UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Giancarlo Souza de Freitas

UM MODELO DE RACIOCÍNIO SOBRE SENSORES PARA AGENTES SIGON EM AMBIENTES DE REALIDADE VIRTUAL

Florianópolis

2018

Giancarlo Souza de Freitas

UM MODELO DE RACIOCÍNIO SOBRE SENSORES PARA AGENTES SIGON EM AMBIENTES DE REALIDADE VIRTUAL

Trabalho de Conclusão de Curso submetido ao Curso de Sistemas de Informação através do Sistema de Gestão de TCCs INE, para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Thiago Ângelo Gelaim, MSc
Coorientador: Prof. Ricardo Azambuja Silveira, Dr.

Florianópolis

2018

Giancarlo Souza de Freitas

UM MODELO DE RACIOCÍNIO SOBRE SENSORES PARA AGENTES SIGON EM AMBIENTES DE REALIDADE VIRTUAL

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Sistemas de Informação", e aprovado em sua forma final pelo Curso de Sistemas de Informação.

Florianópolis, 30 de outubro 2018.

Prof. Cristian Koliver, Dr.
Coordenador do Curso

Thiago Ângelo Gelaim, MSc
Orientador

Prof. Ricardo Azambuja Silveira, Dr.
Coorientador

Banca Examinadora:

Prof. Dr. Elder Rizzon Santos

Prof. Dr. Cecilia Estela Giuffra Palomino

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Freitas, Giancarlo Souza de
Um modelo de raciocínio sobre sensores para agentes Sigon em ambientes de realidade virtual / Giancarlo Souza de Freitas ; orientador, Thiago Angelo Gelaim, coorientador, Ricardo Azambuja Silveira, 2018.
57 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistema de Informação, Florianópolis, 2018.

Inclui referências.

1. Sistema de Informação. 2. Agentes. 3. Percepção ativa. 4. Raciocínio sobre sensores. I. Gelaim, Thiago Angelo. II. Silveira, Ricardo Azambuja. III. Universidade Federal de Santa Catarina. Graduação em Sistema de Informação. IV. Título.

RESUMO

A necessidade de automatização de tarefas e simulação do comportamento de seres humanos em ambientes dinâmicos está crescendo proporcionalmente ao avanço tecnológico. Os desafios tecnológicos que começam a surgir estão indo além da capacidade humana, devido à grande complexidade e urgência de suas resoluções. Em razão disso, agentes inteligentes capazes de simular o comportamento humano são uma alternativa. A simulação do comportamento humano é uma tarefa complexa, e para fazer isso, é necessário que o agente inteligente perceba o ambiente a sua volta, defina em que deve prestar atenção, a razão disso e a relevância das informações que ele está percebendo. Para auxiliar o agente nesta tarefa, existe o conceito de percepção ativa, o qual permite que o agente foque sua percepção nos elementos mais relevantes em um ambiente, dependendo de seus objetivos.

Este trabalho tem como objetivo explorar os conceitos de percepção ativa na implementação de agentes em um ambiente de realidade virtual, mais especificamente, focando no raciocínio sobre os sensores do agente. O raciocínio dos sensores permite que o agente priorize os sensores que percebem mais mudanças ocorrendo no ambiente.

O modelo proposto possibilita a criação e utilização de sensores em agentes. Os sensores coletam percepções do ambiente, e podem, de acordo com sua estratégia, ter suas prioridades alteradas. A validação do modelo é fornecida através da implementação de duas estratégias de priorização diferentes, analisadas em um ambiente com objetos dinâmicos, para melhor visualização.

Palavras-chave: Agentes. Percepção Ativa. Realidade Virtual. Priorização de Sensores.

ABSTRACT

The need for automation of tasks and simulation of human beings behavior in dynamic environments are proportionally growing with the technological advance. Technological challenges that are beginning to emerge are going beyond human capacity, due to the great complexity and urgency of their resolutions. Because of this, intelligent agents capable of simulating human behavior are an alternative. Simulation of human behavior is a complex task, and to do this, it's necessary that the agent perceives the environment around it, define what it needs to pay attention to, the reason for that and the relevance of the information that it is perceiving. To assist the agent on this task, there is the concept of active perception, which allows the agent to focus its perception on the most relevant elements in an environment, depending on its objectives.

This work has the purpose to explore the concepts of active perception on the implementation of agents in a virtual reality environment, more specifically, focusing on the agent's sensors reasoning. Sensors reasoning allows the agent to prioritize sensors that perceive more changes in the environment.

The proposed model enables the sensors creation and utilization on agents. The sensors collect perceptions from the environment, and they can, according to their strategies, have their priorities changed. Model's evaluation is provided through the implementation of two different prioritization strategies, both analyzed in an environment with dynamic objects, for better visualization.

Keywords: Agents. Active Perception. Virtual Reality. Sensors Prioritization.

LISTA DE FIGURAS

Figura 1 Estrutura BDI 22
Figura 2 Modelo de Agente Multi-Contexto 34
Figura 3 Representação Adaptada do Modelo de Sensores 37
Figura 4 Sensores em um Agente 37
Figura 5 Exemplo de Agente neste Modelo 38
Figura 6 Representação da Visão do Agente 40
Figura 7 Diagrama de Atividades de um Sensor 41
Figura 8 Cenário de Testes 44
Figura 9 Prioridades dos Sensores de Visão 45
Figura 10 Novas Percepções dos Sensores de Visão 46
Figura 11 Testes de Percepções Enviadas 47
Figura 12 Tempo de Envio de Percepções 48
Figura 13 Tempo de Publicação no Agente 48
Figura 14 Exemplo de Publicação na Mudança de Informação 50
Figura 15 Prioridades com foco no Sensor de Visão Perto 51
Figura 16 Número de Percepções com foco no Sensor de Visão Perto 51
Figura 17 Prioridades com Sensores de Massa 80 52
Figura 18 Número de Percepções com Sensores de Massa 80 52

SUMÁRIO

1 INTRODUÇÃO 13
1.1 PROBLEMATIZAÇÃO 14
1.2 MOTIVAÇÃO 14
1.3 JUSTIFICATIVA 15
1.4 OBJETIVOS 15
1.4.1 Objetivo Geral 15
1.4.2 Objetivos Específicos 16
1.5 MÉTODO DE PESQUISA 16
2 FUNDAMENTAÇÃO TEÓRICA 19
2.1 AGENTES 19
2.2 AGENTES BDI 21
2.3 PERCEPÇÃO ATIVA 22
2.3.1 Elementos de Percepção Ativa 23
2.4 AMBIENTES 24
2.5 REALIDADE VIRTUAL 26
2.6 CONCLUSÃO 27
3 TRABALHOS CORRELATOS 29
3.1 SUPORTE DE ORIENTAÇÃO DE EVACUAÇÃO USANDO DISPOSITIVOS IOT COOPERATIVOS BASEADOS EM AGENTES 29
3.2 MELHORIA DO COMPORTAMENTO DE CRUZAMENTO DE ESTRADAS COM ABORDAGEM DE PERCEPÇÃO ATIVA 30
3.3 MODELO DE AGENTES E-BDI INTEGRANDO CONFIANÇA BASEADO EM SISTEMAS MULTI-CONTEXTO .. 31
3.4 CONCLUSÃO 33
4 MODELO PROPOSTO 35
4.1 CONTEXTO DE COMUNICAÇÃO 36
4.2 IMPLEMENTAÇÃO DE ESTRATÉGIAS 40
5 TESTES 43
6 CONSIDERAÇÕES FINAIS 53
6.1 TRABALHOS FUTUROS 53
REFERÊNCIAS 55
APÊNDICE A – Artigo 61
APÊNDICE B – Código Fonte 73

1 INTRODUÇÃO

A evolução tecnológica tem, cada vez mais, fornecido à humanidade novas formas de visualizar e interagir com o mundo, seja de forma física ou virtual. Com isso, novas necessidades e desafios continuam surgindo, como citam Rao e Georgeff (2000), o design de sistemas que são requeridos para desempenhar gerenciamento de alto nível e controle de tarefas em ambientes dinâmicos complexos está com uma importância comercial cada vez maior. Os novos desafios e sistemas a serem construídos começam a ser de tamanha complexidade e também urgência de ações rápidas, que a resolução humana não mais consegue resolver em um intervalo de tempo adequado ao problema. Por exemplo, realizar uma boa evacuação em local de desastre. Por outro lado, surgem também ideias de aprimorar certas tarefas já realizadas pela humanidade, como por exemplo, o tratamento de crianças com autismo. No trabalho de Liu et al. (2014), é mostrado que crianças com autismo mostram certo comportamento social positivo enquanto interagem com agentes robóticos, sem a pressão de outras pessoas.

Os seres humanos podem passar por situações críticas onde a sua decisão necessita ser rápida e correta, e nem sempre possuem um bom tempo de resposta para o caso. Por exemplo, em situações de desastres naturais, onde a evacuação de algum local se torna extremamente urgente, e como citam Katayama et al. (2017), soluções para diminuir os danos deste tipo de situação são necessárias, utilizando por exemplo, algum tipo de tecnologia mais autônoma junto a IoT para resultados mais rápidos. Para auxiliar em prover novas formas de solucionar estas novas situações e problemas, existe a abordagem de sistemas multiagentes, que pode ser aplicada nas mais diversas situações, das mais simples como mover um objeto, até as mais complexas como melhorar a evacuação de um local em desastre.

A definição de agente ainda é um tema de grande discussão no cenário científico. Neste trabalho, será considerada a definição adaptada de Wooldridge (2002), a qual define agente como um sistema de computador que está situado em um ambiente, e que é capaz de ações autônomas neste ambiente visando alcançar seus objetivos designados. Um agente inteligente é uma unidade de software capaz de ações autônomas flexíveis visando atingir seus objetivos designados, onde flexibilidade significa três coisas: reatividade, proatividade e habilidade social (WOOLDRIDGE, 2002).

1.1 PROBLEMATIZAÇÃO

Os agentes inteligentes são usados nas mais diversas áreas, como robótica e realidade virtual, e como citam Laukkanen et al. (2004), esse novo campo para adicionar inteligência em ambientes virtuais está emergindo, para auxiliar os mesmos a terem um ambiente que se comporta de forma crível. Nessas aplicações, é importante que o agente inteligente consiga perceber o ambiente a sua volta para tomar alguma ação. Um desafio consiste em utilizar essas aplicações em ambientes simulados do mundo real, onde existe uma grande dinâmica ocorrendo a todo momento. Ao se encontrar neste cenário, um agente inteligente deve estar ciente do que acontece nesse meio e ter a capacidade de priorizar o que deve ter sua atenção em determinado momento. Para este tipo de desafio, o conceito de percepção ativa pode ser aplicado. Bajcsy, Aloimonos e Tsotsos (2017), em sua pesquisa sobre percepção ativa, definem: um agente é um perceptor ativo se ele sabe porque deseja perceber, e então escolhe o que perceber, e determina como, quando e onde deve atingir essa percepção.

A essência da percepção ativa é determinar um objetivo baseado em uma crença atual sobre o mundo e colocar em movimento ações para atingir este objetivo. Em outras palavras, percepção ativa é intencional (ALOIMONOS, 1990) pois tem que combinar percepção e ação, para determinar os elementos que irão auxiliar na realização do objetivo. Os agentes inteligentes ainda não apresentam um comportamento maduro em suas percepções. Bajcsy, Aloimonos e Tsotsos (2017) completam que, apesar do recente sucesso em robótica, Inteligência Artificial e visão computacional, um agente artificial completo necessariamente deve incluir percepção ativa.

Com base nisto, esta pesquisa procura oferecer maior suporte para a percepção ativa ao responder a pergunta "é possível tornar o processo de tomada de decisão menos custoso através do controle de sensores?". Utilizando percepção ativa, um agente pode focar apenas nas informações úteis para seu objetivo designado.

1.2 MOTIVAÇÃO

Existem inúmeros pesquisadores na missão de encontrar métodos para escolher o que processar em uma imagem. Porém Bajcsy, Aloimonos e Tsotsos (2017) chamam atenção ao fato de que a missão de encontrar métodos que determinam que imagens devem ser consi-

deradas, ou seja, qual campo de visão deve ser percebido, não está recebendo muita atenção. Essa missão já se mostrou relevante com o trabalho de Rasouli e Tsotsos (2014), onde é apresentado um aperfeiçoamento de performance com a integração de métodos visuais ativos de ponto de vista de técnicas de atenção visual. Em resposta a isso, esta pesquisa está voltada à percepção ativa do agente, a qual aborda a questão da atenção e também priorização de percepção para interação em um ambiente dinâmico. Com avanços nessa área, os agentes podem se aproximar mais do comportamento humano, tornando estudos com simulações em ambientes virtuais mais fiéis em relação ao mundo real. Paralelamente uma contribuição para a indústria de jogos em realidade virtual também é possível, criando uma experiência virtual mais agradável ou desafiadora.

1.3 JUSTIFICATIVA

A definição de o que um agente deve perceber e como ele interage com o que foi percebido, é uma área importante e complexa de estudo, pois como citam Bajcsy, Aloimonos e Tsotsos (2017), se um agente está envolvido em ações que necessitam de tomada de decisões, ele precisa identificar, reconhecer e manipular diferentes objetos, ele precisa procurar por categorias particulares no cenário, ele precisa reconhecer os efeitos e consequências de diferentes ações. A questão da "percepção de detalhes" unida com a análise dessas percepções, a tomada de decisão e interação com um cenário dinâmico aumenta a complexidade dos agentes.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

O objetivo geral deste trabalho de conclusão de curso é o desenvolvimento de um modelo de raciocínio sobre sensores de um agente, para suporte à utilização de percepção ativa em um ambiente de realidade virtual. Com a utilização deste ambiente, é possível ter uma simulação mais controlada, facilitando a verificação de resultados.

1.4.2 Objetivos Específicos

- Desenvolver sensores para captação de percepções no ambiente virtual;
- Desenvolvimento do modelo, como estudo de caso, através da integração do framework Sigon com o ambiente Unity de realidade virtual;
- Implementar estratégias diferentes de priorização de percepções por meio de sensores para validar o modelo;
- Criação de cenário de testes;
- Validação do comportamento do sistema desenvolvido.

1.5 MÉTODO DE PESQUISA

Este trabalho de conclusão de curso segue, como modelo de pesquisa, o modelo proposto por Takeda et al. (1990). O processo de desenvolvimento é dividido em 5 grandes etapas:

1. Enumeração de problemas

   Nesta fase, é realizado o reconhecimento da existência de um problema, e a partir disso, é feito uma delimitação do problema e a decisão do que será resolvido. O tema estudado nesta pesquisa é a percepção ativa de agentes em um ambiente de realidade virtual.

2. Sugestão

   Após o reconhecimento do problema e a decisão do que pode ser feito, uma sugestão para a resolução é realizada. A sugestão para esta pesquisa é a implementação de agentes em um ambiente de realidade virtual que utilizem da percepção ativa para a sua interação com o cenário.

3. Desenvolvimento

   Na fase do desenvolvimento, a sugestão anteriormente realizada começa a ser implementada.

4. Avaliação

   Para a fase de avaliação, casos de uso são criados para testar o comportamento dos agentes que foram implementados.

5. Conclusão

   Por fim na fase de conclusão, os resultados obtidos das fases anteriores são apresentados, juntamente com possíveis propostas para a continuidade da pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a fundamentação teórica que foi utilizada como base para a realização deste trabalho. A seção 2.1 apresenta o conceito de agente de forma geral, enquanto a 2.2 vai mais a fundo em uma arquitetura de agente mais específica que foi utilizada no trabalho, a arquitetura BDI. O foco maior do trabalho se encontra na seção 2.3 que expõe o conceito de percepção ativa. Na seção 2.4 é comentado sobre ambientes, e na seção 2.5 a realidade virtual. Por fim a seção 2.6 conclui o embasamento teórico.

2.1 AGENTES

A definição de um agente é algo que ainda é debatido. No momento o conceito de ser algo autônomo é o que se tem maior concordância, como cita Wooldridge (2002), mesmo que exista um consenso geral de que autonomia é um conceito central para um agente, existe pouca concordância além deste ponto. Parte da dificuldade em definir um agente, se deve ao fato de que os atributos associados para um agente, diferem em níveis de importância dependendo do tipo de domínio em que está situado (WOOLDRIDGE, 2002).

O trabalho aqui apresentado considera a seguinte definição para agentes: Um agente é um sistema computacional que está situado em algum ambiente, e que é capaz de ações autônomas neste ambiente no intuito de realizar seus objetivos designados (WOOLDRIDGE, 2002). Agentes autônomos estão situados em algum ambiente. Um robô com apenas sensores visuais em um ambiente sem luz não é um agente. Sistemas são agentes ou não em respeito à algum ambiente (FRANKLIN; GRAESSER, 1997). Para exemplificar mais a definição de agente, pode-se observar um programa de folha de pagamento, como citam Franklin e Graesser (1997), pode ser considerado que, um programa de folha de pagamento em um ambiente do mundo real, sente o mundo através de sua entrada e age no mundo através de suas saídas, porém ele não é um agente pelo fato de que sua saída normalmente não afetaria o que ele sente depois. Um programa de folha de pagamento também falha o teste "tempo extra" de continuidade temporal. Este tipo de programa roda uma vez e entra em coma, esperando ser chamado novamente. A maioria dos programas comuns é comandado por uma ou pelas duas condições, independente do esforço para definir um ambiente apropri-

ado. Todos os softwares de agentes são programas, mas nem todos os programas são agentes.

Possuindo a definição de agentes, é possível acrescentar a parte de inteligência na definição, e como cita Wooldridge (2002), um agente inteligente é aquele capaz de ação autônoma flexível para que realize seus objetivos designados, onde flexibilidade envolve reatividade, pró-atividade e habilidade social. Outra visão de agentes que é possível comentar, é a de Brenner, Zarnekow e Wittig (1998), que citam que qualquer programa de software tradicional pode ser considerado um agente não inteligente, pois até mesmo estes programas desempenham alguma tarefa específica e economizam tempo de seus usuários. Apenas a inteligência permite um agente realizar tarefas com alta autonomia, esperando intervenção de seus usuários apenas para decisões importantes. Em questão de aplicação de agentes inteligentes, pode-se observar três grandes áreas, que não necessariamente são exclusivas, podendo um agente inteligente, dependendo de suas tarefas, estar incluso em uma ou em todas as áreas, sendo elas, agentes de informação, agentes de cooperação e agentes de transação.

- Agentes de Informação: São aqueles que possuem como função, o auxílio na busca de informações de seus usuários.

- Agentes de Cooperação: Tem como principal tarefa a solução de problemas complexos com a utilização de comunicação e cooperação com outros objetos, sejam outros agentes, humanos ou recursos externos.

- Agentes de Transação: São agentes projetados com o foco em processamento e monitoramento de transações, cuidando da segurança e robustez da transação.

Aprofundando um pouco mais em agentes inteligentes, Brenner, Zarnekow e Wittig (1998) apresentam características internas e externas que agentes inteligentes podem possuir, sendo internas as características que definem as ações dentro de um agente, e as externas as que afetam os objetos no ambiente em volta do agente. Agentes inteligentes de alta complexidade idealmente possuem um pouco de cada característica, porém é possível um agente inteligente possuir apenas uma ou poucas dessas características. Quando um agente possui a capacidade de reagir apropriadamente às influências ou informações do ambiente, ele possui característica reativa, e se além disso, este agente consegue não apenas reagir, mas tomar iniciativa neste ambiente, também

se torna um agente proativo. Ainda no trabalho de Brenner, Zarnekow e Wittig (1998), características como raciocínio e comunicação são definidas, respectivamente, como a capacidade do agente em utilizar de seu processamento para alcançar seus objetivos, e a sua interação com o ambiente e seus objetos para isso. Por fim é possível citar também as características de mobilidade e personalidade, que descrevem, respectivamente, a possibilidade de um agente navegar entre redes de comunicação eletrônica e a de ter um comportamento mais crível em seu contexto de uso.

