita de maneira programática. Em um modelo complexo estes parâmetros

poderiam facilmente ser obtidos externamente ao programa, sem a necessidade da alteração do mesmo para a sua mudança.

O código-fonte da implementação totalizou 1103 linhas considerando os comentários presentes no código. Desconsiderando os comentários existem no total 857 linhas de código. O tempo total de estudo da ferramenta e do desenvolvimento do modelo podem ser encontrados na Tabela 2:

Tabela 2 - Tempo de desenvolvimento do modelo no Repast J

Atividade	Tempo (em horas)
Estudo da ferramenta	40
Desenvolvimento da implementação	48
Testes	21
Total	109

O Repast J possui uma curva de aprendizagem relativamente moderada. Apesar de não ser necessário aprender nenhuma linguagem, considerando que o implementador já conhece a linguagem Java, o framework é bastante extenso demandando tempo para ser aprendido. Todas as classes e métodos são documentadas facilitando a sua utilização apesar de algumas documentações serem muito reduzidas.

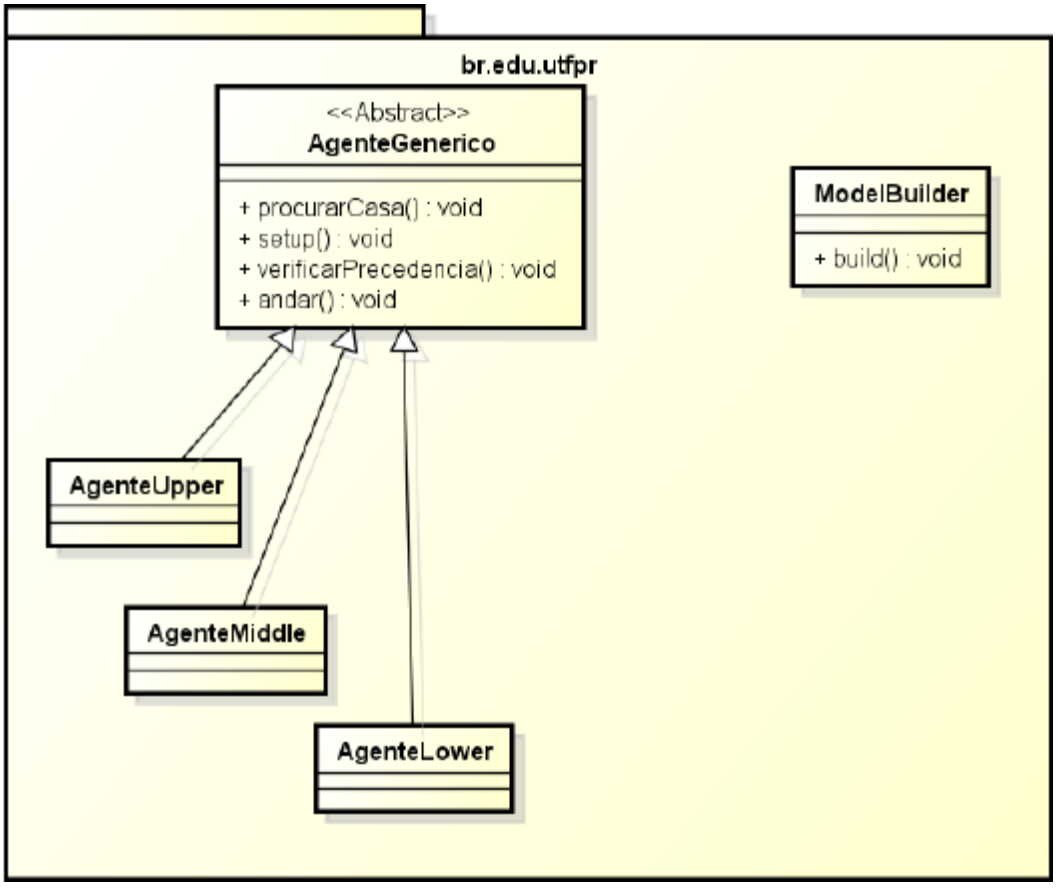

Figura 24- Diagrama de classes da implementação em Repast J

Durante a fase de estudo houveram alguns problemas para criar a implmentação de exemplo. O mais nótavel destes problemas, relacionado a utilização da bilbioteca JOGL, já foi mencionado anteriormente então não será discutido novamente. Outro problema encontrado foi a falta da documentação relativa à implementação de exemplo/referência.

