Valter Jorge da Silva

RESOURCE SHARING BEHAVIORS: Modelagem e Implementação de Simulador para Comportamentos de Divisão de Recursos e Teoria dos Jogos Centrado no Usuário

RECIFE-PE
2016

Valter Jorge da Silva

RESOURCE SHARING BEHAVIORS: Modelagem e Implementação de Simulador Focado no Usuário para Comportamentos de Divisão de Recursos e Teoria dos Jogos

Dissertação de mestrado apresentada ao curso de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Informática Aplicada.

Área de Concentração: Computação Inteligente e Modelagem.

Orientador: Dr. Giordano Ribeiro Eulalio Cabral

RECIFE-PE
2016

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Nome da Biblioteca, Recife-PE, Brasil

S586r Silva, Valter Jorge da
Resource Sharing Behaviors: modelagem e implementação de simulador para comportamentos de divisão de recursos e teoria dos jogos centrado no usuário / Valter Jorge da Silva . – 2016.
118 f. : il.

Orientador(a): Giordano Ribeiro Eulalio Cabral .

Dissertação (Mestrado) – Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, BR-PE, 2016.
Inclui apêndice(s) e referências.

1. Simulação (Computadores) 2. Comportamento humano 3. Teoria dos jogos I. Cabral,Giordano Ribeiro Eulalio, orient. II. Título

CDD 004

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA

VALTER JORGE DA SILVA

RESOURCE SHARING BEHAVIORS

Modelagem e Implementação de Simulador para Comportamentos de Divisão de Recursos e Teoria dos Jogos Centrado no Usuário

Dissertação julgada adequada para obtenção do título de mestre em Informática Aplicada, defendida e aprovada por unanimidade, em 30/08/2016, pela Comissão Examinadora.

Orientador:

_____
Prof. Dr. Giordano Ribeiro Eulálio Cabral
DEINFO/UFRPE

Banca Examinadora:

_____
Prof. Dr. Tiago de Alessandro Espínola Ferreira
DEINFO/UFRPE

_____
Prof. Dr. Geber Lisboa Ramalho
Cin/UFPE

_____
Prof. Dr. Leonardo Rodrigues Sampaio
LD-APP/UNIVASF

Dedico este trabalho à minha noiva Angelica e aos meus pais Jorge e Ivonete por todo incentivo para que eu avançasse na vida.

AGRADECIMENTOS

Agradeço inicialmente a Deus. Aos meus pais, Jorge e Ivonete, com o apoio literal nesta etapa, mesmo estando ambos debilitados. Agradeço a minha noiva Angelica, pela paciência e motivação. Agradeço ao meu orientador Giordano, grande visionário e professional, pela paciência, motivação e direcionamento fornecidos, mesmo em momentos em que passei por problemas sérios de saúde. Agradeço ao pesquisador Guilherme pela ajuda e direcionamento dado durante a elaboração deste trabalho. Agradeço aos amigos e colegas de apartamento Carlos, Emanoel, Joseph e Micael, pelo apoio moral nestes mais de dois anos de mestrado e quatro de graduação. Agradeço à equipe de funcionários do Programa de Pós-Graduação em Informática Aplicada, especialmente ao professor Tiago Ferreira e secretário Eduardo Chaves. Agradeço a todos que contribuíram, diretamente ou indiretamente para a conclusão deste trabalho.

“Quem conhece os outros é inteligente.
Quem conhece a si mesmo é sábio.
Quem vence os outros é forte.
Quem vence a si mesmo é poderoso. ”

(Miyamoto Musashi)

RESUMO

Este trabalho objetiva a criação de uma ferramenta que simule comportamentos entre agentes ao realizarem atividades de divisão de recurso através de jogos típicos da Teoria dos Jogos. O tema de divisão de recursos é bastante estudado entre várias áreas de conhecimento, como Psicologia, Administração e Computação, possuindo várias abordagens ao tema. Uma dessas abordagens é a simulação de modelos de agentes comportamentais realizando as atividades de divisão de recursos. Existem vários problemas na utilização desta abordagem, dependendo da área de pesquisa do usuário, mas geralmente, a maior dificuldade é a disponibilidade de uma ferramenta que possua flexibilidade unida a uma interface gráfica simples e direta como pontos chaves de desenvolvimento. Durante a pesquisa bibliográfica, foram encontradas ferramentas que permitiam ao usuário realizar um experimento de forma facilitada, mas não o permitia alterar as regras ou modelos do experimento. No outro extremo, foram encontradas ferramentas que permitiam ao pesquisador ser flexível, mas com uma complexidade inviável para pessoas sem conhecimento de programação. Então, foi decidido realizar um estudo do estado da arte das ferramentas e modelos aplicados para a simulação destes problemas, objetivando a criação de uma nova ferramenta que atendesse as duas demandas. Para isto, todo o processo de implementação da ferramenta foi acompanhado por um auditor especialista no problema, o qual deveria testar a ferramenta e apontar vantagens e desvantagens para realizar a evolução sistemática do programa. A cada versão foram aprendidas lições do que se deve e não deve ser aplicado a modelagem para se chegar ao resultado esperado. Após a implementação da ferramenta, foi obtido uma ferramenta capaz de simular comportamentos, podendo ser utilizado por pessoas com experiência no tema de Divisão de Recursos e Teoria dos Jogos e sendo flexível para a adição de novas regras, jogos ou funcionalidades para usuários conhecedores de programação. Então, foram realizados vários testes com a finalidade de verificar a corretude das saídas do simulador, sendo este repassado para o auditor e um pequeno grupo de usuários finais testarem e comentarem sobre a ferramenta, fornecendo sugestões e críticas a fim de evoluir a ferramenta. Por fim, foram discutidas as lições aprendidas e as propostas de trabalhos futuros para realizar o aprimoramento da ferramenta.

ABSTRACT

This work aims to create a tool that simulates behavior among agents conducting resource sharing activities through typical games of Game Theory. The Resource Division theme is well studied among various knowledge areas such as Psychology, Management and Computing, counting with various approaches. One of that approach is the simulation of behavioral agents models performing the resource division activities. There are several problems in using this approach depending on the user's search area, but generally, the greatest difficulty is the availability of a tool that has ease of use and flexibility as key points of development. During the literature search, some tools was found that allow the user to carry out an easier way to experiment, but they not allowed the use to change the rules or modeling of the experiment. At the other end, tools were found allowing the researcher to be flexible, but has an unworkable complexity for people without programming knowledge. So it was decided to conduct a study of the state of the art about tools and models for the simulation of these problems, aiming to create a new tool that meets both demands. For this, the entire tool deployment process was accompanied by an expert in the theme, which should test the tool and point advantages and disadvantages to perform systematic evolution of the program. With each version, lessons were learned on what it should and should not be applied to modeling the agents to reach the expected result. After completion of the tool, the result was obtained a simple and powerful tool for the simulation of these behaviors who can be used easily by anyone with experience in the fields of Resources Division and Game Theory, allowing flexibility for adding new rules, games or features, on a easy way, for programming connoisseurs. So several tests were performed in order to verify the correctness of the outputs of the simulator, which was passed on to the auditor and a small group of end users to test and comment on the tool. Finally, we discussed the lessons learned and proposals for future work to carry out, which the objective of the improvement of the tool.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1 INTRODUÇÃO

Atualmente, com o avanço da computação e de outras tecnologias da informação, os seres humanos estão utilizando várias ferramentas em seu cotidiano para diminuir a quantidade de esforço necessário ou automatizar uma dada atividade.

O uso dessas tecnologias e ferramentas atinge várias áreas cientificas, mas como a evolução dessas tecnologias estão interligadas a própria evolução humana, é fato que em algumas dessas áreas ocorrerá uma deficiência em nível quantitativo ou qualitativo para a resolução de problemas, gerais ou específicos.

Existe um tema que é objeto de interesse entre estas várias áreas cientificas, chamado de Divisão de Recursos. O tema de divisão de recursos aborda problemas que envolvem a troca de recursos entre agentes. O tema pode ser aplicado em várias áreas, como as áreas sociais, cognitivas, econômicas, políticas, bélicas e computacionais. Um exemplo de sua aplicação seria a análise do comportamento de pessoas ao realizar a divisão de um valor comum e de interesse de ambos, podendo esta divisão gerar atrito entre os envolvidos.

Quando abordamos o tema de divisão de recursos, existem vários problemas que se utilizam da Teoria dos Jogos na tentativa de se chegar a uma solução viável. Para alguns problemas, existem ferramentas que propõem auxiliar na execução de certas atividades para a sua solução. Como exemplo, existem softwares que realizam a atividade do Jogo do Ultimato entre duas pessoas com o intuito de analisar o comportamento dos jogadores (SOPHIE, 2016). Em contrapartida, existem softwares responsáveis por simular a atividade, com as modelagens dos possíveis comportamentos, partindo para outra abordagem do tema. (MILLISECOND, 2016).

Como citado acima, encontra-se com frequência ferramentas que tentam resolver através de suas atividades um problema especifico para uma das duas possibilidades citadas acima, observar ou simular. Mas a forma em que estes

programas são criados tendem a gerar um outro problema, a qual podemos chamar de especificidade da solução.

Esta especificidade da solução ocorre devido ao fato de um programa criado para a solução de um certo problema não permitir o seu reuso para outras situações, mesmo que estas situações possuam um conjunto de regras de execução ou semelhança considerada. Este problema geralmente ocorre em problemas que envolvem atividades com conjuntos de regras especificas, como um experimento de simulação de comportamento. Exemplificando, para o cenário de observação, a metodologia ou ferramenta dificilmente pode ser reutilizada devido a diferença de pensamento e estratégias do grupo envolvido na atividade. Para o cenário de simulação, são poucas as ferramentas que simulem o comportamento dos agentes envolvidos na atividade em problemas modelados pelo usuário através do próprio programa. Ou seja, dificilmente encontramos uma tecnologia ou ferramenta que trate ou auxilie um ou vários problemas de forma automática, sendo encontradas geralmente ferramentas criadas e orientadas a um caso especifico.

Em conjunto com as dificuldades apresentadas, o pesquisador ao buscar alguma ferramenta para usar em seu experimento, percebe uma carência de ferramentas maleáveis, ou seja, que possam ser modificadas para uso no seu trabalho. Como resultado da busca, é possível encontrar algumas ferramentas, como as citadas no capitulo 3, de Estado da Arte, mas devido ao fato que estas ferramentas foram desenvolvidas com a deficiência da unicidade da solução, este pesquisador não consegue a utilizar da forma desejada.

Visto os problemas acima citados, foi observada a necessidade de criação de ferramentas e tecnologias para simular comportamentos de certas atividades de divisão de recursos utilizando as abordagens explanadas. Devido a isto, visando a abordagem de simulação, desenvolvemos este trabalho com os seguintes objetivos:

Objetivo Geral:

- Modelar e implementar um sistema de simulação comportamental para jogos típicos da área de divisão de recursos que atenda aos requisitos de flexibilidade e precisão.

Objetivos Específicos:

- Desenvolver simulador de jogos típicos de divisão de recursos utilizando uma abordagem que permita a sua modularização;

- Abstrair a necessidade de uso de programação no programa pelo usuário na elaboração e realização de experimentos;

- Permitir a alteração facilitada do programa, via código, para que programadores consigam alterar o simulador sob demanda.

Para atingir estes objetivos, primeiramente foi realizada uma revisão bibliográfica das áreas de Divisão de Recursos, Teoria dos Jogos e Modelagem de Comportamentos Humanos, disponível no capitulo 3. Esta revisão aconteceu para entendermos melhor o que deveria ser abordado no modelo de software e como os potenciais usuários criariam os modelos mentais das atividades e simulações.

Para dar suporte a modelagem do sistema, também foram analisados documentos científicos das áreas correlatas ao tema, que possuam ligação direta a uma possível solução. Nesta análise foi verificado documentos das áreas de Computação Paralela, Computação Inteligente, Inteligência Artificial, entre outros.

Então, foi realizado uma pesquisa da evolução dos modelos até chegarmos no estado da arte, também disposta no capitulo 3. Foram analisados os modelos utilizados antes de sua aplicação se passar através de computadores, os quais possuíam seu foco principal de solução em economia e política, por exemplo. Passamos então para os modelos computacionais iniciais e atuais de softwares utilizados para executar, analisar e/ou simular as atividades de divisão de recursos.

Logo após a finalização da pesquisa dos modelos, os resultados obtidos foram usados como meios de idear os componentes necessários para realizar uma modelagem que permitisse a customização dos dados e regras das atividades propostas pelo usuário, utilizando como motivação e base o modelo do *Steering Behavior*, que é uma técnica de automação de movimentos em Inteligência Artificial, encontrado também no estado da arte.

Com os estudos e a pesquisa bibliográfica finalizadas, foi desenvolvido o simulador, contendo a modelagem de alguns jogos típicos, como o Dilema do Ditador (Kahneman et al, 1986) e o Jogo do Ultimato (Güth et al, 1982), ambos explicados posteriormente. Para o desenvolvimento, foram utilizadas as ideações feitas e os modelos pesquisados como pontos chave para a sua criação, procurando contemplar as funcionalidades cruciais das outras ferramentas encontradas.

Como ponto de inovação, foi possibilitado a customização dos modelos dos agentes dentro do programa, permitindo assim que cada um destes agentes possua um perfil específico dentro de várias possibilidades, aumentando o poder e importância da elaboração, simulação e análise dos dados gerados pelas atividades dos jogos típicos. A abordagem realizada para a customização está descrita no capitulo 4.

Com a modelagem de software e a modelagem dos jogos ideadas, foi implementado o modelo de software utilizando a linguagem de programação C#, o qual utiliza o paradigma de orientação a objetos, permitindo assim o tratamento de cada agente como um indivíduo especifico em toda a população. Foi detalhado como foi feita a diagramação de classe e sua implementação para a replicação do experimento e como foi possibilitado a customização das atividades sem a necessidade de alteração de código por parte dos usuários, possibilitando assim possíveis adendos que possam vir a ser necessários por parte do usuário.

Por fim, os modelos elaborados foram simulados e analisados, observando os comportamentos dos agentes diante de várias possibilidades de atividades que possam vir a surgir em cada jogo, detalhando o que poderia acontecer com cada indivíduo caso este fosse um indivíduo real. Esta simulação e analise foi realizada em

conjunto com pesquisadores da área de Divisão de Recursos e Psicologia Cognitiva, permitindo assim a evolução do simulador para atender melhor as necessidades do usuário.

2 PROBLEMA

Neste capitulo está descrito o problema que levou a pesquisa originaria deste trabalho. Aqui é definido formalmente o problema, demonstradas as dificuldades previstas para a sua solução e as tendências de soluções.

2.1 DEFINIÇÃO FORMAL DO PROBLEMA

Desde o início do interesse de estudos de cooperação até a atualidade, cientistas de várias áreas, como ciências sociais, psicologia, biologia, administração e posteriormente da computação, demonstraram a dificuldade de se realizar experimentos de divisão de recursos com pessoas, especificamente quando estes possuem como objetivo a análise do comportamento dos participantes ou os resultados de suas atividades. Em conjunto com essa dificuldade, após o início da guerra fria, surgiu uma necessidade crescente de se realizar estes experimentos em atividades de nível crítico, como em tomadas de decisão para política ou ações militares. Estes experimentos eram necessários quando a elaboração e execução de um experimento a fim de observar o comportamento eram impossibilitados devido ao impacto direto da ação e a necessidade de se enfrentar um oponente real sem o conhece-lo, podendo acarretar retaliações indesejadas com impactos sérios no meio envolvido.

Com o surgimento dos computadores, observou-se a possibilidade de utilizar os fundamentos e lições aprendidas nas observações realizadas com o objetivo de criar programas que conseguissem simular tais comportamentos e atividades, com o fim de se ter uma previsão do que poderia acontecer em situações críticas sem a necessidade de se correr o risco de uma retaliação na tomada errônea de uma decisão.

Com o sucesso das simulações iniciais, estudiosos utilizaram os conceitos aprendidos, criando simuladores para executar várias atividades modeladas pelo usuário a fim de se obter resultados prévios e observar o possível comportamento dos agentes. Com isso, os cientistas tiveram a possibilidade de experimentarem de forma

segura o comportamento de agentes em uma certa atividade a procura de formulas que otimizassem este comportamento no mundo real.

Apesar do sucesso de uso dos programas simuladores, existem alguns problemas oriundos da modelagem da atividade fora do simulador. O primeiro é a modelagem computacional do que pode ser o “ruído”. Segundo Axelrod (1997), ruído é uma ocorrência ou variável que acarreta na discrepância de interpretação da mensagem entre dois agentes. Por exemplo, o agente *a1* envia uma mensagem considerada positiva para o agente *a2*, mas o agente *a2* a interpreta de modo negativo, devido a uma ocorrência “ruído” acontecida no meio de transmissão da mensagem ou na diferença cultural entre os dois agentes.

O segundo problema é a especificação da modelagem dos agentes, ou seja, como eles se comportariam, já que o mesmo pode ser especificado como um agente unicelular com comportamentos simples e extintivos ou um agente complexo em nível social, como grupos e organizações.

A terceira é a necessidade de criação de uma ferramenta especifica para cada atividade, surgindo o problema da especificidade de solução, impossibilitando assim a reutilização do simulador para outras atividades semelhantes, mas com um conjunto de regras ou modelagem distintas. Ambos os problemas geram um impacto comum de grande peso, a complexidade na codificação e uso do software por pessoas que não seja da área de computação. Essa codificação do software necessita de alguns aspectos que o tornam ainda mais complexo, que são:

- Flexibilidade: A capacidade de generalizar os experimentos, para um ou vários conjuntos de atividades com características distintas, mesmo em um número elevado de agentes;

- Corretude: A capacidade da ferramenta de fornecer o mesmo conjunto de resultados em todas as saídas que possuírem o mesmo conjunto de entradas;

- Facilidade de uso: A capacidade do pesquisador utilizar a ferramenta de uma forma fácil, com uma curva de aprendizagem pequena;

- Precisão: A capacidade de se obter resultados concretos e comparáveis com modelos abstratos, não permitindo analises ambíguas desses resultados.

Seguindo as premissas do problema com o objetivo de seguir os fatores acima citados, identificamos e explanamos o seguinte problema: Apesar de encontrarmos ferramentas em que conseguimos simular os comportamentos de vários agentes em um ambiente composto por regras especificas na área de Divisão de Recursos, dificilmente encontramos uma que permita de forma abstraída a alteração dos parâmetros dos agentes e das regras pelo usuário e/ou especialista, sendo este não conhecedor de linguagem de programação.

2.2 DIFICULDADES EM RESOLVER O PROBLEMA E SUA RELEVÂNCIA

Somente o fato de existir a possibilidade de modelar um software com esses requisitos e impacto na sociedade para simulação já seriam fatores que podem considerar suficientes para a sua elaboração. Mas, caso observemos o fato da ausência de simuladores que ofereçam a flexibilidade desejada por esse projeto, visto que geralmente encontramos apenas ferramentas específicas para cada caso, seria uma adição suficiente para adicionarmos algo útil a sociedade

Para avaliação da complexidade e dificuldade de resolução do problema, podemos analisar a problemática de forma sequencial. Segundo Sommerville (2011), para cada métrica utilizada, a complexidade de modelagem e a engenharia para se manter o *trade off* entre essa complexidade e a usabilidade são maiores. S*e* listarmos primeiramente os requisitos necessários para gerar e manter as quatros métricas acima citadas, já teríamos uma dificuldade inicial de modelagem do software que podemos analisar de nível moderado.

Ao adicionarmos a essa analise a quantidade de modelos possíveis de serem criados para um agente, considerando seus vários tipos de comportamentos (altruísta,

egoísta, etc.), sendo que cada modelagem comportamental pode conter vários valores específicos dependente do tipo e objetivo da simulação, estaremos acrescentando outro fator impactante na complexidade de modelagem do software, necessitando de mais esforço para manter o *trade off* sem adentrar no problema da unicidade de solução.

Ao acrescentarmos a esta modelagem o fato de que nos jogos econômicos, cada agente possui sua “personalidade”, e além dela pode estar seguindo uma estratégia, ocasionando em resultados diferenciados, aumentamos novamente a complexidade, visto a necessidade de inserção deste traço de comportamento.

Finalizando, a criação de um simulador deste tipo permitiria ao pesquisador conseguir informações em um curto espaço de tempo e com pouco esforço, se comparado a realização de um experimento de observação. Essa facilidade ainda poderia permitir que o pesquisador validasse os modelos capturados pelos experimentos de observação, e de realizar simulações quando este tipo de experimento não é possível.

Todo este avanço seria possível ao criar um simulador que possa ser executado em um computador comum da atualidade, podendo ter seu poder acrescentado a desejo do usuário, como citado anteriormente, podendo ter um grande ganho de performance com o uso da computação paralela, por exemplo.

2.3 TENDÊNCIAS DE SOLUÇÕES

Considerando o possível impacto na sociedade de simular experimentos sociais em massa, evitando problemas éticos e retaliações desnecessárias, e mesmo diante das dificuldades encontradas na sua concepção, a modelagem e implementação de um software capaz de simular várias atividades, com várias possibilidades de perfis de agentes permitirá verificar como ocorre a divisão de recursos em grupos de agentes. Esta possibilidade é possível mesmo que este grupo de agentes possua especificações abstratas e distintas, sendo possível realizar várias análises, dependendo do objetivo do experimento. Um exemplo de possibilidade é como evoluir a cooperação entre grupos de pessoas com perfis distintos.

Com uma ferramenta dessa disponível à sociedade, os estudiosos da área poderiam modelar suas atividades específicas com pouca ou nenhuma alteração da sua modelagem ou da própria ferramenta. Caso a ferramenta não seja viável para uma certa atividade, os possíveis interessados poderiam evoluí-la, adicionando módulos que possam cobrir as lacunas existentes, ou então pode-se utilizar a modelagem conceitual para criar outra ferramenta em um espaço menor de tempo, desde que a pessoa que o faça possua conhecimentos de programação.

Por fim, o executor do experimento teria uma ferramenta capaz de chegar a uma solução com o uso de mais recurso de máquina, tempo, ou ambos, a ser decidido pelo modelo de atividade criado pelo usuário. Logo, como possibilidade de resultado, tem-se a uma ferramenta que a ser utilizada por pessoas das diversas áreas que abordam o tema de divisão de recursos, como Psicologia, Computação e Administração, permitindo também a sua modularização e evolução para os conhecedores de linguagem de programação.

3 ESTADO DA ARTE

Criar uma ferramenta que simule as atividades de um jogo econômico não é uma ideia inovadora tampouco recente. Antes mesmo de surgir o computador alguns pesquisadores já pensavam em como automatizar as atividades de cooperação em áreas como Economia, Política e Psicologia. Ou seja, estudar o impacto e a forma que são realizadas atividades de divisão de recurso e cooperação precedem o uso de computadores para qualquer tipo de simulação. Devido a isto, foi realizada a revisão bibliográfica do tema, com o objetivo de levantarmos o estado da arte e os conceitos iniciais para realizar a abordagem do assunto de forma satisfatória. Este levantamento foi realizado por meio de pesquisas em sites de pesquisa de trabalhos acadêmicos, como o Portal de Periódicos da CAPES, o Google Scholar e a Nature.