Voltando para uma visão mais abrangente de agentes, Wooldridge (2002) ainda classifica os agentes em quatro arquiteturas diferentes, as quais definem como é realizada a tomada de decisão dos agentes, sendo elas agentes baseados em lógica, agentes reativos, agentes *Belief-Desire-Intention* e arquiteturas em camadas.

2.2 AGENTES BDI

A arquitetura *belief-desire-intention*, ou BDI, é focada no raciocínio prático, o qual é o processo de decidir qual ação tomar para a realização de algum objetivo. Como cita Wooldridge (2002), raciocínio prático envolve dois processos importantes: decidir que objetivos devem ser atingidos, chamado também de deliberação, e como atingir estes objetivos, que é o raciocínio meio-fim. Para que estes processos sejam realizados, a arquitetura BDI, segundo Brenner, Zarnekow e Wittig (1998), oferece cinco fatores que constituem o estado mental de agentes deliberativos:

- Crenças (Beliefs) contém a visão fundamental do agente em relação ao ambiente em que está situado. Um agente utiliza suas crenças para expressar suas expectativas para possíveis estados futuros.

- Desejos (Desires) são derivados diretamente das crenças, contendo os julgamentos do agente para situações futuras a que ele se encontra, ou seja, o que ele quer realizar ou onde quer chegar. A formulação dos desejos de um agente, não necessariamente é realista, pois neste momento ainda não foi feita nenhuma declaração no sentido de um desejo ser ou não realista.

- Objetivos (Goals) representam o subconjunto dos desejos de um agente que podem ser tomadas ações para realizá-los. Diferente

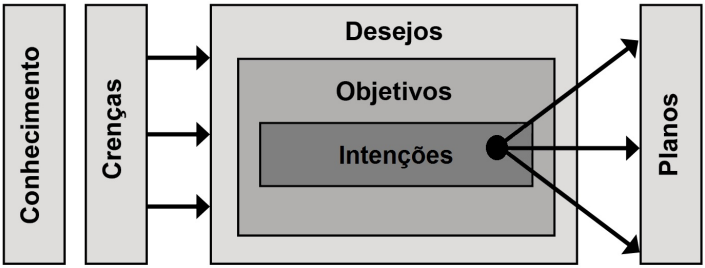

Figura 1 – Estrutura BDI
Fonte: Brenner, Zarnekow e Wittig (1998)

dos desejos do agente, seus objetivos devem ser realistas, e portanto, alcançáveis, como também um objetivo não pode entrar em conflito com outro.

- Intenções (Intentions) são o subconjunto dos objetivos, estando aqui os objetivos que o agente realmente irá seguir. Como um agente normalmente não possui todos os recursos para seus objetivos, ele não pode tentar realizá-los ao mesmo tempo, e por isso, deve configurar prioridades para os seus objetivos, de acordo com sua importância.

- Planos (Plans) possuem uma conexão próxima com as intenções do agente, pois são sequencias de ações que um agente pode realizar para atingir suas intenções.

A estrutura do estado mental do agente pode ser melhor visualizada através da Figura 1.

O agente BDI pode então, utilizar das informações oriundas de onde ele está situado para fazer seu raciocínio. Dependendo do contexto, o agente pode acabar raciocinando diversas vezes sobre uma informação que já percebeu, pois está sendo constantemente enviada para ele. É importante que exista um discernimento sobre percepções, evitando raciocínios desnecessários. Para auxiliar nesta tarefa, é possível utilizar da percepção ativa, assim priorizando as percepções.

2.3 PERCEPÇÃO ATIVA

A maioria das pesquisas passadas e presentes em percepção de máquina envolveu análise de amostras passivas de dados (imagens).

Porém como Aloimonos, Weiss e Bandyopadhyay (1988) já haviam mencionado, a percepção humana não é passiva, e sim ativa. Quando humanos veem e compreendem, eles observam ativamente. Um observador é chamado de ativo quando engajado em algum tipo de atividade cujo propósito é controlar as informações que chegam nos dispositivos sensoriais. Em outras palavras, um agente pode então, ser um observador ativo se utilizar suas capacidades para alterar seus parâmetros visuais, assim adquirindo dados favoráveis de um cenário com o intuito de solucionar a tarefa específica que ele possui em determinado momento (ALOIMONOS, 2013). Ketenci et al. (2012) ainda explicam que, na percepção passiva, o agente adquire tantos dados quanto possível na fase de detecção, e que nessa abordagem, não é necessário que o agente delibere explicitamente sobre suas necessidades de detecção. A percepção ativa, por outro lado, está ligada as intenções e ações do agente, permitindo que o agente perceba o que é necessário para seus objetivos atuais, minimizando informações inúteis.

Para Bajcsy (1988), diferente das análises de amostras passivas de dados, percepção ativa é um problema de processo de aquisição de dados inteligente. Para isso, é preciso definir e mensurar parâmetros e erros de um cenário que em troca podem ser um feedback para controlar o processo de aquisição de dados. Bajcsy, Aloimonos e Tsotsos (2017) definem que um agente é um perceptor ativo se ele sabe porquê ele deseja sentir, e então escolhe o que perceber, e determina como, onde e quando alcançar essa percepção. Para melhor compreensão, Bajcsy, Aloimonos e Tsotsos (2017) ainda classificam os cinco principais constituintes de um agente de percepção ativa em:

2.3.1 Elementos de Percepção Ativa

1. Porquê: Um agente ao se encontrar em determinado estado, gera o que são chamadas de tuplas Expectativa-Ação, que são expectativas de quais ações poderiam ser tomadas a partir deste estado. Essas tuplas Expectativa-Ação vão considerar qualquer tipo de indução inferencial feitas pelo agente, pois o raciocínio indutivo utiliza de uma premissa para chegar em conclusões prováveis.

2. O quê: Cada expectativa é aplicada em um subconjunto específico do mundo, seja um campo visual, tátil ou auditivo, e qualquer ação tomada será aplicada neste campo. Pode-se chamar isso de Seleção de Cenário.

3. Como: Para que um agente possa sentir e/ou perceber, ele deve estar localizado apropriadamente no campo sensorial. O que o agente recebe no campo sensorial deve estar de acordo com suas expectativas de sensação, como também, seu mecanismo de percepção deve estar apto a interpretar os resultados vindos do campo sensorial.

4. Quando: A expectativa de um agente requer Seleção Temporal, ou seja, cada expectativa tem um componente temporal que prescreve quando ela é válida e com qual duração.

5. Onde: Os elementos sensoriais de cada expectativa podem apenas ser sentidos de um ponto de vista específico. Um agente difere sua interpretação de uma cena visual com a de uma superfície tátil. As especificações de cada sensor e sua interação com o seu domínio vão determinar a diferença entre as interpretações. Este processo de diferenciação é chamado de Seleção de Ponto de Vista.

O fator chave é o elemento *porquê*. Um agente de percepção ativa determina dinamicamente o *porquê* de seu comportamento e então controla pelo menos um dos *o quê*, *como*, *onde* e *quando* para cada comportamento (BAJCSY; ALOIMONOS; TSOTSOS, 2017). Quando o *porquê* é definido, o agente pode utilizar de um raciocínio de sensores, por exemplo, para priorizar as percepções ou o sensor que possua maior relação com este *porquê*.

Os sensores de um agente, coletam as informações necessárias para que a percepção ativa seja realizada. Essas percepções podem, tanto vir de objetos dinâmicos, estáticos ou até de outro agente. O que determina o tipo de informação disponível para o agente perceber, é o ambiente em que ele se encontra.

2.4 AMBIENTES

Um agente está situado em um ambiente (WOOLDRIDGE, 2002). Estes ambientes onde os agentes estão situados, podem ter diversas características, e Russell e Norvig (2009) classificam elas como:

- Inteiramente observável vs parcialmente observável: Se os sensores de um agente, conseguem oferecer acesso ao estado completo do ambiente em que está inserido em todos os momentos, então

este ambiente é inteiramente observável. Estes tipos de ambientes são convenientes para o agente, pois não é necessário guardar o estado do ambiente para se manter atualizado. Por outro lado, um ambiente é parcialmente observável quando há falta de dados ou algum sensor está impreciso, por exemplo, caso exista algum objeto que o agente não consegue ter o conhecimento pois está atrás de um outro objeto, este ambiente é parcialmente observável;

- Determinístico vs estocástico: Um ambiente é dito determinístico quando as mudanças de seu estado podem ser determinadas analisando a ação do agente no ambiente, ou seja, são estados previsíveis com os devidos resultados esperados. O ambiente estocástico é um ambiente mais real, onde o estado do mesmo pode mudar de maneira não determinística, sem nenhuma certeza do que pode acontecer a cada mudança de estado;

- Episódico vs sequencial: Quando as experiências e ações de um agente não causam impactos futuros, ou seja, só possuem importância em um momento específico, este agente se encontra em um ambiente episódico. Considerando que as ações de um agente, podem trazer consequências futuras, sejam pequenas ou grandes, ele está situado em um ambiente sequencial, pois existe uma linha temporal de situações que acontecem devido a situações passadas;

- Dinâmico vs estático: Caso o ambiente possa mudar enquanto um agente está decidindo o que fazer, este ambiente pode ser visto como um ambiente dinâmico, pois é independente em relação ao que o agente está fazendo no momento. Se o ambiente não tem um conceito de tempo, e só consegue mudar de estado com as ações do agente, então este ambiente é estático. Ambientes estáticos são mais fáceis de lidar, pois o agente não precisa estar o tempo todo percebendo o mundo, o qual muda apenas com a ação do agente;

- Discreto vs contínuo: A distinção entre discreto e contínuo é feita a partir da análise de como o tempo é manipulado, e das ações e percepções do agente. Por exemplo, um jogo de xadrez possui um número finito de estados distintos, o tornando discreto. Por outro lado, dirigir um táxi é uma situação contínua, onde a velocidade e localização do táxi e dos outros veículos trocam continuamente através do tempo e em um intervalo de valores contínuos.

- Agente único vs multiagente: Um ambiente de agente único é um ambiente onde o agente interage apenas com ele mesmo, como por exemplo, na tentativa de solucionar um jogo de palavras cruzadas por conta própria. Por outro lado, se o ambiente possui mais de um agente interagindo, como por exemplo, dois agentes competindo em um jogo de xadrez, este é um ambiente multiagente.

- Conhecido vs desconhecido: Essa distinção está mais relacionada com o conhecimento do agente em relação as "leis da física" do ambiente. Em um ambiente conhecido, os resultados de todas as ações são fornecidos. Já em um ambiente desconhecido, o agente precisa aprender como o ambiente funciona para tomar boas decisões.

Em ambientes do mundo real, a complexidade pode ser muito elevada para uma boa validação do comportamento de agentes. É importante que, em um primeiro momento, seja possível ter um controle maior das variáveis do ambiente, gerando maior confiança. Para atender essa necessidade, simulações em ambientes de realidade virtual podem ser feitas, possibilitando assim, a criação de ambientes controlados.

2.5 REALIDADE VIRTUAL

A realidade virtual é um tema que causa entusiasmo em grande parte da comunidade, de fato, Burdea e Coiffet (2017) citam que a comunidade científica tem trabalhado nesse campo durante décadas, pois reconhecem que é uma interface humano-computador muito poderosa. A atratividade deste tema fez com que um grande número de publicações, programas de TV e conferências descrevessem a realidade virtual de diversas maneiras, sendo algumas até inconsistentes. Burdea e Coiffet (2017) sugerem que antes de definir o que é realidade virtual, é interessante saber o que não é. Em seu livro eles afirmam que os exemplos de telepresença, nos quais o usuário é imerso em um ambiente remoto, e realidade aumentada, em que algumas imagens computacionais gráficas ou textos são inseridos em cima de imagens do mundo real, já não podem ser considerados como realidade virtual, pois nos dois exemplos existe a presença de imagens que são reais.

O objetivo principal da realidade virtual é, como Parisi (2015) comenta, convencer o seu usuário de que ele se encontra em outro lugar, iludindo o cérebro humano, em particular o córtex visual e partes do cérebro que percebem movimento. De fato, ao iludir o cérebro humano,

existe uma experiência mais imersiva, e a realidade virtual faz isso muito bem, podendo utilizar todos os canais sensoriais humanos. Com isto em mente, Burdea e Coiffet (2017) definem realidade virtual como uma interface humano-computador de ponta que envolve simulações em tempo real e interações através de múltiplos canais sensoriais. Estas modalidades sensoriais são visual, auditiva, tátil, cheiro e gosto.

Ambientes de realidade virtual são bem utilizados nas áreas de jogos virtuais e simulações. Existem ferramentas que auxiliam na criação destes ambientes, como o Unity e o Unreal. É possível, com o uso destas ferramentas, criar agentes virtuais capazes de coletar informações do ambiente de realidade virtual. Um dos problemas disto é, como citam Oijen e Dignum (2011), que sem uma forma de se orientar nas percepções, o agente pode acabar raciocinando muitas informações irrelevantes, pela grande quantidade de informação sensorial.

Um agente para ter um comportamento mais crível nestes ambientes cheio de informações sensoriais, deve possuir algum controle sobre as percepções que chegam até ele. Uma maneira de minimizar o raciocínio de percepções desnecessárias, é implementar raciocínio em seus sensores, possibilitando maior controle de percepções.

2.6 CONCLUSÃO

Com o que foi apresentado nessa seção, é possível notar que o conceito de agente é ainda muito debatido no meio científico, possuindo diversas formas de aplicação. A arquitetura BDI possui um raciocínio que se aproxima do raciocínio humano, tornando o comportamento do agente mais crível, e por isto, foi a arquitetura escolhida para este trabalho. Para aprimorar ainda mais o comportamento, é necessário que o agente consiga filtrar as informações que recebe do ambiente que está situado, necessitando portanto, de percepção ativa. Com o intuito de atuar nessa área de percepção ativa, é necessário ter um agente recebendo informações de um ambiente para validar o comportamento. Foi decidido então, situar o agente em um ambiente de realidade virtual criado na ferramenta Unity.

3 TRABALHOS CORRELATOS

Neste capítulo serão apresentados três projetos já realizados, escolhidos dentre os pesquisados, que possuem relação com o tema e objetivo deste trabalho, que utilizem conceitos exibidos anteriormente na fundamentação teórica. Com o auxílio das bibliotecas digitais IEEE e Google Scholar, a busca de trabalhos foi feita utilizando palavras-chave como agente, percepção ativa e realidade virtual.

3.1 SUPORTE DE ORIENTAÇÃO DE EVACUAÇÃO USANDO DISPOSITIVOS IOT COOPERATIVOS BASEADOS EM AGENTES

No trabalho de Katayama et al. (2017) é descrito um sistema baseado em agentes IoT que oferece suporte à orientação para evacuação em casos de desastres naturais, o qual faz reconhecimento de situação e planejamento de rotas de evacuação.

Katayama et al. (2017) citam que apesar de existirem sinais e placas para auxiliarem na evacuação em casos de desastres, muitas vezes são caros e não tão eficientes, pois podem quebrar ou terem as suas localizações bloqueadas em situações extremas, exigindo assim que existam outros meios dessas orientações chegarem aos que necessitam. Por este motivo, o trabalho em questão utiliza um sistema de agentes inteligentes para determinar situações de risco e agir de acordo, podendo por exemplo, enviar drones para auxiliar a evacuação em algum local. O sistema tem duas principais funções, sendo elas:

1. Reconhecimento de Situação por Cooperação de Dispositivos IoT

   Com a utilização de sensores, os dispositivos conseguem reconhecer uma situação de risco, enviando essa informação para outros dispositivos conectados. Cada dispositivo tem conhecimento de sua localização e a funcionalidade de verificar o grau de perigo na mesma, o qual é calculado fazendo uma diferença no valor de sensores entre uma situação regular e uma situação de desastre, possibilitando a priorização de locais mais críticos para atuar.

2. Planejamento da Orientação de Evacuação pelo Agente

   O agente de orientação de evacuação gera um plano de evacuação a ser seguido pelos agentes auxiliares, como por exemplo, drones, que irão sinalizar e acompanhar as pessoas que estão sendo

evacuadas pelas rotas determinadas. Este plano de evacuação é flexível, podendo ser alterado caso um novo desastre aconteça, fazendo com que os agentes auxiliares mudem a rota.

As duas funções do sistema foram implementadas e testadas separadamente em ambientes reais, alcançando sucesso no reconhecimento de situação com obstáculos e no planejamento da orientação de evacuação, mesmo com pseudo desastres secundários que ocorreram.

3.2 MELHORIA DO COMPORTAMENTO DE CRUZAMENTO DE ESTRADAS COM ABORDAGEM DE PERCEPÇÃO ATIVA

O conceito de percepção ativa é abordado no trabalho de Ketenci et al. (2012), onde é desenvolvido um modelo de percepção ativa para simular o comportamento de cruzamento de estradas dos motoristas. Ketenci et al. (2012) comentam que um dos maiores problemas das cidades são os engarrafamentos, e a origem e a maneira de contorná-los, dependem bastante do comportamento dos motoristas. Por estes motivos, um modelo de percepção ativa se torna útil, podendo fazer simulações para verificar, por exemplo, como uma alteração na estrutura de uma estrada para melhoria do tráfego afetaria os motoristas que a utilizam. No trabalho em questão, a percepção e decisão dos motoristas serão feitas continuamente, para melhor adaptação para as mudanças situacionais, aprimorando o nível de comportamento realista nas áreas de conflito em estradas. Além disso, como o objetivo é simular o comportamento humano, deve-se considerar que a capacidade de percepção humana é limitada, e por isto em um ambiente em que muitas percepções são dados de entrada, só as mais relevantes para o momento devem ser consideradas, ou seja, priorizadas.

A implementação da percepção ativa no contexto do trabalho de Ketenci et al. (2012) é feita utilizando três principais conceitos, sendo eles foco, classificação de percepção em respeito a relevância e percepção de recursos limitada. O foco é o domínio de interesse do agente em um espaço sensorial, ou seja, o espaço que o agente irá se concentrar considerando suas intenções. Já a classificação de percepções é exatamente o agente verificar qual percepção possui relevância com a sua intenção atual, e assim classificá-la como prioritária, a qual pode em um ambiente dinâmico, perder sua prioridade para outra, devido a constante mudança de cenário. Por fim, como citado anteriormente, a percepção humana é limitada, portanto a percepção de recursos limitada torna a simulação da percepção humana mais realista.

Um dos problemas complicados em modelos de tráfegos baseados em agentes é o cruzamento de estradas, onde o problema chave é a complexidade das interações entre os agentes e o número de agentes envolvidos simultaneamente no cruzamento de estradas. Para tentar resolver este problema, e baseando-se na abordagem já existente de considerar o comportamento do motorista no contexto de interseção, a qual um motorista seleciona um número de "jogadores", sendo estes os outros motoristas, quando se aproxima de um cruzamento para decidir se anda ou para, Ketenci et al. (2012) implementaram uma percepção ativa limitada para esta escolha de "jogadores", que agora depende do contexto do tráfego e de como este contexto é percebido pelo agente.

Para realização de testes, foi utilizado uma representação de um cruzamento da vida real em Reggio Calabria na Itália, onde a estrada escolhida foi a estrada leste, que possui três rotas, duas para ida de veículos e uma para a saída. Foram realizadas 100 simulações onde foi verificado o número de acidentes, o tempo de execução e os impasses nos cruzamentos. Analisando as simulações, foi concluído no trabalho que não houveram mudanças significativas no comportamento dos agentes, porém com a utilização da percepção limitada, pode-se perceber que o agente não tem necessidade de calcular as percepções de todo o cenário, podendo calcular apenas as percepções relevantes ao seu contexto no momento.

3.3 MODELO DE AGENTES E-BDI INTEGRANDO CONFIANÇA BASEADO EM SISTEMAS MULTI-CONTEXTO

Gelaim (2016) propôs em 2016, um modelo de agentes BDI onde o conceito de confiança faça parte do raciocínio do agente. Uma de suas motivações para este modelo foi melhorar a interação entre agentes, pois em sistemas multiagente existe a possibilidade de interação com algum agente mal-intencionado, portanto, a utilização de um sistema de confiança ajudaria o agente a decidir com quem é seguro interagir no ambiente.