Apesar de a própria instalação prover a localização da documentação a mesma não é válida. Ao tentar obter a documentação pelo site da ferramenta o mesmo problema foi encontrado. Posteriormente ao se procurar no diretório de instalação a documentação foi encontrada mas não haviam nenhuma indicação que ela se encontrava junto com o framework.

Como a ferramenta Repast J compartilha o mesmo conceito de período que o NetLogo as mesmas medidas de tempo foram aplicadas. Cada dia real seria equivalente a 14 períodos da ferramenta. No total isso corresponde a 5110 periodos por ano (considerando um ano como sendo 365 dias) e 51110 períodos em 10 anos.

Durante a execução dos cenários foram feitas medições de consumo de CPU e memória do NetLogo de maneira a determinar a quantidade de recursos gastos para rodar a simulação. O tempo necessário para rodar cada simulação também foi registrado.

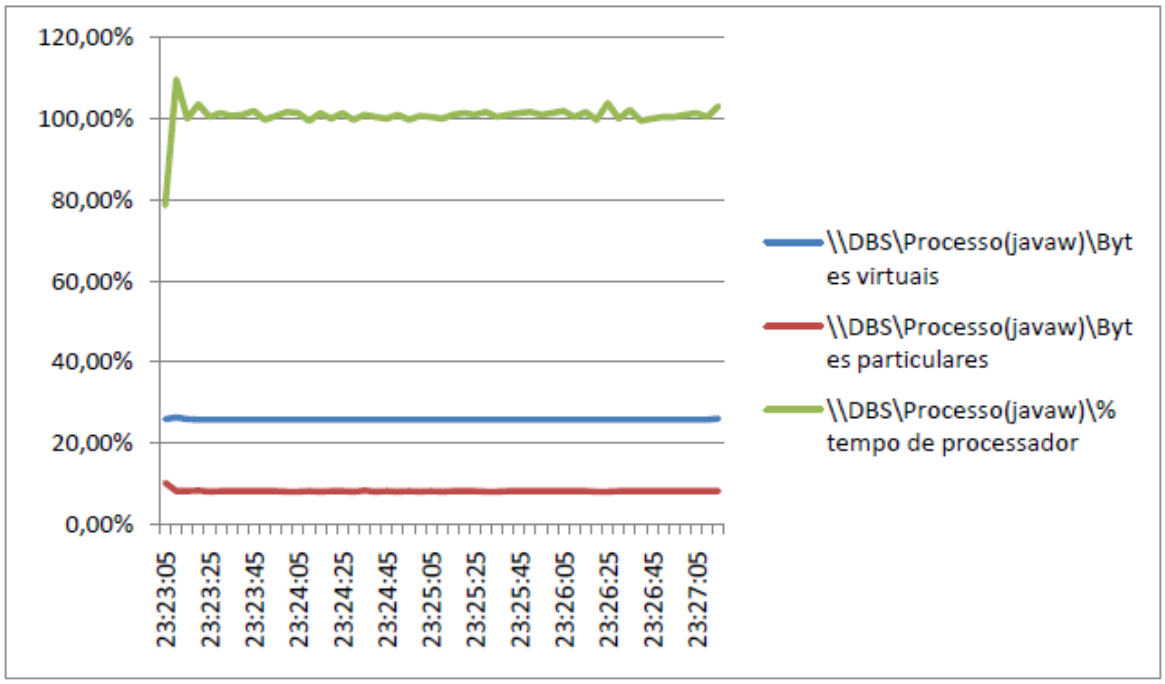

Figura 25 - Gráfico de desempenho da primeira execução do Repast J

O tempo total de execução do primeiro modelo foi de 4 minutos e 15 segundos. Durante esse período o consumo de CPU foi virtualmente constante a 100% o que se justifica pela grande quantidade de agentes co-existindo ao mesmo tempo. A mesma sugestão feita em relação à ferramenta NetLogo, em termos de hardware, se aplica neste caso.