3.1 TEORIA DOS JOGOS

Segundo Myerson (1991), Teoria dos Jogos é o estudo de modelos matemáticos para o conflito e cooperação entre agentes tomadores de decisão racionalmente inteligentes. A Teoria dos Jogos é um estudo multidisciplinar, envolvendo principalmente as áreas de Economia, Ciências Políticas, Psicologia, Ciência da Computação e Biologia.

Na prática, a Teoria dos Jogos contempla basicamente conjuntos de jogos econômicos de soma-zero bem definidos, que envolvem situações sociais modeladas matematicamente com um valor associado. (SALEN; ZIMMERMAN, 2012). Jogos de soma-zero são jogos nos quais o lucro ou pontuação de um jogador acarretará em um prejuízo de mesmo valor para os outros jogadores, totalizando um valor zero, como o jogo do xadrez.

Atualmente, os estudos continuam em um ritmo acelerado para a área de Teoria dos Jogos. Surgiram novos tipos de dilemas e jogos, mas devido à variedade existente, citamos abaixo os considerados importantes para a nossa pesquisa:

- Dilema do Prisioneiro: Dois indivíduos são presos pela polícia. A polícia sem provas suficientes para condená-los, separam os indivíduos, os quais não terão contato um com o outro, e oferecem a ambos o mesmo acordo: o indivíduo terá duas opções, testemunhar contra o outro indivíduo, condenando-o ou ficar em silêncio. Caso um indivíduo testemunhe e o outro fique em silêncio, o primeiro fica livre e o segundo ficará preso por um tempo X. Caso ambos testemunhem, os dois ficarão presos por um tempo $Y < X$. E caso ambos silenciem, os dois ficarão presos por um tempo $Z < Y$.
- Jogo do Ultimato: Um órgão doa um valor para um indivíduo A, para este indivíduo receber esse o valor possui a seguinte condição: O órgão promoverá uma atividade indireta, no qual os indivíduos não se veem e não sabem que é o outro. A atividade será realizada entre o indivíduo A e um indivíduo B. O indivíduo A deverá propor uma oferta de divisão do valor para o indivíduo B, caso o indivíduo B aceite, ambos recebem o valor proposto, caso não aceite, ninguém recebe valor algum.
- Jogo do Ditador: Uma variação do Jogo do Ultimato, no qual o indivíduo A decide a razão de divisão do valor sem interferência do indivíduo B.

Seguindo uma linha cronológica, devido a sua multidisciplinaridade, não se sabe ao certo quando surgiu os estudos de seus conceitos, visto a sua ligação direta com o desenvolvimento da sociedade e da cultura. Mas para os pesquisadores da área, o marco inicial da Teoria dos Jogos foi definido por John Von Neumann ao publicar o seu *paper* "*Zur Theorie der Gesellschaftsspiele"* (NEUMANN, 1928). Neste artigo, Neumann provou através do Teorema do ponto fixo de Brouwer a existência do equilíbrio estratégico em jogos de soma-zero entre duas pessoas, mesmo quando estas utilizam estratégias mistas. Mas o termo Teoria dos Jogos surgiu quando ele publicou seu artigo "*Theory of Games I, general foundations",* se popularizando após a publicação do livro "*Theory of Games and Economic Behavior"* escrito por Von Neumann em conjunto com Oskar Morgenstern em 1944.

Após este livro, vários pesquisadores utilizaram os conceitos definidos por Von Neumann, evoluindo a área, e através da Teoria de Jogos vários destes pesquisadores conseguiram a proeza de receber o Prêmio Nobel, sendo o mais

conhecido pela sociedade o Matemático John Nash, criador do conceito chamado “Equilíbrio Nash”.

Com o passar dos anos, a área de Teoria dos Jogos evoluiu consideravelmente, sendo criados vários tipos de jogos econômicos e várias estratégias para estes. Em conjunto com a evolução da Teoria dos Jogos, várias outras áreas e tópicos surgiram e evoluíram, mas os conceitos que mais se conectaram a esse período foram a Economia e Psicologia, focando nos estudos sobre a cooperação. Este período nós podemos chamar de período “pré-computação”.

Após o surgimento dos computadores, alguns pesquisadores da área se aproveitaram do poder, inicialmente limitado, desta nova tecnologia para realizar pesquisas envolvendo a Teoria dos Jogos e a cooperação de indivíduos. Robert Axelrod, em 1984, publicou um livro, intitulado “*Evolution of Cooperation”* no qual ele define a sua Teoria da Cooperação, iniciando o seu livro com a seguinte pergunta: “Em que condições a cooperação surgirá num mundo de egoístas, sem uma autoridade central? ” (AXELROD, 2010). Neste livro, o autor usa da sua pergunta e se utiliza dos conceitos do Dilema do Prisioneiro da Teoria dos Jogos, para criar um torneio de computador no qual o objetivo era avaliar qual estratégia se saia melhor entre todos os programas. Com os resultados obtidos, ele realizou uma análise crítica dos resultados e definiu a sua Teoria da Cooperação, com métricas e conceitos para criar a cooperação em ambientes não suscetíveis e manter a cooperação quando presente.

Em 1997, Axelrod publicou o seu segundo livro, intitulado “*The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration”*. Nesta obra, o autor se utiliza de coleção de artigos de sua autoria nas áreas de Teoria dos Jogos, Teoria da Complexidade e Computação, para atualizar os avanços realizados até o momento de sua publicação nestas áreas. No livro, o autor comenta cada artigo detalhadamente, com métricas e conceitos a serem utilizados para criar os modelos dos agentes dos experimentos utilizados, além dos conceitos ótimos da época para criar programas que possam ser utilizados para obter os resultados desejáveis na área (AXELROD, 1997). Tanto a obra de 1984 quanto o livro de 1997 são utilizadas como bibliografia necessária pelos pesquisadores iniciais como uma boa base teórica na

área de Teoria dos Jogos, quando o assunto envolver o estudo da cooperação e sua modelagem.

Em conjunto com o avanço da Teoria dos Jogos e sua aplicabilidade em Computação, outra área que conseguiu se utilizar desses estudos para evoluir foi a Biologia. No início da década de 30, os biólogos já utilizavam conceitos que seriam utilizados posteriormente por pesquisadores como base dos estudos a serem incluídos na Teoria dos Jogos, relacionados a cooperação das espécies, antes mesmo do surgimento da Teoria dos Jogos. Um exemplo disto é o trabalho *"The Genetical Theory of Natural Selection"* (FISHER, 1930), no qual o autor apresenta o Teorema Fundamental da Seleção Natural e a Evolução dos membros dominantes, a qual teria impacto direto na definição de alguns conceitos da Teoria dos Jogos.

Apesar do início precoce na Biologia, a Teoria dos Jogos viria a ser amplamente utilizada na área na década de 70, tendo como principal contribuição o livro "*The Selfish Gene"*, de Richard Dawkins (1976). Neste livro, Dawkins demonstra através da visão evolutiva centrada nos genes que, mesmo em nível genético, existem cooperação e altruísmo entre indivíduos semelhantes quando o objetivo é a evolução ou sobrevivência dos genes. Com isto as populações genéticas tendem a utilizar uma estratégia evolucionariamente estável, o qual também é explicado nos livros de Axelrod. Atualmente, em sua 3ª edição (comemorativa de 30 anos), publicado em 2006, Dawkins adiciona os conceitos explanados por Axelrod, utilizando o livro "*The Evolution of Cooperation"* como uma das bases teóricas, reforçando assim os seus resultados. Este livro, até o momento da finalização deste trabalho, é cotado como o maior Best Seller na área de Genética, segundo a empresa Amazon (2016).

Como pesquisas importantes para este trabalho, podemos citar além dos referenciais acima os trabalhos recentes abaixo.

Em seu relatório cientifico, Ichinose e Sayama (2014) apresentam os resultados obtidos em experimentos científicos que utilizaram o Jogo do Ultimato, demonstrando como ocorre a evolução da cooperação neste tipo de abordagem e quais são os resultados esperados para esta ocasião. Eles demonstraram que é possível realizar a cooperação entre dois agentes no jogo do Ultimato, mesmo que os participantes não

tenham conhecimentos adicionais dos outros jogadores, como reputação ou estrutura espacial. Por fim, provaram que esta cooperação é possível se os jogadores decidirem suas ações estatisticamente, desde que seja possível dar continuidade as interações quando uma oferta for rejeitada.

Em sua dissertação de mestrado, Martins (2015) cria um guia para elaboração de instrumentos de pesquisas quando se aborda o comportamento humano. Em seu trabalho, ele apresenta o estado da arte para os vários tipos de problemas que envolvem divisão de recursos utilizando a abordagem de observação, apresentando algumas idealizações e implementações próprias para o problema de divisão de recursos, utilizando em sua maioria de jogos e questionários. Em um destes questionários foi perguntado a pesquisadores da área em todo o mundo quais os pontos chaves importantes em uma pesquisa da área, recebendo os dados da Tabela 1 abaixo. Nesta tabela, é possível verificar que os pontos mais requisitados em uma pesquisa deste tipo são a disponibilidade, estabilidade, facilidade de uso e precisão, pontos chaves interligados com o simulador a ser criado.

Tabela 1. Estatística: Qual critério impacta na escolha de uma ferramenta digital?

Alternativa	Média	Desvio Padrão	N
Price	3,79	1,101	28
Availability	4,36	0,780	28
Ease to use	4,11	1,066	28
Precision	4,00	0,861	28
Type of output data	3,75	0,844	28
Confidence	3,86	1,239	28
Stability	4,18	0,905	28
Scalability	3,21	1,258	28

Fonte: Martins (2015) pagina: 44.

Em seu artigo, Kristin e demais (2016) apresentam os níveis de rejeição de ofertas do Jogo do Ultimato e dissertam sobre o mesmo, quando estão envolvidos indivíduos considerados altruístas, no mundo real. Eles avaliaram os motivos de um indivíduo rejeitar uma oferta no Jogo do Ultimato, mesmo sabendo que a rejeição

acarretará em nenhum ganho, mesmo quando esta rejeição envolve o indivíduo altruísta. Durante os seus experimentos, com indivíduos reais, eles verificaram que doadores de rins e pessoas com níveis educacionais altos tendiam a serem altruístas, aceitando ofertas consideradas desiguais. Apesar desse comportamento altruísta, alguns destes escolhiam rejeitar a oferta como forma de punição para o outro usuário.

3.2 MODELAGEM DE COMPORTAMENTOS E SIMULAÇÕES

Como citado anteriormente, com a criação dos computadores alguns pesquisadores e cientistas de várias áreas de conhecimento começaram a utilizá-los como uma ferramenta para as suas pesquisas. Com os trabalhos na área de Teoria dos Jogos e Divisão de Recursos não foi diferente.

A início, Axelrod propôs alguns torneios computacionais para avaliar qual estratégia sairia melhor em atividades de divisão de recurso (AXELROD, 2010). Posteriormente, vários cientistas, incluindo o próprio Axelrod, iniciaram pesquisas com a finalidade de descobrir como modelar os comportamentos de agentes de forma efetiva a fim de simular comportamentos reais. (MANSURY, 2006) (TOMASINO et Al, 2013)

Com a publicação do livro de Axelrod em 1997 "*The Complexity of Cooperation"* (AXELROD, 1997), foram apresentados modelos de agentes inteligentes e sua base teórica para utilização em agentes competitivos e colaborativos. Novamente, este livro teve grande impacto na época por apresenta meios de se modelar agentes para uso em simulação. A partir deste momento, o uso de métodos de simulação começou a crescer de modo vertiginoso.

Durante anos, pesquisas foram realizadas envolvendo várias áreas de conhecimento, como psicologia, neurociências, ciências políticas e ciência militar, as quais estão disponíveis abaixo as mais importantes para este trabalho, tornando assim o assunto transdisciplinar. Apesar do envolvimento de várias áreas no assunto, poucos eram os trabalhos que puderam ser usados como modelo devido ao fato destes, em sua maioria, serem trabalhos envolvendo objetivos que necessitavam de

métodos específicos que dificilmente seriam utilizados para outras finalidades, decaindo no problema da unicidade de solução.

Tanto é que em 1998, o *National Research Council* dos Estados Unidos publicou um livro, intitulado "*Modeling Human and Organizational Behavior: Application to Military Simulations*" (NATIONAL RESEARCH COUNCIL, 1998). Neste livro é descrito uma base teórica sólida para modelagem de comportamentos humanos e de organizações. Indo além dos trabalhos da época, o livro demonstra os requisitos, modelos e simulações criados até o momento de escrita, explicando detalhadamente em toda a sua obra como criar os vários módulos comportamentais de um humano, incluindo memória, percepção, tomada de decisão e aprendizagem. Apesar do livro ser destinado a simulações militares, o nível de detalhamento descrito no livro permitiu que seus métodos fossem utilizados em outras áreas, desde que envolvessem a modelagem de comportamentos humanos.

Após a publicação dos dois livros, de Axelrod e do Conselho Nacional de Pesquisas, foram levantadas várias problemáticas sobre o uso de simulações.

A primeira problemática se referia sobre a validação das simulações, visto que os resultados das mesmas são correlacionados ao método utilizado para a modelagem dos agentes (MOSS; DAVIDSSON, 2001). Vários estudiosos pesquisaram métodos de validação para as simulações, mas um trabalho considerado de grande impacto para a área foi o trabalho "*Micro- and Macro-Level Validation in Agent-Based Simulation: Reproduction of Human-Like Behaviors and Thinking in a Sequential Bargaining Game."* (TAKADAMA; KAWAI; KOYAMA, 2008). Neste trabalho, os autores realizaram um levantamento teórico sobre os métodos utilizados para validação de simulações, implementando uma simulação própria de um jogo de barganha para testar os métodos de validação.

A segunda problemática se refere ao uso de ruído nas simulações, o qual Axelrod em seu segundo livro citava como uma falha de comunicação entre os agentes, os quais poderiam receber uma resposta inversa a situação e tratá-la de diversas formas. Com o passar do tempo, pesquisas foram efetuadas com o objetivo de criar agentes cada vez mais realísticos por estes agirem de forma racional. Até que

em 2005, Pynadath e Marsella demonstraram que os humanos nem sempre tomam decisões racionais por agirem com razões emocionais. Esta comprovação acabou invalidando vários agentes e modelos criados e publicados até o momento por estes não reproduzirem comportamentos realísticos (PYNADATH; MARSELLA, 2005). Várias discussões foram feitas, até o momento em que Zhang e Leezer publicaram o trabalho *"Simulating human-like decisions in a memory-based agent model"* (ZHANG; LEEZER, 2010). Neste trabalho, os pesquisadores realizaram outro levantamento teórico a fim de implementar um modelo de agente racional que agisse também em um modo deliberativo. A pesquisa se utilizou da teoria de Kahneman (2002), a qual permitiu a criação de agentes que tomariam decisões com informações incompletas ou incorretas.

Após o trabalho de Zhang e Leezer, muitos outros foram publicados com aplicações em diversas áreas que utilizavam os conceitos diretamente pela pesquisa de ambos ou por trabalhos evolutivos destes (AN, 2012) (LEE, 2012). Atualmente, foi publicado o artigo "*A game-theory based agent-cellular model for use in urban growth simulation: A case study of the rapidly urbanizing Wuhan area of central China"* (TAN et al., 2015), no qual os autores fizeram o levantamento da arte na área de Teoria dos Jogos, Modelagem e Simulação, aplicando os conceitos em um modelo de simulação para crescimento da população na China.

Para este trabalho, as referências acima foram utilizadas como base para o desenvolvimento do simulador, utilizando como inspiração para a sua mecânica o *Steering Behavior.* O *Steering Behavior* é uma técnica de Inteligência Artificial (IA) aplicada em agentes de jogos e animação, o qual permite a criação de comportamentos para a movimentação destes agentes no mundo de uma forma improvisada e semelhante a humana. O ápice desta técnica acontece quando combinamos os comportamentos simples, gerando novas condutas complexas, como por exemplo, fazer um agente ir de um ponto a outro enquanto se desvia de obstáculos. Uma referência concreta para esta técnica pode ser encontrada no trabalho de Reynolds (1999).

3.3 COMPUTAÇÃO PARALELA

Apesar do projeto não fazer uso de computação paralela em seu estado atual, foi ideado o seu uso em versões futuras do simulador, visto as possíveis vantagens que essa tecnologia poderia proporcionar. Então, foi realizado um estudo sistemático do tema, a fim de disponibilizar o suporte a essa tecnologia em versões futuras do simulador.

Desde o início da computação, o paralelismo na execução de tarefas computacionais era tratado por cientistas da computação como um tema difícil de se trabalhar. Com a evolução da tecnologia, iniciaram os trabalhos com computação paralela, utilizando como meio os processadores dos computadores. Atualmente, a programação paralela já trabalha com as placas de vídeos. Hoje, as placas de vídeo são hardwares híbridos, o qual consegue desempenhar a sua função original de processamento de imagens e a função de computação paralela de cálculos, ao mesmo tempo.

Acompanhando esta evolução, surgiram várias empresas que fabricavam o *hardware* e a tecnologia das placas de vídeos, como a AMD e a NVIDIA. A NVIDIA pioneira na fabricação de hardware híbrido, conseguiu com suas placas a maioria do mercado atual, visto que seus equipamentos objetivam o público de jogadores e cientistas.

Com o advento das placas de vídeo como hardware principal para computação paralela, o poder computacional evoluiu rapidamente. Considerando uma CPU atual, como o Intel Core i7, obteríamos um desempenho considerado ao trabalhar paralelamente com 8 núcleos a uma média de 3.7Ghz à 64 bits, a um preço médio de 2000 reais. Já considerando uma GPU atual na mesma faixa de preço, é possível utilizar o modelo GeForce GTX 970, o qual possui 1664 núcleos com 1.1GHz em uma interface de 256bits.

Apesar do aumento de desempenho óbvio, programar paralelamente não é considerado fácil. Primeiramente, o problema necessita ser paralelizável, ou seja, divisível em trabalhos independentes. Caso o problema seja paralelizável, deparamos

com a dificuldade de programação dos programas. Atualmente a NVIDIA criou a sua linguagem de programação, chamada CUDA C, a qual permite criar softwares que executem na placa gráfica. A linguagem facilitou bastante a sua programação, mas a mesma possui uma complexidade alta, exigindo um tempo considerável para criação e testes.

Devido a essa dificuldade, a NVIDIA lançou o livro "*CUDA BY EXAMPLE: An Introduction to General-Purpose GPU Programming"*. Neste livro são explicados os conceitos básicos de programação paralela e as aplicações dos conceitos, sendo criados programas para a explicação desses conceitos (SANDERS, 2012).

A partir deste momento, a aplicação de CUDA cresceu, principalmente em aplicações que exigem grande poder de processamento ou que gere grande montante de dados. Este crescimento possibilitou vários estudos na área de Inteligência Artificial. Dentre estes crescimentos, surgiu o tema de *Deep Learning,* o qual é um tipo de aprendizagem de máquina que utiliza uma quantidade de dados e processamento massivos para criar modelos e representações melhores de aprendizagem. Atualmente, o *Deep Learning* está incluso em pesquisas de grande porte como a simulação do cérebro humano para comportamentos de pedestres (NVIDIA, 2016ª), a análise dos dados coletados em Marte (NVIDIA, 2016b) e a proteção de ataques cibernéticos (NVIDIA, 2016c).

3.4 APLICAÇÕES

Na Teoria dos Jogos, especificamente na área de Divisão de Recursos, a aplicação da teoria se dá de várias formas, desde questionários eletrônicos ou físicos a utilização de *Serious Games* (Jogos Sérios). Abaixo está descrito o estado da arte nas seções relacionadas ao nosso trabalho.

3.4.1 Frameworks para experimentos econômicos

Segundo Martins (2015), frameworks para experimentos econômicos são ferramentas que visam facilitar o processo de pesquisa em teoria dos jogos e áreas multidisciplinares, como economia e psicologia. Martins complementa que "[...] elas

permitem aos pesquisadores configurarem sem dificuldades experimentos onde eles podem testar hipóteses a respeito do comportamento de indivíduos, simular o comportamento de uma população fazendo uso de autômatos com inteligência artificial e estudar a interação entre agentes virtuais e pessoas reais. ” (MARTINS, 2015).

Ao realizarmos a pesquisa por estes frameworks, descobrimos uma solução citada por Martins. Esta ferramenta é o Z-TREE, acrônimo para *Zurich Toolbox for Ready-made Econômic Experiments*. O Z-TREE é um sistema no qual o pesquisador, sem necessidade do domínio de alguma linguagem de programação, pode conceber e executar experimentos via rede através do modelo cliente-servidor (FISCHBACHER, 2007). O framework possui uma base de experimento na qual o pesquisador consegue montar jogos socioeconômicos autônomos ou com interação do usuário através de botões. A maior limitação da ferramenta, como citado por Martins, é o controle preciso do tempo de transição dos estados, o qual pode dificultar os experimentos autônomos e exigir um estudo mais amplo de psicólogos.

O matemático Alexander Kasprzyk publicou em seu site um compilado de ferramentas utilizados em experimentos matemáticos e sociais, chamado *Kasprzyk A-Life* (KASPRZYK, 2002), as ferramentas foram desenvolvidas para Mac-OS, possuindo cada uma sua finalidade. Podemos citar com destaque a *Langton’s Ant v1.3.1*, a qual se utiliza de autômatos celulares para simular o comportamento das formigas, o qual pode ser utilizado para estudos de cooperação. Kasprzyk também disponibilizou em seu site a sua implementação para o Dilema do Prisioneiro, no qual o usuário joga o jogo com o dilema com o computador, este utilizando a estratégia do Olho por Olho. Essa ferramenta foi utilizada pelo matemático para avaliar os resultados do experimento de Axelrod em relação a um jogador real.

Serendip Studio é um site (SERENDIP, 2010) com vários experimentos biológicos e sociais. Dentre os vários experimentos, estão disponíveis as ferramentas com o Jogo do Dilema do Prisioneiro, no qual o usuário joga com o servidor o jogo com o Dilema do Prisioneiro Iterado, com a quantidade de rodadas definida pelo desempenho dos dois envolvidos. Ao final do jogo, a ferramenta demonstra o resultado do jogo e tenta informar qual a estratégia utilizada pelo jogador.

Outra ferramenta disponível pelo Serendip Studio é o Ant Colonies, que é um jogo que simula o comportamento de formigas, permitindo o estudo de comportamento e cooperação em grupo. A ferramenta permite a inserção e remoção de formigas e tarefas com o objetivo de estudar a cooperação e a resposta para perturbações e intrusões em grupos.

Prisoner's Dilemma (DAVIS, 2007a) é um site que permite ao usuário jogar o jogo com o Dilema do Prisioneiro Iterado com o computador. O maior diferencial desta ferramenta é a possibilidade do jogador simular o experimento escolhendo a estratégia a ser utilizada por ambos os jogadores, utilizando as várias estratégias já definidas ou inserindo uma nova. O jogador pode ainda alterar várias variáveis, como a chance do jogador computadorizado utilizar uma escolha randômica, o tempo de cada rodada, a quantidade de rodadas, e a chance da Inteligência Artificial utilizar cada estratégia.