O agente proposto no trabalho em questão, utiliza de diversos contextos para seu funcionamento, sendo eles contextos de comunicação, emoções, confiança, planejamento, intenções, desejos e crenças, além de regras de ponte para a troca de informações entre os mesmos. A seguir uma breve explicação sobre os contextos utilizados por Gelaim (2016):

- Contexto de Comunicação (CC): Este é o contexto responsável

pela comunicação do agente com o seu ambiente, sendo então o local para receber percepções do ambiente e enviar ações do agente. É possível neste contexto receber experiências de entidades já existentes com a entidade que está sendo avaliada se é confiável ou não.

- Contexto de Emoções (EC): A representação das emoções do agente e suas manipulações são realizadas neste contexto. O fator emocional do agente pode afetar o cálculo da confiança, apesar de não ser de uma forma tão significativa como as crenças. Seguindo o modelo usado no trabalho, o estado emocional atual do agente é integrado em uma crença do tipo julgamento ao ser criada. O contexto de emoções possui ligação significativa com o de crenças, assim como com o de intenções e comunicação, isso se deve ao fato de que mudanças nesses contextos podem afetar as emoções do agente.

- Contexto de Crenças (BC): O que o agente acredita em relação ao mundo em sua volta está inserido neste contexto, podendo medir as crenças também com um grau de certeza de sua credibilidade. Foram adicionados componentes de julgamento e confiança na linguagem de crenças já existentes de Casali, Godo e Sierra (2005).

- Contexto de Intenções (IC): Aqui ficam armazenadas as intenções do agente, sendo geradas pelas suas crenças e desejos. Essas intenções possuem uma graduação, para avaliar os custos e benefícios envolvidos nas ações que o agente pode tomar.

- Contexto de Planejamento (PC): O planejamento para realizar os desejos que o agente possui são feitos neste contexto. Os planos de um agente devem ser compatíveis com as condições atuais, ou seja, devem ser planos que a sua realização se torna possível devido às circunstâncias do ambiente em que o agente se encontra.

- Contexto de Confiança (TC): Com a utilização de um modelo de confiança, informações do próprio contexto e de informações provenientes do contexto de crenças, este contexto determina a confiança do agente em outras entidades. O contexto de confiança utiliza de diversas informações para seu funcionamento, sendo elas interação direta, observação direta, informação de testemunho, viés, norma, dimensões e contexto, avaliação da confiança e resultado na tomada de decisão. A interação direta é o resultado de uma interação entre o agente e outra entidade do ambiente,

e ao avaliar o comportamento desta outra entidade, está sendo feita a observação direta. O agente pode interagir com outras entidades no ambiente para ter o julgamento delas sobre a entidade alvo, ou seja, obter a informação de testemunho. O viés pode ser visto, de certa forma, como otimismo ou pessimismo do agente, fazendo com que em caso otimista, julgamentos positivos tenham mais atenção. As normas podem ser vistas como a lei em uma sociedade de agentes. Dimensões podem criar uma nova confiança ao se unirem, por exemplo, a confiança de conhecimento de peças de carro e a confiança de saber como utilizá-las, podem se unir em uma confiança de ser mecânico. Os contextos descrevem um cenário para definição de confiança com maior precisão. Por fim, a avaliação de confiança pode decidir entre qual modelo de confiança usar caso exista mais que um, e ao possuir as novas crenças criadas e a decisão oriunda dos contextos de planejamento e intenções, temos o resultado na tomada de decisão.

- Contexto de Desejos (DC): Neste contexto estão os objetivos gerais do agente, o que o agente quer que ocorra e também o que ele não quer que aconteça. Por exemplo, um agente pode desejar se exercitar ao ar livre, e também desejar que não chova enquanto ele sai para se exercitar. Além disso, os desejos podem ter um nível de priorização, fazendo com que um agente deseje mais um objetivo do que o outro.

- Regras de Ponte: As regras de ponte são o que fazem a comunicação entre os contextos ser possível. Um contexto só pode se comunicar com outro, caso exista uma regra de ponte que permita essa comunicação. Os contextos e as regras de ponte que possibilitam sua comunicação, podem ser conferidos na figura 2, onde cada aresta é uma regra de ponte ligando dois contextos.

O trabalho de Gelaim (2016) focou mais na parte teórica do modelo, preparando a base para que a implementação de um agente seguindo este modelo fosse possível em um trabalho futuro.

3.4 CONCLUSÃO

Analisando os trabalhos correlatos, fica claro a necessidade da utilização dos conceitos presentes neste trabalho em agentes inteligentes. No primeiro trabalho a utilização dos sensores é de considerável

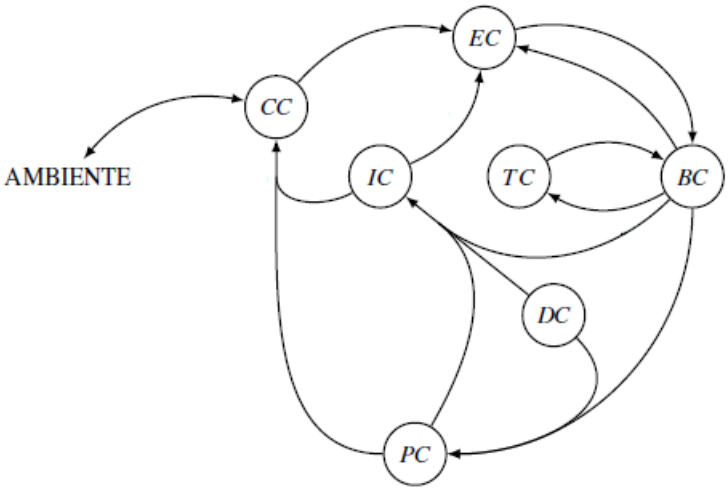

Figura 2 – Modelo de Agente Multi-Contexto
Fonte: Gelaim (2016)

importância para minimizar os impactos de desastres naturais. Utilizando uma pequena comparação de cenários observados pelos sensores, o agente pode definir qual área pode ser priorizada. Existe a possibilidade de que a eficiência dessa decisão possa ser melhorada ao adicionar um maior raciocínio nos sensores do agente.

A simulação do comportamento humano com agentes de percepção limitada é o foco do segundo trabalho. Apesar de que essa limitação não tenha melhorado substancialmente o comportamento dos agentes, ela salva o agente de realizar cálculos desnecessários relacionados ao ambiente. Além disso a limitação de percepções torna a simulação mais real, pois a percepção humana é também limitada.

O modelo proposto no terceiro trabalho não foi ainda implementado, porém oferece uma base para a implementação de agentes BDI, o qual este trabalho de conclusão utiliza. O ciclo racional do agente começa a partir das informações vindas do ambiente através dos sensores, possivelmente criando a relação de maior raciocínio sobre sensores com maior raciocínio do agente. O raciocínio sobre sensores demonstra-se então, um tema de importância no contexto de agentes, podendo ser o primeiro passo para a percepção ativa e uma simulação aprimorada do comportamento humano.

4 MODELO PROPOSTO

O objetivo deste trabalho é desenvolver um modelo de raciocínio sobre sensores para agentes em realidade virtual. Um agente inteligente deve ter a capacidade de perceber o ambiente em sua volta e utilizar destas percepções para realizar alguma tarefa a ele designada. Os sensores do agente são responsáveis por receber as informações do ambiente em que se encontram. Caso o agente esteja em um ambiente dinâmico, existem diversas variáveis e novas situações ocorrendo para o agente tomar conhecimento, e por este motivo, o agente deve possuir certo discernimento sobre o que ele deve perceber naquele momento, ou seja, quais das diversas percepções que ele recebe são relevantes para suas atuais intenções, fazendo com que o comportamento do agente se aproxime mais do comportamento humano.

A percepção humana é seletiva, dependendo da situação que uma pessoa se encontra, ela pode estar mais ou menos focada no ambiente à sua volta. Quando uma pessoa faz um passeio por um parque repleto de outras pessoas, muitas percepções chegam ao mesmo tempo, porém grande parte dessas percepções são filtradas por não possuírem relação com o objetivo da pessoa naquele momento. Por outro lado, caso a pessoa esteja andando em uma rua pouco iluminada durante a noite, qualquer mudança no ambiente será considerada, pois existe uma preocupação, ou intenção, em estar seguro o tempo todo. Este tipo de situação demonstra a necessidade de um agente em saber principalmente o que perceber e o por que perceber.

Este trabalho utilizará como base o modelo de agente proposto por Gelaim (2016) mencionado na seção de trabalhos correlatos, com um enfoque maior no contexto de comunicação, pois é o contexto que se comunica diretamente com o ambiente, sendo então o primeiro a receber as percepções deste ambiente e onde os sensores do agente estão localizados. Os sensores recebem percepções do ambiente o tempo todo, e faz seu raciocínio em virtude de todas essas percepções. No recente trabalho de Gelaim et al. (2018), o modelo de agentes citado anteriormente sai da teoria para a prática, com a denominação de agente Sigon.

4.1 CONTEXTO DE COMUNICAÇÃO

Como citado anteriormente, o contexto de comunicação é responsável por fazer a interface entre o agente e o ambiente em que está situado. Gelaim et al. (2018) explicam que este contexto possui em sua implementação, sensores e atuadores para que o agente possa perceber o ambiente e atuar no mesmo. Os sensores são responsáveis por receber os dados oriundos do ambiente, e em sua estrutura existe um identificador e uma implementação do mesmo. Os atuadores servem para que o agente possa modificar o ambiente, atuando no mesmo, e sua estrutura é similar à dos sensores, com um identificador e uma implementação.

$$sensor : SENSOR`(`'identificador`,`' implementacao`)`'`.`'$$

$$atuador : ATUADOR`(`'identificador`,`' implementacao`)`'`.`'$$

Gelaim et al. (2018) definem que um agente deve possuir ao menos um sensor ou atuador, porém pode possuir inúmeros sensores e atuadores, dependendo de sua aplicação. O modelo de sensor possui um publicador literal, o qual vai publicar esses literais, que são strings representando as percepções do agente, no contexto de comunicação. A figura 3 demonstra a relação dos sensores com o contexto de comunicação. É de responsabilidade do desenvolvedor a implementação dos sensores e a publicação dos literais, que deve estar de acordo com a especificação da linguagem usada do contexto de comunicação do agente Sigon. O contexto de comunicação é um contexto de observação, podendo se inscrever a um ou mais sensores. Após receber os literais dos sensores, o contexto pode armazená-los e começar o ciclo de raciocínio do agente. Este trabalho então propõe que, neste contexto, seja feita uma filtragem das percepções do agente, podendo assim dar suporte a percepção ativa, ao decidir o que será priorizado para o agente na situação em que se encontra.

O ser humano possui inúmeros sensores para ajudar na percepção do ambiente em que se encontra, portanto pode-se imaginar que um agente, para ter um comportamento mais parecido com o do ser humano, deve também possuir diversos sensores. A figura 4 demonstra alguns sensores que o agente pode possuir. O agente percebe um obstáculo logo à sua frente (1), com um sensor relacionado a visão. O sensor responsável pelo tato da mão do agente (2), detecta o toque em um bloco. O chão também pode ser percebido (3), neste caso através do sensor do pé do agente.

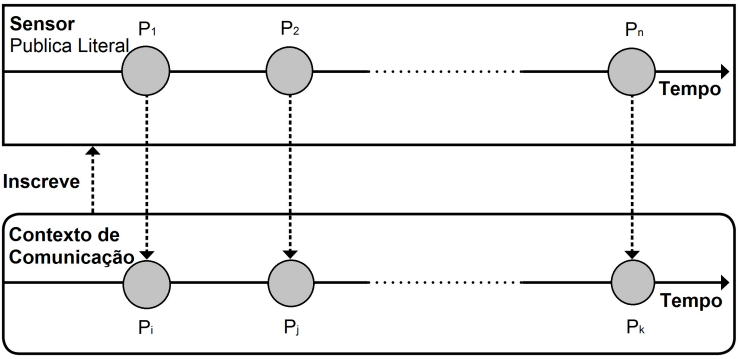

Figura 3 – Representação Adaptada do Modelo de Sensores
Fonte: Gelaim et al. (2018)

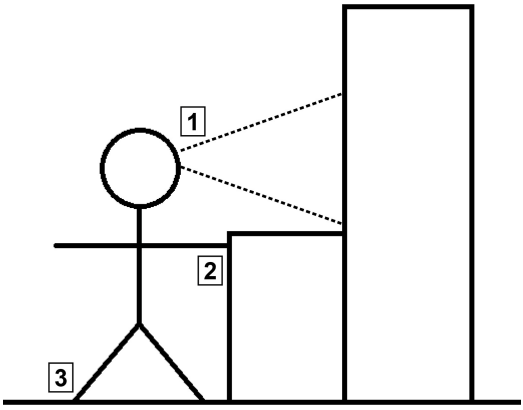

Figura 4 – Sensores em um Agente
Fonte: O Autor

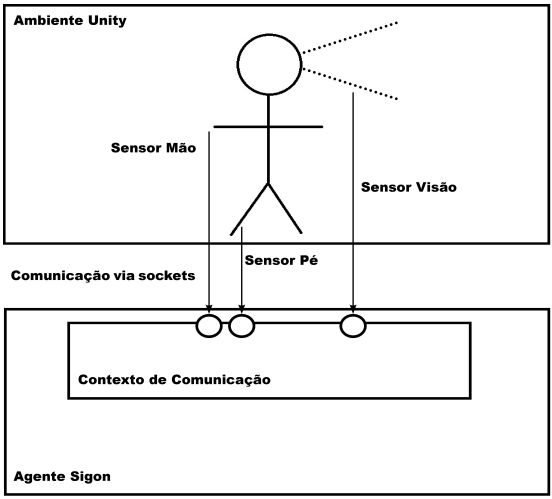

Figura 5 – Exemplo de Agente neste Modelo
Fonte: O Autor

Para o contexto deste trabalho, foi feita a integração do ambiente de realidade virtual Unity com o Sigon, um framework para o desenvolvimento de agentes como sistemas multi-contextos, introduzido por Gelaim et al. (2018). Em questões de comunicação entre as tecnologias, é utilizada uma arquitetura cliente-servidor com comunicação via sockets, onde cada um dos sensores do agente Sigon é um servidor com uma porta específica. Esses servidores foram feitos para receber as informações dos clientes, que são as representações dos sensores no ambiente criado na ferramenta do Unity, a qual possibilita a implementação de ambientes virtuais 3D para a verificação do funcionamento das percepções do agente. Nos sensores pode ser feito a filtragem de percepções e um raciocínio sobre o que está recebendo para enviar ao agente. A figura 5 mostra um exemplo de como um agente pode ser construído nesse modelo.

Os clientes Unity podem, enviar mensagens contendo todo o tipo de informação coletada do ambiente para seus sensores específicos. Um exemplo de mensagem enviada é o nome de um objeto específico no ambiente junto com as suas coordenadas em relação ao mesmo.

$$msg = position(\text{“}objeto, x, y, z'')$$

Neste exemplo, uma possível ação de um atuador ligado ao agente

seria, ao tomar conhecimento da posição de determinado objeto, se deslocar até ele.

O agente conhece todos os seus sensores, e ao ser criado, já os inicia para que comecem a receber informações do ambiente assim que conectados. O ambiente envia informações o tempo todo para os sensores, e diversas vezes o cliente responsável por enviar para um sensor específico não possui mudanças, ou seja, nenhuma informação nova está sendo enviada. Para estes casos que não ocorre variância das informações, faz sentido que este sensor diminua de prioridade com o passar do tempo, fazendo com que o agente tenha um foco maior nos sensores que estão recebendo novas percepções.

A priorização de sensores, portanto, auxilia para que o agente receba, na maior parte do tempo, mais informações dos sensores que estão sendo mais utilizados. Cada sensor do agente é uma thread Java e um socket, começando com um valor de prioridade 5 e variando esse valor em um intervalo de 1 à 10, de acordo com a necessidade e implementação. O intervalo da prioridade pode variar dependendo do tipo de sensor. Por exemplo, a visão do agente pode ser dividida em 3 sensores para representar sua distância em relação ao agente, fazendo com que o sensor de visão mais próximo do agente, possua valores de intervalo maiores que o sensor de visão mais distante. A motivação para intervalos diferentes nos sensores de visão é baseada no comportamento humano, que em cenários onde algo desconhecido acontece próximo ao ser humano, e ao mesmo tempo, algo distante do mesmo, a priorização de atenção será no que está mais próximo.

É possível, através do modelo presente no trabalho, criar tantos sensores quanto necessário para determinada função. A função de visão V do agente, por exemplo, pode possuir n sensores para representar, como citado anteriormente, a distância. Este comportamento é exibido na figura 6. O sensor de maior área contém os sensores de menor área, percebendo objetos que não estão na área dos sensores menores.

$$V = \sum_{i=1}^{n} S_{i-1} \subseteq S_i$$

A priorização de sensores pode utilizar de estratégias diferentes, de acordo com a necessidade de contexto, sendo de responsabilidade do programador a escolha ou a criação da estratégia que melhor o atende. O modelo possibilita implementar diferentes estratégias de priorização, conforme a necessidade dos agentes.

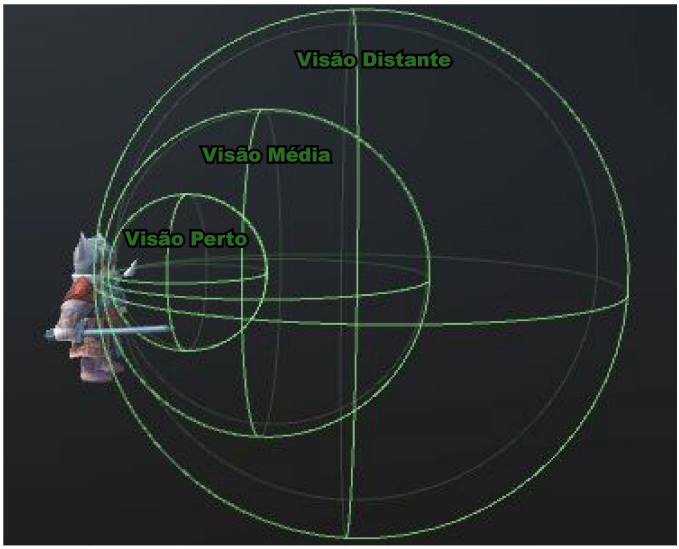

Figura 6 – Representação da Visão do Agente
Fonte: O Autor

4.2 IMPLEMENTAÇÃO DE ESTRATÉGIAS

Para este trabalho, foram implementadas e validadas duas estratégias de priorização de sensores, como exemplos de estratégias possíveis. A primeira estratégia é chamada de estratégia gradativa, e a segunda de estratégia repentina.

A estratégia gradativa leva em consideração as percepções diferentes que um sensor recebe. Para que os sensores consigam verificar se as informações que estão recebendo são sempre as mesmas ou não, é necessário que cada um guarde, até certo tempo, as percepções que chegaram anteriormente. A partir disso, o sensor pode checar regularmente as percepções novas com as antigas, diminuindo ou aumentando sua prioridade de acordo com a conclusão. As percepções antigas que o sensor conhece devem possuir um tempo limite para ficarem no mesmo, para que essa checagem faça sentido, pois caso contrário, o sensor poderia em certo momento parar de receber a percepção de um objeto qualquer e ainda considerar que não houve mudança no ambiente. É possível analisar a figura 7 para melhor visualização.