O consumo de memória da ferramenta é constante o que é normal dado que a máquina virtual Java pré-alocou a quantidade de memória necessária e durante a execução não houve necessidade de aumento dessa reserva. Nota-se que o consumo de memória do Repast J foi menor que o do NetLogo.

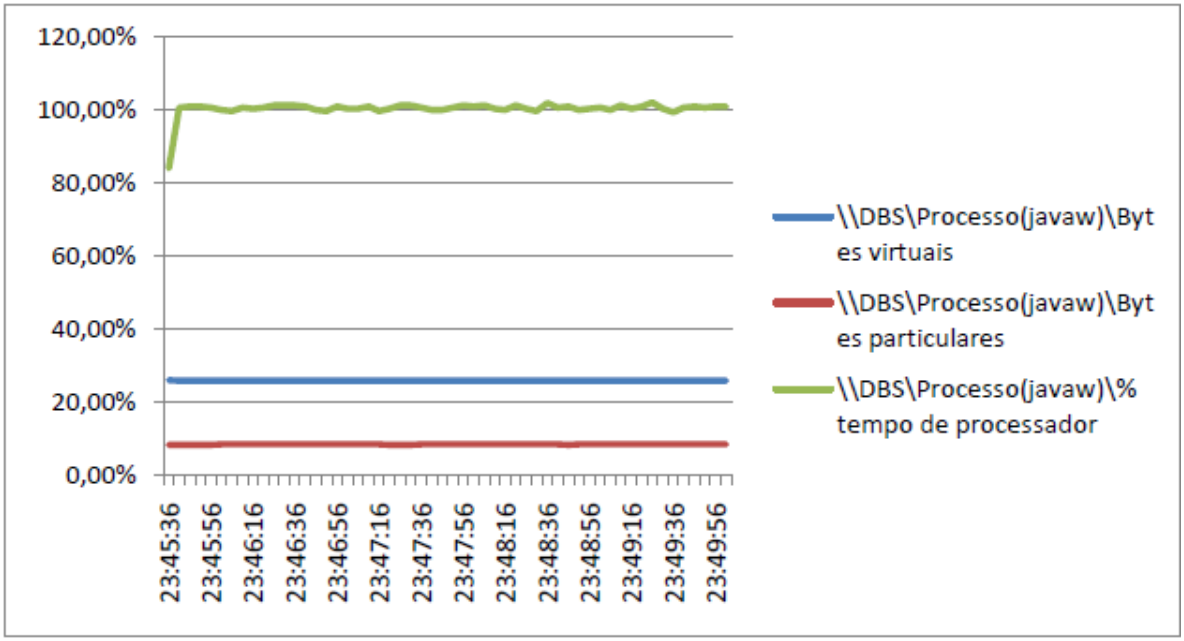

Figura 26 - Gráfico de desempenho da segunda execução do Repast J

O tempo total da execução da segunda implementação foi de 4 minutos e 25 segundos. De maneira similar à primeira execução o consumo de CPU se manteve constante em 100%.

O consumo de memória na segunda execução foi bastante similar ao da primeira execução. O mesmo padrão se repete devido a pouca diferença existente entre cada cenário em termos de quantidade de agente e novamente é necessário considerar o padrão de alocação de memória da máquina virtual Java.

9 Comparação entre duas ferramentas

Ambas as ferramentas apresentadas neste estudo possuem vantagens e desvantagens. Algumas destas foram apresentadas anteriomente quando da discussão de cada ferramenta. De maneira resumida uma análise entre estas vantagens e desvantagens será realizado nesta seção.

O NetLogo possui como maior vantagem a baixa curva de aprendizagem necessária para sua utilização. Conforme já mencionado esta ferramenta foi criada de maneira a propiciar um ambiente de estudo de modelagem baseada em agentes. Considerando este objetivo é importante ressaltar que o NetLogo pode ser recomendado como um ponto de

partida de estudos práticos sobre a àrea de agentes de *software* e modelagem baseada em agentes.

