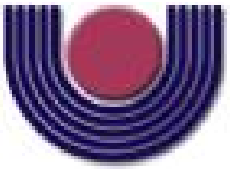

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Implementação do Módulo Simulador de Mundos Virtuais para o SimBot - Simulador de Robôs para LEGO NXT

Juliano Richetti

CASCAVEL
2014

JULIANO RICHETTI

IMPLEMENTAÇÃO DO MÓDULO SIMULADOR DE MUNDOS VIRTUAIS PARA O SIMBOT - SIMULADOR DE ROBÔS PARA LEGO NXT

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientadora: Prof. Adriana Postal

CASCAVEL
2014

JULIANO RICHETTI

IMPLEMENTAÇÃO DO MÓDULO SIMULADOR DE MUNDOS VIRTUAIS PARA O SIMBOT - SIMULADOR DE ROBÔS PARA LEGO NXT

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Adriana Postal (Orientadora)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Josué Pereira de Castro
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Reginaldo Aparecido Zara
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 8 de dezembro de 2014

AGRADECIMENTOS

Agradeço primeiramente a minha orientadora Adriana Postal que me ajudou muito a escrever e completar este trabalho. Aos meus amigos Cezar, Thiago, Mauriverti, Remi, Natal entre outros que me ajudaram de alguma forma ou outra no desenvolvimento do trabalho. Agradeço também pelo apoio moral que ofereceram para terminar este desafio. Agradeço a minha família por ter me dado a oportunidade de concluir este curso de Ciência da Computação na Unioeste. Também agradeço aos usuários do fórum do Unity, que responderam, mesmo sem saber, grande parte das minhas perguntas e ajudaram indiretamente no desenvolvimento deste trabalho. Agradeço ao meu irmão, Jeferson Richetti, a Julliane Brita e a Kássia Yuri por terem ajudado na escrita deste documento.

Lista de Figuras

Sumário

Resumo

Este trabalho descreve o módulo criador de mundos para o SimBot, simulador para robôs LEGO NXT. Este módulo é classificado como uma aplicação de Realidade Virtual e utiliza um Ambiente Virtual, desenvolvido utilizando a plataforma Unity, para simular um ambiente real e possibilitar a simulação das ações do robô LEGO, interagindo com os objetos inseridos no ambiente.

Palavras-chave: LEGO NXT, Realidade Virtual, Ambiente Virtual, Simulador, Unity3D

Capítulo 1

Introdução

1.1 Definição do problema

O curso de Ciência da Computação da UNIOESTE oferece, como optativa, a disciplina de Introdução a Robótica. Nesta disciplina, utilizam-se os kits LEGO NXT 2.0 (LEGO, 2014) para a realização das aulas práticas. Como o custo de aquisição dos kits é alto, o que limita o número de alunos matriculados na disciplina, surgiu a ideia de construir o - Simulador de Robôs para LEGO NXT, uma ferramenta de simulação que seja capaz de resolver este problema de custo e permita que mais alunos possam se matricular nesta disciplina.

Segundo o dicionário Houaiss (HOUAISS, 2009), simulação pode significar: imitação do funcionamento de um processo por meio da função de outro, ou teste, experiência ou ensaio em que se reproduz artificialmente uma situação, ou as condições reais de um meio, fenômeno etc., frequentemente realizado com modelos.

Porém estas definições não são suficientes para explicar a simulação feita neste trabalho, ela tem um caráter mais técnico. Portanto, separamos simulação em dois focos: Simulação Computacional, aquela que precisa de um computador para poder ser feita, e Simulação Não-Computacional (CHWIF; MEDINA, 2010), que não utiliza computadores para ser realizada. A simulação aqui proposta trata-se da imitação do mundo real em um cenário virtual, portanto é classificada como simulação computacional.

O SimBot tem como objetivo permitir a modelagem livre de protótipos de robôs com as peças do LEGO NXT e simular o seu comportamento dentro de ambientes virtuais regidos pelas leis da física, mais especificamente as leis da mecânica. A movimentação do robô LEGO ocorrerá pela tradução dos comandos do robô em NXC (HANSEN, 2013), para comandos Unity

(UNITY, 2014a). Dessa forma, o SimBot consiste em três partes e pode ser visto na Figura 1.1. A primeira parte é o tradutor de comandos NXC, a segunda parte é o Modelador de Robôs, onde ocorre a montagem do robô LEGO, e a terceira parte é o módulo de simulação do robô LEGO em um ambiente virtual 3D regido por leis da mecânica.