Aprofundando um pouco no nível de implementação, existem

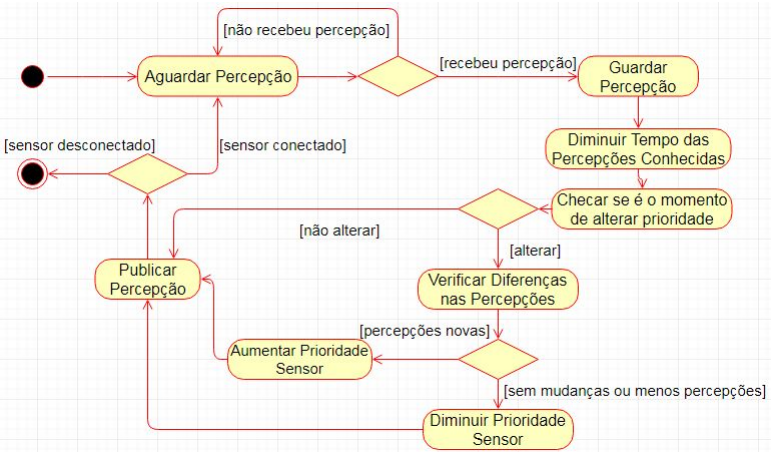

Figura 7 – Diagrama de Atividades de um Sensor
Fonte: O Autor

dois HashSets de percepções, um para as percepções que estão chegando, e outro para as percepções antigas. A cada momento de checagem para alteração de prioridade, o HashSet de percepções antigas recebe as percepções que estavam no outro HashSet, para que seja verificado se houveram mudanças. Como o comportamento do HashSet não permite que exista mais de um elemento igual dentro de si, caso o HashSet de percepções antigas não mude, significa que nenhuma percepção nova chegou desde a última checagem. A prioridade então, é aumentada caso tenha mudanças, e caso não tenha, a prioridade do sensor é diminuída. Isto vale para todos os sensores.

A estratégia repentina para a priorização de sensores, foca mais na quantidade de percepções que chegaram em dado momento. A prioridade do sensor será de acordo com a seguinte fórmula:

$$P = (1 - \frac{m}{F}) \cdot 10$$

Onde P é a prioridade do sensor, F é a força que está sendo colocada no sensor, neste caso o número de percepções, e m é o ponto de referência configurável, denominado como massa. Quanto maior a massa de um sensor, menor será a tolerância com o número de percepções, fazendo com que a prioridade se mantenha baixa. Por outro lado, um sensor que receba um número de percepções maior que a sua massa, terá sua prioridade elevada de acordo. Se o resultado da equação der

negativo, significa que o número das percepções é menor que a massa, portanto a prioridade do sensor será mínima.

A implementação dessa estratégia utiliza do HashSet de percepções para guardar o número de percepções até o momento de checagem. Quando chega o momento de modificar a prioridade do sensor, a fórmula é aplicada, utilizando a massa configurada para o contexto e a quantidade de percepções que chegaram. Após a modificação de sua prioridade, o sensor esvazia o HashSet utilizado, para receber a nova leva de percepções.

As duas estratégias aqui apresentadas possuem maneiras diferentes de alterar a prioridade do sensor. A primeira estratégia altera a prioridade de forma gradativa, ou seja, a prioridade de um sensor não pula níveis. Já na segunda estratégia, é possível que uma prioridade saia de 5 para 10 de uma vez, dependendo do número de percepções. Um cenário possível para a segunda estratégia, é ser utilizada em um agente responsável pela segurança, onde alterações no ambiente podem causar estado de alerta rapidamente. Por outro lado, a primeira estratégia pode ser usada em agentes auxiliares em jogos virtuais, onde a prioridade de seus sensores é menos crítica.

5 TESTES

O agente implementado no framework Sigon para este trabalho possui 5 sensores, sendo 3 sensores responsáveis pela visão do agente, entre os quais o alcance é o fator diferencial de cada um, 1 sensor dedicado para a audição do agente e 1 sensor que recebe informações sobre o tato.

Para a verificação do funcionamento da priorização de sensores, foi criado um cenário simples de testes para o agente, que pode ser visualizado a partir da figura 8. O cenário consiste em objetos que o agente pode perceber, como chão, paredes e blocos estáticos, podendo em suas percepções verificar por exemplo, a posição de tais objetos em relação aos eixos x, y e z, dentro do ambiente virtual 3D. Além dos objetos fixos no cenário, foi implementada a criação de blocos dinâmicos em tempo de execução, os quais caem no cenário em posições aleatórias à frente do agente. Os sensores perceberão cada posição de um bloco em movimento como uma nova percepção sobre o mesmo, e quando este bloco estiver em uma posição de repouso, os sensores continuarão percebendo o objeto, mas conseguem verificar que não é uma percepção nova.

O primeiro teste realizado neste cenário foi feito utilizando a estratégia gradativa de priorização e desconsiderando a publicação de percepções para o raciocínio do agente, tendo foco apenas no raciocínio dos sensores para a priorização. A comunicação do ambiente Unity com o agente Sigon foi feita através de uma conexão TCP. Foi verificado que a priorização de sensores funcionou corretamente, mostrando que os sensores que estavam recebendo percepções diferentes ao longo do tempo, tiveram suas prioridades aumentadas, enquanto os que estavam recebendo as mesmas percepções, tiveram suas prioridades diminuídas.

Para melhor visualização, foram coletados dados referentes aos sensores de visão perto, média e distante, os quais tem um intervalo de prioridade de 5 a 10, 3 a 8 e 1 a 6, respectivamente, e colocados nos gráficos das figuras 9 e 10. Estes valores de intervalo de prioridade seguem a ideia, apresentada no capítulo 4, sobre diferentes prioridades para sensores de visão em relação à distância que alcançam. O gráfico da figura 9 mostra como a prioridade do sensor se comportou em relação ao tempo, em milissegundos (ms), enquanto o gráfico da figura 10, mostra o número de percepções novas que chegaram nesse sensor durante o mesmo tempo do gráfico de prioridade. Os gráficos devem ser analisados em conjunto para melhor compreensão. É possível notar

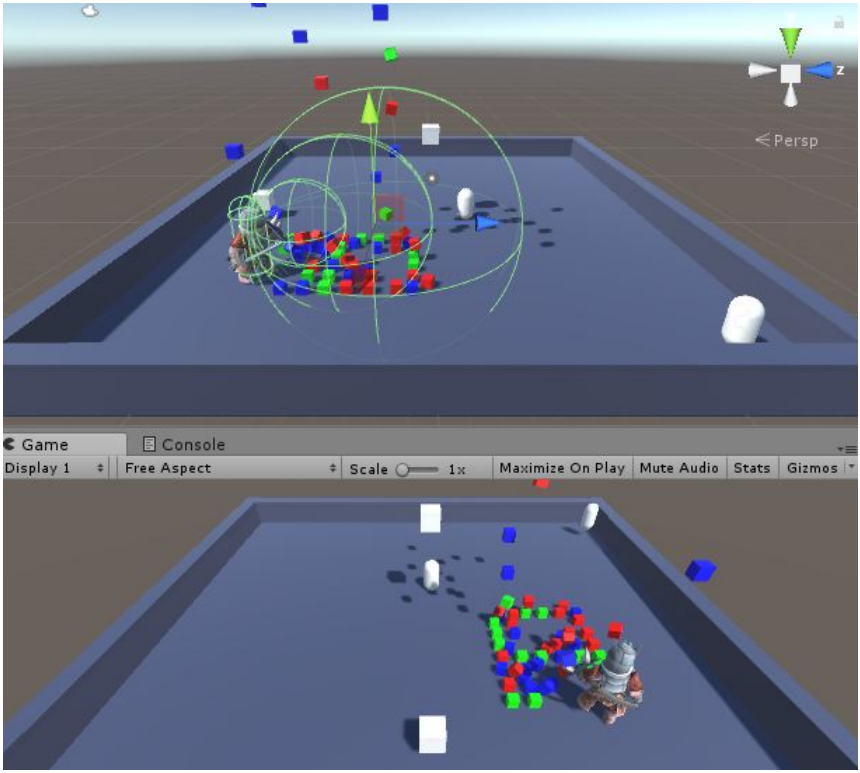

Figura 8 – Cenário de Testes
Fonte: O Autor

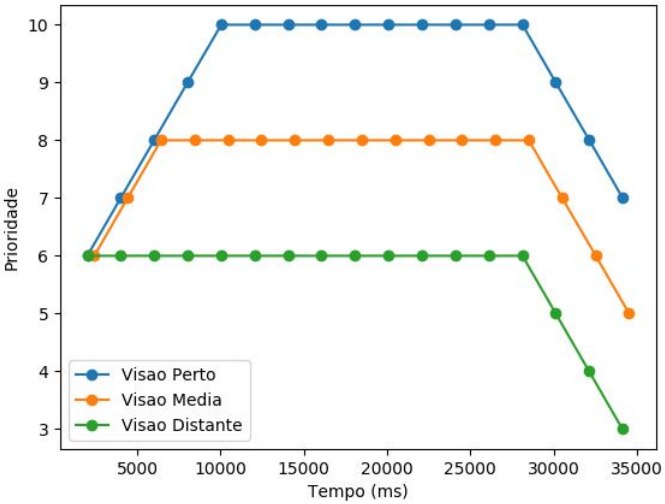

Figura 9 – Prioridades dos Sensores de Visão
Fonte: O Autor

que, nos momentos que haviam novas percepções, a prioridade dos sensores aumentava até seu limite máximo. Ao chegar no limite máximo de cada sensor, a prioridade se mantinha constante, até o momento que os sensores parassem de receber novas percepções, fazendo com que suas prioridades começassem a diminuir.

O segundo teste realizado neste cenário, levou em consideração também a publicação das percepções para o raciocínio do agente, além da priorização dos sensores com a estratégia gradativa. O tipo de conexão se manteve como TCP, e como resultado, foi verificado que quando existe em um sensor, um número muito elevado de percepções por segundo, sendo publicadas no agente para seu raciocínio, com uma média de 508,5 percepções como demonstra o gráfico da figura 11, o ambiente Unity trava. Este comportamento de trava se deve ao fato de que o protocolo TCP garante a segurança da comunicação, entregando as mensagens de forma ordenada e sem a perda das mesmas. Por estes motivos, caso uma mensagem ainda esteja sendo processada, existe um limite de mensagens que podem ser enviadas nesse intervalo de tempo, e se este limite é atingido, a conexão trava o envio de novas mensagens. No cenário testado, como os blocos dinâmicos não somem, existiam inúmeras percepções sendo enviadas por segundo, e mesmo que o sensor consiga fazer sua priorização, a publicação de cada mensagem para

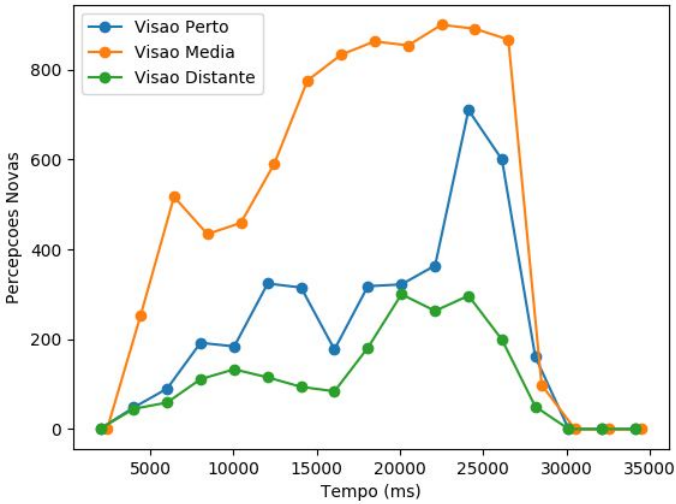

Figura 10 – Novas Percepções dos Sensores de Visão
Fonte: O Autor

o raciocínio do agente não acompanhava em velocidade o número de percepções recebidas, fazendo com que o limite da conexão fosse alcançado. O gráfico da figura 11, demonstra que 20 execuções foram realizadas para chegar na média de 508,5 percepções por segundo.

Em virtude do segundo teste realizado, foi decidido fazer uma verificação mais a fundo sobre o tempo do envio de percepções do ambiente, e também o tempo da publicação de uma percepção no agente. Com isto em mente, foi retirado novamente a publicação da percepção do processo, para verificar apenas o raciocínio do sensor e o tempo de envio de percepções pelo ambiente. Durante um tempo médio de 30 segundos, foram coletadas percepções do ambiente, e ao analisar estes dados, pode-se verificar que um intervalo entre o envio de uma percepção e outra, podia chegar a 0,02 milissegundos. Por outro lado, o tempo médio de uma publicação é de 42,8 milissegundos, um tempo superior ao de envio de percepções do ambiente. Este tempo de publicação foi verificado ao isolar a publicação de percepção do agente, publicando percepções em uma iteração dentro do contexto de comunicação.

Os gráficos das figuras 12 e 13, demonstram visualmente os resultados dos testes de tempo do envio de percepções e da publicação no agente, com uma amostra de 20 valores para cada. Considerando um cenário com um nível muito elevado de objetos dinâmicos para per-

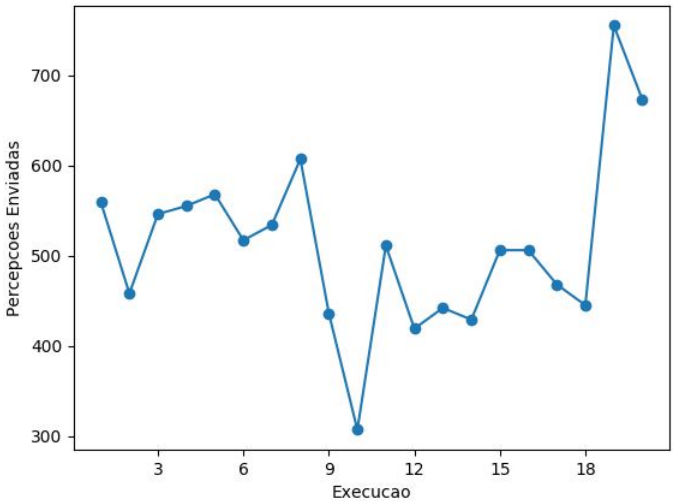

Figura 11 – Testes de Percepções Enviadas
Fonte: O Autor

ceber, em média podem haver 2140 percepções sendo enviadas pelo ambiente, no intervalo de tempo que uma percepção está sendo publicada no agente. Estes resultados explicam o comportamento de trava do protocolo TCP, pois a fila de espera de mensagens que ele possui, acaba atingindo o limite.

Considerando o foco deste trabalho, foi decidido tentar adaptar o envio de percepções em relação ao raciocínio do agente. Esta decisão também teve como base uma pequena pesquisa sobre o tempo médio da discriminação de tempo humana. De acordo com a revisão de Grondin (2010), a sensibilidade máxima para discriminação do tempo, está localizada em um intervalo de 300 milissegundos até 800 milissegundos. O tempo do ciclo de raciocínio do agente já é mais rápido que a discriminação de tempo humana, portanto foi decidido não ser alterado neste primeiro momento.

O ambiente em que o agente está situado foi então configurado para que o envio de percepções não sobrecarregue a conexão, por conta de sua velocidade de envio. Foi determinado que este envio só pode acontecer a cada 50 milissegundos, o qual é aproximadamente o tempo médio do ciclo de raciocínio do agente, e em um intervalo de 0,5 milissegundos até que o tempo reinicie. Este intervalo foi feito para que o ambiente possa enviar o máximo de percepções possíveis em um ci-

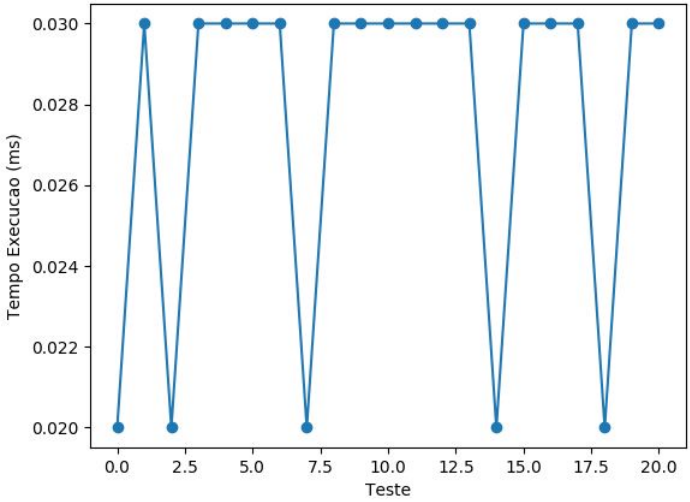

Figura 12 – Tempo de Envio de Percepções
Fonte: O Autor

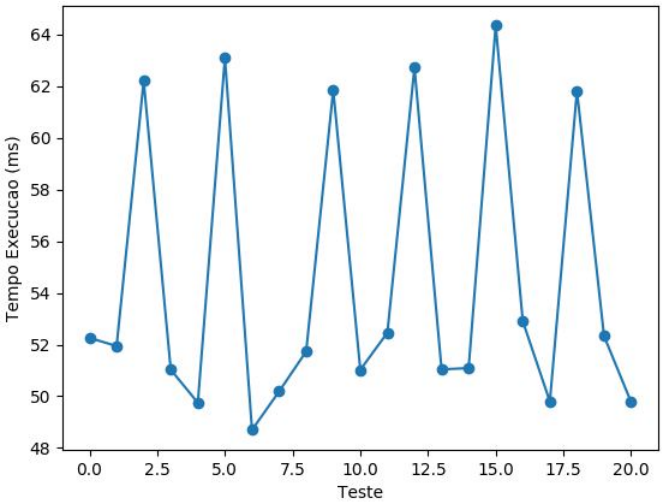

Figura 13 – Tempo de Publicação no Agente
Fonte: O Autor

clo de envio, sem sobrecarregar a conexão. O tempo para envio de percepções e seu intervalo são configuráveis, para atender contextos diferentes. Para este trabalho, essa regra foi implementada para os sensores de visão, que são os sensores que recebem o maior número de informações em um ambiente dinâmico. Foi considerado que o toque e a audição são sensores com maior discernimento em relação a mudanças, para que por exemplo, se uma música parar completamente por alguns milissegundos, isso seja percebido.

Após a realização dos ajustes, foi feito outro teste, para verificar se o problema registrado no segundo teste foi resolvido. Como resultado, a trava de conexão não ocorria mais facilmente, precisando ter mais de 1000 objetos enviando suas informações para que a conexão comece a travar novamente.

Com o intuito de tornar o raciocínio dos sensores um suporte ainda melhor para o agente, foi criado também uma lógica em relação a atualização de percepções sobre um objeto. O agente Sigon armazena as informações sobre determinado objeto, portanto, se o ambiente está enviando sempre as mesmas informações de um objeto para um sensor, só faz sentido publicar no agente se alguma mudança ocorreu no objeto. Por exemplo, se um sensor recebe nove percepções seguidas que informam que o objeto está na posição x, e após isso uma que mostra que o objeto mudou para a posição y, só é necessário publicar duas vezes no agente, a primeira percepção da posição x e a atualização para a posição y. Esta alteração nos sensores permite que o agente processe menos informações desnecessárias, possibilitando maior tolerância ao envio de percepções pelo ambiente, pois não perde tempo recebendo a publicação de algo que já conhece. A figura 14 mostra um exemplo visual onde P1, P2, P3 e P4 são percepções, que acontecem uma após a outra, de um sensor de visão sobre a posição de um cubo verde caindo.

O terceiro teste realizado, foi feito para avaliar o funcionamento da estratégia repentina de priorização. Este teste também responde o seguinte questionamento sobre essa estratégia: “Qual é a influência da massa quando o agente possui um número de percepções muito maior que a mesma?”. Para uma melhor visualização deste comportamento, foi utilizada uma massa de tamanho 50 para todos os sensores, e feito com que um sensor específico recebesse mais percepções que os outros. O sensor escolhido para receber mais percepções neste caso foi o sensor de visão perto. Os gráficos das figuras 15 e 16, demonstram o comportamento dos sensores neste teste.