De maneira análoga é necessário ressaltar que o objetivo de simplificação da ferramenta acaba por limitar o seu escopo de utilização. Existem algumas restrições em relação a ferramenta como a entrada de dados e outros que limitam o seu uso. Dado estas limitações é passível somente a construção de modelos simplificados. Outra utilização da ferramenta seria para a construções de protótipos ou provas de conceitos envolvendo agentes de *software*.

O Repast J é uma ferramenta extremamente complexa que permite, por extensão, a construção de qualquer tipo de modelo baseado em agentes de software. O objetivo do Repast J, o qual é incluso dentro do Repast Simphony, é prover uma ferramenta avançada de criação de modelos de *software*, neste caso fornecendo uma implementação da ferramenta para tecnologia Java.

A curva de aprendizagem do *framework* é consideravelmente mais elevada em comparação a do NetLogo (Figura 27), mas possivelmente esta comparação em especial não pode ser feita devido aos objetivos diferentes de cada ferramenta. Independentemente disso é necessário ressaltar a simplicidade da documentação fornecida junto com a ferramenta que, apesar de não impedir por completo a sua utlização por um iniciante, definitavamente torna o uso excessivamente complexo numa primeira análise.

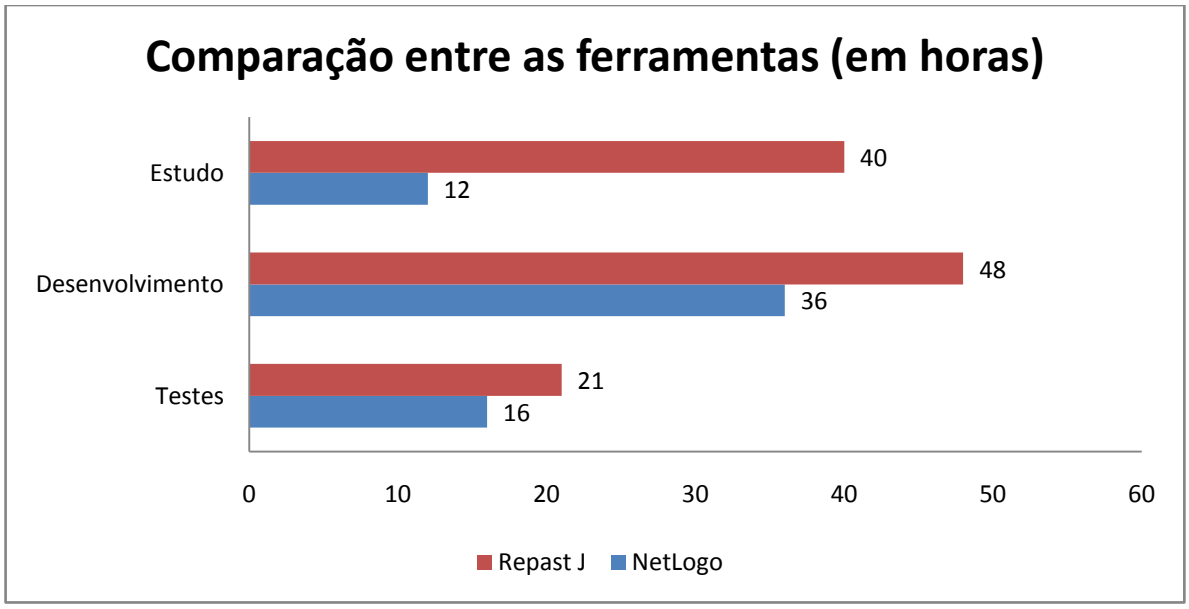

Figura 27 - Comparação do tempo de implementação

A API disponibilizada pela ferramenta é extensa e aparenta contemplar várias funcionalidades apesar de que as implementações criadas não terem feito uso excessivo delas.

Neste quesito existe uma clara vantagem do Repast J em relação ao NetLogo apesar do argumento sobre os objetivos distintos de cada ferramenta poder ser aplicado neste caso.

Em termos de performance é perceptível algumas similaridades entre as duas ferramentas. Ambas as ferramentas estressaram ao máximo a CPU da máquina utilizada para os testes. Isto demonstra que a utilização de um hardware que possibilite um melhor paralelismo real seria benéfico a ambas as ferramentas.