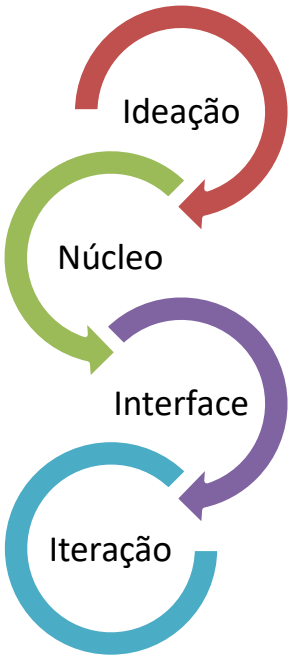
Spatialised Prisoner's Dilemma: Evolution Simulator (DAVIS, 2007b) é outra ferramenta, similar ao *Prisoner's Dilemma*, com o diferencial de que este utiliza conceitos evolutivos, citados por Richard Dawkins em seu livro *The Selfish Gene*. Nesta ferramenta é possível somente simular o experimento, permitindo ao usuário alterar a quantidade de gerações, o nível evolucionário, a longevidade de cada célula e a estratégia a ser utilizada. A principal deficiência nesta ferramenta é o fato da mesma apenas permitir que os valores das variáveis sejam alterados por valores pré-fixados.

Como observado acima, existem várias ferramentas disponíveis, mas estas estão datadas até o ano de 2010, não sendo incluídos os experimentos e estudos mais novos, como os estudos de Ichimose e Sayama (2014) e os estudos de Kristin e demais (2016), relacionados ao Jogo do Ultimato para o comportamento e cooperação de indivíduos.

4 MÉTODOS DA PESQUISA

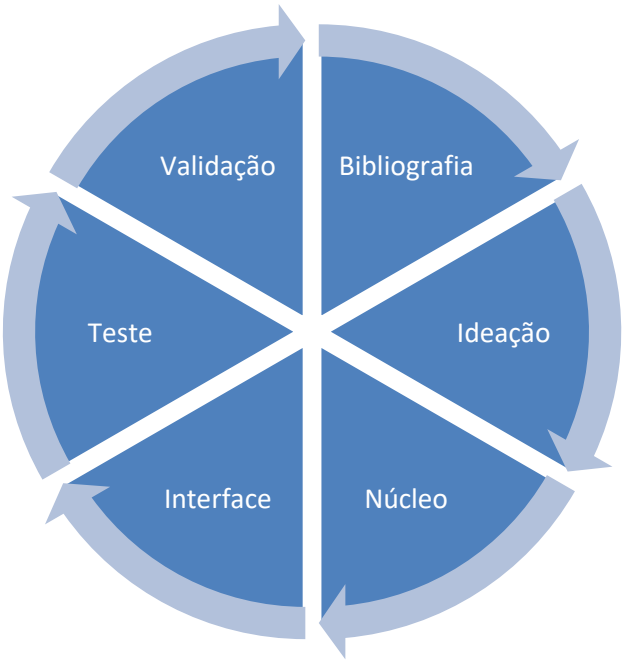
Este trabalho foi idealizado com um método de pesquisa dividido em quatro etapas sequenciais e distintas. Como método de execução, foi utilizado o método hibrido de desenvolvimento disposto na Figura 1.

Figura 1. Processo de desenvolvimento da ferramenta



Este método é composto inicial pelo método em cascata para construção do núcleo da ferramenta, passando então para o desenvolvimento inicial da interface. Após a finalização da etapa da interface, inicia o método iterativo, no qual o trabalho como um todo perpassa por etapas iterativas e complementares de desenvolvimento, executadas em ciclos, como disposto na Figura 2.

Figura 2. Método iterativo de desenvolvimento da ferramenta



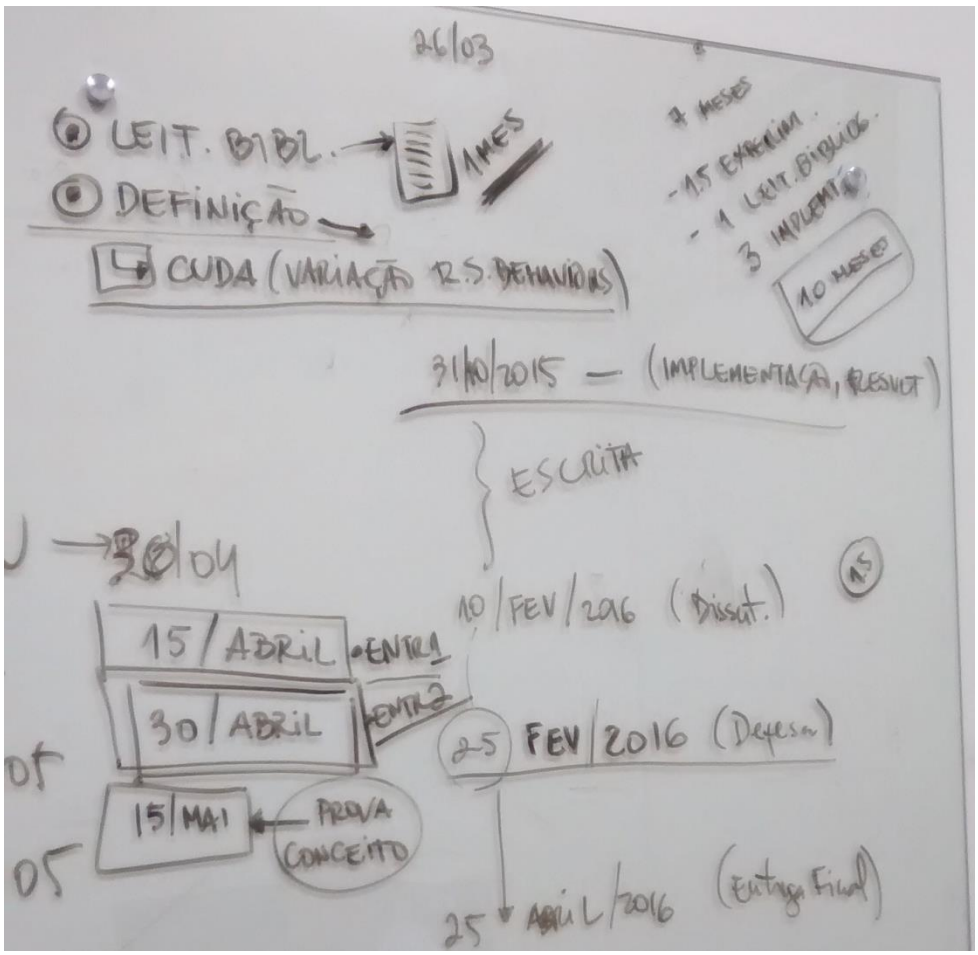
Na etapa de Bibliografia, é revisado toda a literatura para procura de conceitos novos. Com esta revisão bibliográfica, é iniciado o novo ciclo de Ideação, realizada em conjunto com um especialista da área, para a elaboração de novas ideias e melhoramento das implementações realizadas. Com isto é iniciado o desenvolvimento das ideias geradas para o Núcleo e Interface. Após a implementação das ideias, são realizados os Testes para verificação da corretude das saídas e então a ferramenta é repassada para o especialista realizar a Validação, apontando os conceitos a serem estudados para melhoramento da ferramenta, retornando assim para a Bibliografia e reiniciando o ciclo.

Os resultados dessas etapas estão descritos ao longo deste trabalho, onde cada etapa é utilizada como base para a próxima, cada uma contribuindo para o resultado final. Para o acompanhamento do trabalho como um todo, foram definidos marcos, que são:

1. Processo de ideação da ferramenta;
2. Processo de implementação da ferramenta;
3. Processo de simulação da ferramenta;
4. Teste com o usuário;

Após reunirmos alguns trabalhos relevantes a área, foi estabelecido o plano de projeto com as datas previstas para a finalização de marcos e algumas propostas de avanço, dispostos na Figura 3.

Figura 3. Datas previstas para o plano de projeto.



Segue abaixo cada uma das etapas descritas detalhadamente com os métodos utilizados em cada uma dessas etapas.

4.1 PROCESSO DE IDEAÇÃO DA FERRAMENTA

Após a revisão bibliográfica, foi iniciado o processo de ideação da ferramenta, no qual consistiu em analisar a revisão e o estado da arte e tentar idealizar algo que acrescentasse conhecimento.

No processo de ideação foi planejada a prova do conceito do trabalho, sendo firmado o objetivo inicial de criar uma ferramenta de simulação que permitisse sua customização pelo usuário, intitulado *Resource Sharing Behavior* (RSB). O nome da ferramenta foi decidido devido a forma de execução idealizada inicialmente, o qual seria um comportamento semelhante ao *Steering Behavior* (REYNOLDS, 1999).

Os pontos chaves da ferramenta deveriam ser a facilidade de uso e a flexibilidade. Focando nestes pontos, foi decidido iniciar a implementação pelo núcleo da ferramenta (*backend)*, com o intuito de permitir a sua flexibilidade seguindo o conceito do *Steering Behavior*, no qual um conjunto de comportamentos simples, ao serem unidos, origina um comportamento complexo. Para isto, decidimos criar dois alicerces para o programa, que é o Principio e a Susceptibilidade, que deveriam suportar inicialmente esta modulação dos comportamentos complexos, com o Principio definido para o tipo de personalidade do jogador (altruísta, egoísta), e a Susceptibilidade definido para o tipo de estratégia do jogador (Olho-por-Olho, Cabeça-Dura). Esta definição está explicada detalhadamente no capitulo 5, durante a definição e implementação da ferramenta.

Para deixarmos a ferramenta fácil de utilizar, foi criado uma interface gráfica simples e intuitiva (*frontend)* ao usuário, que possibilitaria a este a criação dos experimentos de forma fácil e intuitiva sem a necessidade de conhecimento de programação. Caso o usuário possuísse o conhecimento na linguagem desenvolvida, ele poderia então adentrar o *backend* e criar novos módulos para funções, regras e jogos que desejasse ou criar uma nova interface gráfica a seu desejo.

4.1.1 Modelo da Ferramenta

Após a ideação inicial da ferramenta ser concluída, foi obtido o modelo da ferramenta com suas especificações básicas. Após vários encontros com o auditor, todos os aspectos comportamentais dos agentes foram resumidos de acordo com a experiência e conselho do especialista, reduzindo para quatro variáveis fundamentais: Valor, Principio, Susceptibilidade e Memória.

O Valor seria a representação de um valor de comum interesse entre os agentes, como uma moeda ou poder;

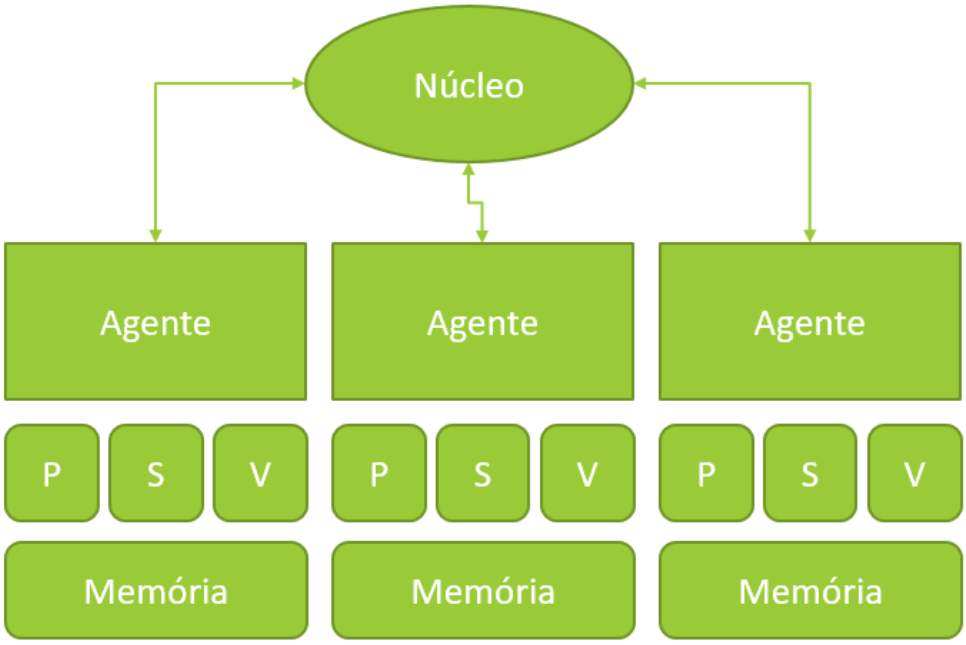
O Princípio seria um valor real representando o princípio e as atitudes naturais de um indivíduo, como uma pessoa egoísta, altruísta, igualitária, etc.;

A Susceptibilidade seria outro valor real que daria suporte ao Princípio, representando o potencial de um indivíduo de ser suscetível ou sensível a atitudes alheias, podendo alterar a sua ação natural. Esta variável suportaria o Principio representando em grande parte dos jogos o tipo de estratégia a ser utilizada;

Por fim, a Memória seria um *array* de valores reais representando a memória que um indivíduo possui em relação a atitudes realizadas com os outros agentes da População.

Então, com a junção das quatro variáveis, tem-se a representação do agente, o qual, em conjunto com os outros agentes e com as funções da População, obtém-se o modelo do núcleo do programa disponível na Figura 4 abaixo.

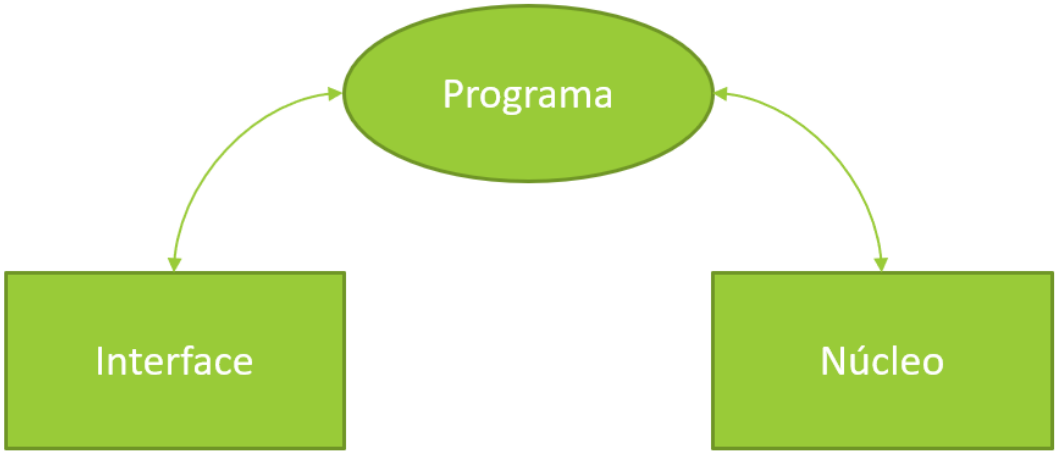
Figura 4. Modelo do Núcleo do Simulador



Com este modelo, caso o usuário deseje utilizar uma metodologia diferente da que está disponível no simulador, em um ou vários módulos, basta alterar ou inserir o código necessário neste modulo. Como exemplo, pode-se citar a necessidade de alteração da memória pelo auditor, o qual necessitou apenas da alteração do modulo Memória; outro exemplo ocorre na necessidade de inclusão ou exclusão de componentes, o qual pode ocorrer a inclusão de outra variável sem a necessidade de alteração das outras, necessitando apenas a inclusão da função necessária na População.

Por fim, para integralizar o projeto, basta inserir a Interface Gráfica que utilize este Núcleo, a fim de proporcionar a experiência visual desejada para o usuário, tudo isto de forma autônoma, conforme modelo da Figura 5. Com isto, caso seja exigido uma alteração da interface do simulador, esta pode ser feita sem impactos no Núcleo do simulador.

Figura 5. Modelo Geral do Programa.



4.2 PROCESSO DE IMPLEMENTAÇÃO DA FERRAMENTA

Com o processo de ideação inicial finalizado, foi iniciado o processo de implementação. Para o processo de implementação foi decidido utilizar o Visual Studio 2013, ferramenta proprietária da Microsoft, com a linguagem de programação C# para a ferramenta comum e C++ para a ferramenta em CUDA. O desenvolvimento da

ferramenta em CUDA foi postergada com o objetivo de implementar mais uma abordagem na ferramenta comum.

Para a realização do projeto, foi utilizado o método de desenvolvimento hibrido descrito na Figura 1, o qual se utiliza inicialmente do método de desenvolvimento em cascata e após os conceitos básicos estarem desenvolvidos, inicia a segunda etapa utilizando o método iterativo, para evolução da ferramenta e alterações com os conceitos recebidos pelo especialista consultado. Devido ao uso do método iterativo, nenhuma ideia foi descartada. Com as iterações seguintes, disponível no Capitulo 5, algumas ideias foram recicladas ou moldadas para se adequar ao escopo do projeto.

No próximo capitulo está detalhado os métodos utilizados na implementação da ferramenta. Também está presente a evolução seguida em seu desenvolvimento.

4.3 PROCESSO DE SIMULAÇÃO DA FERRAMENTA

Após a finalização da ferramenta foi iniciado o processo de simulação. Este processo consiste da modelagem e simulação de experimentos contidos nos trabalhos pesquisados como referência bibliográfica, com o objetivo de validar a ferramenta. Também foram realizados experimentos adaptados da literatura para tentar provar a corretude da ferramenta.

Neste processo, assim como no processo de implementação, um especialista da área foi contatado, ajudando com os conceitos necessários e fornecendo apoio na modelagem do simulador e das simulações.

Inicialmente, foram realizados testes unitários para verificar possíveis erros de cálculos e corretude das saídas e entradas para posteriormente executar os experimentos. Após a finalização dos testes iniciou-se a modelagem dos experimentos, fixando uma quantidade mínima de execuções para cada um, com o objetivo de realizarmos uma validação dos próprios resultados.

No capítulo 6 estão descritos todos os testes realizados, com o motivo, modelo e resultado destes. Devido à grande quantidade de testes, os considerados

semelhantes ou menos importantes para a validação da ferramenta foram movidos para o anexo.

4.4 TESTES COM O USUÁRIO

Após a finalização das simulações, foram realizados testes fechados (*Closed Beta*) da ferramenta com um conjunto de usuários finais. Estes testes objetivam verificar erros na ferramenta e obter *feedback* de usuários que possam ser considerados usuários reais.

Para este tipo de teste, a ferramenta foi repassada para 10 pessoas, incluindo o especialista auditado no desenvolvimento e modelagem do experimento. Os participantes deveriam ser especialistas na área de Teoria dos Jogos ou possuir experiência com a aplicação de seus conceitos ou de conceitos ligados a divisão de recurso a qual a ferramenta é destinada. O quantitativo de pessoas a testar a ferramenta foi definido em uma quantidade pequena devido a dificuldade de encontrar pessoas com o perfil exigido para a realização do teste.

Após os testes, foi repassado um questionário opcional para os usuários participantes avaliarem a ferramenta e fornecessem críticas e sugestões para a melhoria do programa. Este questionário foi repassado com o intuito de verificar se o simulador possui as características necessárias para ser considerado uma ferramenta útil e fácil de usar. O questionário está disponível no Apêndice A.

Infelizmente, foram recebidas apenas 2 respostas dos questionários e o *feedback* pessoal do auditor. Apesar da quantidade baixa, os comentários informaram a utilidade da ferramenta, apontando melhorias e necessidades extras no programa.

5 DESENVOLVIMENTO DA FERRAMENTA

Neste capitulo estão descritas as etapas de desenvolvimento da ferramenta, intitulada *Resource Sharing Behaviors* (RSB). É descrito a ideia inicial, sua modelagem e implementação. Então a versão apresentada é iterada com os aprimoramentos do conceito inicial, seguindo por seus respectivos modelos e desenvolvimento, chegando na ferramenta final. Finalmente, são apresentadas as possibilidades de expansão do software e os meios de realizar esta expansão.

5.1 IDEIA INICIAL

Inicialmente foi idealizado um software capaz de simular o jogo do ditador entre dois agentes. O simulador poderia armazenar uma quantidade máxima de 10 agentes, objetivando a facilidade de escolha em quais agentes iriam “jogar” o jogo e persistir os dados destes para realização de novas atividades.

Os agentes possuiriam em sua estrutura três valores: Valor, Expressividade e Susceptibilidade. O valor seria a variável que conteria o valor do agente para a população, o qual poderia ser traduzido para o mundo real como um valor em recursos ou reputação.

A Expressividade seria a variável que conteria o estilo do jogador, o qual podemos estipular os extremos como um agente egoísta com valor 1 e altruísta com valor 0. Na atividade, o agente ativo ficará com a parcela do valor referente ao valor da Expressividade.

A Susceptibilidade seria a variável que conteria a probabilidade de o agente ser suscetível ao ato de desertar do outro jogador, adicionando a possibilidade do mesmo agir em ambientes com ruído, o agente se utilizar de estratégias como o olho por olho ou ser generoso. O valor 0 seria a modelagem do agente cabeça-dura, o qual não aceita objeções, ações ou dicas de outros agentes, seguindo sempre o seu Principio. O valor 1 seria a modelagem do agente que utiliza a estratégia olho por olho,

realizando a mesma ação sofrida de um agente, contra ele mesmo. Outros valores entre os extremos demarcaria a chance dele executar uma ou outra estratégia.

No conceito inicial, o usuário deveria adicionar os agentes, com os seus respectivos dados, e posteriormente o adicionar a população. Ao rodar o experimento, o usuário deveria escolher quais agentes participariam do jogo e o valor de receita da atividade. Então, o software simularia a atividade, de acordo com os dados dos agentes, atualizaria o valor dos agentes e retornaria para o menu de escolha de agentes.

5.2 SEGUNDA ITERAÇÃO

Para a segunda versão do RSB, o auditor verificou que as possibilidades de experimentação eram bastantes limitadas. Ao executar a ferramenta e verificar sua bibliografia privada, este verificou que o uso de uma memória permitiria uma maior quantidade de experimentações. Então, foi ideado a adição da memória dos agentes, a qual deveria ser realizada progressivamente entre as versões que surgissem, permitindo assim a realização de testes individuais na modelagem da memória.

A adição da memória deveria ser feita de forma que o usuário escolhesse a probabilidade do agente participante com poder de escolha lembrar-se da última ação do agente passivo, caso esta memória exista. Para esta modelagem da memória, foram utilizados os conceitos dispostos no livro do Conselho Nacional de Pesquisa (NATIONAL RESEARCH COUNCIL, 1998).

Ao revisar os materiais bibliográficos, foi decido a alteração do nome da variável de Expressividade para Principio. Isto foi idealizado devido ao fato de cada agente, representado no mundo real, possuir em seu poder de escolha um princípio, o qual define que o mesmo seja egoísta, altruísta, igualitário e etc. Para o software, esta mudança foi considerada puramente estética, mas para a interação entre o usuário e a ferramenta, mitigou-se um problema de valor semântico que o mesmo poderia causar.