Os resultados do terceiro teste demonstram o funcionamento da estratégia repentina de priorização. Além disso, ao analisar os dois grá-

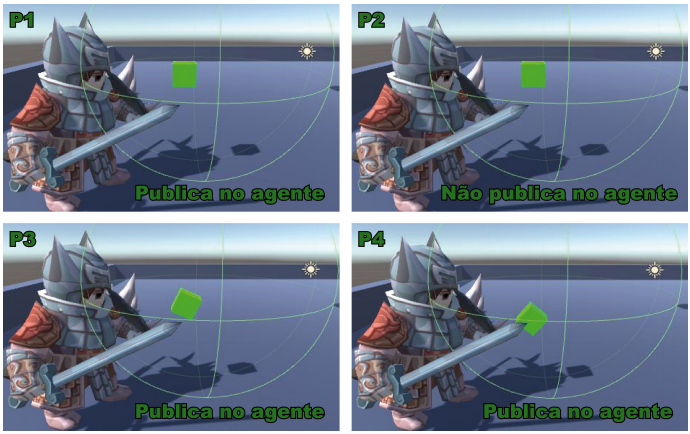

Figura 14 – Exemplo de Publicação na Mudança de Informação
Fonte: O Autor

ficos em conjunto, verifica-se que quando um sensor recebe um número de percepções muito maior que a massa do mesmo, o sensor tende a manter sua prioridade alta.

Mantendo a linha de testes na estratégia repentina, pode-se fazer o questionamento: "Qual é a influência de uma massa grande quando um sensor não tem muitas percepções?". A massa neste teste foi colocada com um valor de 80, e o sensor de visão perto foi normalizado. É possível verificar com os gráficos das figuras 17 e 18, que os sensores mantiveram na maior parte do tempo suas prioridades no mínimo. Este comportamento demonstra que, utilizando uma massa grande, os sensores tendem a manter suas prioridades baixas.

Os testes apresentados neste capítulo, tiveram como objetivo validar o modelo proposto e diferentes estratégias de priorização. Como resultado, pode-se perceber que o modelo funcionou corretamente, e as estratégias implementadas, funcionaram de acordo com suas definições apresentadas no capítulo 4. O raciocínio de sensores demonstrou-se como um bom suporte para o agente, evitando o processamento desnecessário de informações já conhecidas. Vale ressaltar também que, além da validação, através dos testes foi possível verificar possíveis pontos de melhoria com relação ao agente Sigon, que podem ser trabalhados no futuro.

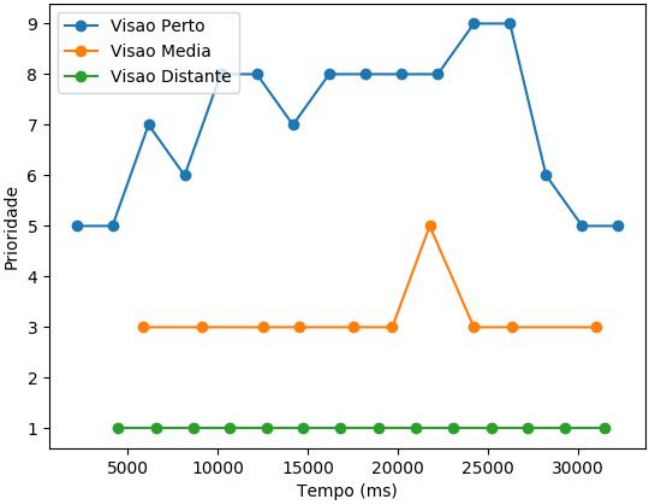

Figura 15 – Prioridades com foco no Sensor de Visão Perto
Fonte: O Autor

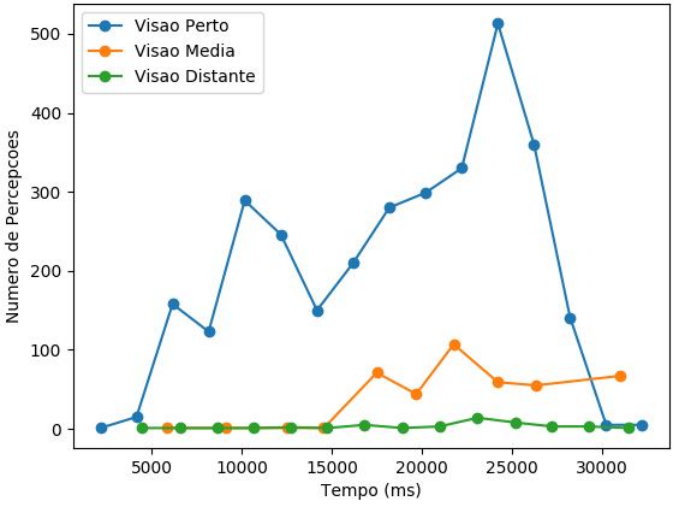

Figura 16 – Número de Percepções com foco no Sensor de Visão Perto
Fonte: O Autor

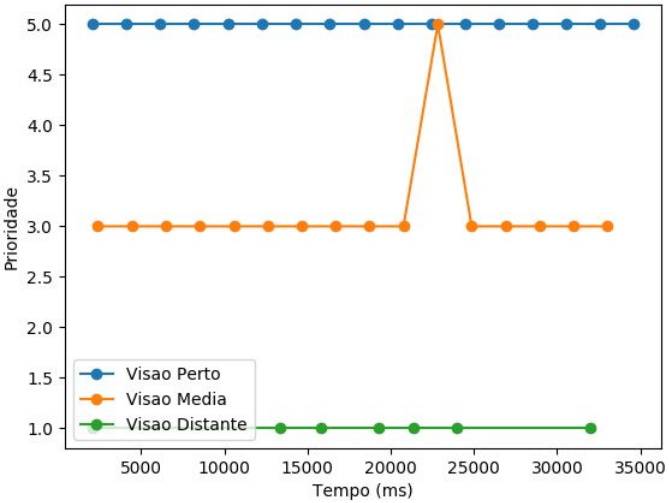

Figura 17 – Prioridades com Sensores de Massa 80
Fonte: O Autor

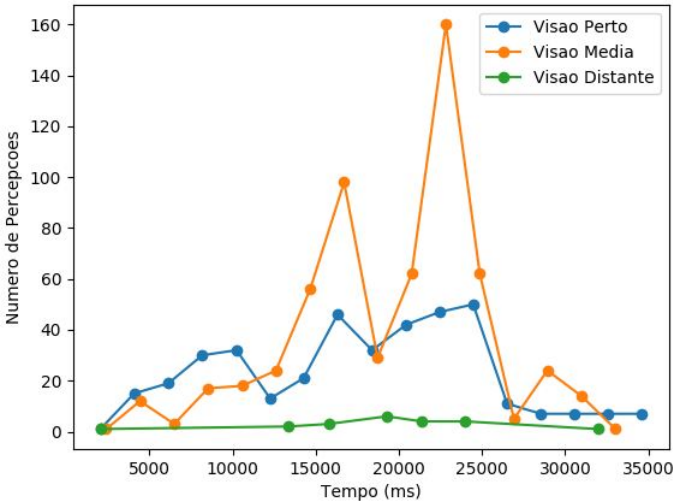

Figura 18 – Número de Percepções com Sensores de Massa 80
Fonte: O Autor

6 CONSIDERAÇÕES FINAIS

Este trabalho inicialmente apresentou no capítulo 1, a necessidade atual de novas pesquisas na área de percepção ativa em agentes. No capítulo 2 foi feito o embasamento necessário sobre agentes, percepção ativa e ambientes virtuais para a realização do trabalho. Os trabalhos correlatos no capítulo 3 demonstraram três trabalhos realizados na área de agentes, onde o modelo de agentes proposto pelo trabalho correlato de Gelaim (2016), serviu como base para este projeto. O capítulo 4 apresentou o modelo proposto com duas estratégias de priorização de sensores, e no capítulo 5 foram realizados testes para validar o comportamento.

Através dos testes realizados, é possível concluir que este trabalho cumpre o objetivo por ele proposto, apresentando um modelo de raciocínio sobre sensores capaz de priorizar os sensores por ele criados. Foram apresentadas duas estratégias possíveis para a priorização dos sensores, sendo que o programador pode optar também, se preferir, por criar sua própria estratégia dentro do modelo. O modelo também da suporte para a criação de outros sensores, podendo ser variações de sensores já existentes, como de visão.

Por consequência do funcionamento do controle de sensores, é possível que o agente processe menos informações desnecessárias, pois é feita a filtragem de percepções que não são diferentes das que o agente já conhece. Este comportamento faz com que o ambiente possa enviar um número maior de percepções, pelo fato de que o agente não precisa processar todas, assim não afogando a conexão.

O trabalho aqui apresentado, também contempla a sugestão de trabalho futuro feita por Prandi (2017), sobre o modo que a percepção chega ao agente. É sugerido que o agente não pergunte dados específicos do ambiente, e sim, apenas receba o que está em seu campo de visão para que possa interpretar. Com a implementação de sensores realizada neste trabalho, que enviam o tempo todo percepções que entram em seus campos de atuação, este objetivo foi cumprido.

6.1 TRABALHOS FUTUROS

Uma primeira sugestão de trabalho futuro é aprimorar o tempo de execução do ciclo de raciocínio do agente Sigon, após a publicação de uma percepção. Quanto mais rápido este raciocínio ocorrer, menor

pode ser a limitação de envio de percepções do ambiente. Vale ressaltar que, quanto mais sensores no agente, maior o impacto desse raciocínio, pois no agente Sigon, a publicação de percepções dos sensores é síncrona. Pode ser interessante além de aprimorar o tempo, fazer com que a publicação seja assíncrona.

Outra sugestão para o futuro, é verificar a possibilidade de utilizar o protocolo UDP ao invés do TCP para a comunicação do agente com um ambiente virtual. Considerando que o protocolo UDP não possui todas as garantias do protocolo TCP, a comunicação não deve travar, porém percepções podem ser perdidas por conta disto. Podem existir cenários onde não é necessário garantir que todas as percepções enviadas sejam recebidas, sendo portanto, válido avaliar os prós e contras desta abordagem.

REFERÊNCIAS

ALOIMONOS, J. Purposive and qualitative active vision. *Proceedings of 10th IEEE International Conference on Pattern Recognition*, v. 1, p. 346–360, 1990.

ALOIMONOS, J.; WEISS, I.; BANDYOPADHYAY, A. Active vision. *International Journal of Computer Vision*, v. 1, n. 4, p. 333–356, Jan 1988. ISSN 1573-1405. <https://doi.org/10.1007/BF00133571>.

ALOIMONOS, Y. *Active Perception*. Taylor & Francis, 2013. (Computer Vision Series). ISBN 9781134776092. <https://books.google.com.br/books?id=C64kxWgRvQEC>.

BAJCSY, R. Active perception. *Proceedings of the IEEE*, v. 76, n. 8, p. 966–1005, Aug 1988. ISSN 0018-9219.

BAJCSY, R.; ALOIMONOS, Y.; TSOTSOS, J. K. Revisiting active perception. 2017. <https://link.springer.com/article/10.1007/s10514-017-9615-3citeas>. Acessado em 25/10/2017.

BRENNER, W.; ZARNEKOW, R.; WITTIG, H. *Intelligent Software Agents: Foundations and Applications*. Springer, 1998. ISBN 9783540634119. <https://books.google.com.br/books?id=xcHaAAAAMAAJ>.

BURDEA, G.; COIFFET, P. *Virtual Reality Technology*. Wiley, 2017. (Wiley - IEEE). ISBN 9781119485728. <https://books.google.com.br/books?id=hMQ8DwAAQBAJ>.

CASALI, A.; GODO, L.; SIERRA, C. *Graded BDI Models for Agent Architectures*. [S.l.: s.n.], 2005. 126-143 p.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: MÜLLER, J. P.; WOOLDRIDGE, M. J.; JENNINGS, N. R. (Ed.). *Intelligent Agents III Agent Theories, Architectures, and Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. p. 21–35. ISBN 978-3-540-68057-4.

GELAIM, T. Ângelo. *Modelo de Agentes E-BDI Integrando Confiança Baseado em Sistemas Multi-Contexto*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, 2016.

GELAIM, T. Ângelo et al. Sigon: A multi-context system framework for intelligent agents. *Expert Systems with Applications*, 2018. ISSN 0957-4174. <http://www.sciencedirect.com/science/article/pii/S0957417418307000>.

GRONDIN, S. Timing and time perception: A review of recent behavioral and neuroscience findings and theoretical directions. *Attention, Perception, & Psychophysics*, v. 72, n. 3, p. 561–582, Apr 2010. ISSN 1943-393X. <https://doi.org/10.3758/APP.72.3.561>.

KATAYAMA, K. et al. Evacuation guidance support using cooperative agent-based iot devices. In: *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*. [S.l.: s.n.], 2017. p. 1–2.

KETENCI, U.-G. et al. Improved road crossing behavior with active perception approach. 01 2012.

LAUKKANEN, S. et al. Adding intelligence to virtual reality. 2004. <http://www.frontiersinai.com/ecai/ecai2004/ecai04/pdf/Laukkanen.pdf>. Acessado em 27/10/2017.

LIU, L. et al. Interactive robots as social partner for communication care. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2014. p. 2231–2236. ISSN 1050-4729.

OIJEN, J. van; DIGNUM, F. Scalable perception for bdi-agents embodied in virtual environments. In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. [S.l.: s.n.], 2011. v. 2, p. 46–53.

PARISI, T. *Learning Virtual Reality: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile*. O'Reilly Media, 2015. ISBN 9781491922804. <https://books.google.com.br/books?id=bXvPCgAAQBAJ>.

PRANDI, H.

Uma Arquitetura para a Atuação de Agentes Inteligentes — Universidade Federal de Santa Catarina, Florianópolis, 2017.

RAO, A.; GEORGEFF, M. P. Bdi agents: From theory to practice. 2000. <http://www.ppgia.pucpr.br/ fabricio/ftp/Aulas/Mestrado/AS/Artigos-Apresentacoes/BDI-Agents/rao95bdi.pdf>. Acessado em 25/10/2017.

RASOULI, A.; TSOTSOS, J. K. Visual saliency improves autonomous visual search. *2014 Canadian Conference on Computer and Robot Vision (CRV)*, p. 111–118, 2014.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594.

TAKEDA, H. et al. Modeling design processes. *AI Mag.*, American Association for Artificial Intelligence, Menlo Park, CA, USA, v. 11, n. 4, p. 37–48, out. 1990. ISSN 0738-4602.

WOOLDRIDGE, M. Intelligent agents: The key concepts. 2002. Acessado em 25/10/2017.

APÊNDICE A – Artigo

Um modelo de raciocínio sobre sensores para agentes Sigon em ambientes de realidade virtual

Giancarlo Souza de Freitas[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brazil

giancarlo.souza@grad.ufsc.br

Abstract. *The need for automation of tasks and simulation of human beings behavior in dynamic environments are proportionally growing with the technological advance. Technological challenges that are beginning to emerge are going beyond human capacity, due to the great complexity and urgency of their resolutions. Because of this, intelligent agents capable of simulating human behavior are an alternative. For a better simulation, there is the concept of active perception, which allows the agent to focus its perception on the most relevant elements in an environment, depending on its objectives.*
This article has the purpose to explore the concepts of active perception on the implementation of agents in a virtual reality environment, more specifically, focusing on the agent's sensors reasoning. Sensors reasoning allows the agent to prioritize sensors that perceive more changes in the environment.
The proposed model enables the sensors creation and utilization on agents. The sensors collect perceptions from the environment, and they can, according to their strategies, have their priorities changed. Model's evaluation is provided through the implementation of two different prioritization strategies, both analyzed in an environment with dynamic objects, for better visualization.

Resumo. *A necessidade de automatização de tarefas e simulação do comportamento de seres humanos em ambientes dinâmicos está crescendo proporcionalmente ao avanço tecnológico. Os desafios tecnológicos que começam a surgir estão indo além da capacidade humana, devido à grande complexidade e urgência de suas resoluções. Em razão disso, agentes inteligentes capazes de simular o comportamento humano são uma alternativa. Para uma melhor simulação, existe o conceito de percepção ativa, o qual permite que o agente foque sua percepção nos elementos mais relevantes em um ambiente, dependendo de seus objetivos.*
Este artigo tem como objetivo explorar os conceitos de percepção ativa na implementação de agentes em um ambiente de realidade virtual, mais especificamente, focando no raciocínio sobre os sensores do agente. O raciocínio dos sensores permite que o agente priorize os sensores que percebem mais mudanças ocorrendo no ambiente.
O modelo proposto possibilita a criação e utilização de sensores em agentes. Os sensores coletam percepções do ambiente, e podem, de acordo com sua estratégia, ter suas prioridades alteradas. A validação do modelo é fornecida através da implementação de duas estratégias de priorização diferentes, analisadas em um ambiente com objetos dinâmicos, para melhor visualização.

1. Introdução

A evolução tecnológica tem, cada vez mais, fornecido à humanidade novas formas de visualizar e interagir com o mundo, seja de forma física ou virtual. Com isso, novas necessidades e desafios continuam surgindo. Estes novos desafios e sistemas a serem construídos começam a ser de tamanha complexidade e também urgência de ações rápidas, que a resolução humana não mais consegue resolver em um intervalo de tempo adequado ao problema. A abordagem de sistemas multiagentes, onde cada agente é um sistema de computador autônomo, pode auxiliar nessa nova era tecnológica.

Situações críticas onde a decisão necessita ser rápida e correta, como em situações de desastres naturais, podem utilizar de agentes inteligentes para reduzir danos. Ao utilizar de conceitos de percepção ativa um agente pode perceber o ambiente e priorizar as percepções que recebe, podendo se mostrar um bom aliado em todo o tipo de situação.

Neste artigo, é apresentado um modelo de raciocínio sobre sensores de um agente, utilizando como base o modelo de agente proposto por *Gelaim 2016* [Ângelo Gelaim 2016], com um enfoque maior no contexto de comunicação. Este modelo de agente sai da teoria para a prática, com a denominação de agente Sigon. Com a utilização de um raciocínio sobre sensores é possível obter um suporte para que o agente possa perceber de maneira mais inteligente, priorizando os sensores mais utilizados. Também são implementadas duas estratégias de priorização para os sensores, que podem ser utilizadas de acordo com a necessidade do problema que o agente deve resolver.

O artigo está organizado da seguinte forma: na seção 2 é apresentado o contexto de percepções em agentes. Na seção 3 são apresentadas as políticas de percepções desenvolvidas para agentes Sigon. Na seção 4 são descritos os testes realizados. Por fim, na seção 5 são apresentadas as conclusões e trabalhos futuros.

2. Percepção em Agentes

A maioria das pesquisas passadas e presentes em percepção de máquina envolveu análise de amostras passivas de dados (imagens). Porém como *Aloimonos et al. 1988* [Aloimonos et al. 1988] já haviam mencionado, a percepção humana não é passiva, e sim ativa. Quando humanos veem e compreendem, eles observam ativamente. *Bajcsy et al. 2017* [Bajcsy et al. 2017] definem então que, um agente é um perceptor ativo se ele sabe porquê ele deseja sentir, e então escolhe o que perceber, e determina como, onde e quando alcançar essa percepção. *Bajcsy et al. 2017* ainda definem *onde*, *o quê*, *como*, *quando* e *porquê* como elementos da percepção ativa. O fator chave é o elemento *porquê*, pois quando é definido, o agente pode utilizar de um raciocínio de sensores, por exemplo, para priorizar as percepções ou o sensor que possua maior relação com este *porquê*.

No trabalho de *Ketenci et al. 2012* [Ketenci et al. 2012], por exemplo, é desenvolvido um modelo de percepção ativa para simular o comportamento de cruzamento de estradas dos motoristas. O objetivo era simular como seres humanos agiriam com uma alteração na estrutura de uma estrada para melhoria do tráfego. Para isto, a percepção do agente deve ser limitada, para simular melhor o comportamento humano.

A implementação da percepção ativa no contexto do trabalho de *Ketenci, 2012* é feita utilizando três principais conceitos, sendo eles foco, classificação de percepção em

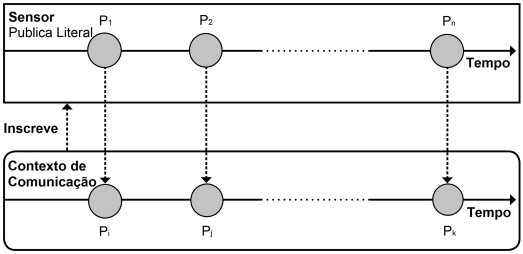

Figura 1. Representação Adaptada do Modelo de Sensores

respeito a relevância e percepção de recursos limitada. O foco é o domínio de interesse do agente em um espaço sensorial, ou seja, o espaço que o agente irá se concentrar considerando suas intenções. Já a classificação de percepções é exatamente o agente verificar qual percepção possui relevância com a sua intenção atual, e assim classificá-la como prioritária, a qual pode em um ambiente dinâmico, perder sua prioridade para outra, devido a constante mudança de cenário.