Apesar disto nota-se que o Repast J conseguiu concluir de maneira mais eficiente as tarefas necessárias para o modelo as realizando em menor tempo. Isto pode se atribuir ao fato de que o NetLogo necessita interpretar a linguagem Logo e então a traduzir para Java antes de executa-lá, desconsiderando neste caso a segunda tradução necessária entre bytecodes e o código nativo de máquina. No caso do Repast J somente é necessário realizar a tradução entre bytecodes e o código nativo de máquina.

No quesito de utilização de memória existem padrões diferentes para cada feramenta. O NetLogo demonstrou na primeira execução (Figura 16) um crescimento constante de utilização de memória enquanto que na segunda execução (Figura 17) o consumo de memória foi constante. Em ambos os casos nota-se um baixo consumo de memória o que é justificável devido a simplicidade do modelo implementado.

O Repast J demonstrou em ambas as execuções um consumo constante de memória para a sua execução (Figura 25 e Figura 26). De maneira similar ao NetLogo o consumo de memória foi bastante reduzido e a mesma justificativa se aplica. Apesar disso o consumo de memória do Repast J foi menor que o do NetLogo em média. Isto é justificável considerando a necessidade do NetLogo em realizar a tradução da linguagem Logo.

Tabela 3 - Comparação entre as ferramentas

Critério de comparação	NetLogo	Repast J
Facilidade de uso	Extremamente fácil de usar. Possui uma linguagem de programação de fácil aprendizado e uma interface simples e funcional.	Possui um framework complexo que dificulta o entendimento inicial. Possui uma interface familiar para quem já utilizou Eclipse. Se baseia na linguagem Java o que facilita seu uso.
Funcionalidades	Componentes gráficos de interação com o modelo. Suporte a visualização 2D e 3D. Possibilita obter gráficos em tempo real sobre a execução do modelo.	Framework extenso que disponibiliza várias funcionalidades aos implementadores. Suporte a visualização 2D e 3D utilizando JOGL. Baseada em Java.

Desvantagens	Não possui suporte a banco de dados. Não possui debug.	Documentação inicial confusa e mal organizada. Não disponibiliza bibliotecas necessárias a sua utilização.
Documentação	API documentada e exemplos iniciais bem compreensivos. Contém uma biblioteca de exemplos variados.	API documentada. Conforme mencionado documentação inicial confusa.
Uso destinado	Introdução a programação baseada em agentes. Implementação de modelos e protótipos simples.	Implementação robusta de modelos complexos que necessitem de robustez e performance.

10 Conclusão

A utilização da modelagem baseada em agentes é uma ferramenta útil para a simulação de fenômenos os quais não podem ser facilmente reproduzidos. Em especial a simulação social, e em específico a simulação de crescimento urbano, é beneficiada pela utilização de agentes de *software*. Ao se utilizar agentes de software é possível implantar regras a nível micro enquanto se observa em nível macro a interação entre os agentes.

O uso de agentes de *software* para desenvolver estes modelos também permite uma maior flexibilidade aos mesmos. A implementação da regra pode ser realizada somente uma vez enquanto se altera os dados sobre os quais os agentes irão agir. Desta maneira é possível construir diversos cenários utilizando o mesmo conjunto de regras e então executá-los para posterior estudo.

O desenvolvimento de um modelo de crescimento urbano utilizando parâmetros reais e seu posterior uso por uma prefeitura seria de extrema valia para a gestão municipal. A possibilidade de simular o crescimento urbano nas futuras décadas possibilitaria um melhor planejamento dos gastos públicos. Isto resultaria em melhores condições de infra-estrutura pública como hospitais, vias de tráfego e segurança sendo planejadas e implementadas de antemão.

Ambas as ferramentas apresentadas neste estudo permitem a construção de modelos baseados em agentes. Existem no entanto diferenças entre cada ferramenta os que as tornam interessantes para usos distintos. O NetLogo mostra-se uma ferramenta claramente voltada ao

meio acadêmico sendo mais indicada para o estudo inicial sobre agentes de software do que para construções de modelos complexos. Outra possível utilidade para esta ferramenta é para a construção de protótipos ou provas de conceito.