5.3 TERCEIRA ITERAÇÃO

A terceira versão do RSB contou com algumas melhorias envolvendo a execução das atividades, o uso da memória e a automação de sua execução.

Inicialmente foi adicionado uma variável para conter a porcentagem de decaimento da memória. Este decaimento deveria ser informado no início da execução do software e definiria a taxa de redução da probabilidade de o agente ativo lembrar da ação do agente passivo ao executar uma atividade com o tempo, estando este último declarado como a quantidade de atividades executadas na qual o par de agentes não coparticipam.

A principal melhoria para a execução da atividade é a definição de um padrão de reações dos agentes para todas as possibilidades previstas para uma simulação, envolvendo as possíveis entradas até o momento. As reações envolveram desde o agente “ditador” ser egoísta e pegar todo valor até o mesmo repassar tudo para o segundo agente. Foi incluído também as exceções com as ações inversas do mesmo. Essas ações inversas ocorreriam em casos raros ou quando o agente possuir um “temperamento” de executar uma ação contrária a prevista com a entrada.

Para automatizar a execução das simulações foi inserido a opção para o usuário executar uma quantidade desejável de atividades de forma aleatória. Esta funcionalidade foi adicionada após teste com o auditor resultarem em um retrabalho e perca de tempo deste por ter que escolher manualmente os agentes participantes para realizar a atividade. A automatização permitiu então a realização de vários testes para verificar a validade das saídas em vários cenários possíveis.

Por último, uma adição para esta versão foi a possibilidade de remoção automática de agentes que ficassem com seu valor negativo. Este valor ficaria negativo quando a execução da atividade envolvesse um custo para ambos os agentes e estes não conseguissem receber um valor maior que este custo. A aplicação desta funcionalidade foi aprovada pelo especialista auditor, com a finalidade de realizarmos uma evolução simplificada da população através das estratégias utilizadas pelo agente. Com essa abordagem seria possível verificar de forma

simplificada, após um determinado número de atividades escolhidas pelo usuário, quais estratégias sobrepõem as outras.

5.4 QUARTA ITERAÇÃO

Para a quarta versão foi adicionado a funcionalidade do usuário criar agentes com dados aleatórios. Esta adição ajudaria o usuário eximindo-o da necessidade de digitar os dados de cada agente nos casos em que os mesmos possuam esta liberdade de criação, o que aconteceu com o especialista durante os testes.

Com esta adição aleatória, o usuário necessitaria apenas informar a quantidade de agentes a adicionar e o valor desejado, sendo retornado a quantidade desejada de agentes com o Principio e a Susceptibilidade valoradas aleatoriamente.

5.5 QUINTA ITERAÇÃO

Na quinta versão do RSB foi decidido alterar o funcionamento da memória em conjunto com a tomada de decisão do agente. Esta alteração passou a utilizar não somente a porcentagem do agente lembrar a ação, mesmo com o fator de decaimento, mas também a informação do fator utilizado pelo outro agente. Desta forma caso o agente “ditador” lembre da ação utilizada pelo outro agente para com ele, o mesmo poderá escolher entre utilizar um ramo maior de ações e estratégias, como o descrito na tabela abaixo.

Tabela 2. Possibilidades de ação do agente.

Ação Estratégia	Positiva	Negativa
Cabeça-Dura	Utilizar o Principio	Perdoar
Olho por Olho	Cooperar	Retaliar

Resumindo, os agentes podem realizar as seguintes ações:

- Ser cabeça dura e utilizar apenas seu princípio;
- Utilizar a olho por olho, ou variantes, e repetir a ação;
- Se a ação for negativa, perdoar ou retaliar;
- Se a ação for positiva, colaborar ou desertar.

Uma abordagem especifica para esta adição é a possibilidade de uma ação positiva de um agente ser identificada como ação negativa pelo outro agente, visto que ambos podem possuir princípios diferentes. Com isso, temos um aumento de complexidade da ferramenta e tratamos o conceito inicial de ruído citado por Axelrod (1997).

Uma segunda adição ao software foi a possibilidade do usuário criar os agentes em lotes, com o objetivo de abstrair a carga de trabalho manual. Desta forma, o usuário poderá escolher quantos agentes com os dados informados por ele deverão ser criados, evitando assim que este tenha que reinserir os dados repetidamente. Assim, o usuário poderá criar os agentes de forma mais rápida e futuramente poderá automatizar essa criação.

5.6 SEXTA ITERAÇÃO

A sexta versão do programa foi criada com o foco na usabilidade do usuário. O objetivo principal seria permitir ao usuário criar e/ou refazer os experimentos sem a necessidade de este adicionar os agentes novamente, salvando uma cópia dos mesmos.

Para atingir o objetivo, inicialmente foi criado uma memória geral para o programa. Esta memória, diferente da memória individual do agente, permitiria ao usuário salvar o estado atual de toda a população de agentes, com todos os seus dados, no momento desejado. Posteriormente, quando necessário, estes dados poderiam ser lidos ou reescritos pelo usuário como este desejar. Com isto evitamos a reinserção de dados já fornecidos ao programa, economizando tempo e trabalho e aumentando a potencialidade do simulador.

A segunda funcionalidade adicionada para contemplar o objetivo foi a permissão do usuário alterar os dados inseridos no início do programa a qualquer momento. Os dados a se alterar foram o decaimento da memória, o custo e a receita das atividades. Esta permissão, em conjunto com a primeira funcionalidade adicionada nesta versão, permitiu ao usuário modelar vários cenários nos quais os ambientes eram estáticos e imutáveis, sem a necessidade de remodelar os agentes e de reiniciar o programa. Outra possibilidade adicionada com esta funcionalidade é a possibilidade de o usuário realizar vários experimentos em sequência, com os agentes em seu estado inicial ou alterado pelo cenário anterior.

Por fim, para facilitar o uso do programa, foi adicionado a opção de o usuário repetir o experimento realizado, sem a necessidade deste ficar salvando e carregando dados na memória. Para isto, foi adicionado uma outra memoria temporária que armazenava os estados dos agentes antes do experimento, podendo este estado ser recuperado ao fim dos cenários e assim este ser refeito.

5.7 SÉTIMA ITERAÇÃO

Para a sétima versão foi ideada a funcionalidade de salvar os resultados em arquivos. O motivo para a criação desta função seguiu da seguinte premissa: o usuário pode criar e simular pequenas populações e quantidade de atividades. Mas com as funcionalidades existentes, o mesmo pode criar e simular grandes populações com altas quantidades de atividades, mas este não conseguiria analisar estes resultados de forma fácil por ter como saída de dados apenas a janela do console.

Para esta funcionalidade, primeiramente foi criado uma variável para conter o caminho de execução do programa, o qual será o local onde os arquivos serão salvos. Após a coleta dessa informação, no início da atividade, é solicitado ao usuário o nome do arquivo de saída, então o arquivo é criado e são adicionados os resultados de cada atividade da simulação realizada neste arquivo.

Para diversificar os dados que o usuário pode utilizar, foi adicionado à função de criação de um arquivo de texto contendo todos os agentes com suas respectivas informações e valores antes da atividade e outro arquivo depois da atividade. Com

isto, o usuário poderá analisar de forma mais detalhada as mudanças decorrentes das atividades na população.

Após os testes de unidade serem realizados em conjunto com auditor especialista, foi considerado a adição de suporte a arquivos no formato CSV. A adição desse formato facilitaria a análise devido a automação de funções analíticas utilizando softwares externos de manipulação de planilhas, como o Microsoft Excel, LibreOffice, MatLab ou outros. Em conjunto com essa funcionalidade, foi decidido adicionar as informações contidas nas variáveis do simulador no início dos arquivos, para informar ao usuário em qual circunstância os resultados foram gerados. Após a finalização da implementação, o *RSB* passou a gerar todos os resultados em arquivo texto e no formato CSV, contendo todas as informações do ambiente simulado.

Ao se realizar o teste geral da ferramenta, em conjunto com o especialista, verificou-se através da análise dos arquivos de saída que o decaimento da memória se dava de forma correta, mas não completa. Após a realização de uma atividade entre dois agentes as memórias de todos os agentes da população decaiam, assemelhando-se a uma situação na qual existe apenas uma estação consumidora da atividade e todos os agentes observam a atividade ser realizada. Para suprir a possível necessidade de simular um ambiente com várias estações consumidoras da atividade ou uma situação em que os agentes não participantes da atividade não tem conhecimento da atividade, foi adicionado a opção do usuário definir que o decaimento da memória só acontece nos agentes participantes.

Para finalizar, adicionamos a permissão do usuário escolher e alterar de forma facilitada a morte dos agentes endividados. Esta funcionalidade foi inserida com o intuito de suportar as atividades com custo negativo e o suporte a atividade que não sejam de ganho zero.

5.8 OITAVA ITERAÇÃO

Após a sétima versão do RSB ser finalizada, o software seria capaz de abstrair a modelagem e simulação de agentes inteligentes no jogo do Dilema do Ditador.

Apesar dessa capacidade, foi decidido adicionar mais uma funcionalidade, a de utilizar o jogo do Ultimato.

A funcionalidade deveria utilizar todo o conjunto de funções já existente, visto a sua modularidade, de forma a mudar apenas as regras, permitindo ao usuário adicionar algum requisito a mais que viesse a necessitar.

Primeiramente, foi modelado o conjunto de regras do jogo do Ultimato no RSB. Como este jogo é uma adaptação do Dilema do Ditador, possuindo como diferença a possibilidade do agente receptor recusar a oferta, a inserção foi de fácil concepção, necessitando apenas colocar a condição de escolha para o segundo agente.

Por fim, foi necessário adicionar mais uma variável ao agente, chamado de fator de rejeição. Este fator funcionaria de forma semelhante ao princípio, impactando diretamente na tomada de decisão do segundo agente ao receber a proposta.

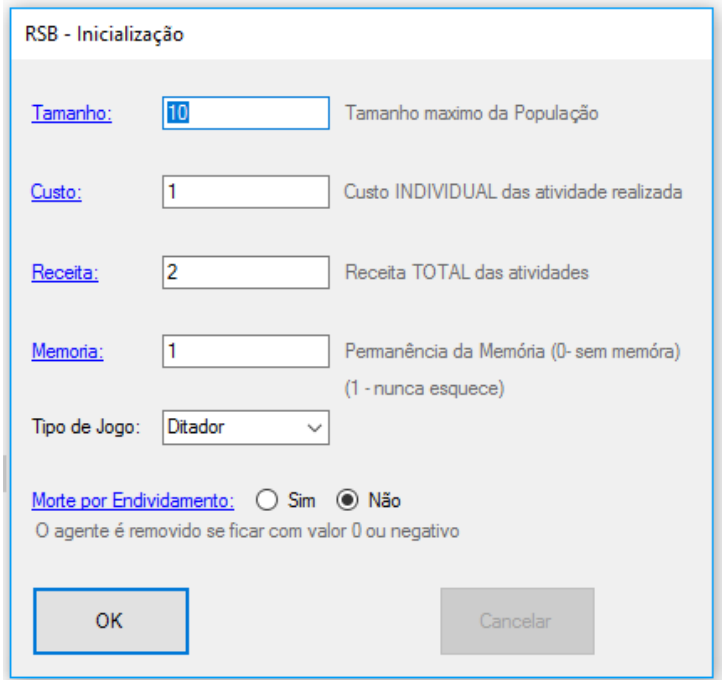
Finalizando, o programa utiliza o mesmo conjunto de agentes para ambos os jogos, inserindo os conteúdos e variáveis necessários à medida que forem necessários. O RSB realizará as perguntas de sua inicialização e por fim perguntará o tipo de jogo a se utilizar, moldando-se ao jogo e a necessidade do usuário para realizar a modelagem e simulação dos agentes em atividades de divisão de recursos.

5.9 ÚLTIMA ITERAÇÃO PRÉ-TESTES

Para a última versão foi criada a interface gráfica do aplicativo. A interface gráfica visa possibilitar um fácil entendimento da ferramenta, facilitando o seu uso. Para isto, são fornecidas informações sobre cada valor a ser utilizado com o usuário e a possibilidade de o mesmo conseguir informações detalhadas de algumas variáveis utilizada pelo programa através de mensagens na própria interface.

Segue abaixo as imagens da interface gráfica, com o detalhamento das funcionalidades.

Figura 6. Janela de Inicialização



Ao iniciar o programa, o usuário é solicitado a informar dados relacionados a inicialização do cenário, descrito na Figura 6 acima.

Após a inserção dos dados iniciais, o usuário é levado a um menu contendo as principais funções do aplicativo, descrito na Figura 7 abaixo. O usuário então pode adicionar um agente, remover, visualizar toda a população, realizar as atividades e alterar as variáveis iniciais, podendo também carregar ou salvar o estado atual da população. Por fim, ele pode acessar os detalhes de versão e equipe de desenvolvimento para envio de sugestões, críticas e informações sobre falhas.

Ao tentar inserir um agente, o usuário o pode fazer de duas formas, inserindo os dados manualmente ou preenchendo as informações de forma aleatória. Escolhendo a opção manual, o usuário é apresentado a janela disposta na Figura 8 abaixo, onde pode inserir as informações desejadas para a criação do número desejado de agentes.

Figura 7. Menu principal do RSB.

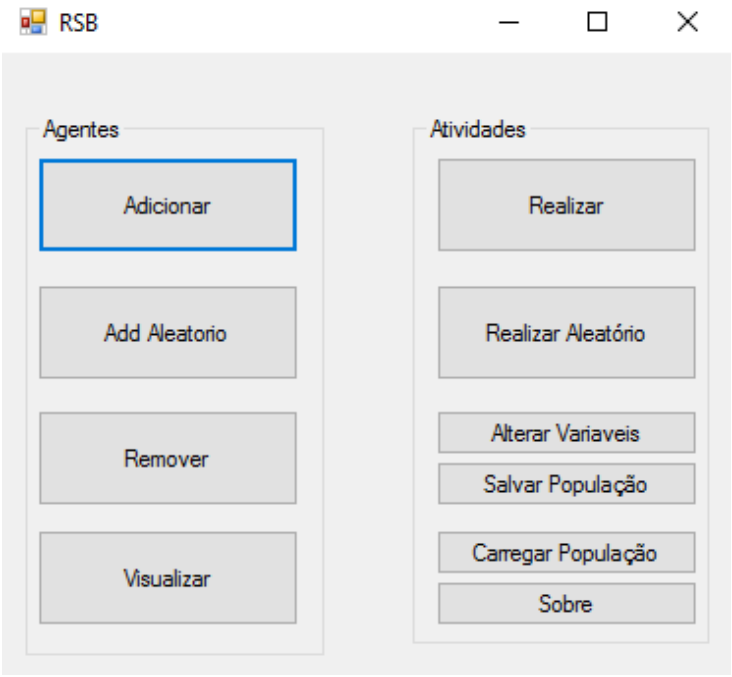
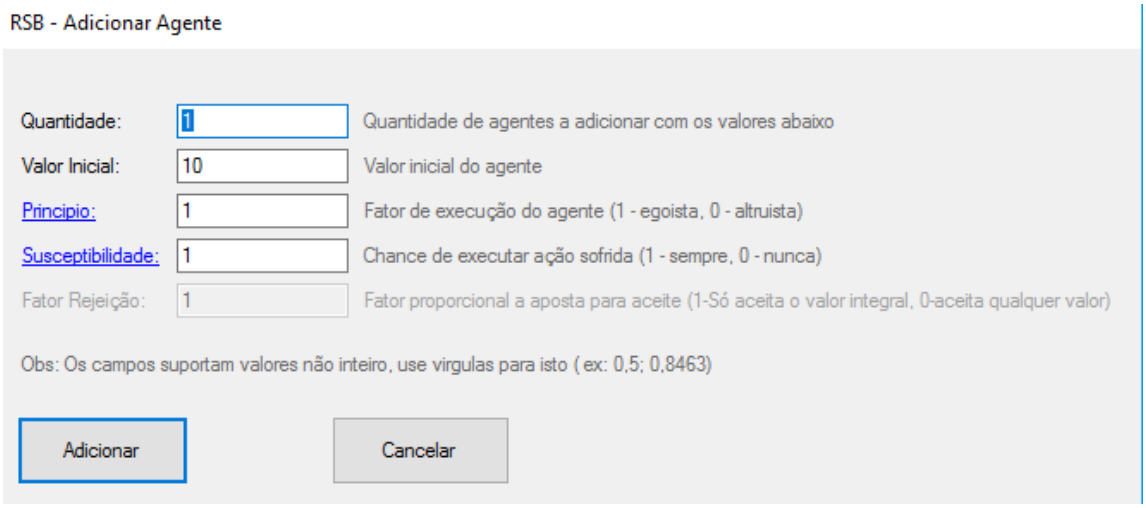
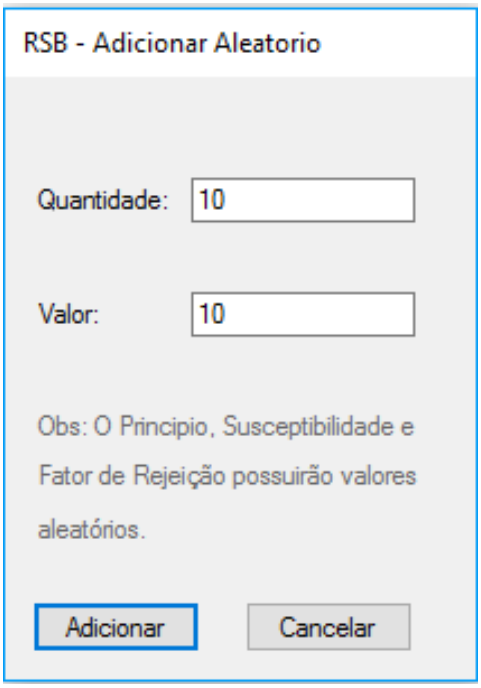


Figura 8. Janela de criação de agente



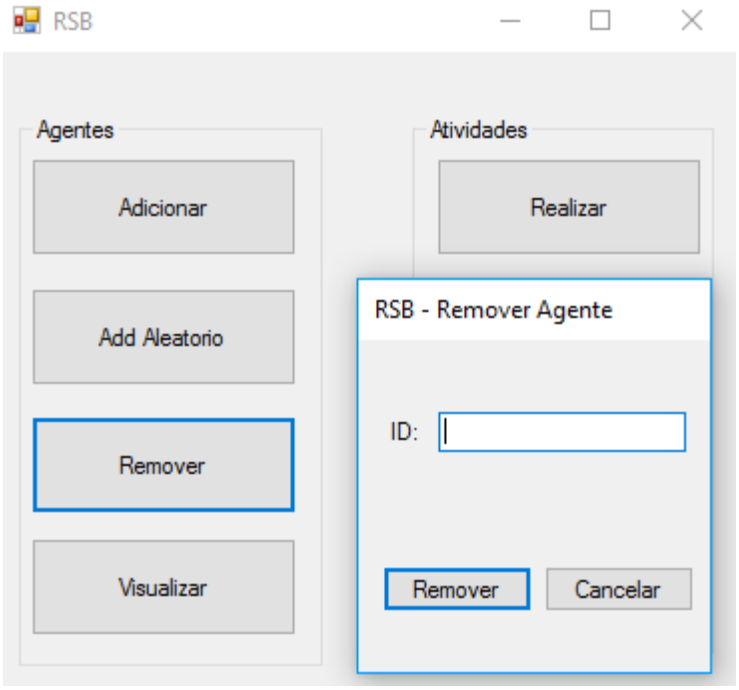
Caso o usuário deseje realizar a opção de inserção aleatória, é apresentado a ele a janela da Figura 9, mais simples em relação a outra, para informar o número de agentes e o valor inicial.

Figura 9. Janela de adição aleatória de agentes



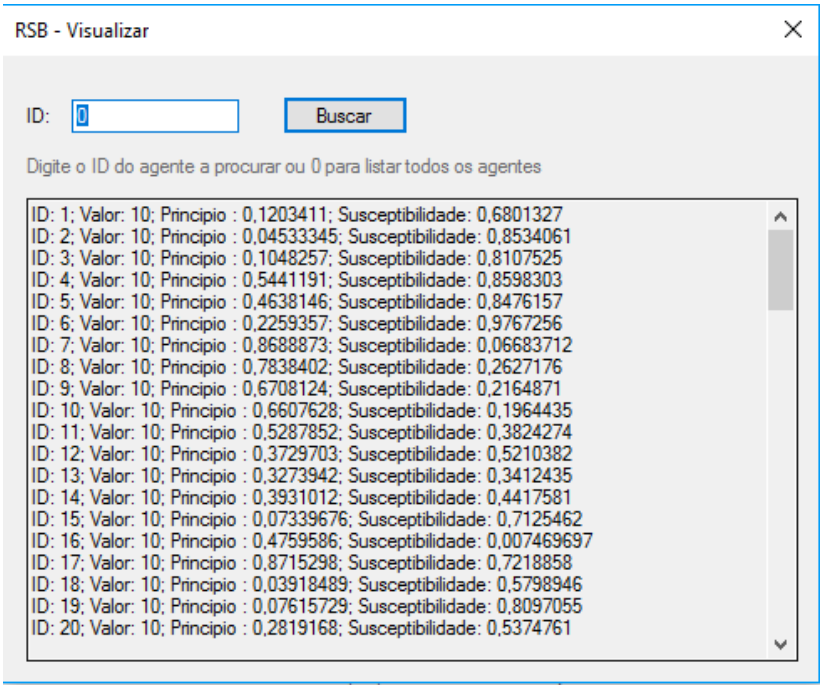
Ao selecionar a opção de remover agente, o usuário pode inserir o ID do agente através da janela demonstrada na Figura 10, removendo-o da população, caso ele exista.

Figura 10. Janela de remoção de agentes.



Caso o usuário queira visualizar a situação atual dos agentes na população, ele o pode fazer através da janela Visualizar disponível na Figura 11. O usuário pode inserir o ID do agente para procura-lo ou não informar o ID e exibir todos os agentes da população. Ao realizar a busca, serão fornecidos todos os dados disponíveis dos agentes.

Figura 11. Janela de visualização de agentes



Para realizar atividades, o usuário pode utilizar a opção para realizar uma atividade especifica, digitando o ID dos dois agentes desejados, como mostrando na Figura 12.

Caso deseje realizar atividades em lote ou não queira escolher os agentes, o usuário pode utilizar a opção de Realizar Aleatório para realizar as atividades. Como descrito na Figura 13. Com esta opção, os agentes serão escolhidos aleatoriamente, podendo escolher o quantitativo de atividades e o quantitativo de experimentos a se realizar.