Para tentar resolver a complexidade de interações entre agentes, é considerado o comportamento do motorista no contexto de interseção, a qual um motorista seleciona um número de "jogadores", sendo estes os outros motoristas, quando se aproxima de um cruzamento para decidir se anda ou para. *Ketenci, 2012* implementaram uma percepção ativa limitada para esta escolha de "jogadores", que agora depende do contexto do tráfego e de como este contexto é percebido pelo agente.

Ketenci, 2012 [Ketenci et al. 2012] apresenta um modelo de agente inspirado em BDI que integra um modelo computacional de emoções. A abordagem utilizada para o modelo é baseada em um Sistema Multi-Contexto. No modelo proposto, as crenças são graduadas com graus de certeza do agente sobre elas e podem ser rotuladas como especiais. As crenças especiais são chamadas de julgamentos e são utilizadas para intermediar a relação entre confiança e emoções. O modelo é formalmente definido como:

3. Políticas de percepção em agentes Sigon

Nesta seção, é apresentado como ocorre a percepção do agente Sigon, o qual está situado em um ambiente de realidade virtual criado na ferramenta Unity. Na arquitetura de um agente Sigon, o contexto de comunicação é responsável por fazer a interface entre o agente e o ambiente em que está situado. Este contexto possui em sua implementação, sensores e atuadores para que o agente possa perceber o ambiente e atuar no mesmo. Os sensores são responsáveis por receber os dados oriundos do ambiente, e em sua estrutura existe um identificador e uma implementação do mesmo.

$$sensor : SENSOR`('identificador`,'implementacao`)'`.'$$

A figura 1 demonstra a relação dos sensores com o contexto de comunicação. O modelo de sensor possui um publicador literal, o qual vai publicar esses literais, que são strings representando as percepções do agente, no contexto de comunicação. É de

responsabilidade do desenvolvedor a implementação dos sensores e a publicação dos literais, que deve estar de acordo com a especificação da linguagem usada do contexto de comunicação do agente Sigon.

A comunicação entre o agente Sigon e o ambiente Unity foi realizada através de uma arquitetura cliente-servidor com comunicação via sockets, onde cada um dos sensores do agente Sigon é um servidor com uma porta específica. Estes sensores podem enviar mensagens contendo todo o tipo de informação coletada do ambiente para o agente. Um exemplo de mensagem enviada é o nome de um objeto específico no ambiente junto com as suas coordenadas em relação ao mesmo.

$$msg = position(\text{“}objeto, x, y, z'')$$

As percepções oriundas do ambiente chegam o tempo todo nos sensores do agente, porém é possível que um sensor fique sem receber novas percepções por um certo período de tempo. Para estes casos que não ocorre variância das informações, faz sentido que o sensor diminua de prioridade com o passar do tempo, fazendo com que o agente tenha um foco maior nos sensores que estão sendo mais utilizados.

Para que o agente receba, na maior parte do tempo, mais informações dos sensores que estão sendo mais utilizados, é utilizado uma priorização de sensores. Cada sensor do agente é uma thread Java e um socket, começando com um valor de prioridade 5 e variando esse valor em um intervalo de 1 à 10, de acordo com a necessidade e implementação.

O modelo de sensores do agente Sigon, permite criar tantos sensores quanto necessário para determinada função. Os sensores específicos de cada função do agente, são agrupados de tal forma que, o sensor de maior área contém os sensores de menor área, percebendo objetos que não estão na área dos sensores menores.

$$V = \sum_{i=1}^{n} S_{i-1} \subseteq S_i$$

A figura 2, demonstra um exemplo de como poderia ser criado sensores para a visão do agente. A função de visão, neste exemplo, é separada em 3 sensores diferentes, cada um representando uma nível de distância diferente entre o agente e um objeto a ser percebido.

A priorização de sensores pode utilizar de políticas diferentes, de acordo com a necessidade de contexto, sendo de responsabilidade do programador a escolha ou a criação da política que melhor o atende. É possível implementar diferentes políticas de priorização, conforme a necessidade dos agentes.

3.1. Política gradativa de percepções

A primeira política de priorização é denominada de política gradativa. Essa política leva em consideração as percepções diferentes que chegam em um sensor. Ao armazenar, até certo tempo, percepções que chegaram em um ciclo anterior de raciocínio do sensor, é possível checar regularmente as percepções novas com as antigas, diminuindo ou aumentando sua prioridade de acordo com a conclusão. As percepções antigas que o sensor conhece devem possuir um tempo limite para ficarem no mesmo, para que essa checagem

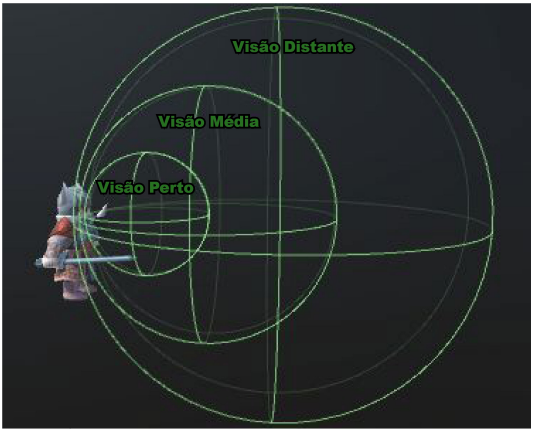

Figura 2. Exemplo da Função Visão do Agente

faça sentido, pois caso contrário, o sensor poderia em certo momento parar de receber a percepção de um objeto qualquer e ainda considerar que não houve mudança no ambiente.

Aprofundando um pouco no nível de implementação, existem dois HashSets de percepções, um para as percepções que estão chegando, e outro para as percepções antigas. A cada momento de checagem para alteração de prioridade, o HashSet de percepções antigas recebe as percepções que estavam no outro HashSet, para que seja verificado se houveram mudanças. Como o comportamento do HashSet não permite que exista mais de um elemento igual dentro de si, caso o HashSet de percepções antigas não mude, significa que nenhuma percepção nova chegou desde a última checagem. A prioridade então, é aumentada caso tenha mudanças, e caso não tenha, a prioridade do sensor é diminuída. Isto vale para todos os sensores.

3.2. Política repentina de percepções

A segunda política de priorização de sensores é chamada de política repentina. Essa política foca mais na quantidade de percepções que chegaram em dado momento, modificando a prioridade do sensor de acordo com a seguinte fórmula:

$$P = (1 - \frac{m}{F}) \cdot 10$$

Onde P é a prioridade do sensor, F é a força que está sendo colocada no sensor, neste caso o número de percepções, e m é o ponto de referência configurável, denominado como massa. Quanto maior a massa de um sensor, menor será a tolerância com o número de percepções, fazendo com que a prioridade se mantenha baixa. Por outro lado, um sensor que receba um número de percepções maior que a sua massa, terá sua prioridade elevada de acordo. Se o resultado da equação der negativo, significa que o número das percepções é menor que a massa, portanto a prioridade do sensor será mínima.

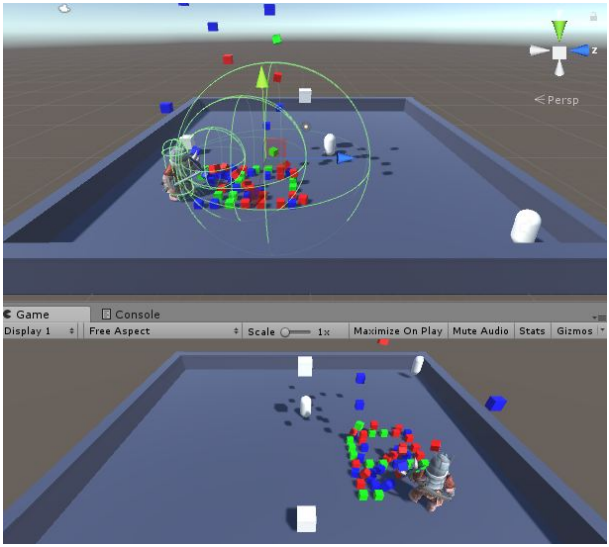

Figura 3. Cenário de Testes

A implementação dessa política utiliza do HashSet de percepções para guardar o número de percepções até o momento de checagem. Quando chega o momento de modificar a prioridade do sensor, a fórmula é aplicada, utilizando a massa configurada para o contexto e a quantidade de percepções que chegaram. Após a modificação de sua prioridade, o sensor esvazia o HashSet utilizado, para receber a nova leva de percepções.

4. Avaliação

Para a verificação do funcionamento da priorização de sensores, foi criado um cenário simples de testes para o agente, que pode ser visualizado a partir da figura 3. A comunicação do ambiente Unity com o agente Sigon foi feita através de uma conexão TCP. O cenário criado consiste em objetos que o agente pode perceber, como chão, paredes e blocos estáticos, podendo em suas percepções verificar por exemplo, a posição de tais objetos em relação aos eixos x, y e z, dentro do ambiente virtual 3D. Além dos objetos fixos no cenário, foi implementada a criação de blocos dinâmicos em tempo de execução, os quais caem no cenário em posições aleatórias à frente do agente. Os sensores perceberão cada posição de um bloco em movimento como uma nova percepção sobre o mesmo, e quando este bloco estiver em uma posição de repouso, os sensores continuarão percebendo o objeto, mas conseguem verificar que não é uma percepção nova.

O primeiro teste realizado neste cenário foi feito utilizando a política gradativa de priorização e desconsiderando a publicação de percepções para o raciocínio do agente, tendo foco apenas no raciocínio dos sensores para a priorização. Para melhor visualização, foram coletados dados referentes aos sensores de visão perto, média e distante, os quais tem um intervalo de prioridade de 5 a 10, 3 a 8 e 1 a 6, respectivamente, e

colocados nos gráficos das figuras 4 e 5.

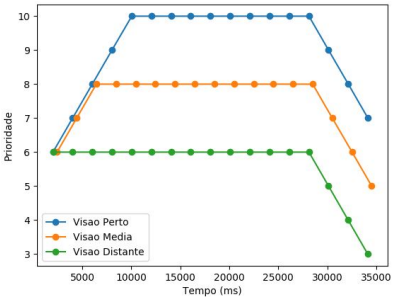

Figura 4. Prioridades dos Sensores

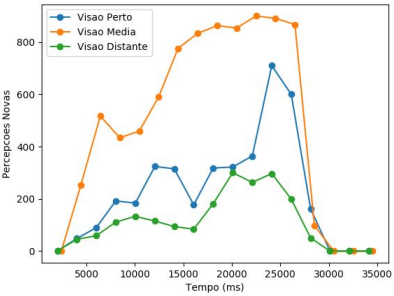

Figura 5. Novas Percepções dos Sensores

O segundo teste realizado no cenário, levou em consideração também a publicação das percepções para o raciocínio do agente, além da priorização dos sensores com a política gradativa. Foi então verificado que, o tempo de raciocínio do agente não acompanhava o tempo de envio de percepções do ambiente. Isto se torna um problema devido as políticas do protocolo TCP, que suspende a comunicação quando existe um número elevado de mensagens ainda não processadas, para garantir a entrega de todas as mensagens. Os gráficos das figuras 6 e 7 demonstram o tempo entre o envio de percepções e o tempo de raciocínio do agente sobre uma percepção, respectivamente.

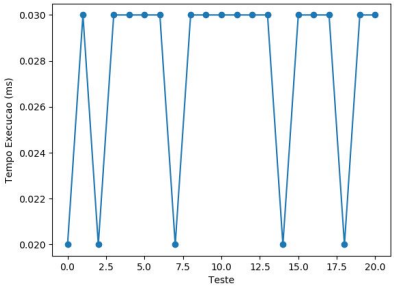

Figura 6. Tempo de Envio de Percepções

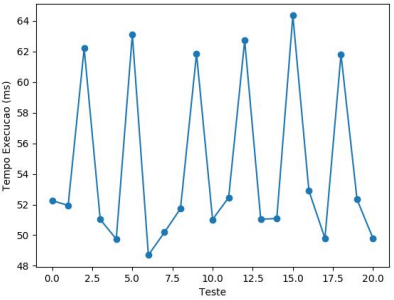

Figura 7. Tempo de Publicação no Agente

Em comparação com seres humanos, que possuem uma discriminação de tempo de 300 milissegundos para mais, o agente já possui um raciocínio bem rápido, portanto foi decidido limitar as percepções do ambiente neste primeiro momento. Foi determinado que o envio de percepções só pode acontecer a cada 50 milissegundos, o qual é aproximadamente o tempo médio do ciclo de raciocínio do agente, e em um intervalo de 0,5 milissegundos até que o tempo reinicie. Este intervalo foi feito para que o ambiente possa enviar o máximo de percepções possíveis em um ciclo de envio, sem sobrecarregar a conexão TCP.

Com o intuito de tornar o raciocínio dos sensores um suporte ainda melhor para o agente, e um processo menos custoso, foi criado também uma lógica em relação a atualização de percepções sobre um objeto. O agente Sigon armazena as informações sobre determinado objeto, portanto, se o ambiente está enviando sempre as mesmas informações de um objeto para um sensor, só faz sentido publicar no agente se alguma mudança ocorreu no objeto. Esta alteração nos sensores permite que o agente processe menos informações desnecessárias, possibilitando maior tolerância ao envio de percepções pelo ambiente, pois não perde tempo recebendo a publicação de algo que já conhece.

O terceiro teste realizado, foi feito para avaliar o funcionamento da política repentina de priorização. Este teste também responde o seguinte questionamento sobre essa política: "Qual é a influência da massa quando o agente possui um número de percepções muito maior que a mesma?". Para uma melhor visualização deste comportamento, foi utilizada uma massa de tamanho 50 para todos os sensores, e feito com que um sensor específico recebesse mais percepções que os outros. O sensor escolhido para receber mais percepções neste caso foi o sensor de visão perto. Os gráficos das figuras 8 e 9, demonstram o comportamento dos sensores neste teste.

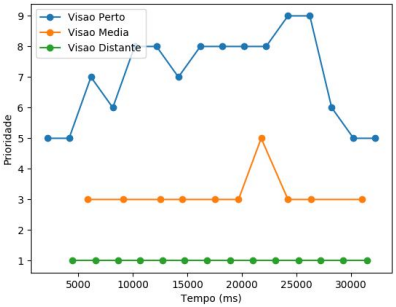

Figura 8. Prioridades com foco no Sensor de Visão Perto

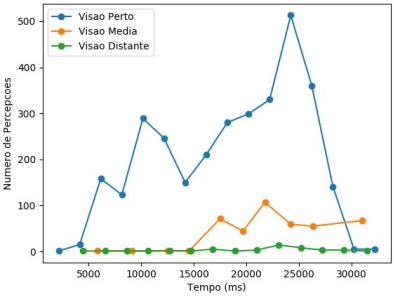

Figura 9. Número de Percepções com foco no Sensor de Visão Perto

Mantendo a linha de testes na estratégia repentina, pode-se fazer o questionamento: "Qual é a influência de uma massa grande quando um sensor não tem muitas percepções?". A massa neste teste foi colocada com um valor de 80, e o sensor de visão perto foi normalizado. É possível verificar com os gráficos das figuras 10 e 11, que os sensores mantiveram na maior parte do tempo suas prioridades no mínimo. Este comportamento demonstra que, utilizando uma massa grande, os sensores tendem a manter suas prioridades baixas.

5. Conclusão

Neste artigo são apresentadas políticas para sensores de agentes situados em ambientes dinâmicos de realidade virtual. Um modelo para raciocínio sobre sensores em agentes é também apresentado, possibilitando o uso das políticas. O modelo também da suporte para a criação de outros sensores, podendo ser variações de sensores já existentes, como de visão.

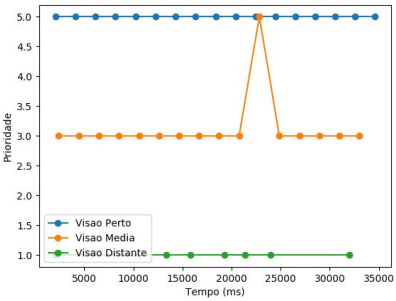

Figura 10. Prioridades com Sensores de Massa 80

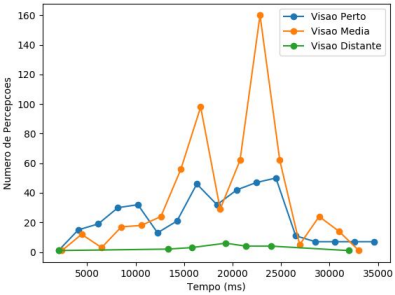

Figura 11. Número de Percepções com Sensores de Massa 80

Por consequência do funcionamento do controle de sensores, é possível que o agente processe menos informações desnecessárias, pois é feita a filtragem de percepções que não são diferentes das que o agente já conhece. Este comportamento faz com que o ambiente possa enviar um número maior de percepções, pelo fato de que o agente não precisa processar todas, assim não afogando a conexão.

5.1. Trabalhos futuros

Uma primeira sugestão de trabalho futuro é aprimorar o tempo de execução do ciclo de raciocínio do agente Sigon, após a publicação de uma percepção. Quanto mais rápido este raciocínio ocorrer, menor pode ser a limitação de envio de percepções do ambiente. Vale ressaltar que, quanto mais sensores no agente, maior o impacto desse raciocínio, pois no agente Sigon, a publicação de percepções dos sensores é síncrona. Pode ser interessante além de aprimorar o tempo, fazer com que a publicação seja assíncrona.

Outra sugestão para o futuro, é verificar a possibilidade de utilizar o protocolo UDP ao invés do TCP para a comunicação do agente com um ambiente virtual. Considerando que o protocolo UDP não possui todas as garantias do protocolo TCP, a comunicação não deve travar, porém percepções podem ser perdidas por conta disto. Podem existir cenários onde não é necessário garantir que todas as percepções enviadas sejam recebidas, sendo portanto, válido avaliar os prós e contras desta abordagem.

Referências

Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *International Journal of Computer Vision*, 1(4):333–356.

Bajcsy, R., Aloimonos, Y., and Tsotsos, J. K. (2017). Revisiting active perception.

Ketenci, U.-G., Auberlet, J.-M., Brémond, R., and Grislin-Le Strugeon, E. (2012). Improved road crossing behavior with active perception approach.

Ângelo Gelaim, T. (2016). Modelo de agentes e-bdi integrando confiança baseado em sistemas multi-contexto. Master's thesis, Universidade Federal de Santa Catarina, Florianópolis.

APÊNDICE B – Código Fonte

O código produzido neste trabalho utilizou do framework Sigon e da ferramenta Unity. Devido a grande quantidade de códigos já presentes nessas tecnologias, este apêndice possui apenas os códigos criados para este trabalho. É possível acessar o código completo através dos links https://github.com/giancarlosouza/UnityGame e https://github.com/giancarlosouza/sigon-examples.