O Repast J no entanto é uma ferramenta mais madura e sendo desenvolvido por mais de uma década. Esta ferramenta é parte constituinte de outra ferramenta chamada Repast Symphony que tem como objetivo fornecer uma implementação sólida para a construção de modelos baseados em agentes. Ressalta-se inclusive a existência de uma implementação voltada para computadores de alto desempenho.

O *framework* disponibilizado pela ferramenta é complexo e possui uma grande quantidade de funcionalidades disponíveis para utilização. No entanto é necessário ressaltar que a documentação existente é simples e carece de detalhes que apesar de não impedir a sua utilização a torna mais penosa. Outro ponto negativo em relação a documentação da ferramenta é relativo as versões das bibliotecas a serem utilizadas em especial o JOGL. Esta bilbioteca não é fornecida junto com o Repast J e existem certas restrições em relação a versão da mesma que não são especificadas.

11 Trabalhos Futuros

Em específico a modelagem baseado em agentes e simulação social percebe-se uma grande área de estudo passível de ser explorada. Conforme já mencionado o impacto resultante da construção e utilização de um modelo robusto de crescimento urbano é imensurável para a gestão pública. Este diferencial na gestão pública resultaria em melhorias diretas para a população. A continuação de um estudo neste área no entanto necessita de estudos mais aprofundados na teoria de agentes de *software* os quais provalvemente seriam melhor conduzidos a nível de mestrado.

A utilização de agentes de software é ilimitada e pode ser expandida a diversas áreas. Através do estudo apresentado duas ferramentas, que se baseiam em Java, se destacam como sendo passíveis de serem utilizadas para o desenvolvimento de outras aplicações. Podemos citar como exemplo a utilização de agentes de software em sistemas colaborativos, simulações de interface humano máquina e outros. As ferramentas apresentadas neste estudo podem ser utilizadas para a criação destes modelos e implementações. A análise e comparação realizada entre cada ferramenta auxilia a decisão de qual ferramenta deveria ser empregada.

Futuramente é possível realizar novos estudos sobre cada uma das ferramentas. O Repast J se destaca por fornecer o maior potencial em termos de funcionalidade para a construção de modelos complexo já que disponiboliza várias funcionalidades. O NetLogo se destaca por ser um ferramenta acadêmica de notável capacidade educacional principalmente para usuários que nunca tiveram um contato com agentes de software.

12 Referências Bibliográficas

Agent-Based Modeling. Disponível em <http://en.wikipedia.org/wiki/Social_simulation#Agent-Based_Modeling>. Acesso em: 30 mai. 2005.

Agent-Based social simulation. Disponível em <http://en.wikipedia.org/wiki/Agent_based_social_simulation>. Acesso em: 30 mai. 2005.

BAKER, William L. (1989). A Review of Models of Landscape Change. Landscape Ecology, v. 2, n. 2, p. 11-33.

BARROS, J. X. (2004). Urban Growth in Latin American Cities: Exploring urban dynamics through agent-based simulation. Doctoral Thesis, Univesity of London, London.

CAMPOS, Milton C. C.; CARDOZO Nilceu P. e MARQUES JÚNIOR, José. (2006). Modelos de Paisagem e sua Utilização em Levantamentos Pedológicos, v. 6, n. 1, 1º Semestre de 2006.

CÂMARA, Gilberto e MONTEIRO, Antônio M. V. (2003). Introdução à Modelagem Dinâmica Espacial. São José dos Campos.

CORRÊA, R. L. (1995). O espaço urbano. Ed. Ática, São Paulo.

CASTELLS, Manuel. (1974). Lá cuestión urbana. 7ª ed., Madrid y Mexico, Siglo XXI.

CASTELLS, Manuel. (1983). The City and the Grassroots: A Cross-Cultural Theory of Urban Social Movements. Editora da Universidade da Califórnia, Berkeley, Califórnia.

COUCLELIS, Helen. From Cellular Automata to Urban Models: New Principles for Model Development and Implementation. Environment and Planning B: Planning and Design, 1997, v. 24, 165-174.