Figura 12. Janela de realização de atividade

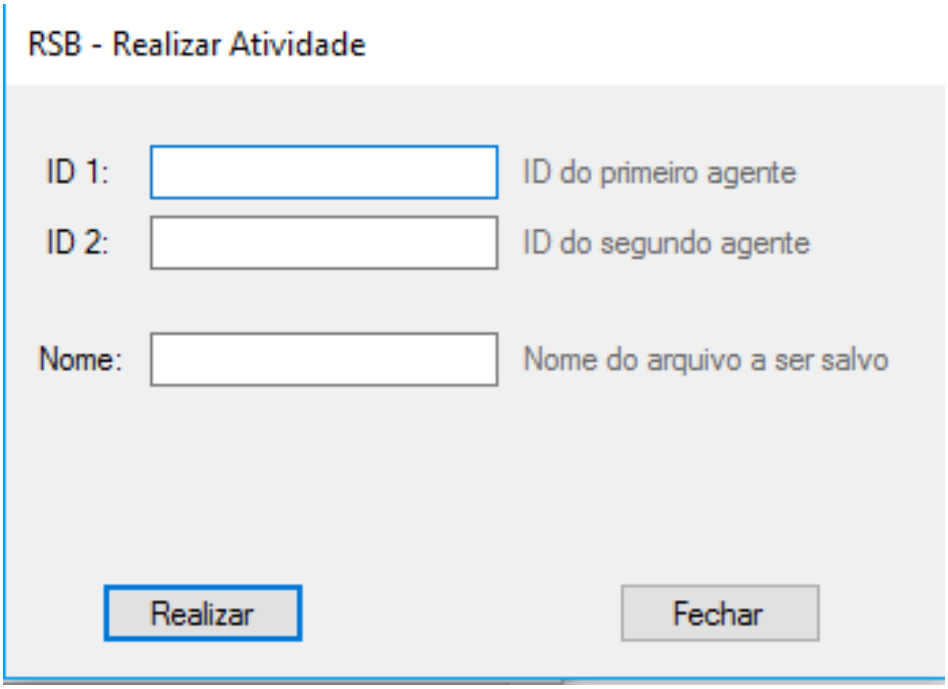
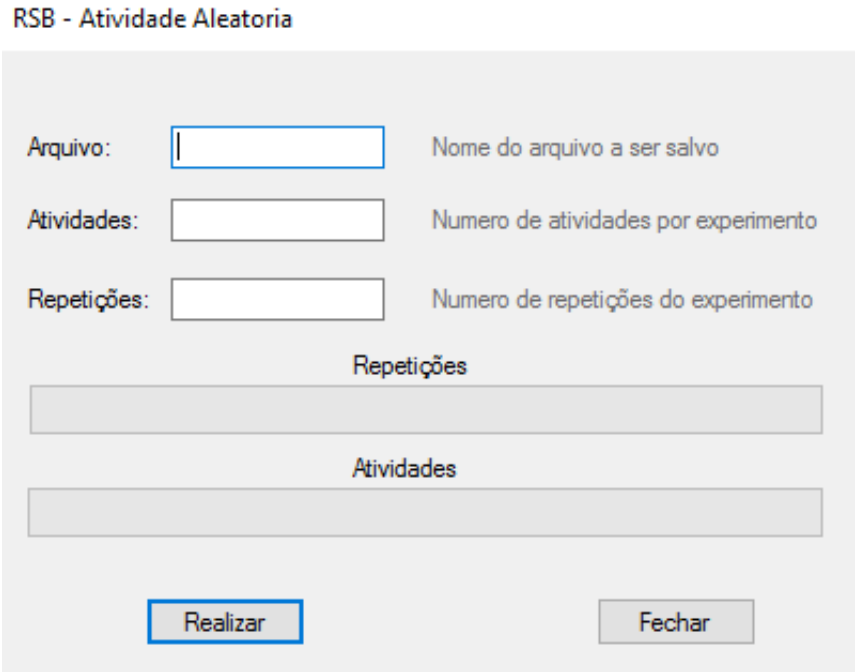


Figura 13. Janela de realização de atividade aleatória ou experimento.



Com estas opções, o usuário poderá então modelar e simular o experimento, podendo visualizar os resultados pelo próprio programa ou analisar os dados através de um arquivo no formato .CSV. Caso este queira visualizar os resultados através do próprio programa, este pode utilizar a janela de visualização de agentes (Figura 11),

no qual o sistema informa todos os dados de todos os agentes existentes. Caso este queira visualizar através do arquivo CSV, o usuário pode visualizar através de qualquer software de planilha todos os dados disponíveis na janela de visualização de agentes, além de detalhes de todas as atividades realizadas e como cada agente se comportou ao realizar atividades, tudo isto em planilhas separadas para cada caso (Figura 14) e para cada agente (Figura 15).

Figura 14. Visualização do CSV para os experimentos.

Identificador	Principio	Suceptibilidade	Valor Inicial	Valor Final
1	0,5	0,4823915	10	90
2	0,5	0,7669456	10	116
3	0,5	0,8568038	10	110,5
4	0,5	0,5640233	10	99
5	0,5	0,8356255	10	102
6	0,5	0,2808262	10	90,5
7	0,5	0,5122721	10	107
8	0,5	0,7505797	10	105,5
9	0,5	0,2113839	10	99
10	0,5	0,6018743	10	106,5
11	0,5	0,4813521	10	99
12	0,5	0,2089653	10	87
13	0,5	0,6019216	10	109,5
14	0,5	0,2265122	10	109
15	0,5	0,2620254	10	114,5

Por fim, para permitir a evolução do software por qualquer pessoa com o conhecimento necessário, o código-fonte é fornecido em conjunto com a ferramenta, liberado após autorização por e-mail. O código-fonte das etapas essenciais ao simulador está descrito no Apêndice B.

Figura 15. Visualização dos experimentos através de arquivo CSV

Agente 1	Agente 2	Lembrou	Suscetivel	Fator	Valor Final
149	33	False	False	1	8
33	34	False	False	0,5	8,5
33	52	False	False	0,5	9
33	80	False	False	0,5	9,5
42	33	False	False	0,5	10
18	33	False	False	0,5	10,5
33	75	False	False	0,5	11
64	33	False	False	0,5	11,5
21	33	True	True	0,5	12
46	33	False	False	0,5	12,5
45	33	False	False	0,5	13
33	25	False	False	0,5	13,5
60	33	False	False	0,5	14
33	82	True	False	0,5	14,5
33	12	True	False	0,5	15

6 SIMULAÇÃO DA FERRAMENTA

Para a realização de testes da ferramenta, foram realizadas algumas simulações. Estas simulações objetivam certificar que o RSB fornecerá resultados corretos.

6.1 MÉTODO

Para os resultados serem considerados corretos, o programa deverá fornecer saídas semelhantes a alguns resultados obtidos na literatura. Os resultados podem possuir alguma variação a literatura, visto a existência de discrepâncias na metodologia aplicada e ao fato de boa parte do algoritmo de execução do RSB ser considerado estocástico.

Para a execução das simulações foi definido pelo especialista a realização de um conjunto de atividades. Para padronizar as entradas, ficou definido que os experimentos deveriam conter os seguintes parâmetros:

1. O tamanho da população total será de 100 agentes, os quais deveriam iniciar com 10 de valor;
2. A receita total para cada atividade seria de 5 valor;
3. Caso exista custo, o valor de custo deveria ser em 2 valor;
4. Deveriam ser realizados 3 experimentos com 10000 atividades cada.
5. Os experimentos deveriam ser realizados com permanência de memória em 100% (sem perca).

Para a realização das simulações de testes, foram utilizados em sua completude ou adaptação os exemplos de modelagens dispostos no capitulo de Estado da Arte, sendo executados os conjuntos de experimentos abaixo:

- População totalmente egoísta e cabeça-dura;

- População totalmente egoísta e totalmente suscetível;
- População totalmente altruísta e cabeça-dura;
- População totalmente altruísta e totalmente suscetível;
- População igualitário e cabeça-dura;
- População igualitária e totalmente suscetível;
- População igualitária;
- Teste de sobrevivência celular (99 para 1):
  - Egoístas com altruístas;
  - Egoístas com igualitários;
  - Altruístas com igualitários;
- Teste de sobrevivência em conjunto (9 para 1):
  - Egoístas com altruístas;
  - Egoístas com igualitários;
  - Altruístas com igualitários.

Os testes acimas foram realizados nos dois modos de execução, que são os modos com custo e modo sem custo. Com isso totalizaram 60 testes, com os casos acima citados.

6.2 RESULTADOS

Devido a alta quantidade de testes, estão sendo apresentados e discutidos os resultados dos testes que forem considerados mais importantes para este trabalho. Então, o acesso a todas as planilhas geradas para os testes realizados está disponível no seguinte *link:* https://1drv.ms/f/s!ApVTEY1ZQuyugoE2BEppyPFZB8xD3w

Segue abaixo os resultados das simulações realizadas. Após a apresentação dos resultados, explicaremos o porquê de eles ocorrerem.

6.2.1 Teste de corretude

Nestes testes, foram testados vários cenários com diferenciados modelos de agentes para a verificação da corretude do algoritmo do programa.

Para cada teste, estará disposto uma tabela como a seguinte.

Tabela 3. Tabela modelo a ser utilizado na verificação de dados.

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	x	y	z	0
Com Custo	X	Y	Z	10

Segue as especificações de cada campo da tabela:

- Na linha Tipo, estão descritos o tipo da atividade, podendo se referir ao modo de execução da atividade, a estratégia do agente ou ambos;
- O Valor Total (x e X) é a média do valor de todos os agentes de um certo tipo, em todos os experimentos;
- O Valor Mínimo (y e Y) é a média dos menores valores dos agentes em todos os experimentos. No caso especifico dos testes de sobrevivências, são tomados os menores valores em um quantitativo igual a proporção do agente invasor;
- O Valor Máximo (z e Z) é a média dos maiores valores dos agentes em todos os experimentos. No caso especifico dos testes de sobrevivências, são tomados os menores valores em um quantitativo igual a proporção do agente invasor;
- Mortes é o percentual médio de mortes em todos os experimentos.

O Valor Total deverá seguir a formula 1, descrita abaixo. Seguem abaixo os testes executados.

$$Receita \times \left(\frac{Atividade}{Agente}\right) + Valor\ Inicial \qquad (1)$$

6.2.1.1 População Totalmente Egoísta Cabeça-Dura

Para o teste dos agentes egoístas cabeças-duras, o resultado ficou aleatório, com vencedores e perdedores diferentes. As médias dos valores estão dispostos na tabela abaixo.

Tabela 4. Resultado População Egoísta com Estratégia Cabeça-Dura

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	408,33	630	0
Com Custo	142,99	45,67	223	23

O resultado acima era esperado devido ao fato de todos os agentes com poder de decisão decidirem pegar todo o valor. Com essa atitude, os agentes que fossem escolhidos mais vezes como o ditador conseguiria o maior valor.

Como o fator de escolha é aleatório, com todos os agentes possuindo a mesma probabilidade de serem escolhidos, o resultado deveria ser aleatório com valores máximos e mínimos próximos entre as repetições, o que ocorreu.

Para o experimento com custo, a média de mortes entre cada experimento ficou em 23%, visto que todos são egoístas e pegam todo o valor possível para tentar sobreviver. Os valores totais, máximos e mínimos ficaram baixos devido a essa disputa por recurso, já que se um agente ficar sem valor será removido da população.

6.2.1.2 População Totalmente Egoísta e Suscetível

No teste dos agentes egoístas e suscetíveis, o resultado ficou aleatório, com vencedores e perdedores variados.

Tabela 5. Resultado População Egoísta com estratégia Olho por Olho

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	415	626,67	0
Com Custo	129,03	13,33	214,33	15,33

O resultado acima também era esperado, como no teste anterior, devido ao fato dos agentes totalmente suscetíveis executarem a estratégia do olho por olho. Como a população é completamente egoísta, independentemente dos agentes lembrarem ou não da ação do outro, estes sempre executarão a ação egoísta.

6.2.1.3 População Totalmente Altruísta e Cabeça-Dura

Para este teste, executamos o experimento com todos os agentes com valor de princípio 0, sendo estes considerados totalmente altruístas. Colocamos o valor de Suscetibilidade como sendo 0, para que eles tenham o comportamento Cabeça-Dura, ou seja, sempre seguirão o seu princípio.

Neste teste, o resultado também ficou aleatório, como esperado. O principal diferencial em relação aos testes com agentes egoístas é que neste o agente passivo, o que não possui poder de escolha, receberá todo o valor da proposta. Logo esperávamos que os agentes que fossem escolhidos mais vezes como segundo agente saíssem vencedores.

Tabela 6. Resultado População Altruísta com estratégia Cabeça-Dura

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	390	633,33	0
Com Custo	133,76	51,33	233,33	17,67

Ao observamos a tabela acima, conseguimos visualizar uma pequena melhora em relação ao comportamento de uma população totalmente egoísta, principalmente para atividades com custo.

6.2.1.4 População Totalmente Altruísta e Totalmente Suscetível

Neste teste, executamos o experimento com todos os agentes com valor de Princípio 0, para serem considerados altruístas, e valor de Suscetibilidade 1, para que eles tenham o comportamento Olho-por-Olho.

Assim como esperado, os resultados foram aleatórios, semelhante ao comportamento do teste totalmente altruísta anterior a este. Como explicado anteriormente, isto ocorre devido a todos terem o mesmo comportamento, logo não importa se os agentes “retaliem” as ações sofridas. Com este teste conseguimos testar se os agentes altruístas conseguiriam retaliar, o que foi comprovado.

Tabela 7. Resultado População Altruísta com estratégia Olho por Olho

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	396,67	640	0
Com Custo	137,39	62,67	236	19,67

Como podemos observar, a população acima obteve um resultado superior aos outros experimentos, apesar de ocorrerem mais mortes em relação ao outro experimento com agente altruísta. Ao observarmos as atividades de forma detalhada, visualizamos que os agentes que foram escolhidos mais vezes como o agente passivo venceram.

6.2.1.5 População Igualitária Cabeça-Dura

Para este teste, executamos o experimento com todos os agentes com valor de princípio 0,5, sendo considerados agentes igualitários, dividindo o valor recebido

igualmente. O valor de Suscetibilidade foi marcado em 0, sendo estes considerados Cabeças-duras, com o objetivo destes sempre dividirem o valor igualmente seguindo o seu princípio. Com esta configuração, os agentes que mais participarem, independentemente de serem ativos ou passivos, conseguiriam a maior pontuação, mas os vencedores e valores finais devem ser aleatórios devido a sua homogeneidade.

Assim como esperado, os resultados foram aleatórios, devido ao motivo de igualdade dos agentes. Os agentes vencedores foram os que mais participaram de atividades, e todos os valores finais foram múltiplos de 0,5.

Tabela 8. Resultado População Igualitária com estratégia Cabeça-Dura

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	419,17	602,5	0
Com Custo	110	91,33	126,67	0

Os valores ficaram próximos, comparados com os outros resultados, devido ao motivo de cada agente participante receber o mesmo valor por atividade.

Como esperado, na atividade com custo, não ocorreram mortes. Isto ocorreu por causa da divisão igualitária gerar uma receita de 0,5 para cada agente.

Para a média do valor total, observamos que a atividade sem custo obteve valor bruto 5 vezes maior que o valor da atividade sem custo. Isso ocorre devido a diferença de receita por atividade.

Como a receita da atividade sem custo é de 5 e a da atividade com custo é 1, a razão entre os dois deverá ficar em exatamente 5 para 1, finalmente acrescentando o valor 10.

6.2.1.6 População Igualitária Totalmente Suscetível

Neste teste, executamos o experimento com todos os agentes com valor de Princípio 0,5 e valor de Susceptibilidade 1, para que estes utilizem a estratégia olho por olho. Com esta configuração, deverá acontecer o mesmo do teste anterior, independente destes lembrarem ou não da ação sofrida.

Os resultados também foram aleatórios, pelo motivo do teste anterior, sendo os vencedores os que mais participarem. Como cada participação concedia 2,5 de valor para o agente, muitos deles ficaram com os valores não inteiros.

Tabela 9. Resultado População Igualitária com estratégia Olho por Olho

Tipo	Valor Total	Valor Mínimo	Valor Máximo	Mortes
Sem Custo	510(1)	428,33	601,67	0
Com Custo	110	93	134,16	0

Os valores ficaram bem próximos aos valores do teste anterior, mas conseguimos validar os dados com a informação de que os vencedores são os que mais participaram e que os valores tiveram esta proximidade pelo motivo da distribuição de escolha dos agentes está mais próxima.

6.2.1.7 Teste de Sobrevivência Celular Egoísta com Invasor Altruísta

Neste teste, os experimentos possuem um cenário com uma população completamente egoísta, então é inserido um agente altruísta para verificar como estes se comportam.

Para a realização deste experimento, criamos 99 agentes egoístas com valores de Susceptibilidade aleatórios. Para o agente invasor, criamos duas vertentes deste: Um altruísta cabeça-dura e um altruísta olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente altruísta deverá ficar abaixo da média do valor final dos agentes egoístas;
- No experimento com custo do agente cabeça-dura, se ele for escolhido para uma atividade 5 ou mais vezes, este deverá ficar sem valor e morrerá;
- No experimento do agente olho-por-olho, as chances do agente inserido morrer são baixas. A possibilidade de ganho do agente ocorre caso este seja escolhido como agente ativo em conjunto com um agente egoísta do qual ele já sofreu uma ação, retaliando. Outra possibilidade é no caso do agente egoísta utilizar também a estratégia do olho por olho.

Segue abaixo a tabela de execução dos experimentos:

Tabela 10. Resultado Teste de Sobrevivência entre Egoísta e Altruísta

Tipo	Valor Total Egoísta	Valor Total Altruísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	513,05	208,33	208,33	641,67	52,39	0
Cabeça Dura Com Custo	138,88	0	45,33	229	52,51	20,67
Olho por Olho Sem Custo	510,66	445	391,67	625	52,39	0
Olho por Olho Com Custo	136,8	0	46	230,33	52,43	19,33

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- O agente altruísta cabeça-dura ficou com o pior resultado em todos os experimentos, conseguindo o valor apenas quando os agentes egoístas “colaboravam” com o agente altruísta, utilizando a sua susceptibilidade;
- Quando a atividade envolvia custo, o agente altruísta morreu;

6.2.1.8 Teste de Sobrevivência Celular Egoísta com Invasor Igualitário

Neste teste, os experimentos possuem um cenário com uma população completamente egoísta, então é inserido um agente igualitário para verificar como eles se comportam.

Para a realização deste experimento, criamos 99 agentes egoístas com valores de Susceptibilidade aleatórios. Para o agente invasor, criamos duas vertentes deste: Um igualitário cabeça-dura e um igualitário olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente igualitário ficará abaixo da média do valor final dos agentes egoístas, mas com uma distância menor comparado ao experimento do agente altruísta;
- No experimento com custo do agente cabeça-dura, as chances dele ser removido são altas, o que ocorre por existir apenas 1% de chance deste ser escolhido como agente ativo;
- No experimento do agente olho-por-olho, as chances do agente inserido morrer são médias. A possibilidade de ganho do agente ocorre caso este seja escolhido como agente ativo ou caso um agente egoísta utilize a estratégia do olho por olho.

Segue abaixo a tabela de execução dos experimentos:

Tabela 11. Resultado Teste de Sobrevivência entre Egoísta e Igualitário

Tipo	Valor Total Egoísta	Valor Total Igualitário	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	511,49	362,5	355,83	629,17	52,39	0
Cabeça Dura Com Custo	135,38	0	50,17	229,33	52,7	18,67
Olho por Olho Sem Custo	510,6	450,83	371,67	645,83	52,39	0
Olho por Olho Com Custo	135,01	80	61,33	244,17	52,54	18,33

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- O agente igualitário cabeça-dura ficou com o pior resultado na maioria dos experimentos. Os casos em que este resultado não ocorreu foram os que ele foi selecionado muitas vezes como agente ativo;
- Quando a atividade envolvia custo, o agente igualitário morreu em sua maioria, sobrevivendo apenas em alguns casos que se utilizava da estratégia olho por olho;
- Quando o agente igualitário usou da estratégia olho por olho e não morria, ele não ficava com o pior resultado, apesar de ficar em resultados considerados ruins;

6.2.1.9 Teste de Sobrevivência Celular Altruísta com Invasor Egoísta

Neste teste, os experimentos contam com um cenário totalmente altruísta, então é inserido um agente egoísta para verificar como eles se comportam.

Para a realização deste experimento, criamos 99 agentes altruístas com valores de Susceptibilidade aleatórios. Para o agente invasor, criamos duas vertentes deste: Um egoísta cabeça-dura e um egoísta olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente egoísta ficará acima da média do valor final dos agentes altruístas;
- No experimento com custo do agente cabeça-dura, as chances dele ser removido são baixas, o que ocorre por ele só perder valor quando for escolhido para ser o agente passivo e o agente ativo retaliar;
- No experimento, as chances do agente inserido vencer são consideradas altas. Isso ocorre devido ao fato dele explorar sempre na primeira atividade, saindo com vantagem sempre na primeira interação com um agente altruísta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 12. Resultado Teste de Sobrevivência entre Altruísta e Egoísta

Tipo	Valor Total Altruísta	Valor Total Egoísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	506,47	860	381,67	860	46,14	0
Cabeça Dura	130,69	554,67	34	554,67	45,78	19

Com Custo						
Olho por Olho Sem Custo	508,11	696,67	370	696,67	46,14	0
Olho por Olho Com Custo	134,1	240	49	240	46,01	18,67

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- O agente egoísta ficou com o melhor resultado em todos os experimentos;
- Nas atividades que envolviam custo, os agentes altruístas que sobreviveram possuíam um valor de susceptibilidade abaixo do valor normal, o que raramente acontecia nos outros experimentos.

6.2.1.10 Teste de Sobrevivência Celular Altruísta com Invasor Igualitário

Neste teste, os experimentos contam com um cenário totalmente altruísta, então é inserido um agente igualitário para verificar como a população se comporta.

Para a realização deste experimento, criamos 99 agentes altruístas com valores de Susceptibilidade aleatórios. Para o agente invasor, criamos duas vertentes deste: Um igualitário cabeça-dura e um igualitário olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente igualitário ficará acima da média do valor final dos agentes altruístas;

- No experimento com custo do agente cabeça-dura, as chances dele ser removido são nulas, visto que ele nunca perderá valor;
- No experimento, as chances do agente inserido vencer são consideradas médias. Caso o agente seja escolhido várias vezes como agente ativo, este não conseguirá uma pontuação tão alta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 13. Resultado Teste de Sobrevivência entre Altruísta e Igualitário.

Tipo	Valor Total Altruísta	Valor Total Igualitário	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	508,48	645	392,5	662,5	46,14	0
Cabeça Dura Com Custo	128,94	360,67	32,5	360,67	46,05	16,33
Olho por Olho Sem Custo	509,39	570	404,17	630,83	46,14	0
Olho por Olho Com Custo	133,56	185,5	50,67	235	48,01	17,67

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- O agente igualitário cabeça-dura ficou com o melhor resultado na maioria dos experimentos;

- O agente igualitário olho por olho ficou com um desempenho médio por utilizar a estratégia após o primeiro contato com outro agente, que é altruísta;
- Nas atividades que envolviam custo com o agente igualitário utilizando a estratégia olho por olho, os sobreviventes tenderam a utilizar a mesma estratégia.

6.2.1.11 Teste de Sobrevivência Celular Igualitário com Invasor Egoísta

Neste teste, os experimentos contam com um cenário composto por agentes igualitário, então é inserido um novo agente egoísta para verificar como a população se comporta.