B.1 UNITY

B.1.1 Arquivo Assets/Scripts/CharController.cs

```
1  public class CharController : MonoBehaviour {
2
3    public float speed;
4    public float turnSpeed;
5
6    public string conHost = "127.0.0.1";
7    public int conPort = 2600;
8    public TcpListener server;
9    public TcpClient mySocket;
10
11   public Thread mThread;
12   private Thread threadTimer;
13   private bool running = true;
14
15   public ClientVisionFar visionFar;
16   public ClientVisionMed visionMed;
17   public ClientVisionClose visionClose;
18   public ClientTact tact;
19   public ClientHearing hearing;
20
21   void Start(){
22     speed = 10;
23     turnSpeed = 2;
24     mThread = new Thread (new ThreadStart(Listen));
25     mThread.IsBackground = true;
26     mThread.Start();
27     threadTimer = new Thread ((new
       ↪  ThreadStart(timeController)));
28     threadTimer.Start ();
```

```
29    }
30
31    public void FixedUpdate () {
32
33      CharacterController controller =
        ↪  GetComponent<CharacterController> ();
34      Animation an = GetComponent<Animation> ();
35
36      Vector3 move = new Vector3 (0, 0, Input.GetAxis
        ↪  ("Vertical"));
37
38      if (Input.GetAxis ("Vertical") != 0) {
39        an.Play ("run");
40      } else {
41        an.Play ("idle");
42      }
43
44      controller.transform.Rotate (0, Input.GetAxis
        ↪  ("Horizontal") * turnSpeed, 0);
45      move = transform.TransformDirection (move);
46      controller.SimpleMove (move * speed);
47
48    }
49
50    public void Listen(){
51      try {
52        // Create listener on localhost port 2600.
53        server = new
          ↪  TcpListener(IPAddress.Parse("127.0.0.1"),
          ↪  conPort);
54        server.Start();
55        Debug.Log("Server is listening");
56        Byte[] bytes = new Byte[1024];
57        while (running) {
58          using (mySocket = server.AcceptTcpClient()) {
59            // Get a stream object for reading
60            using (NetworkStream stream =
              ↪  mySocket.GetStream()) {
61              int length;
62              // Read incomming stream into byte arrary.
```

```
63              while ((length = stream.Read(bytes, 0,
                ↪  bytes.Length)) != 0) {
64                var incommingData = new byte[length];
65                Array.Copy(bytes, 0, incommingData, 0,
                  ↪  length);
66                // Convert byte array to string message.
67                string clientMessage =
                  ↪  Encoding.ASCII.GetString(incommingData);
68                Debug.Log("client message received as: " +
                  ↪  clientMessage);
69              }
70            }
71          }
72        }
73      }
74      catch (SocketException socketException) {
75        Debug.Log("SocketException " +
          ↪  socketException.ToString());
76      }
77    }
78
79    public void timeController(){
80      while(running){
81        visionFar.resetTimer ();
82        visionMed.resetTimer ();
83        visionClose.resetTimer ();
84        tact.resetTimer ();
85        hearing.resetTimer ();
86      }
87    }
88
89    void OnApplicationQuit() { // stop listening thread
90      running = false;
91      threadTimer.Join(500);
92      mThread.Join(500);
93    }
94  }
```

B.1.2 Arquivo Assets/Scripts/ClientHearing.cs

```
1  public class ClientHearing : MonoBehaviour {
2
3   public TcpClient mySocket;
4
5   public string conHost = "127.0.0.1";
6   public int conPort = 2405;
7
8   public NetworkStream theStream;
9   public StreamWriter theWriter;
10
11  public bool socketReady = false;
12  private float gameTime;
13  private float timeNow;
14  public float timeToWaitBeforeSendPerception = 0.05f;
15  public float timeRangeToSendPerception = 0.0005f;
16  private float sendCycleTime;
17
18  void Start () {
19   gameTime = Time.realtimeSinceStartup;
20   sendCycleTime = timeToWaitBeforeSendPerception +
     ↪  timeRangeToSendPerception;
21   try {
22    mySocket = new TcpClient();
23    var result =
      ↪  mySocket.BeginConnect(conHost,conPort,null, null);
24    var success = result.AsyncWaitHandle.WaitOne(500);
25
26    if (!success)
27    {
28     throw new Exception("Failed to connect.");
29     socketReady = false;
30    }else{
31
32     theStream = mySocket.GetStream();
33     theWriter = new StreamWriter(theStream);
34     socketReady = true;
35    }
36
37   }
```

```
38      catch (Exception e) {
39       Debug.Log("Socket error:" + e);
40      }
41     }
42
43     void OnTriggerStay(Collider other){
44      if(other.CompareTag ("sound")){
45       audioTransformations (other);
46      }
47      timeNow = Time.realtimeSinceStartup;
48     }
49
50     void audioTransformations(Collider other){
51      AudioSource audioSource =
        ↪  other.gameObject.GetComponent<AudioSource>();
52      float actualVolume = audioSource.volume;
53
54      theWriter.Write ("p(" + other.transform.name + ","
55       + other.transform.position.x + ","
56       + other.transform.position.y + ","
57       + other.transform.position.z + ","
58       + actualVolume + ").,"
59       + Time.realtimeSinceStartup.ToString () + "\n");
60      theWriter.Flush ();
61
62      float distance = Vector3.Distance
        ↪  (this.gameObject.transform.position,
        ↪  other.transform.position);
63      audioSource.volume = (other.bounds.extents.y - distance)
        ↪  / other.bounds.extents.y;
64     }
65
66     void OnTriggerEnter(Collider other){
67      if (other.CompareTag ("surprise")) {
68       AudioSource audioSource =
         ↪  other.gameObject.GetComponent<AudioSource>();
69       audioSource.Play ();
70       audioSource.volume = 1f;
71       print ("Danger");
72      }
73     }
```

```
74
75  public void resetTimer(){
76   if(timeNow - gameTime > sendCycleTime){
77    gameTime = timeNow;
78   }
79  }
80 }
```

B.1.3 Arquivo Assets/Scripts/ClientTact.cs

```
1  public class ClientTact : MonoBehaviour {
2
3   public TcpClient mySocket;
4
5   public string conHost = "127.0.0.1";
6   public int conPort = 2404;
7
8   public NetworkStream theStream;
9   public StreamWriter theWriter;
10
11  public bool socketReady = false;
12
13  private float gameTime;
14  private float timeNow;
15  public float timeToWaitBeforeSendPerception = 0.05f;
16  public float timeRangeToSendPerception = 0.0005f;
17  private float sendCycleTime;
18
19  void Start () {
20   gameTime = Time.realtimeSinceStartup;
21   sendCycleTime = timeToWaitBeforeSendPerception +
     ↪  timeRangeToSendPerception;
22   try {
23    mySocket = new TcpClient();
24    var result =
      ↪  mySocket.BeginConnect(conHost,conPort,null, null);
25    var success = result.AsyncWaitHandle.WaitOne(500);
26
27    if (!success)
28    {
```

```
29      throw new Exception("Failed to connect.");
30      socketReady = false;
31     }else{
32
33      theStream = mySocket.GetStream();
34      theWriter = new StreamWriter(theStream);
35      socketReady = true;
36     }
37
38    }
39    catch (Exception e) {
40     Debug.Log("Socket error:" + e);
41    }
42   }
43
44   void OnTriggerStay(Collider other){
45    if(!other.name.Equals("Character")){
46     theWriter.Write ("p(" + other.transform.name + ","
47      + other.transform.position.x + ","
48      + other.transform.position.y + ","
49      + other.transform.position.z + ").,"
50      + Time.realtimeSinceStartup.ToString () + "\n");
51     theWriter.Flush ();
52    }
53    timeNow = Time.realtimeSinceStartup;
54   }
55
56   public void resetTimer(){
57    if(timeNow - gameTime > sendCycleTime){
58     gameTime = timeNow;
59    }
60   }
61  }
```

B.1.4 Arquivo Assets/Scripts/ClientVisionClose.cs

```
1  public class ClientVisionClose : MonoBehaviour {
2
3   public TcpClient mySocket;
4
```

```
5   public string conHost = "127.0.0.1";
6   public int conPort = 2403;
7
8   public NetworkStream theStream;
9   public StreamWriter theWriter;
10
11  public bool socketReady = false;
12  private float gameTime;
13  private float timeNow;
14  public float timeToWaitBeforeSendPerception = 0.05f;
15  public float timeRangeToSendPerception = 0.0005f;
16  private float sendCycleTime;
17
18  void Start () {
19   gameTime = Time.realtimeSinceStartup;
20   sendCycleTime = timeToWaitBeforeSendPerception +
     ↪  timeRangeToSendPerception;
21   try {
22    mySocket = new TcpClient();
23    var result =
     ↪  mySocket.BeginConnect(conHost,conPort,null, null);
24    var success = result.AsyncWaitHandle.WaitOne(500);
25
26    if (!success)
27    {
28     throw new Exception("Failed to connect.");
29     socketReady = false;
30    }else{
31
32     theStream = mySocket.GetStream();
33     theWriter = new StreamWriter(theStream);
34     socketReady = true;
35    }
36
37   }
38   catch (Exception e) {
39    Debug.Log("Socket error:" + e);
40   }
41  }
42
43  void OnTriggerStay(Collider other){
```

```
44    if(!other.name.Equals("Character")){
45     if(Time.realtimeSinceStartup - gameTime >
       ↪  timeToWaitBeforeSendPerception){
46      if (!checkOnChildrenIntersection (other)) {
47       theWriter.Write ("p(" + other.transform.name + ","
48        + other.transform.position.x + ","
49        + other.transform.position.y + ","
50        + other.transform.position.z + ").,"
51        + Time.realtimeSinceStartup.ToString () + "\n");
52       theWriter.Flush ();
53      }
54     }
55    }
56    timeNow = Time.realtimeSinceStartup;
57   }
58
59   Boolean checkOnChildrenIntersection(Collider other){
60    if (this.gameObject.transform.childCount != 0) {
61     for(int i = 0; i <
       ↪  this.gameObject.transform.childCount; i++){
62      Bounds childBounds =
        ↪  this.gameObject.transform.GetChild
        ↪  (i).GetComponent<Collider> ().bounds;
63      if(childBounds.Intersects(other.bounds)){
64       return true;
65      }
66     }
67    }
68    return false;
69   }
70
71   public void resetTimer(){
72    if(timeNow - gameTime > sendCycleTime){
73     gameTime = timeNow;
74    }
75   }
76  }
```

B.1.5 Arquivo Assets/Scripts/ClientVisionFar.cs

```
1  public class ClientVisionFar : MonoBehaviour {
2
3   public TcpClient mySocket;
4
5   public string conHost = "127.0.0.1";
6   public int conPort = 2401;
7
8   public NetworkStream theStream;
9   public StreamWriter theWriter;
10
11  public bool socketReady = false;
12  private float gameTime;
13  private float timeNow;
14  public float timeToWaitBeforeSendPerception = 0.05f;
15  public float timeRangeToSendPerception = 0.0005f;
16  private float sendCycleTime;
17
18  void Start () {
19   gameTime = Time.realtimeSinceStartup;
20   sendCycleTime = timeToWaitBeforeSendPerception +
      ↪  timeRangeToSendPerception;
21   try {
22    mySocket = new TcpClient();
23    var result =
      ↪  mySocket.BeginConnect(conHost,conPort,null, null);
24    var success = result.AsyncWaitHandle.WaitOne(500);
25
26    if (!success)
27    {
28     throw new Exception("Failed to connect.");
29     socketReady = false;
30    }else{
31
32     theStream = mySocket.GetStream();
33     theWriter = new StreamWriter(theStream);
34     socketReady = true;
35    }
36
37   }
```

```
38      catch (Exception e) {
39       Debug.Log("Socket error:" + e);
40      }
41     }
42
43     void OnTriggerStay(Collider other){
44      if(!other.name.Equals("Character")){
45       if (Time.realtimeSinceStartup - gameTime >
         ↪  timeToWaitBeforeSendPerception) {
46        if (!checkOnChildrenIntersection (other)) {
47         theWriter.Write ("p(" + other.transform.name + ","
48          + other.transform.position.x + ","
49          + other.transform.position.y + ","
50          + other.transform.position.z + ").,"
51          + Time.realtimeSinceStartup.ToString () + "\n");
52         theWriter.Flush ();
53        }
54       }
55      }
56      timeNow = Time.realtimeSinceStartup;
57     }
58
59     Boolean checkOnChildrenIntersection(Collider other){
60      if (this.gameObject.transform.childCount != 0) {
61       for(int i = 0; i <
         ↪  this.gameObject.transform.childCount; i++){
62        Bounds childBounds =
          ↪  this.gameObject.transform.GetChild
          ↪  (i).GetComponent<Collider> ().bounds;
63        if(childBounds.Intersects(other.bounds)){
64         return true;
65        }
66       }
67      }
68      return false;
69     }
70
71     public void resetTimer(){
72      if(timeNow - gameTime > sendCycleTime){
73       gameTime = timeNow;
74      }
```

```
 }
}
```

B.1.6 Arquivo Assets/Scripts/ClientVisionMed.cs

```
public class ClientVisionMed : MonoBehaviour {

 public TcpClient mySocket;

 public string conHost = "127.0.0.1";
 public int conPort = 2402;

 public NetworkStream theStream;
 public StreamWriter theWriter;

 public bool socketReady = false;
 private float gameTime;
 private float timeNow;
 public float timeToWaitBeforeSendPerception = 0.05f;
 public float timeRangeToSendPerception = 0.0005f;
 private float sendCycleTime;

 void Start () {
  gameTime = Time.realtimeSinceStartup;
  sendCycleTime = timeToWaitBeforeSendPerception +
  ↪  timeRangeToSendPerception;
  try {
   mySocket = new TcpClient();
   var result =
   ↪  mySocket.BeginConnect(conHost,conPort,null, null);
   var success = result.AsyncWaitHandle.WaitOne(500);

   if (!success)
   {
    throw new Exception("Failed to connect.");
    socketReady = false;
   }else{

    theStream = mySocket.GetStream();
    theWriter = new StreamWriter(theStream);
```

```
34       socketReady = true;
35      }
36
37     }
38     catch (Exception e) {
39      Debug.Log("Socket error:" + e);
40     }
41    }
42
43    void OnTriggerStay(Collider other){
44     if(!other.name.Equals("Character")){
45      if(Time.realtimeSinceStartup - gameTime >
        ↪  timeToWaitBeforeSendPerception){
46       if (!checkOnChildrenIntersection (other)) {
47        theWriter.Write("p(" + other.transform.name + ","
48         + other.transform.position.x + ","
49         + other.transform.position.y + ","
50         + other.transform.position.z + ").,"
51         + Time.realtimeSinceStartup.ToString() + "\n");
52        theWriter.Flush();
53       }
54      }
55     }
56     timeNow = Time.realtimeSinceStartup;
57    }
58
59    Boolean checkOnChildrenIntersection(Collider other){
60     if (this.gameObject.transform.childCount != 0) {
61      for(int i = 0; i <
        ↪  this.gameObject.transform.childCount; i++){
62       Bounds childBounds =
         ↪  this.gameObject.transform.GetChild
         ↪  (i).GetComponent<Collider> ().bounds;
63       if(childBounds.Intersects(other.bounds)){
64        return true;
65       }
66      }
67     }
68     return false;
69    }
70
```

```
71   public void resetTimer(){
72    if(timeNow - gameTime > sendCycleTime){
73     gameTime = timeNow;
74    }
75   }
76  }
```

B.1.7 Arquivo Assets/Scripts/FallingCubes.cs

```
1   public class FallingCubes : MonoBehaviour {
2
3    public float delay = 0.1f;
4    private int count = 0;
5    public GameObject cube;
6    public Material cubeMaterial;
7    private string cubeName = "FallingCube";
8    string[] colorNames = { "Green", "Red", "Blue" };
9    IDictionary<string, Color> colors = new
     ↪  Dictionary<string, Color>();
10
11   void Start () {
12    colors.Add ("Green", Color.green);
13    colors.Add ("Red", Color.red);
14    colors.Add ("Blue", Color.blue);
15    InvokeRepeating ("Spawn", delay, delay);
16   }
17
18   void FixedUpdate(){
19    this.count++;
20    if(this.count > 1000){
21     CancelInvoke ();
22    }
23   }
24
25   void Spawn () {
26    GameObject cubeClone = Instantiate (cube, new Vector3
      ↪  (Random.Range (4, 10), 15, Random.Range (-8, 0)),
      ↪  Quaternion.identity);
27    string colorName = colorNames [Random.Range (0, 3)];
```

```
28    cubeClone.GetComponent<Renderer> ().material.color =
      ↪  colors[colorName];
29    cubeClone.name = colorName + cubeName + count;
30   }
31
32   void OnApplicationQuit() {
33    cube.name = cubeName;
34   }
35  }
```

B.1.8 Arquivo Assets/Scripts/GroundScript.cs

```
1  public class GroundScript : MonoBehaviour {
2
3   void OnTriggerEnter(Collider other){
4    if (other.name.Contains ("Falling")) {
5     Destroy (other.gameObject);
6    }
7   }
8  }
```

B.2 SIGON

B.2.1 Arquivo src/unity/Main.java

```
1  public class Main {
2
3   public static void main(String[] args) {
4    startAgent();
5   }
6
7   private static void startAgent(){
8    try {
9
10    File agentFile = new File("unity.on");
11    CharStream stream =
      ↪  CharStreams.fromFileName(agentFile.getAbsolutePath());
12    AgentLexer lexer = new AgentLexer(stream);
```

```
13      CommonTokenStream tokens = new
        ↪  CommonTokenStream(lexer);
14
15      AgentParser parser = new AgentParser(tokens);
16      parser.removeErrorListeners();
17      parser.addErrorListener(new VerboseListener());
18
19      ParseTree tree = parser.agent();
20      ParseTreeWalker walker = new ParseTreeWalker();
21
22      AgentWalker agentWalker = new AgentWalker();
23      walker.walk(agentWalker, tree);
24
25      Agent agent = new Agent();
26      agent.run(agentWalker, null);
27
28
29
30    } catch (IOException e) {
31      System.out.println("I/O exception.");
32    }
33   }
34  }
```

B.2.2 Arquivo src/unity/Perception.java

```
1   public class Perception {
2
3    private String value;
4    private int timeCount = 1000;
5
6    public Perception(String value){
7     this.value = value;
8    }
9
10   @Override
11   public int hashCode() {
12    final int prime = 31;
13    int result = 1;
```

```
14      result = prime * result + ((value == null) ? 0 :
        ↪  value.hashCode());
15      return result;
16    }
17
18    @Override
19    public boolean equals(Object obj) {
20      if (this == obj)
21        return true;
22      if (obj == null)
23        return false;
24      if (getClass() != obj.getClass())
25        return false;
26      Perception other = (Perception) obj;
27      if (value == null) {
28        if (other.value != null)
29          return false;
30      } else if (!value.equals(other.value))
31        return false;
32      return true;
33    }
34
35    public int getTimeCount() {
36      return timeCount;
37    }
38
39    public void setTimeCount(int timeCount) {
40      this.timeCount = timeCount;
41    }
42
43    public String getValue() {
44      return value;
45    }
46
47    public void setValue(String value) {
48      this.value = value;
49    }
50  }
```