DAVIDSSON, Paul. Multi Agent Based Simulation: Beyond Social Simulation. LINCS vol. 1979, Springer Verlag.

EPSTEIN, J. M. e AXTELL, R. (1996). Growing Artificial Societies: Social Science from the Bottom Up. Brooking Institution Press, Washington DC, EUA.

GODOY, Marcela M. G. Modelagem Dinâmica de Ocupação do Solo no Bairro Savassi, Belo Horizonte, Brasil. 2004. 84 f. Dissertação (Mestrado em Geografia) - Universidade Federal de Minas Gerais, Belo Horizonte, 2004.

HARVEY, David. (1975). Class Structure in a Capitalist Society and the Theory of Residential Differentiation. In Processs in Physical and Human Geography, p. 354-369, Londres.

LANA, Raquel M. Modelos Dinâmicos Acoplados para Simulação da Ecologia do vetor do Aedes aegypti. 2009. 89 f. Dissertação (Mestrado em Ecologia de Biomas Tropicais) - Universidade Federal de Ouro Preto, Ouro Preto, 2009.

NetLogo. Disponível em <http://ccl.northwestern.edu/netlogo/>.

MARETO, Raian V.; ASSIS, Talita O. e GAVLAK, André A. (2010). Simulating Urban Growth and Residential Segregation through Agente-Based Modeling, Brazilian Workshop on Social Simulation.

NORTH, M.J.; COLLIER N.T.; VOS J.R. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Transactions on Modeling and Computer Simulation, Vol. 16, ed. 1, p. 1-25, ACM, Nova Iorque, Nova Iorque, EUA, Janeiro de 2006.

PEDROSA, Bianca M. e CÂMARA, Gilberto. (2002). Modelagem Dinâmica e Geoprocessamento. INPE. São Paulo.

Repast J. <http://repast.sourceforge.net/index.html>.

SABATINI, F.; CÁCERES, G.; CERDÁ, J. (2001). Segregación residencial em las principales ciudades chilenas: tendencias de las tres últimas décadas y posibles cursos de acción. EURE, vol. 27, p. 21-42, Santiago, Chile.

SANTOS, Milton. (1996). A Natureza do Espaço: Técnica e Tempo, Razão e Emoção. São Paulo. HUCITEC.

SCHELLING, T. C. (1971). Dynamic models of segregation. Journal of Mathematical Sociology, vol. 1, p. 143-186.

SJOBERG, G. (1960). The Pre Industrial City: past and present, American journal of sociology, p. 143-186.

SOARES-FILHO, Britaldo S. (1998), Análise de Paisagem: Fragmentação e Mudanças. Departamento de Cartografia. Centro de Sensoriamento Remoto. Instituto de Geociências. Universidade Federal de Minas Gerais, Belo Horizonte.

SOARES-FILHO, Britaldo S.; CERQUEIRA, Gustavo C.; ARAÚJO, William L. e VOLL, Eliane. (2003). Modelagem de Dinâmica de Paisagem: Concepção e Potencial de Aplicação de Modelos de Simulação baseados em Autômato Celular.

SOARES-FILHO, Britaldo S. e TEIXEIRA, Gustavo G. (2009). Simulação de tendência de desmatamento nas Cabeceiras do Rio Xingu, Mato Grosso - Brasil, Anais XIV Simpósio de Sensoriamento Remoto, INPE, p. 5483-5490, Natal, Brasil, Abril de 2009.
Software Agent. Disponível em <http://en.wikipedia.org/wiki/Software_agent
http://agents.umbc.edu/introduction/ao/>. Acesso em: 30 mai 2005.

TEIXEIRA, Gustavo G. e SOARES-FILHO, Britaldo S. (2009), Simulação da tendência do desmatamento nas Cabaceiras do Rio Xingu, Mato Grosso - Brasil, Anais XIV Simpósio Brasileiro de Sensoriamento Remoto, p. 5483-5490.

UMBELINO, Glauco. e BARBIERI, Alisson (2010), Uso de Autômatos Celulares em estudos de população, espaço e ambiente, Associação Brasileira de Estudos Populacionais.