Para a realização deste experimento, criamos 99 agentes igualitários com valores de Susceptibilidade aleatórios. Para o agente invasor, criamos duas vertentes deste: Um egoísta cabeça-dura e um egoísta olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente egoísta ficará acima da média do valor final dos agentes igualitários;
- No experimento com custo do agente cabeça-dura, as chances dele ser removido são nulas, visto que ele nunca perderá valor;
- No experimento, as chances do agente inserido vencer são consideradas altas. Caso o agente seja escolhido várias vezes, este conseguirá uma pontuação alta;
- Nenhum agente morrerá.

Segue abaixo a tabela de execução dos experimentos:

Tabela 14.Teste de Sobrevivência entre Igualitário e Egoísta

Tipo	Valor Total Igualitário	Valor Total Egoísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	508,55	653,33	424,17	653,33	48,64	0
Cabeça Dura Com Custo	108,52	256,5	91,5	256,5	48,64	0
Olho por Olho Sem Custo	509,52	557,5	431,67	584,17	48,64	0
Olho por Olho Com Custo	109,25	184,33	90,83	184,33	48,64	0

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- O agente egoísta ficou com o melhor resultado na maioria dos experimentos;
- Nenhum agente morreu;
- Apesar do comportamento geral da população, os agentes egoístas mantiveram um desvio-padrão baixo em relação a toda população.

6.2.1.12 Teste de Sobrevivência Celular Igualitário com Invasor Altruísta

Neste teste, o cenário é composto por agentes igualitário, então é inserido um novo agente altruísta para verificar como a população se comporta.

Para a realização deste experimento, criamos 99 agentes igualitários com valores aleatórios de Susceptibilidade. Para o agente invasor, criamos duas vertentes deste: Um altruísta cabeça-dura e um altruísta olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente altruísta ficará abaixo da média do valor final dos agentes igualitários;
- No experimento com custo do agente cabeça-dura, as chances dele ser removido são altas, visto que ele perderá 2 de valor sempre que for o agente ativo, ganhando apenas 0,5 de valor quando for o passivo caso o outro agente não retribua;
- No experimento sem custo do agente olho por olho, a média do valor final do agente altruísta deverá ficar abaixo, mas próximo da média dos agentes igualitários;
- No experimento com custo do agente olho por olho, este poderá ser removido caso seja escolhido várias vezes como agente ativo sem ter sofrido ação dos agentes passivos anteriormente;
- No experimento, as chances do agente inserido vencer são consideradas baixas. Caso o agente seja escolhido várias vezes como agente passivo, este conseguirá uma pontuação alta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 15. Resultado Teste de Sobrevivência entre Igualitário e Altruísta

Tipo	Valor Total Igualitário	Valor Total Altruísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	511,41	370	370	608,33	48,64	0
Cabeça Dura	111,12	0	94,83	126,67	48,64	1

Com Custo						
Olho por Olho Sem Custo	510,34	475,83	431,67	588,33	48,64	0
Olho por Olho Com Custo	110,62	49,17	81,17	127	48,64	0,33

Observando os resultados dispostos na tabela acima, observamos os seguintes pontos:

- O agente altruísta ficou com o pior resultado em todos os experimentos;
- Somente o agente altruísta morreu;
- Os agentes altruístas somente não morreram nas atividades com custo quando estes eram escolhido várias vezes como agente passivo.

6.2.1.13 Teste de Sobrevivência Conjunto Egoísta com Invasor Altruísta

Neste teste, os experimentos possuem um cenário com uma população completamente egoísta, então é inserido um grupo de agentes altruístas para verificar como estes se comportam.

Para a realização deste experimento, criamos 90 agentes egoístas com valores de Susceptibilidade aleatórios. Para os agentes invasores, criamos duas vertentes deste: 10 altruístas cabeça-dura e 10 altruístas olho-por-olho. A modificação da quantidade dos agentes ocorreu para criar a ideia de grupo, podendo assim um integrante de um grupo testado realizar atividades com outro do mesmo grupo. Com isto, definisse o teste de sobrevivência em conjunto, no qual um pequeno grupo de agentes tenta sobreviver a ataques de um grupo maior já fixado.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final dos agentes altruístas deverá ficar abaixo da média do valor final dos agentes egoístas;
- No experimento com custo do agente cabeça-dura, os agentes altruístas deverão morrer se executarem muitas atividades;
- No experimento do agente olho-por-olho, as chances dos agentes inseridos morrer são altas. Eles morrerão caso executem várias tarefas como agente passivo, exceto se o executarem entre seu próprio grupo.

Segue abaixo a tabela de execução dos experimentos:

Tabela 16. Resultado do Teste de Grupo entre Egoísta e Altruísta

Tipo	Valor Total Egoísta	Valor Total Altruísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	544,19	202,33	202,33	678,33	44,97	0
Cabeça Dura Com Custo	146,92	0	56,67	253,33	43,92	25
Olho por Olho Sem Custo	518,52	433,33	407	658,33	44,97	0
Olho por Olho Com Custo	144,52	26,67	85,7	254,33	45,02	23,67

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- Os agentes altruístas ficaram com o pior resultado em todos os experimentos, conseguindo o valor apenas quando os agentes egoístas “colaboravam” com o agente altruísta, utilizando a sua susceptibilidade;
- Quando a atividade envolvia custo, 97% dos agentes altruístas morreram e os sobreviventes ficaram com valor quase nulo;

6.2.1.14 Teste de Sobrevivência Conjunto Egoísta com Invasor Igualitário

Neste teste, os experimentos possuem um cenário com uma população completamente egoísta, então é inserido um grupo de agentes igualitários para verificar como eles se comportam.

Para a realização deste experimento, criamos 90 agentes egoístas com valores de Susceptibilidade aleatórios. Para o grupo de agente invasor, criamos duas vertentes deste: Um grupo com 10 agentes igualitário cabeça-dura e outro com 10 agentes igualitário olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do agente cabeça-dura, a média do valor final do agente igualitário ficará abaixo da média do valor final dos agentes egoístas, mas com uma distância menor comparado ao experimento do agente altruísta;
- No experimento com custo do agente cabeça-dura, as chances dos agentes invasores serem removido são altas, o que ocorre por existir apenas 10% de chance deste ser escolhido como agente ativo, lucrando 0,5 por atividade, enquanto possui uma probabilidade de aproximadamente 9% de ser escolhido como agente passivo contra um agente egoísta, acumulando um prejuízo de 2.

Segue abaixo a tabela de execução dos experimentos:

Tabela 17.Resultado Teste em Grupo entre Egoísta e Igualitário

Tipo	Valor Total Egoísta	Valor Total Igualitário	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	528,1	347,08	347,33	615,58	44,97	0
Cabeça Dura Com Custo	144,66	0	45,83	251,33	45,88	23,67
Olho por Olho Sem Custo	515,54	460,17	425,67	603,58	44,97	0
Olho por Olho Com Custo	138,25	87,98	74,33	205,17	45,04	18,33

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- Os agentes igualitário cabeça-dura ficaram com o pior resultado na maioria dos experimentos. Os casos em que este resultado não ocorreu foram os que ele foi selecionado muitas vezes como agente ativo;
- Quando a atividade envolvia custo, os agentes igualitários morreram em sua maioria, sobrevivendo apenas em alguns casos que se utilizava da estratégia olho por olho;
- Quando os agentes igualitários usaram da estratégia olho por olho e não morria, ele não ficava com o pior resultado, apesar de ficar em resultados considerados ruins;
- Os resultados se assemelharam ao do teste de sobrevivência celular do mesmo tipo.

6.2.1.15 Teste de Sobrevivência Conjunto Altruísta com Invasor Egoísta

Neste teste, os experimentos foram compostos por um cenário totalmente altruísta, então é inserido um grupo de agentes egoístas para verificação de seus comportamentos.

Para a realização deste experimento, criamos 90 agentes altruístas com valores de Susceptibilidade aleatórios. Para o grupo de agentes invasores, criamos duas vertentes: Um grupo de 10 agentes egoísta cabeça-dura e outro grupo com 10 agentes egoísta olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo dos agentes cabeça-dura, a média do valor final dos agentes invasores ficará acima da média do valor final dos agentes altruístas;
- No experimento com custo dos agentes cabeça-dura, as chances deles serem removido são baixas, o que ocorre por ele só perder valor quando for escolhido para ser o agente passivo e o agente ativo retaliar ou for outro agente egoísta;
- No experimento, as chances dos agentes inseridos vencerem são consideradas altas. Isso ocorre devido ao fato dele explorar sempre na primeira atividade, saindo com vantagem sempre na primeira interação com outro agente.

Segue abaixo a tabela de execução dos experimentos:

Tabela 18. Resultado Teste de Grupo entre Altruísta e Egoísta

Tipo	Valor Total Altruísta	Valor Total Egoísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	474,76	818,17	426,5	818,17	46,89	0

Cabeça Dura Com Custo	102,1	567,33	45,73	567,33	48,97	37,67
Olho por Olho Sem Custo	500,69	593,83	427,83	613,33	46,89	0
Olho por Olho Com Custo	132,51	235,47	81,3	239,57	48,5	24,67

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- Os agentes egoístas ficaram entre os melhores resultados em todos os experimentos;
- Quando os agentes egoístas utilizavam a estratégia cabeça-dura, eles obtiveram os melhores resultados;
- Nas atividades que envolviam custo, os agentes altruístas morreram em uma taxa superior ao normal, o que aconteceu devido ao fato de disputarem com 10 agentes egoístas.

6.2.1.16 Teste de Sobrevivência Conjunto Altruísta com Invasor Igualitário

Neste teste, os experimentos contam com um cenário totalmente altruísta, então é inserido um grupo de agentes igualitários para verificar como a população se comporta.

Para a realização deste experimento, criamos 90 agentes altruístas com valores de Susceptibilidade aleatórios. Para os agentes invasores, criamos duas vertentes deste: Um grupo com 10 agentes igualitários cabeça-dura e um grupo com 10 agentes igualitários olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo com invasores cabeça-dura, a média do valor final dos agentes igualitários ficará acima da média do valor final dos agentes altruístas;
- No experimento com custo dos agentes cabeça-dura, as chances deles serem removidos são nulas, visto que ele nunca perderá valor;
- No experimento, as chances do agente inserido vencer são consideradas médias. Caso o agente seja escolhido várias vezes como agente ativo, este não conseguirá uma pontuação tão alta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 19. Teste de Grupo entre Altruísta e Igualitário

Tipo	Valor Total Altruísta	Valor Total Igualitário	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	490,7	684,33	410,83	689,75	46,89	0
Cabeça Dura Com Custo	114,01	314,93	57,52	314,93	46,58	21
Olho por Olho Sem Custo	504,08	563,33	422,25	603,75	46,14	0
Olho por Olho Com Custo	126,7	184,98	72,03	203,97	47,72	17,67

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- Os agentes igualitários cabeça-dura ficaram com o melhor resultado na maioria dos experimentos;
- Os agentes igualitários olho por olho ficaram com um desempenho médio por utilizar a estratégia após o primeiro contato com outro agente, que é altruísta, ficando em média com 50% dos 10 melhores resultados.

6.2.1.17 Teste de Sobrevivência Conjunto Igualitário com Invasor Egoísta

Neste teste, os experimentos contam com um cenário composto por agentes igualitário, então é inserido um grupo com 10 agentes egoístas para verificar como a população se comporta.

Para a realização deste experimento, criamos 90 agentes igualitários com valores de Susceptibilidade aleatórios. Para o conjunto de agentes invasores, criamos dois grupos: Um grupo com 10 egoístas cabeça-dura e outro com 10 egoístas olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo do conjunto de agentes cabeça-dura, a média do valor final dos agentes egoístas ficará acima da média do valor final dos agentes igualitários;
- No experimento com custo dos agentes cabeça-dura, as chances deles serem removidos são nulas, visto que eles nunca perderão valor;
- No experimento, as chances dos agentes inseridos vencer são consideradas altas. Caso o agente seja escolhido várias vezes, este conseguirá uma pontuação alta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 20. Resultado Teste de Grupo entre Igualitário e Egoísta

Tipo	Valor Total Igualitário	Valor Total Egoísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	493,02	662,83	433,67	663,08	47,82	0
Cabeça Dura Com Custo	93,57	257,88	75,43	257,88	47,82	0
Olho por Olho Sem Custo	505,79	547,92	444,42	585,08	47,82	0
Olho por Olho Com Custo	104,98	155,2	90,32	155,77	47,82	0

Observando os resultados dispostos na tabela, observamos os seguintes pontos:

- Os agentes egoístas ficaram com o melhor resultado na maioria dos experimentos;
- Nenhum agente morreu;
- Apesar do comportamento geral da população, os agentes egoístas mantiveram um desvio-padrão baixo em relação a toda população.

6.2.1.18 Teste de Sobrevivência Conjunto Igualitário com Invasor Altruísta

Neste teste, o cenário é composto por agentes igualitário, então é inserido um grupo de agentes altruístas para verificarmos como a população se comporta.

Para a realização deste experimento, criamos 90 agentes igualitários com valores de Susceptibilidade aleatórios. Para os agentes invasores, criamos duas vertentes deste: Um grupo com 10 altruístas cabeça-dura e um grupo com 10 altruístas olho-por-olho.

Como resultado, esperávamos pelos acontecimentos seguintes:

- No experimento sem custo dos agentes cabeça-dura, a média do valor final dos agentes altruístas ficará abaixo da média do valor final dos agentes igualitários;
- No experimento com custo do agente cabeça-dura, as chances de um agente ser removido são altas, visto que ele perderá 2 de valor sempre que for o agente ativo, ganhando apenas 0,5 de valor quando for o passivo caso o outro agente não retribua ou 3 de valor caso o outro agente seja altruísta;
- No experimento sem custo do agente olho por olho, a média do valor final do agente altruísta deverá ficar abaixo, mas próximo da média dos agentes igualitários;
- No experimento com custo do agente olho por olho, este poderá ser removido caso seja escolhido várias vezes como agente ativo sem ter sofrido ação dos agentes passivos anteriormente;
- No experimento, as chances dos agentes inseridos vencerem são consideradas baixas. Caso o agente seja escolhido várias vezes como agente passivo, este conseguirá uma pontuação alta.

Segue abaixo a tabela de execução dos experimentos:

Tabela 21. Resultado Teste de Grupo Igualitário com agente Altruísta

Tipo	Valor Total Igualitário	Valor Total Altruísta	Valor Mínimo	Valor Máximo	Suscep	Morte
Cabeça Dura Sem Custo	527,51	352,42	351,75	584,67	47,82	0

Cabeça Dura Com Custo	122,29	0	109,69	136	47,82	10
Olho por Olho Sem Custo	515,03	464,75	435,75	573,33	47,82	0
Olho por Olho Com Custo	117,29	71,1	80,77	132,07	47,82	3,67

Observando os resultados dispostos na tabela acima, observamos os seguintes pontos:

- Os agentes altruístas ficaram com o pior resultado em todos os experimentos;
- Somente os agentes altruístas morreram;
- Os agentes altruístas somente não morreram nas atividades com custo quando estes eram escolhido várias vezes como agente passivo.

7 CONCLUSÃO E TRABALHOS FUTUROS

Com os estudos realizados, foram analisados os principais jogos utilizados em divisão de recursos e a evolução da Teoria dos Jogos. Observou-se também que existem duas abordagens ao tema, que é a analise comportamental das pessoas em atividades de divisão de recursos e a modelagem e simulação dos comportamentos.

Como podemos observar, apesar dos pesquisadores trabalharem a um bom tempo com computação com o objetivo de auxiliar na exploração dessas abordagens, não encontramos uma ferramenta fácil de usar para que outros pesquisadores e atuantes da área a utilizem.

Então, com o auxílio de um especialista Doutor da área de Psicologia Cognitiva, foi criado uma ferramenta com o objetivo de modelar e simular estas atividades, sendo este simulador prático, flexível e correto. Demonstramos a evolução na construção desta ferramenta, os testes realizados para verificação da corretude e a disponibilizamos para alguns pesquisadores da área para que estes pudessem testar.

O simulador foi criado de forma a ser simples em seu núcleo e interface gráfica, com o objetivo de ser utilizado por pessoas com pouco ou nenhum conhecimento em linguagem de programação de forma efetiva e ser estendido sem dificuldade por pesquisadores que possuam esse conhecimento, podendo assim de forma rápida acrescentar novos jogos, variáveis, comportamentos e possibilidades.

Como resultado dos testes realizados com os usuários, não foram obtidos resultados suficientes para realizar uma análise qualitativa ou quantitativa, utilizando os resultados obtidos como *feedback* para melhorias futuras. Das respostas obtidas, em conjunto com a ajuda do auditor especialista, conseguimos extrair a informação necessária verificar os pontos forte e fracos da ferramenta criada.

7.1 VANTAGENS E DESVANTAGENS DA FERRAMENTA

Como vantagem, a ferramenta possui flexibilidade na modelagem dos agentes, permitindo a criação de uma infinidade de agentes utilizando o conceito de Principio e

Susceptibilidade. Com o RSB, é possível realizar todo esse conjunto de modelagem e simulá-lo através de uma interface simples e fácil de usar, mesmo apresentando pontos de melhorias, sem a necessidade de conhecimento em linguagem de programação. Por fim, é possível acrescentar novos módulos na ferramenta sem impactar as criações já realizadas, desde que o usuário possua conhecimento na linguagem C#.

Como desvantagem, existem funcionalidades que necessitam de maior abstração, como a criação de agentes. Também foi julgado necessário a modelagem de outros jogos típicos, como o Dilema do Prisioneiro e o Jogo do Bem Comum, ficando esta modelagem apresentada como melhorias da ferramenta.

7.2 CONTRIBUIÇÕES

Ao longo desse trabalho foi possível verificar os conceitos básico para a ideação e implementação de um simulador para a divisão de recursos. Foram observados os modelos que já existem e as necessidades desses modelos para aplicabilidade aos pesquisadores da área sem conhecimentos gerais de programação. Esta pesquisa possibilitou a criação de um simulador simples e fácil de utilizar para os pesquisadores da área com esse perfil, mas flexível para que os pesquisadores com o conhecimento especifico na linguagem pudessem expandir.

O software recebeu boas críticas dos testadores, mas como foi observado, existem melhorias a serem realizadas. Apesar disto, os envolvidos nos testes relataram a possibilidade de utilizarem a ferramenta como ponto referencial em suas pesquisas.

7.3 TRABALHOS FUTUROS

A início, como possibilidade de trabalho futuro, pode-se observar como a ferramenta será recebida pelo público-alvo, a fim de receber *feedback* dos usuários para realizar melhorias.

Como observado, a ferramenta possui uma flexibilidade em seu núcleo permitindo a sua expansão facilitada, mas até o momento de finalização deste

trabalho, o *RSB* possuía apenas a modelagem de dois jogos típicos, o Dilema do Ditador e o Jogo do Ultimato. Então, uma proposta para dar continuidade ao trabalho é a modelagem de outros jogos típicos da área de Divisão de Recursos e de Teoria dos Jogos. Em conjunto com esta etapa, é possível realizar um teste de facilidade de uso com os testadores a fim de verificar a facilidade de uso proporcionada pela interface gráfica do simulador. Com isto, seria possível realizar também melhorias nesta interface.

Outra abordagem que pode ser tomada para melhorias desse trabalho é a ideação de meios de abstração para que qualquer usuário, sem necessidade de conhecimentos de programação, consiga expandir o núcleo da ferramenta. Para isto, a ideação possui várias abordagens, como a criação de uma Linguagem de Domínio Especifico para o tema tratado ou o uso de outras tecnologias para realizar a abstração.

REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, I. **Engenharia de Software.** 9ª edição. São Paulo: Pearson Education, 2011.

MYERSON, R. B. **Game theory:** analysis of conflict. Paperback ed. Cambridge: First Harvard University Press, 1997.

SALEN, K.; ZIMMERMAN, E. **Regras do Jogo:** Fundamentos do Design de Jogos. São Paulo: Blucher, 2012.

NEUMANN, J. von. **Zur Theorie der Gesellschaftsspiele**. Mathematische Annalen, vol 100, pg. 295-320. 1928.

AXELROD, R. **A Evolução da Cooperação**. Tradução de Jusella Santos. São Paulo: Leopardo Editora, 2010.

AXELROD, R. M. **The Complexity of Cooperation:** agente based models of competition and collaboration. New Jersey: Princeton University Press, 1997.

FISHER, R. A. **The Genetical Theory of Natural Selection**. Oxford: Oxford University Press, 1930.

DAWKINS, R. **The Selfish Gene**. Oxford: Oxford University Press, 1976.

AMAZON. **Amazon best-sellers**. Disponível em: <http://www.amazon.com/gp/bestsellers/books/13535/ref=zg_b_bs_13535_1>. Acesso em 12 de Fevereiro de 2016.

ICHINOSE, G.; SAYAMA, H. **Evolution of Fairness in the Not Quite Ultimatum Game**. Nature, 2014. Disponível em: <http://www.nature.com/articles/srep05104> Acesso em 17 de Fevereiro de 2016.

MARTINS, S. L. **Lições aprendidas sobre a escolha e desenvolvimento de ferramentas de entretenimento digitais em experimento sobre divisão de recursos**. 2015. 154 f. Dissertação (Mestrado em Informática Aplicada) – Universidade Federal Rural de Pernambuco, Recife.

KRISTIN, M. et al. **Is costly punishment altruistic? Exploring rejection of unfair offers in the Ultimatum Game in realworld altruists**. Nature, 2016. Disponível em: <http://www.nature.com/articles/srep18974>. Acesso em: 12 de Fevereiro de 2016.

MANSURY, Y.; DIGGORY, M.; DEISBOECK, T. S. **Evolutionary game theory in an agent-based brain tumor model:** Exploring the ‘Genotype-Phenotype’ link. Journal of Theoretical Biology, vol 238, p. 146-156, 2006.

TOMASINO, B, et al. **Framing the ultimatum game: the contribution of simulation**. Frontiers in Human Neuroscience, 2013.

NATIONAL RESEARCH COUNCIL. **Modeling Human and Organizational Behavior:** Application to Military Simulations. National Academy Press, 1998;

MOSS, S.; DAVIDSSON, P. **Multi-Agent-Based Simulation, Lecture Notes** in Artificial Intelligence, Vol. 1979, Springer-Verlag, 2001.

TAKADAMA, K.; KAWAI, T; KOYAMA. Y. **Micro- and Macro-Level Validation in Agent-Based Simulation: Reproduction of Human-Like Behaviors and Thinking in a Sequential Bargaining Game**. Journal of Artificial Societies & Social Simulation, Vol. 11 Issue 2, 2008.
.
PYNADATH, D.V.; MARSELLA, S.C. **PsychSim: modeling theory of mind with decision-theoretic agents**. In: IJCAI'05, p. 1181, 2005.

ZHANG, Y.; LEEZER, J. **Simulating human-like decisions in a memory-based agent model**. Comput Math Organ Theory, vol 16, p. 373–399, 2010.

KAHNEMAN D. **Maps of bounded rationality:** a perspective on intuitive judgment and choice. The Nobel Prizes Lecture, 2002.