B.2.3 Arquivo src/unity/SensorHearing.java

```
1  public class SensorHearing extends Sensor {
2
3   private String sensorName = "Hearing";
4   private Socket socket;
5   private int port = 2405;
6   private float mass = 80.0f;
7   private HashSet<Perception> perceptions = new
    ↪  HashSet<>();
8   private HashSet<Perception> oldPerceptions = new
    ↪  HashSet<>();
9   private HashMap<String, Perception>
    ↪  latestObjectPerception = new HashMap<>();
10  private double timeToCheckPerceptions = 0;
11  private int minPriority = 1;
12  private int maxPriority = 10;
13
14  @Override
15  public void run() {
16   try {
17    ServerSocket serverSocket = new ServerSocket(port);
18    System.out.println("Server Started and listening to the
      ↪  port " + port);
19    socket = serverSocket.accept();
20
21    InputStream is = socket.getInputStream();
22    InputStreamReader isr = new InputStreamReader(is);
23    BufferedReader br = new BufferedReader(isr);
24    while (true) {
25     String msg = br.readLine();
26     if (msg == null) {
27      break;
28     }
29     differentPerceptionPriorityStrategy(msg);
30    }
31   } catch (Exception e) {
32    e.printStackTrace();
33   } finally {
34    try {
35     socket.close();
```

```
36      } catch (Exception e) {
37       e.printStackTrace();
38      }
39     }
40    }
41
42    public void differentPerceptionPriorityStrategy(String
      ↪  msg){
43     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
44     perceptions.remove(p);
45     perceptions.add(p);
46
47     HashSet<Perception> temp = new HashSet<>();
48
49     perceptions.stream().forEach(perception -> {
50      perception.setTimeCount(perception.getTimeCount() - 1);
51      if (perception.getTimeCount() == 0) {
52       temp.add(perception);
53      }
54     });
55
56     perceptions.removeAll(temp);
57     oldPerceptions.removeAll(temp);
58
59     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
60      int oldPerceptionsSize = oldPerceptions.size();
61      oldPerceptions.addAll(perceptions);
62
63      if (oldPerceptions.size() > oldPerceptionsSize) {
64       if (this.getPriority() < this.getMaxPriority()) {
65        this.setPriority(this.getPriority() + 1);
66       }
67      } else {
68       if (this.getPriority() > this.getMinPriority()) {
69        this.setPriority(this.getPriority() - 1);
70       }
71      }
```

```
72      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
73        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  (oldPerceptions.size() - oldPerceptionsSize));
74      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
75     }
76     checkLatestPerceptionToPublish(p);
77    }
78
79    public void effectiveMassPriorityStrategy(String msg){
80     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
81     perceptions.remove(p);
82     perceptions.add(p);
83
84     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
85      float priorityCalculation = 1 - (mass /
        ↪  perceptions.size());
86      int newPriority = Math.round(priorityCalculation * 10);
87      if(newPriority > this.getMaxPriority()){
88       newPriority = this.getMaxPriority();
89      } else if (newPriority < this.getMinPriority()){
90       newPriority = this.getMinPriority();
91      }
92      this.setPriority(newPriority);
93      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
94        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  this.perceptions.size());
95      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
96      perceptions.clear();
97     }
98     checkLatestPerceptionToPublish(p);
99    }
100
101   public void checkLatestPerceptionToPublish(Perception p){
```

```
102     String objectKey = p.getValue().substring(0,
        ↪  p.getValue().indexOf(","));
103     if(latestObjectPerception.get(objectKey) != null){
104      if(!latestObjectPerception.get(objectKey).getValue()
105       .equals(p.getValue())){
106       latestObjectPerception.put(objectKey,p);
107       super.publisher.onNext(p.getValue());
108      }
109     } else {
110      latestObjectPerception.put(objectKey,p);
111     }
112    }
113
114    public int getMinPriority() {
115     return minPriority;
116    }
117
118    public void setMinPriority(int minPriority) {
119     this.minPriority = minPriority;
120    }
121
122    public int getMaxPriority() {
123     return maxPriority;
124    }
125
126    public void setMaxPriority(int maxPriority) {
127     this.maxPriority = maxPriority;
128    }
129
130   }
```

B.2.4 Arquivo src/unity/SensorTact.java

```
1   public class SensorTact extends Sensor {
2
3    private String sensorName = "Tact";
4    private Socket socket;
5    private int port = 2404;
6    private float mass = 80.0f;
```

```
7   private HashSet<Perception> perceptions = new
    ↪  HashSet<>();
8   private HashSet<Perception> oldPerceptions = new
    ↪  HashSet<>();
9   private HashMap<String, Perception>
    ↪  latestObjectPerception = new HashMap<>();
10  private double timeToCheckPerceptions = 0;
11  private int minPriority = 1;
12  private int maxPriority = 10;
13
14  @Override
15  public void run() {
16   try {
17    ServerSocket serverSocket = new ServerSocket(port);
18    System.out.println("Server Started and listening to the
      ↪  port " + port);
19    socket = serverSocket.accept();
20
21    InputStream is = socket.getInputStream();
22    InputStreamReader isr = new InputStreamReader(is);
23    BufferedReader br = new BufferedReader(isr);
24    while (true) {
25     String msg = br.readLine();
26     if (msg == null) {
27      break;
28     }
29     differentPerceptionPriorityStrategy(msg);
30    }
31   } catch (Exception e) {
32    e.printStackTrace();
33   } finally {
34    try {
35     socket.close();
36    } catch (Exception e) {
37     e.printStackTrace();
38    }
39   }
40  }
41
42  public void differentPerceptionPriorityStrategy(String
    ↪  msg){
```

```
43      Perception p = new Perception(msg.substring(0,
        ↪  msg.lastIndexOf(",")));
44      perceptions.remove(p);
45      perceptions.add(p);
46
47      HashSet<Perception> temp = new HashSet<>();
48
49      perceptions.stream().forEach(perception -> {
50       perception.setTimeCount(perception.getTimeCount() - 1);
51       if (perception.getTimeCount() == 0) {
52        temp.add(perception);
53       }
54      });
55
56      perceptions.removeAll(temp);
57      oldPerceptions.removeAll(temp);
58
59      if
        ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
        ↪  - timeToCheckPerceptions > 2) {
60       int oldPerceptionsSize = oldPerceptions.size();
61       oldPerceptions.addAll(perceptions);
62
63       if (oldPerceptions.size() > oldPerceptionsSize) {
64        if (this.getPriority() < this.getMaxPriority()) {
65         this.setPriority(this.getPriority() + 1);
66        }
67       } else {
68        if (this.getPriority() > this.getMinPriority()) {
69         this.setPriority(this.getPriority() - 1);
70        }
71       }
72       System.out.println(this.sensorName + ", " +
         ↪  this.getPriority() + ", "
73         + msg.substring(msg.lastIndexOf(",")+1) + ", " +
           ↪  (oldPerceptions.size() - oldPerceptionsSize));
74       timeToCheckPerceptions =
         ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
75      }
76      checkLatestPerceptionToPublish(p);
77     }
```

```
78
79  public void effectiveMassPriorityStrategy(String msg){
80   Perception p = new Perception(msg.substring(0,
      ↪  msg.lastIndexOf(",")));
81   perceptions.remove(p);
82   perceptions.add(p);
83
84   if
      ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
      ↪  - timeToCheckPerceptions > 2) {
85    float priorityCalculation = 1 - (mass /
       ↪  perceptions.size());
86    int newPriority = Math.round(priorityCalculation * 10);
87    if(newPriority > this.getMaxPriority()){
88     newPriority = this.getMaxPriority();
89    } else if (newPriority < this.getMinPriority()){
90     newPriority = this.getMinPriority();
91    }
92    this.setPriority(newPriority);
93    System.out.println(this.sensorName + ", " +
       ↪  this.getPriority() + ", "
94      + msg.substring(msg.lastIndexOf(",")+1) + ", " +
         ↪  this.perceptions.size());
95    timeToCheckPerceptions =
       ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
96    perceptions.clear();
97   }
98   checkLatestPerceptionToPublish(p);
99  }
100
101 public void checkLatestPerceptionToPublish(Perception p){
102  String objectKey = p.getValue().substring(0,
      ↪  p.getValue().indexOf(","));
103  if(latestObjectPerception.get(objectKey) != null){
104   if(!latestObjectPerception.get(objectKey).getValue()
105    .equals(p.getValue())){
106    latestObjectPerception.put(objectKey,p);
107    super.publisher.onNext(p.getValue());
108   }
109  } else {
110   latestObjectPerception.put(objectKey,p);
```

```
111     }
112    }
113
114    public int getMinPriority() {
115     return minPriority;
116    }
117
118    public void setMinPriority(int minPriority) {
119     this.minPriority = minPriority;
120    }
121
122    public int getMaxPriority() {
123     return maxPriority;
124    }
125
126    public void setMaxPriority(int maxPriority) {
127     this.maxPriority = maxPriority;
128    }
129
130   }
```

B.2.5 Arquivo src/unity/SensorVisionClose.java

```
1   public class SensorVisionClose extends Sensor {
2
3    private String sensorName = "VisionClose";
4    private Socket socket;
5    private int port = 2403;
6    private float mass = 80.0f;
7    private HashSet<Perception> perceptions = new
     ↪  HashSet<>();
8    private HashSet<Perception> oldPerceptions = new
     ↪  HashSet<>();
9    private HashMap<String, Perception>
     ↪  latestObjectPerception = new HashMap<>();
10   private double timeToCheckPerceptions = 0;
11   private int minPriority = 5;
12   private int maxPriority = 10;
13
14   @Override
```

```
15  public void run() {
16   try {
17    ServerSocket serverSocket = new ServerSocket(port);
18    System.out.println("Server Started and listening to the
      ↪  port " + port);
19    socket = serverSocket.accept();
20
21    InputStream is = socket.getInputStream();
22    InputStreamReader isr = new InputStreamReader(is);
23    BufferedReader br = new BufferedReader(isr);
24    while (true) {
25     String msg = br.readLine();
26     if (msg == null) {
27      break;
28     }
29     differentPerceptionPriorityStrategy(msg);
30    }
31   } catch (Exception e) {
32    e.printStackTrace();
33   } finally {
34    try {
35     socket.close();
36    } catch (Exception e) {
37     e.printStackTrace();
38    }
39   }
40  }
41
42  public void differentPerceptionPriorityStrategy(String
    ↪  msg){
43   Perception p = new Perception(msg.substring(0,
     ↪  msg.lastIndexOf(",")));
44   perceptions.remove(p);
45   perceptions.add(p);
46
47   HashSet<Perception> temp = new HashSet<>();
48
49   perceptions.stream().forEach(perception -> {
50    perception.setTimeCount(perception.getTimeCount() - 1);
51    if (perception.getTimeCount() == 0) {
52     temp.add(perception);
```

```
53      }
54     });
55
56     perceptions.removeAll(temp);
57     oldPerceptions.removeAll(temp);
58
59     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
60      int oldPerceptionsSize = oldPerceptions.size();
61      oldPerceptions.addAll(perceptions);
62
63      if (oldPerceptions.size() > oldPerceptionsSize) {
64       if (this.getPriority() < this.getMaxPriority()) {
65        this.setPriority(this.getPriority() + 1);
66       }
67      } else {
68       if (this.getPriority() > this.getMinPriority()) {
69        this.setPriority(this.getPriority() - 1);
70       }
71      }
72      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
73        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  (oldPerceptions.size() - oldPerceptionsSize));
74      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
75     }
76     checkLatestPerceptionToPublish(p);
77    }
78
79    public void effectiveMassPriorityStrategy(String msg){
80     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
81     perceptions.remove(p);
82     perceptions.add(p);
83
84     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
```

```
85      float priorityCalculation = 1 - (mass /
        ↪  perceptions.size());
86      int newPriority = Math.round(priorityCalculation * 10);
87      if(newPriority > this.getMaxPriority()){
88       newPriority = this.getMaxPriority();
89      } else if (newPriority < this.getMinPriority()){
90       newPriority = this.getMinPriority();
91      }
92      this.setPriority(newPriority);
93      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
94        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  this.perceptions.size());
95      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
96      perceptions.clear();
97     }
98     checkLatestPerceptionToPublish(p);
99    }
100
101   public void checkLatestPerceptionToPublish(Perception p){
102    String objectKey = p.getValue().substring(0,
       ↪  p.getValue().indexOf(","));
103    if(latestObjectPerception.get(objectKey) != null){
104     if(!latestObjectPerception.get(objectKey).getValue()
105      .equals(p.getValue())){
106      latestObjectPerception.put(objectKey,p);
107      super.publisher.onNext(p.getValue());
108     }
109    } else {
110     latestObjectPerception.put(objectKey,p);
111    }
112   }
113
114   public int getMinPriority() {
115    return minPriority;
116   }
117
118   public void setMinPriority(int minPriority) {
119    this.minPriority = minPriority;
120   }
```

```
121
122  public int getMaxPriority() {
123   return maxPriority;
124  }
125
126  public void setMaxPriority(int maxPriority) {
127   this.maxPriority = maxPriority;
128  }
129 }
```

B.2.6 Arquivo src/unity/SensorVisionFar.java

```
1  public class SensorVisionFar extends Sensor {
2
3   private String sensorName = "VisionFar";
4   private Socket socket;
5   private int port = 2401;
6   private float mass = 80.0f;
7   private HashSet<Perception> perceptions = new
    ↪  HashSet<>();
8   private HashSet<Perception> oldPerceptions = new
    ↪  HashSet<>();
9   private HashMap<String, Perception>
    ↪  latestObjectPerception = new HashMap<>();
10  private double timeToCheckPerceptions = 0;
11  private int minPriority = 1;
12  private int maxPriority = 6;
13
14  @Override
15  public void run() {
16   try {
17    ServerSocket serverSocket = new ServerSocket(port);
18    System.out.println("Server Started and listening to the
      ↪  port " + port);
19    socket = serverSocket.accept();
20
21    InputStream is = socket.getInputStream();
22    InputStreamReader isr = new InputStreamReader(is);
23    BufferedReader br = new BufferedReader(isr);
24    while (true) {
```

```
25      String msg = br.readLine();
26      if (msg == null) {
27       break;
28      }
29      differentPerceptionPriorityStrategy(msg);
30     }
31    } catch (Exception e) {
32     e.printStackTrace();
33    } finally {
34     try {
35      socket.close();
36     } catch (Exception e) {
37      e.printStackTrace();
38     }
39    }
40   }
41
42   public void differentPerceptionPriorityStrategy(String
     ↪  msg){
43    Perception p = new Perception(msg.substring(0,
      ↪  msg.lastIndexOf(",")));
44    perceptions.remove(p);
45    perceptions.add(p);
46
47    HashSet<Perception> temp = new HashSet<>();
48
49    perceptions.stream().forEach(perception -> {
50     perception.setTimeCount(perception.getTimeCount() - 1);
51     if (perception.getTimeCount() == 0) {
52      temp.add(perception);
53     }
54    });
55
56    perceptions.removeAll(temp);
57    oldPerceptions.removeAll(temp);
58
59    if
      ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
      ↪  - timeToCheckPerceptions > 2) {
60     int oldPerceptionsSize = oldPerceptions.size();
61     oldPerceptions.addAll(perceptions);
```

```
62
63      if (oldPerceptions.size() > oldPerceptionsSize) {
64       if (this.getPriority() < this.getMaxPriority()) {
65        this.setPriority(this.getPriority() + 1);
66       }
67      } else {
68       if (this.getPriority() > this.getMinPriority()) {
69        this.setPriority(this.getPriority() - 1);
70       }
71      }
72      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
73        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  (oldPerceptions.size() - oldPerceptionsSize));
74      timeToCheckPerceptions =
         ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
75     }
76     checkLatestPerceptionToPublish(p);
77    }
78
79    public void effectiveMassPriorityStrategy(String msg){
80     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
81     perceptions.remove(p);
82     perceptions.add(p);
83
84     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
85      float priorityCalculation = 1 - (mass /
         ↪  perceptions.size());
86      int newPriority = Math.round(priorityCalculation * 10);
87      if(newPriority > this.getMaxPriority()){
88       newPriority = this.getMaxPriority();
89      } else if (newPriority < this.getMinPriority()){
90       newPriority = this.getMinPriority();
91      }
92      this.setPriority(newPriority);
93      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
```

```
94        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪ this.perceptions.size());
95      timeToCheckPerceptions =
          ↪ Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
96      perceptions.clear();
97     }
98     checkLatestPerceptionToPublish(p);
99    }
100
101   public void checkLatestPerceptionToPublish(Perception p){
102    String objectKey = p.getValue().substring(0,
         ↪ p.getValue().indexOf(","));
103    if(latestObjectPerception.get(objectKey) != null){
104     if(!latestObjectPerception.get(objectKey).getValue()
105      .equals(p.getValue())){
106      latestObjectPerception.put(objectKey,p);
107      super.publisher.onNext(p.getValue());
108     }
109    } else {
110     latestObjectPerception.put(objectKey,p);
111    }
112   }
113
114   public int getMinPriority() {
115    return minPriority;
116   }
117
118   public void setMinPriority(int minPriority) {
119    this.minPriority = minPriority;
120   }
121
122   public int getMaxPriority() {
123    return maxPriority;
124   }
125
126   public void setMaxPriority(int maxPriority) {
127    this.maxPriority = maxPriority;
128   }
129  }
```

B.2.7 Arquivo src/unity/SensorVisionMed.java

```
1  public class SensorVisionMed extends Sensor {
2
3   private String sensorName = "VisionMed";
4   private Socket socket;
5   private int port = 2402;
6   private float mass = 80.0f;
7   private HashSet<Perception> perceptions = new
    ↪  HashSet<>();
8   private HashSet<Perception> oldPerceptions = new
    ↪  HashSet<>();
9   private HashMap<String, Perception>
    ↪  latestObjectPerception = new HashMap<>();
10  private double timeToCheckPerceptions = 0;
11  private int minPriority = 3;
12  private int maxPriority = 8;
13
14  @Override
15  public void run() {
16   try {
17    ServerSocket serverSocket = new ServerSocket(port);
18    System.out.println("Server Started and listening to the
      ↪  port " + port);
19    socket = serverSocket.accept();
20
21    InputStream is = socket.getInputStream();
22    InputStreamReader isr = new InputStreamReader(is);
23    BufferedReader br = new BufferedReader(isr);
24    while (true) {
25     String msg = br.readLine();
26     if (msg == null) {
27      break;
28     }
29     differentPerceptionPriorityStrategy(msg);
30    }
31   } catch (Exception e) {
32    e.printStackTrace();
33   } finally {
34    try {
35     socket.close();
```

```
36      } catch (Exception e) {
37       e.printStackTrace();
38      }
39     }
40    }
41
42    public void differentPerceptionPriorityStrategy(String
      ↪  msg){
43     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
44     perceptions.remove(p);
45     perceptions.add(p);
46
47     HashSet<Perception> temp = new HashSet<>();
48
49     perceptions.stream().forEach(perception -> {
50      perception.setTimeCount(perception.getTimeCount() - 1);
51      if (perception.getTimeCount() == 0) {
52       temp.add(perception);
53      }
54     });
55
56     perceptions.removeAll(temp);
57     oldPerceptions.removeAll(temp);
58
59     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
60      int oldPerceptionsSize = oldPerceptions.size();
61      oldPerceptions.addAll(perceptions);
62
63      if (oldPerceptions.size() > oldPerceptionsSize) {
64       if (this.getPriority() < this.getMaxPriority()) {
65        this.setPriority(this.getPriority() + 1);
66       }
67      } else {
68       if (this.getPriority() > this.getMinPriority()) {
69        this.setPriority(this.getPriority() - 1);
70       }
71      }
```

```
72      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
73        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  (oldPerceptions.size() - oldPerceptionsSize));
74      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
75     }
76     checkLatestPerceptionToPublish(p);
77    }
78
79    public void effectiveMassPriorityStrategy(String msg){
80     Perception p = new Perception(msg.substring(0,
       ↪  msg.lastIndexOf(",")));
81     perceptions.remove(p);
82     perceptions.add(p);
83
84     if
       ↪  (Double.valueOf(msg.substring(msg.lastIndexOf(",")+1))
       ↪  - timeToCheckPerceptions > 2) {
85      float priorityCalculation = 1 - (mass /
        ↪  perceptions.size());
86      int newPriority = Math.round(priorityCalculation * 10);
87      if(newPriority > this.getMaxPriority()){
88       newPriority = this.getMaxPriority();
89      } else if (newPriority < this.getMinPriority()){
90       newPriority = this.getMinPriority();
91      }
92      this.setPriority(newPriority);
93      System.out.println(this.sensorName + ", " +
        ↪  this.getPriority() + ", "
94        + msg.substring(msg.lastIndexOf(",")+1) + ", " +
          ↪  this.perceptions.size());
95      timeToCheckPerceptions =
        ↪  Double.valueOf(msg.substring(msg.lastIndexOf(",")+1));
96      perceptions.clear();
97     }
98     checkLatestPerceptionToPublish(p);
99    }
100
101   public void checkLatestPerceptionToPublish(Perception p){
```

```
102     String objectKey = p.getValue().substring(0,
        ↪  p.getValue().indexOf(","));
103     if(latestObjectPerception.get(objectKey) != null){
104      if(!latestObjectPerception.get(objectKey).getValue()
105       .equals(p.getValue())){
106       latestObjectPerception.put(objectKey,p);
107       super.publisher.onNext(p.getValue());
108      }
109     } else {
110      latestObjectPerception.put(objectKey,p);
111     }
112    }
113
114    public int getMinPriority() {
115     return minPriority;
116    }
117
118    public void setMinPriority(int minPriority) {
119     this.minPriority = minPriority;
120    }
121
122    public int getMaxPriority() {
123     return maxPriority;
124    }
125
126    public void setMaxPriority(int maxPriority) {
127     this.maxPriority = maxPriority;
128    }
129   }
```