AN, L. **Modeling human decisions in coupled human and natural systems:** Review of agente-based models. Ecological Modelling, vol. 229, p. 25-36, 2012.

LEE, C. S. **Multi-objective game-theory models for conflict analysis in reservoir watershed management.** Chemosphere, vol. 87; p. 608-613, 2012.

TAN, R. et al. **A game-theory based agent-cellular model for use in urban growth simulation:** A case study of the rapidly urbanizing Wuhan area of central China. Computers, Environment and Urban Systems, vol. 49; p. 15-29, 2015.

SANDERS, J.; KANDROT, E. **CUDA BY EXAMPLE:** Na Introduction to General-Purpose GPU Programming. Pearson, 2012.

NVIDIA CORPORATION (a). **Stanfords social robot jackrabbot seeks to understand pedestrian behavior**. Disponível em: <https://news.developer.nvidia.com/stanfords-social-robot-jackrabbot-seeks-to-understand-pedestrian-behavior/>. Acesso em: 19 de Junho de 2016.

NVIDIA CORPORATION (b). **Analyzing scientific data from Mars with Deep Learning.** Disponível em: <https://news.developer.nvidia.com/analyzing-scientific-data-from-mars-with-deep-learning/>. Acesso em 19 de Junho de 2016.

NVIDIA CORPORATION (c). **Share Your Science: Machine Learning helps fortune 500 companies hunt cyber-attacks and outages.** Disponível em: <https://news.developer.nvidia.com/share-your-science-machine-learning-helps-fortune-500-companies-hunt-cyber-attacks-and-outages/>. Acesso em 19 de Junho de 2016.

KASPRZYK, A. **Kasprzyk A-Life.** <Online>, 2002. Disponível em: <*http://www.kasprzyk.demon.co.uk/www/alife/index.html*> Acesso em 22 de janeiro de 2016.
FISCHBACHER, U. **z-tree**: Zurich toolbox for ready-made economic experiments, Experimental Economics, vol. 10, no. 2, p. 171–178, 2007.

SERENDIP STUDIO; **Serendip Studio**. <Online>, 2010. Disponível em: <http://serendip.brynmawr.edu/bb/pd.html>. Acesso em 22 de Janeiro de 2016.

DAVIS, W (a). **Iterated Prisoners Dilemma.net.** <Online>, 2007. Disponível em <http://www.iterated-prisoners-dilemma.net/>. Acesso em 22 de Janeiro de 2016.

DAVIS, W (b). **Spartialised Prisoners Dilemma.net.** <Online> , 2007. Disponível em: <http://www.spatialised-prisoners-dilemma.net/>. Acesso em 22 de Janeiro de 2016.

REYNOLDS, C. W. **Steering Behaviors For Autonomous Characters**. in the proceedings of Game Developers Conference 1999. Miller Freeman Game Group, San Francisco, p. 763-782, 1999.

SOPHIE. **SoPHIE Tutorial: Ultimatum Game.** <Online>, 2016. Disponível em <http://www.sophie.uni-osnabrueck.de/docs/tutorials/UG.1.4.pdf>. Acesso em 12 de Setembro de 2016.

MILLISECOND. **Millisecond Ultimatum Game.** <Online>, 2016. Disponível em < http://www.millisecond.com/download/library/UltimatumGame/>. Acesso em 17 de Setembro de 2016.

KAHNEMAN, D.; KNETSCH, J. L.; THALER, R. H. **Fairness and the assumptions of economics**. Journal of business, p. S285-S300, 1986.

GÜTH, W., SCHMITTBERGER, SCHWARZE. **An Experimental Analysis of Ultimatum Bargaining**. Journal of Economic Behavior and Organization. 3 (4): 367–388, 1982.

APÊNDICE A – QUESTIONÁRIO APLICADO SOBRE USO DA FERRAMENTA

RSB – Questionário

Este questionário visa obter informações sobre o programa Resource Sharing Behaviors (RSB). As informações serão utilizadas para validação de resultados parciais e para a realização de melhoramentos na ferramenta.

1. Você achou o RSB uma ferramenta fácil de utilizar?

    a. Sim

    b. Não

2. Você achou o RSB uma ferramenta útil?

    a. Sim

    b. Não

3. Você utilizaria o RSB como uma (ou a) ferramenta para simulações das atividades envolvendo divisão de recursos abordados?

    a. Sim

    b. Não

4. Você recomendaria o RSB para algum pesquisador da área?

    a. Sim

    b. Não

5. Se fossem acrescentados novos jogos, você acredita que a abordagem utilizada pelo programa seria simples e funcional para estes jogos?

   a. Sim

   b. Não

6. Você teve alguma dificuldade ou dúvida em alguma funcionalidade?

   a. Sim

   b. Não

   6.1. Em qual (ou quais) funcionalidade (s) você teve dificuldade?

   6.2. Qual a dificuldade encontrada?

7. Você encontrou algum erro ou inconsistência na ferramenta?

   a. Sim

   b. Não

   7.1. Por favor, descreva os erros encontrados.

8. Você acha que poderíamos melhorar a ferramenta em algum aspecto?

   a. Sim

   b. Não

   8.1. O que poderíamos melhorar?

9. Em uma escala de 0 a 10 (sendo 0 muito insatisfeito e 10 muito satisfeito), o quanto você está satisfeito com os resultados da ferramenta?

10. Em uma escala de 0 a 10 (sendo 0 muito difícil e 10 muito fácil), o quanto você achou essa ferramenta fácil de utilizar?

11. Em uma escala de 0 a 10 (sendo 0 totalmente inútil e 10 totalmente útil), o quanto você achou essa ferramenta útil?

12. Qual a sua experiência na área abordada pelo programa?

13. Caso deseje, poderia inserir seu nome?

14. Caso deseje, poderia inserir um e-mail para contato?

APÊNDICE B – CÓDIGO-FONTE DO NÚCLEO DA FERRAMENTA

```
    public class Agent
    {
        public int id;
        public float valor, valorInicial, principio, suceptibilidade, rejeicao,
evolucao, evoluiMenos, evoluiMais;
        public List<Memoria> memoria;
        public Agent(int id, float value, float princip, float sucept, int tamanho)
        {
            this.id = id;
            valor = value;
            principio = princip;
            suceptibilidade = sucept;
            if (principio > 1)
                principio = 1;
            if (principio < 0)
                principio = 0;
            if (suceptibilidade > 1)
                suceptibilidade = 1;
            if (suceptibilidade < -1)
                suceptibilidade = -1;
            memoria = new List<Memoria>(tamanho);
        }
        public Agent(int id, float value, float princip, float sucept, float reject,
int tamanho)
        {
            this.id = id;
            valor = value;
            principio = princip;
            suceptibilidade = sucept;
            rejeicao = reject;
            if (principio > 1)
                principio = 1;
            if (principio < 0)
                principio = 0;
            if (suceptibilidade > 1)
                suceptibilidade = 1;
            if (suceptibilidade < -1)
                suceptibilidade = -1;
            if (rejeicao < 0)
                rejeicao = 0;
            if (rejeicao > 1)
                rejeicao = 1;
            memoria = new List<Memoria>(tamanho);
        }
        public Agent(int id, float value, float princip, float sucept, int tamanho,
float fatorEvolucao, float menos, float mais)
        {
            this.id = id;
            valor = value;
            principio = princip;
            suceptibilidade = sucept;
            evolucao = fatorEvolucao;
            evoluiMenos = menos;
            evoluiMais = mais;
            if (principio > 1)
                principio = 1;
            if (principio < 0)
                principio = 0;
```

```
            if (suceptibilidade > 1)
                suceptibilidade = 1;
            if (suceptibilidade < -1)
                suceptibilidade = -1;
            if (evolucao < 0)
                evolucao = 0;
            if (evolucao > 1)
                evolucao = 1;
            if (evoluiMenos < 0)
                evoluiMenos = 0;
            if (evoluiMenos > 1)
                evoluiMenos = 1;
            if (evoluiMais < 0)
                evoluiMais = 0;
            if (evoluiMais > 1)
                evoluiMais = 1;
            memoria = new List<Memoria>(tamanho);
        }
        public Agent(int id, float value, float princip, float sucept, float reject,
int tamanho, float fatorEvolucao, float menos, float mais)
        {
            this.id = id;
            valor = value;
            principio = princip;
            suceptibilidade = sucept;
            rejeicao = reject;
            evolucao = fatorEvolucao;
            evoluiMenos = menos;
            evoluiMais = mais;
            if (principio > 1)
                principio = 1;
            if (principio < 0)
                principio = 0;
            if (suceptibilidade > 1)
                suceptibilidade = 1;
            if (suceptibilidade < -1)
                suceptibilidade = -1;
            if (rejeicao < 0)
                rejeicao = 0;
            if (rejeicao > 1)
                rejeicao = 1;
            if (evolucao < 0)
                evolucao = 0;
            if (evolucao > 1)
                evolucao = 1;
            if (evoluiMenos < 0)
                evoluiMenos = 0;
            if (evoluiMenos > 1)
                evoluiMenos = 1;
            if (evoluiMais < 0)
                evoluiMais = 0;
            if (evoluiMais > 1)
                evoluiMais = 1;
            memoria = new List<Memoria>(tamanho);
        }
        public void decairMemoria(float decaimento)
        {
            foreach (Memoria mem in memoria)
            {
                mem.porcentagem = mem.porcentagem * decaimento;
                mem.porcentagemRecusa = mem.porcentagemRecusa * decaimento;
            }
```

```
        }
        public void removerMemoria(int id)
        {
            memoria.Remove(memoria.Find(x => x.id == id));
        }
        public void atualizarMemoria(List<Memoria>memory)
        {
            for(int x = 0; x < memoria.Count; x++)
            {
                if(memoria[x].porcentagem < 50)
                {
                    if (memoria[x].porcentagem < memory[x].porcentagem)
                    {
                        if(memory[x].porcentagem < 50)
                        {
                            memoria[x].porcentagem = memory[x].porcentagem;
                            memoria[x].fator = memory[x].fator;
                        }
                        else
                        {
                            memoria[x].porcentagem = 50;
                            memoria[x].fator = memory[x].fator;
                        }
                    }
                }
            }
        }
    }
}


public class Memoria
    {
        public int id;
        public float porcentagem;
        public float fator;
        public float porcentagemRecusa;
        public float fatorRecusa;

        public Memoria(int ID)
        {
            id = ID;
            porcentagem = 0;
            fator = 0;
            porcentagemRecusa = 0;
            fatorRecusa = 0;
        }
    }

public class Population
    {
        List<Agent> agentes;

        public Population(int n)
        {
            agentes = new List<Agent>(n);
        }


        public void inserir(Agent agente)
        {
            if(agentes.Count == agentes.Capacity)
```

```
            {
                Console.WriteLine("Impossivel adicionar agente, população cheia!");
            }
            else
            {
                agentes.Add(agente);
                //otimizar para o ultimo caso
                foreach (Agent agents in agentes)
                {
                    agents.memoria.Add(new Memoria(agente.id));
                }
                if(agentes.Count > 1)
                {
                    agentes[agentes.Count - 1].memoria.Clear();
                    for (int x = 0; x < agentes[agentes.Count - 2].memoria.Count; x++)
                    {
                        agentes[agentes.Count - 1].memoria.Add(new
Memoria(agentes[agentes.Count - 2].memoria[x].id));
                    }
                }
                Console.WriteLine("Agente adicionado com sucesso!");
            }

        }

        public void excluir(int id)
        {
            Agent agente = buscar(id);
            if (agente != null)
            {
                agentes.Remove(agente);

                foreach(Agent agents in agentes)
                {
                    agents.removerMemoria(id);
                }
                Console.WriteLine("Agente removido com sucesso!");
            }
        }

        public Agent buscar(int id)
        {
            Agent agente = agentes.Find(x => x.id == id);
            //if (agente == null)
            //{
            //    Console.WriteLine("Agente não encontrado.");
            //}
            return agente;
        }

        //public void buscar()
        //{
        //    if (agentes.Count == 0)
        //        Console.WriteLine("Nenhum Agente encontrado.");
        //    else
        //    {
        //        foreach (Agent agente in agentes)
        //        {
        //            Console.WriteLine("ID do agente: " + agente.id);
        //            Console.WriteLine("Valor do agente: " + agente.valor);
        //            Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
```

```
        //            Console.WriteLine("Principio do agente: " + agente.principio);
        //        }
        //    }
        //}

        public String[] Buscar(bool ultimato)
        {
            int i = agentes.Count;
            String[] resultado;
            if (i == 0)
            {
                resultado = new String[1];
                resultado[0] = "Nenhum Agente encontrado.";
            }
            else
            {
                resultado = new String[i];
                int j = 0;

                if (ultimato)
                {
                    foreach (Agent agente in agentes)
                    {
                        resultado[j] = "ID: " + agente.id + "; Valor: " + agente.valor
+ "; Principio : " + agente.principio + "; Susceptibilidade: " +
agente.suceptibilidade + "; Rejeição: " + agente.rejeicao;
                        j++;
                    }
                }
                else
                {
                    foreach (Agent agente in agentes)
                    {
                        resultado[j] = "ID: " + agente.id + "; Valor: " + agente.valor
+ "; Principio : " + agente.principio + "; Susceptibilidade: " +
agente.suceptibilidade;
                        j++;
                    }
                }
            }
            return resultado;
        }

        public void Buscar(bool ultimato, bool evolui)
        {
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (evolui)
                {
                    if (ultimato)
                    {
                        foreach (Agent agente in agentes)
                        {
                            Console.WriteLine("ID do agente: " + agente.id);
                            Console.WriteLine("Valor do agente: " + agente.valor);
                            Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            Console.WriteLine("Principio do agente: " +
agente.principio);
```

```
                            Console.WriteLine("Fator de rejeição do agente: " +
agente.rejeicao);
                            Console.WriteLine("Chance de Evolução do agente: " +
agente.evolucao);
                            Console.WriteLine("Fator de Evolução Positiva do agente: "
+ agente.evoluiMais);
                            Console.WriteLine("Fator de Evolução Negativa do agente: "
+ agente.evoluiMenos);
                        }
                    }
                    else
                    {
                        foreach (Agent agente in agentes)
                        {
                            Console.WriteLine("ID do agente: " + agente.id);
                            Console.WriteLine("Valor do agente: " + agente.valor);
                            Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            Console.WriteLine("Principio do agente: " +
agente.principio);
                            Console.WriteLine("Chance de Evolução do agente: " +
agente.evolucao);
                            Console.WriteLine("Fator de Evolução Positiva do agente: "
+ agente.evoluiMais);
                            Console.WriteLine("Fator de Evolução Negativa do agente: "
+ agente.evoluiMenos);
                        }
                    }
                }
                else
                {
                    if (ultimato)
                    {
                        foreach (Agent agente in agentes)
                        {
                            Console.WriteLine("ID do agente: " + agente.id);
                            Console.WriteLine("Valor do agente: " + agente.valor);
                            Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            Console.WriteLine("Principio do agente: " +
agente.principio);
                            Console.WriteLine("Fator de rejeição do agente: " +
agente.rejeicao);
                        }
                    }
                    else
                    {
                        foreach (Agent agente in agentes)
                        {
                            Console.WriteLine("ID do agente: " + agente.id);
                            Console.WriteLine("Valor do agente: " + agente.valor);
                            Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            Console.WriteLine("Principio do agente: " +
agente.principio);
                        }
                    }
                }
                foreach (Agent agente in agentes)
                {
                    Console.WriteLine("ID do agente: " + agente.id);
                    Console.WriteLine("Valor do agente: " + agente.valor);
```

```
                    Console.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                    Console.WriteLine("Principio do agente: " + agente.principio);
                    Console.WriteLine("Fator de rejeição do agente: " +
agente.rejeicao);
                }
            }
        }


        public void Atividade(Agent a1, Agent a2, float custo, float receita, float
fator, float decaimento, string save, bool morre, bool evolui, Random rand)
        {
            float receita1, receita2;
            a1.valor -= custo;
            a2.valor -= custo;

            //Console.WriteLine("Os agentes " + a1.id + " e " + a2.id + " deram " +
custo + " como valor de custo.");

            receita1 = receita * fator;
            receita2 = receita - receita1;

            a1.valor += receita1;
            a2.valor += receita2;

            //Console.WriteLine("O agente " + a1.id + " recebeu " + receita1 + ".
Totalizando " + a1.valor);
            //Console.WriteLine("O agente " + a2.id + " recebeu " + receita2 + ".
Totalizando " + a2.valor);
            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
            //{
            //    file.WriteLine("O agente " + a1.id + " recebeu " + receita1 + ".
Totalizando " + a1.valor);
            //    file.WriteLine("O agente " + a2.id + " recebeu " + receita2 + ".
Totalizando " + a2.valor);
            //}

            //Console.WriteLine("Atividade finalizada. A receita total foi de " +
(receita1 + receita2) + " de um previsto de " + receita);

            //Console.WriteLine("Atualizando as memorias dos agentes");
            a1.atualizarMemoria(a2.memoria);
            a2.atualizarMemoria(a1.memoria);
            //Console.WriteLine("Memorias atualizadas");
            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
            //{
            //    file.WriteLine("Memorias atualizadas");
            //}

            //Console.WriteLine("Adicionando nova memoria do agente " + a2.id);
            int index = a2.memoria.FindIndex(x => x.id == a1.id);
            if (a2.memoria[index].id == a1.id)
            {
                a2.memoria[index].porcentagem = 100;
                a2.memoria[index].fator = fator;
                //Console.WriteLine("Nova memoria adicionada com sucesso");
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
```

```
                //    file.WriteLine("Nova memoria adicionada com sucesso no agente "
+ a2.id);
                //}
            }
            else
            {
                Console.WriteLine("Problema na adição da memória");
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("Problema na adicao da memória");
                //}
                Console.ReadKey(true);
            }

            //Console.WriteLine("Atividade finalizada. Decaindo memoria de todos os
agentes");
            //Console.WriteLine("Atividade finalizada. Decaindo a memoria dos dois
agentes");
            //foreach (Agent agente in agentes) decaimento para todos os agentes
            //{
            //    agente.decairMemoria(decaimento);
            //}

            a1.decairMemoria(decaimento);
            a2.decairMemoria(decaimento);

            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
            //{
            //    file.WriteLine("Decaimento de memória realizada com sucesso");
            //}

            if (morre)
            {
                if (a1.valor <= 0)
                {
                    Console.WriteLine("Agente " + a1.id + " ficou sem valor, este está
sendo removido");
                    excluir(a1.id);
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Agente " + a1.id + " ficou sem valor e foi
removido");
                    //}
                }

                if (a2.valor <= 0)
                {
                    Console.WriteLine("Agente " + a2.id + " ficou sem valor, este está
sendo removido");
                    excluir(a2.id);
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Agente " + a2.id + " ficou sem valor e foi
removido");
                    //}
                }
            }
            if (evolui) //evolução no jogo do ditador
```

```
            {
                if (Convert.ToSingle(rand.NextDouble()) <= a2.evolucao)
                {
                    //Console.WriteLine("Evoluindo o agente 2...");
                    if (receita2 < (a2.principio + a2.suceptibilidade) / 2)
                    {
                        float diferenca = receita1 - receita2;
                        if (diferenca < 0)
                            a2.suceptibilidade = a2.suceptibilidade + (diferenca *
a2.evoluiMenos);
                        else
                            a2.suceptibilidade = a2.suceptibilidade - (diferenca *
a2.evoluiMenos);

                        if (a2.suceptibilidade < 0)
                        {
                            a2.suceptibilidade = 0.5f;
                            a2.principio -= 0.01f;
                        }
                    }
                    else
                    {
                        if (receita2 > (a2.principio + a2.suceptibilidade) / 2)
                        {
                            float diferenca = receita1 - receita2;
                            if (diferenca < 0)
                                a2.suceptibilidade = a2.suceptibilidade - (diferenca *
a2.evoluiMais);
                            else
                                a2.suceptibilidade = a2.suceptibilidade + (diferenca *
a2.evoluiMais);

                            if(a2.suceptibilidade > 1)
                            {
                                a2.suceptibilidade = 0.5f;
                                a2.principio += 0.01f;
                            }
                        }
                    }
                }
            }
        }

        public void Atividade(Agent a1, Agent a2, float custo, float receita, float
proposta, float decaimento, bool aceita, string saveDir, bool morre, bool evolui,
Random rand)
        {
            a1.valor -= custo;
            a2.valor -= custo;

            //Console.WriteLine("Os agentes " + a1.id + " e " + a2.id + " deram " +
custo + " como valor de custo.");

            if (aceita)
            {
                float receita1 = receita * proposta;
                float receita2 = receita - receita1;

                a1.valor += receita1;
                a2.valor += receita2;
```

```
                //Console.WriteLine("O agente " + a1.id + " recebeu " + receita1 + ".
Totalizando " + a1.valor);
                //Console.WriteLine("O agente " + a2.id + " recebeu " + receita2 + ".
Totalizando " + a2.valor);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("O agente " + a1.id + " recebeu " + receita1 + ".
Totalizando " + a1.valor);
                //    file.WriteLine("O agente " + a2.id + " recebeu " + receita2 + ".
Totalizando " + a2.valor);
                //}

                //Console.WriteLine("Atividade finalizada. A receita total foi de " +
(receita1 + receita2) + " de um previsto de " + receita);

                //Console.WriteLine("Adicionando nova memoria do agente " + a2.id);
                int index = a2.memoria.FindIndex(x => x.id == a1.id);
                if (a2.memoria[index].id == a1.id)
                {
                    a2.memoria[index].porcentagem = 100;
                    a2.memoria[index].fator = proposta;
                  //  Console.WriteLine("Nova memoria adicionada com sucesso");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Nova memoria adicionada com sucesso no
agente " + a2.id);
                    //}
                }
                else
                {
                    Console.WriteLine("Problema na adição da memória");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Problema na adicao da memória");
                    //}
                    Console.ReadKey(true);
                }
            }
            else
            {
                //Console.WriteLine("Nao houve atividade devido a recusa da
proposta");

                //Console.WriteLine("Adicionando nova memoria do agente " + a1.id);
                int index = a1.memoria.FindIndex(x => x.id == a2.id);
                if (a1.memoria[index].id == a2.id)
                {
                    a1.memoria[index].porcentagemRecusa = 100;
                    a1.memoria[index].fatorRecusa = proposta;
                  //  Console.WriteLine("Nova memoria adicionada com sucesso");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Nova memoria adicionada com sucesso no
agente " + a1.id);
                    //}
                }
                else
                {
```

```
                    Console.WriteLine("Problema na adição da memória");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Problema na adicao da memória");
                    //}
                    Console.ReadKey(true);
                }
            }

            //Console.WriteLine("Atividade finalizada. Decaindo a memoria dos dois
agentes");

            a1.decairMemoria(decaimento);
            a2.decairMemoria(decaimento);

            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
            //{
            //    file.WriteLine("Decaimento de memória realizada com sucesso");
            //}

            if (morre)
            {
                if (a1.valor <= 0)
                {
                    Console.WriteLine("Agente " + a1.id + " ficou sem valor, este está
sendo removido");
                    excluir(a1.id);
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Agente " + a1.id + " ficou sem valor e foi
removido");
                    //}
                }

                if (a2.valor <= 0)
                {
                    Console.WriteLine("Agente " + a2.id + " ficou sem valor, este está
sendo removido");
                    excluir(a2.id);
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("Agente " + a2.id + " ficou sem valor e foi
removido");
                    //}
                }
            }
            if (evolui)//evolução no jogo do Ultimato
            {
                if (!aceita)
                {
              //      Console.WriteLine("Evoluindo o agente 1...");
                //    Console.WriteLine("Evoluindo o agente 2...");
                }
            }
        }
```

```
        public void AtividadeDitador(Agent a1, Agent a2, float custo, float receita,
float decaimento, string saveDir, string agentDir, bool morre, bool evolui)
        {
            Memoria mem = a1.memoria.Find(x => x.id == a2.id);
            float fator = 0;
            Random aleatorio = new Random();
            bool lembrou = false, suscetivel = false;


            if (mem.porcentagem == 0) //se não possui memoria, faz utilizando o
principio
            {
                lembrou = false;
                suscetivel = false;

                //Console.WriteLine("Agente " + a1.id + " não possui memoria de acao
do agente " + a2.id);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("Agente " + a1.id + " não possui memoria de acao
do agente " + a2.id);
                //}

                //Console.WriteLine("Agente " + a1.id + " tomará a ação com o agente "
+ a2.id + " e decidira quanto cada um ganhará");
                fator = a1.principio;
                //Console.WriteLine("O agente " + a1.id + " utilizou o fator " +
fator);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("O agente " + a1.id + " utilizou o fator " +
fator);
                //}

                Atividade(a1, a2, custo, receita, fator, decaimento,saveDir, morre,
evolui, aleatorio);

                UpdateAgentCSV(agentDir,a1, a2, lembrou, suscetivel, fator);

            }else // se possui memoria, verifica se lembra da ação memorizada
            {
                //Console.WriteLine("Agente " + a1.id + " possui memoria de acao do
agente " + a2.id + " com porcentagem " + mem.porcentagem);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("Agente " + a1.id + " possui memoria de acao do
agente " + a2.id + " com porcentagem " + mem.porcentagem);
                //}

                //Random aleatorio = new Random();

                if(Convert.ToSingle(aleatorio.NextDouble())*100 <=
mem.porcentagem)//se lembrou da ação, probabiliza se utiliza a ação anterior
                {
                    lembrou = true;
                  //  Console.WriteLine("O agente " + a1.id + " lembrou da ultima ação
realizada pelo agente " + a2.id);
                    //Console.WriteLine("Verificando se irá executar a mesma ação ou
não");
```

```
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("O agente " + a1.id + " lembrou da ultima
ação realizada pelo agente " + a2.id);
                    //}

                    if(a1.suceptibilidade < 0)// se caso inverso
                    {
                        if (Convert.ToSingle(aleatorio.NextDouble()) <=
a1.suceptibilidade * -1)//se sucetivel, utiliza a inversa
                        {
                      //      Console.WriteLine("O agente " + a1.id + "decidiu
realizar a ação inversa do agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + "decidiu
realizar a ação inversa do agente " + a2.id);
                            //}

                            //fator = 1 - a2.principio;
                            fator = 1 - mem.fator;
                        //    Console.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //}

                            Atividade(a1, a2, custo, receita, fator, decaimento,
saveDir, morre, evolui, aleatorio);
                        }
                        else //se não sucetivel, utiliza o principio
                        {
                          //  Console.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para realizar a ação com o agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para realizar a ação com o agente " + a2.id);
                            //}

                            fator = a1.principio;
                            //Console.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //}

                            Atividade(a1, a2, custo, receita, fator, decaimento,
saveDir, morre, evolui, aleatorio);
                        }
                    }
                    else // caso normal
                    {
```

```
                        if(Convert.ToSingle(aleatorio.NextDouble()) >=
a1.suceptibilidade)//se não sucetivel, utiliza o principio
                        {
                            suscetivel = false;
                            //Console.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para realizar a ação com o agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para realizar a ação com o agente " + a2.id);
                            //}

                            fator = a1.principio;
                            //Console.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //}

                            Atividade(a1, a2, custo, receita, fator, decaimento,
saveDir, morre, evolui, aleatorio);
                            UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel,
fator);
                        }
                        else //se olho por olho
                        {
                            suscetivel = true;

                            //Console.WriteLine("O agente " + a1.id + " decidiu
realizar a mesma ação realizada pelo agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " decidiu
realizar a mesma ação realizada pelo agente " + a2.id);
                            //}

                            fator = mem.fator;
                            //Console.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //}

                            Atividade(a1, a2, custo, receita, fator,
decaimento,saveDir, morre, evolui, aleatorio);
                            UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel,
fator);
                        }
                    }
                }
                else // se não lembrou da ação
                {
                    lembrou = false;
                    suscetivel = false;
```

```
                    //Console.WriteLine("O agente " + a1.id + " não lembrou da ação.
Utilizando o principio");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("O agente " + a1.id + " não lembrou da ação.
Utilizando o principio");
                    //}

                    fator = a1.principio;
                    //Console.WriteLine("O agente " + a1.id + " utilizou o fator " +
fator);
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("O agente " + a1.id + " utilizou o fator " +
fator);
                    //}
                    Atividade(a1, a2, custo, receita, fator, decaimento,saveDir,
morre, evolui, aleatorio);
                    UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel, fator);
                }
            }

            //double fator = (a1.principio + (a2.principio * a1.suceptibilidade)) / 2;

            //a1.valor -= custo;
            //a2.valor -= custo;

            //double receita1 = receita / 2;
            //double receita2 = receita1;

            //receita1 = receita1 + (receita1 * fator);
            //receita2 = receita - receita1;
            //a1.valor = a1.valor + receita1;
            //a2.valor = a2.valor + receita2;

            //Console.WriteLine("Atividade Realizada com sucesso!");
            //Console.WriteLine("Receita do agente " + a1.id + "= " + a1.valor);
            //Console.WriteLine("Receita do agente " + a2.id + "= " + a2.valor);
        }

        private void UpdateAgentCSV(string agentPath, Agent a1, Agent a2, bool
lembrou, bool suscetivel, float fator)
        {
            string delimitador = ";";
            string agent1Archive = agentPath + "\\agente" + a1.id + ".csv";
            string agent2Archive = agentPath + "\\agente" + a2.id + ".csv";

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agent1Archive, true))
            {
                file.WriteLine(a1.id + delimitador + a2.id + delimitador +
lembrou.ToString() + delimitador + suscetivel.ToString() + delimitador + fator +
delimitador + a1.valor);
            }

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agent2Archive, true))
            {
```

```
                file.WriteLine(a1.id + delimitador + a2.id + delimitador +
lembrou.ToString() + delimitador + suscetivel.ToString() + delimitador + fator +
delimitador + a2.valor);
            }
        }

        private void UpdateAgentCSV(string agentPath, Agent a1, Agent a2, bool
lembrou, bool suscetivel, float fator, bool aceita)
        {
            string delimitador = ";";
            string agent1Archive = agentPath + "\\agente" + a1.id + ".csv";
            string agent2Archive = agentPath + "\\agente" + a2.id + ".csv";

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agent1Archive, true))
            {
                file.WriteLine(a1.id + delimitador + a2.id + delimitador +
lembrou.ToString() + delimitador + suscetivel.ToString() + delimitador + fator +
delimitador + aceita.ToString() + delimitador + a1.valor);
            }

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agent2Archive, true))
            {
                file.WriteLine(a1.id + delimitador + a2.id + delimitador +
lembrou.ToString() + delimitador + suscetivel.ToString() + delimitador + fator +
delimitador + aceita.ToString() + delimitador + a2.valor);
            }
        }

        public void AtividadeUltimato(Agent a1, Agent a2, float custo, float receita,
float decaimento, string saveDir, string agentDir, bool morre, bool evolui)
        {
            Memoria mem = a1.memoria.Find(x => x.id == a2.id);
            float fator = 0;
            Random aleatorio = new Random();
            bool lembrou = false, suscetivel = false;

            if (mem.porcentagem == 0) //se não possui memoria da proposta, faz
utilizando o principio
            {
                lembrou = false;
                suscetivel = false;

                //Console.WriteLine("Agente " + a1.id + " não possui memoria de acao
do agente " + a2.id);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("Agente " + a1.id + " não possui memoria de acao
do agente " + a2.id);
                //}

//                Console.WriteLine("Agente " + a1.id + " fara uma proposta ao agente
" + a2.id + " usando o seu principio");
                fator = a1.principio;
  //              Console.WriteLine("O agente " + a1.id + " propos ficar com " + fator
+ "do valor total");
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
```

```
                //    file.WriteLine("O agente " + a1.id + " utilizou o fator " +
fator);
                //}

                bool aceita = Proposta(a2, a1.id, fator, saveDir);

                Atividade(a1, a2, custo, receita, fator, decaimento, aceita, saveDir,
morre, evolui, aleatorio);
                UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel, fator, aceita);
            }
            else // se possui memoria, verifica se lembra da ação memorizada
            {
    //            Console.WriteLine("Agente " + a1.id + " possui memoria de acao do
agente " + a2.id + " com porcentagem " + mem.porcentagem);
                //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                //{
                //    file.WriteLine("Agente " + a1.id + " possui memoria de acao do
agente " + a2.id + " com porcentagem " + mem.porcentagem);
                //}

                if (Convert.ToSingle(aleatorio.NextDouble()) * 100 <=
mem.porcentagem)//se lembrou da ação, probabiliza se utiliza a ação anterior
                {
                    lembrou = true;
      //              Console.WriteLine("O agente " + a1.id + " lembrou da ultima ação
realizada pelo agente " + a2.id);
        //            Console.WriteLine("Verificando se irá executar a mesma ação ou
não");
                    //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                    //{
                    //    file.WriteLine("O agente " + a1.id + " lembrou da ultima
ação realizada pelo agente " + a2.id);
                    //}

                    if (a1.suceptibilidade < 0)// se caso inverso
                    {
                        if (Convert.ToSingle(aleatorio.NextDouble()) <=
a1.suceptibilidade * -1)//se sucetivel, faz o inverso
                        {
          //                  Console.WriteLine("O agente " + a1.id + "decidiu
realizar a ação inversa do agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + "decidiu
realizar a ação inversa do agente " + a2.id);
                            //}

            //                Console.WriteLine("Agente " + a1.id + " fara uma
proposta ao agente " + a2.id);
                            fator = 1 - mem.fator;
              //              Console.WriteLine("O agente " + a1.id + " propos ficar
com " + fator + "do valor total");
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " utilizou o
fator " + fator);
                            //}
```

```
                            bool aceita = Proposta(a2, a1.id, fator, saveDir);

                            Atividade(a1, a2, custo, receita, fator, decaimento,
aceita, saveDir, morre, evolui, aleatorio);

                        }
                        else //se não sucetivel, utiliza o principio
                        {
                //            Console.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para propor um valor para o agente " + a2.id);
                            //using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@saveDir, true))
                            //{
                            //    file.WriteLine("O agente " + a1.id + " decidiu
utilizar seu principio para propor um valor para o agente " + a2.id);
                            //}

                            fator = a1.principio;

                            bool aceita = Proposta(a2, a1.id, fator, saveDir);

                            Atividade(a1, a2, custo, receita, fator, decaimento,
aceita, saveDir, morre, evolui, aleatorio);
                        }
                    }
                    else // caso normal
                    {
                        if (Convert.ToSingle(aleatorio.NextDouble()) >=
a1.suceptibilidade)//se não sucetivel, utiliza o principio
                        {
                            suscetivel = false;

                            fator = a1.principio;

                            bool aceita = Proposta(a2, a1.id, fator, saveDir);

                            Atividade(a1, a2, custo, receita, fator, decaimento,
aceita, saveDir, morre, evolui, aleatorio);
                            UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel,
fator, aceita);
                        }
                        else //se olho por olho
                        {
                            suscetivel = true;

                            fator = mem.fator;

                            bool aceita = Proposta(a2, a1.id, fator, saveDir);

                            Atividade(a1, a2, custo, receita, fator, decaimento,
aceita, saveDir, morre, evolui, aleatorio);
                            UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel,
fator, aceita);
                        }
                    }
                }
                else // se não lembrou da ação
                {
                    lembrou = false;
                    suscetivel = false;

                    fator = a1.principio;
```

```
                    bool aceita = Proposta(a2, a1.id, fator, saveDir);

                    Atividade(a1, a2, custo, receita, fator, decaimento, aceita,
saveDir, morre, evolui, aleatorio);
                    UpdateAgentCSV(agentDir, a1, a2, lembrou, suscetivel, fator,
aceita);
                }
            }
        }

        private bool Proposta(Agent a2, int a1Id, float proposta, string saveDir)
        {
            if (proposta < a2.rejeicao)
            {
                Memoria mem = a2.memoria.Find(x => x.id == a1Id);

                if (mem.porcentagemRecusa == 0) //se não possui memoria da recusa do
agente 1, então recusa
                {
                    return false;
                }
                else // se possui memoria, verifica se lembra da ação memorizada
                {
                    Random aleatorio = new Random();

                    if (Convert.ToSingle(aleatorio.NextDouble()) * 100 <=
mem.porcentagemRecusa)//se lembrou da ação, probabiliza se aceita
                    {
                        if (a2.suceptibilidade < 0)// se caso inverso
                        {
                            if (Convert.ToSingle(aleatorio.NextDouble()) <=
a2.suceptibilidade * -1)//se sucetivel, faz o inverso
                            {
                                if(proposta<mem.fatorRecusa)
                                {
                                    return true;
                                }
                                else
                                {
                                    return false;
                                }
                            }
                            else //se não sucetivel, recusa
                            {
                                return false;
                            }
                        }
                        else // caso normal
                        {
                            if (Convert.ToSingle(aleatorio.NextDouble()) >
a2.suceptibilidade)//se não sucetivel, recusa
                            {
                                return false;
                            }
                            else //se sucetivel, verifica
                            {
                                if(proposta<mem.fatorRecusa)
                                {
                                    return false;
                                }
                                else
```

```
                                {
                                    return true;
                                }
                            }
                        }
                    }
                    else // se não lembrou da ação
                    {
                        return false;
                    }
                }
            }
            else
            {
                return true;
            }
        }

        public void RealizarAtividadeDitador(float custo, float receita, bool all,
Random random, float decaimento, string logArc, string agentDir, bool morre, bool
evolui)
        {
            Agent agente1, agente2;

            if (all)
            {}
            else
            {
                if (agentes.Count > 1)//otimizado para Count = 2
                {
                    do
                    {
                        int busca1 = random.Next(agentes[0].id, (agentes[agentes.Count
- 1].id) + 1);
                        agente1 = buscar(busca1);
                    } while (agente1 == null);
                    do
                       {
                        int busca2 = random.Next(agentes[0].id, (agentes[agentes.Count
- 1].id) + 1);
                        agente2 = buscar(busca2);
                       } while (agente1.Equals(agente2)|| agente2 == null);
                    AtividadeDitador(agente1, agente2, custo, receita, decaimento,
logArc, agentDir, morre, evolui);
                }
                else
                {
                    //Console.WriteLine("População abaixo de 2 agentes.");
                }
            }
        }
        public void RealizarAtividadeUltimato(float custo, float receita, bool all,
Random random, float decaimento, string saveDir, string agentDir, bool morre, bool
evolui)
        {
            Agent agente1, agente2;
            if (all)
            {
            }
            else
            {
                if (agentes.Count > 1)//otimizado para Count = 2
```

```
                {
                    do
                    {
                        int busca1 = random.Next(agentes[0].id, (agentes[agentes.Count
- 1].id) + 1);
                        agente1 = buscar(busca1);
                    } while (agente1 == null);
                    do
                    {
                        int busca2 = random.Next(agentes[0].id, (agentes[agentes.Count
- 1].id) + 1);
                        agente2 = buscar(busca2);
                    } while (agente1.Equals(agente2) || agente2 == null);
                    AtividadeUltimato(agente1, agente2, custo, receita, decaimento,
saveDir, agentDir, morre, evolui);
                }
                else
                {
                    Console.WriteLine("População abaixo de 2 agentes.");
                }
            }
        }
        public void copyAll(Population destino, bool ultimato)
        {
            destino.agentes.Clear();
            if (ultimato)
            {
                foreach (Agent agente in agentes)
                {
                    Agent novoAgente = new Agent(agente.id, agente.valor,
agente.principio, agente.suceptibilidade, agente.rejeicao, agentes.Capacity);
                    foreach (Memoria memory in agente.memoria)
                    {
                        Memoria memoria = new Memoria(memory.id);
                        memoria.fator = memory.fator;
                        memoria.porcentagem = memory.porcentagem;
                        novoAgente.memoria.Add(memoria);
                    }
                    destino.agentes.Add(novoAgente);
                }
            }
            else
            {
                foreach (Agent agente in agentes)
                {
                    Agent novoAgente = new Agent(agente.id, agente.valor,
agente.principio, agente.suceptibilidade, agentes.Capacity);
                    foreach (Memoria memory in agente.memoria)
                    {
                        Memoria memoria = new Memoria(memory.id);
                        memoria.fator = memory.fator;
                        memoria.porcentagem = memory.porcentagem;
                        novoAgente.memoria.Add(memoria);
                    }
                    destino.agentes.Add(novoAgente);
                }
            }
        }
        public bool PossuiAgente()
        {
            if (agentes.Count == 0)
                return false;
```

```
            else
            return true;
        }
        public void SaveToArchive(string archiveName, bool evolui)
        {
            string arcPath =
Path.GetDirectoryName(System.AppDomain.CurrentDomain.BaseDirectory.ToString()) + "\\"
+ archiveName + ".txt";
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (evolui)
                {
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@arcPath, true))
                        {
                            file.WriteLine("ID do agente: " + agente.id);
                            file.WriteLine("Valor do agente: " + agente.valor);
                            file.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            file.WriteLine("Principio do agente: " +
agente.principio);
                            file.WriteLine("Fator de Evolução do agente: " +
agente.evolucao);
                            file.WriteLine("");
                        }
                    }
                }
                else
                {
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@arcPath, true))
                        {
                            file.WriteLine("ID do agente: " + agente.id);
                            file.WriteLine("Valor do agente: " + agente.valor);
                            file.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            file.WriteLine("Principio do agente: " +
agente.principio);
                            file.WriteLine("");
                        }
                    }
                }
            }
        }
        public void SaveToCSV(string archivePath, bool evolui)
        {
            string delimitador = ";";
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (evolui)
                {
                    using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                    {
```

```
                        file.WriteLine("Identificador" + delimitador + "Valor" +
delimitador + "Suceptibilidade" + delimitador + "Principio" + delimitador +
"Evolucao");
                    }
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                        {
                            file.WriteLine(agente.id + delimitador + agente.valor +
delimitador + agente.suceptibilidade.ToString() + delimitador + agente.principio +
delimitador + agente.evolucao);
                        }
                    }
                }
                else
                {
                    using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                    {
                        file.WriteLine("Identificador" + delimitador + "Principio" +
delimitador + "Suceptibilidade" + delimitador + "Valor Inicial" + delimitador + "Valor
Final");
                    }
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                        {
                            file.WriteLine(agente.id + delimitador + agente.principio
+ delimitador + agente.suceptibilidade + delimitador + agente.valorInicial +
delimitador + agente.valor);
                        }
                    }
                }
            }
        }
        public void USaveToArchive(string archiveName, bool evolui)
        {
            string arcPath =
Path.GetDirectoryName(System.AppDomain.CurrentDomain.BaseDirectory.ToString()) + "\\"
+ archiveName + ".txt";
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (evolui)
                {
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@arcPath, true))
                        {
                            file.WriteLine("ID do agente: " + agente.id);
                            file.WriteLine("Valor do agente: " + agente.valor);
                            file.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            file.WriteLine("Principio do agente: " +
agente.principio);
                            file.WriteLine("Fator de Rejeição: " + agente.rejeicao);
                            file.WriteLine("Fator de Evolução do agente: " +
agente.evolucao);
```

```
                            file.WriteLine("");
                        }
                    }
                }
                else
                {
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@arcPath, true))
                        {
                            file.WriteLine("ID do agente: " + agente.id);
                            file.WriteLine("Valor do agente: " + agente.valor);
                            file.WriteLine("Suceptibilidade do agente: " +
agente.suceptibilidade.ToString());
                            file.WriteLine("Fator de Rejeição: " + agente.rejeicao);
                            file.WriteLine("Principio do agente: " +
agente.principio);
                            file.WriteLine("");
                        }
                    }
                }
            }
        }
        public void USaveToCSV(string archivePath, bool evolui)
        {
            string delimitador = ";";
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (evolui)
                {
                    using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                    {
                        file.WriteLine("Identificador" + delimitador + "Principio" +
delimitador + "Susceptibilidade" + delimitador + "Rejeicao" + delimitador + "Evolucao"
+ delimitador + "Valor Inicial" + delimitador + "Valor Final");
                    }
                    foreach (Agent agente in agentes)
                    {
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                        {
                            file.WriteLine(agente.id + delimitador + agente.principio
+ delimitador + agente.suceptibilidade + delimitador + agente.rejeicao + delimitador +
agente.evolucao + delimitador + agente.valorInicial + delimitador + agente.valor);
                        }
                    }
                }
                else
                {
                    using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                    {
                        file.WriteLine("Identificador" + delimitador + "Principio" +
delimitador + "Susceptibilidade" + delimitador + "Rejeicao" + delimitador + "Valor
Inicial" + delimitador + "Valor Final");
                    }
                    foreach (Agent agente in agentes)
                    {
```

```
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@archivePath, true))
                        {
                            file.WriteLine(agente.id + delimitador + agente.principio
+ delimitador + agente.suceptibilidade + delimitador + agente.rejeicao + delimitador +
agente.valorInicial + delimitador + agente.valor);
                        }
                    }
                }
            }
        }
        internal void SaveValue()
        {
            foreach (Agent agente in agentes)
            {
                agente.valorInicial = agente.valor;
            }
        }
        internal void CreateAgentsArchive(string agentPath, bool ultimato, bool apend)
        {
            string delimitador = ";";
            if (agentes.Count == 0)
                Console.WriteLine("Nenhum Agente encontrado.");
            else
            {
                if (ultimato)
                {
                    foreach (Agent agente in agentes)
                    {
                        string agentArchive = agentPath + "\\agente" + agente.id +
".csv";
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agentArchive, apend))
                        {
                            file.WriteLine("Agente 1" + delimitador + "Agente 2" +
delimitador + "Lembrou" + delimitador + "Suscetivel" + delimitador + "Fator" +
delimitador + "Aceitou" + delimitador + "Valor Final");
                        }
                    }
                }
                else
                {
                    foreach (Agent agente in agentes)
                    {
                        string agentArchive = agentPath + "\\agente" + agente.id +
".csv";
                        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@agentArchive, apend))
                        {
                            file.WriteLine("Agente 1" + delimitador + "Agente 2" +
delimitador + "Lembrou" + delimitador + "Suscetivel" + delimitador + "Fator" +
delimitador + "Valor Final");
                        }
                    }
                }
            }
        }
    }
```