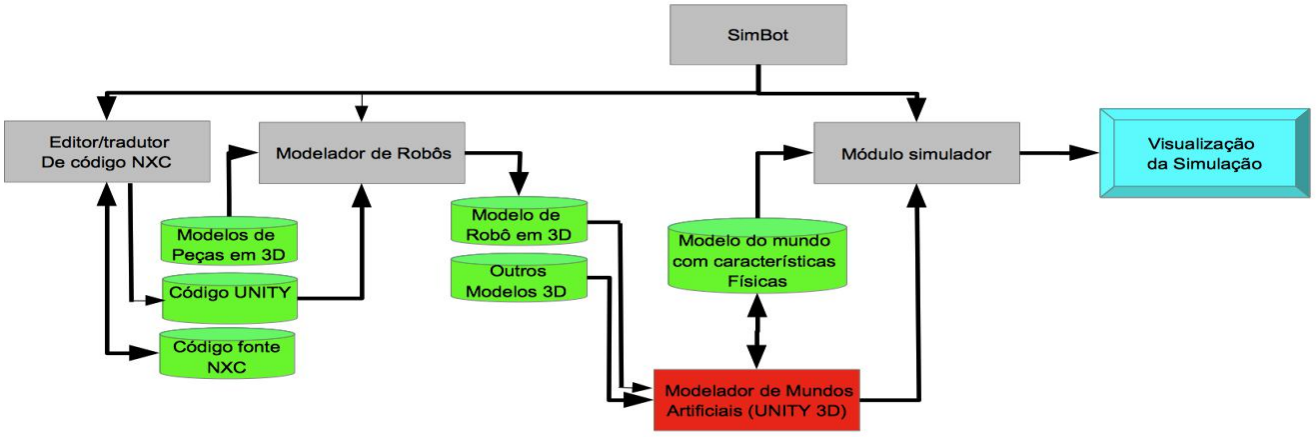

Figura 1.1: Os três módulos do SimBot

O software proposto apresenta alguns desafios: a importação de um robô LEGO funcional montado no Modelador de Robôs, a criação de um ambiente virtual para a simulação do robô, a aplicação das leis da física neste ambiente e neste robô LEGO, e a execução da própria simulação.

Este trabalho desenvolveu o `Módulo Simulador`, que consiste em criar um ambiente virtual pré-definido como uma mesa de testes, onde o utilizador do programa poderá adicionar objetos como cubos, rampas e esferas, podendo também alterar suas posições no mundo, rotacioná-los e aumentar ou diminuir o tamanho de cada objeto. No ambiente foi utilizado um componente do Unity para simular o robô, para que seja possível simular a interação dele com o ambiente montado pelo usuário. A movimentação do robô será realizada diretamente pelo usuário.

Este trabalho foi desenvolvido utilizando a ferramenta Unity (UNITY, 2014a). Nela foi desenvolvido o ambiente virtual onde ocorrerá a simulação, e foram adicionadas as leis da mecânica, adicionando aos objetos pertencentes ao ambiente virtual, o componente *RigidBodies*, que será visto na seção 3.2. O Unity utiliza o motor de física PhysX (NVIDIA, 2014), descrito na sessão 3.1, para simular estas leis da física em suas aplicações.

1.2 Objetivos

O objetivo geral deste trabalho é o desenvolvimento do Módulo Simulador, que pode ser visto na figura 1.2. Os objetivos específicos são: a criação do ambiente virtual e seus objetos e criação da simulação permitindo a interação do robô com o ambiente virtual. Para a criação do ambiente virtual e seus objetos foram feitos os seguintes passos:

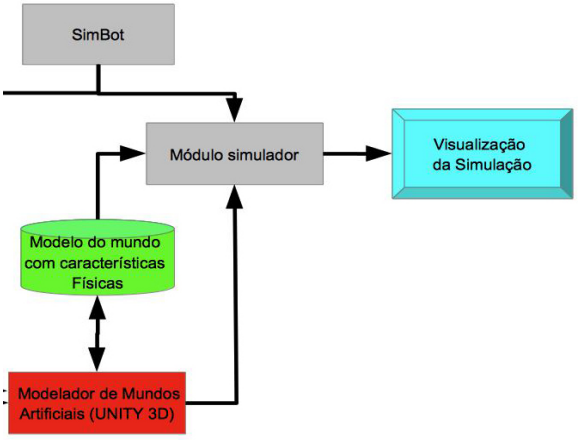

Figura 1.2: Módulo Simulador - Duas Etapas

- Criado o Ambiente Virtual onde ocorrerá a simulação.
- Criados os obstáculos presentes no ambiente para incrementar a interação do robô com o ambiente.
- Importado o robô LEGO do Modelador de Robôs.
- Criado o sistema para adicionar novos obstáculos ao ambiente virtual e a modificação de suas características como tamanho, rotação e posição.

Para a criação da simulação foi adicionado o componente que permitirá que o robô LEGO utilizado seja controlado durante a simulação e uma nova câmera ao robô que permitirá seguí-lo enquanto este se move durante a simulação.

1.3 Trabalhos Relacionados

1.3.1 SimTwo

SimTwo (COSTA, 2012) é um simulador realístico que permite a implementação de diversos tipos de robôs: Robôs móveis, Manipuladores, Quadrúpedes, Humanóides, Veículos leves com ou sem propulsores, robôs consistindo de misturas entre corpos rígidos e juntas e veículos aquáticos com propulsores.

O realismo da dinâmica implementada no SimTwo é adquirido desconstruindo os robôs em um sistema de corpos rígidos, com juntas controladas por motores elétricos, e todos os aspectos mecânicos são simulados numericamente considerando suas propriedades físicas, como formato, massa, momento, fricção e inércia. Ele também oferece a possibilidade de entregar sinais de referência para os controles primários do SimTwo, utilizando controles criados pelos próprios usuários. Eles podem ser implementados usando uma linguagem de *script* editável e compatível com o SimTwo no próprio programa, ou por um programa remoto que conversa com o SimTwo por internet, ou porta serial.

Ao executar o simulador, abre-se imediatamente a janela onde ocorre a simulação, como mostra a Figura 1.3. Dentro do ambiente virtual mostrado é possível perceber três tipos de objetos: os que não podem ser movidos, os que podem ser movidos e o robô que é o foco da simulação. A simulação nativa considera o robô como uma esteira, e é possível colocar os objetos em cima dela para ver sua movimentação. Porém, não foi encontrada uma maneira de modificar qual objeto é considerado como robô, ou mudar o robô, nem como mudar as posições dos objetos imóveis.

1.3.2 MORSE

O simulador MORSE (LAAS-CNRS, 2010) é feito em Python, e utiliza o Blender para criar seus objetos. Ao criar um projeto de simulação, o programa cria automaticamente um ambiente virtual com diversos objetos, móveis e fixos, e um robô. Podem-se utilizar as setas do teclado

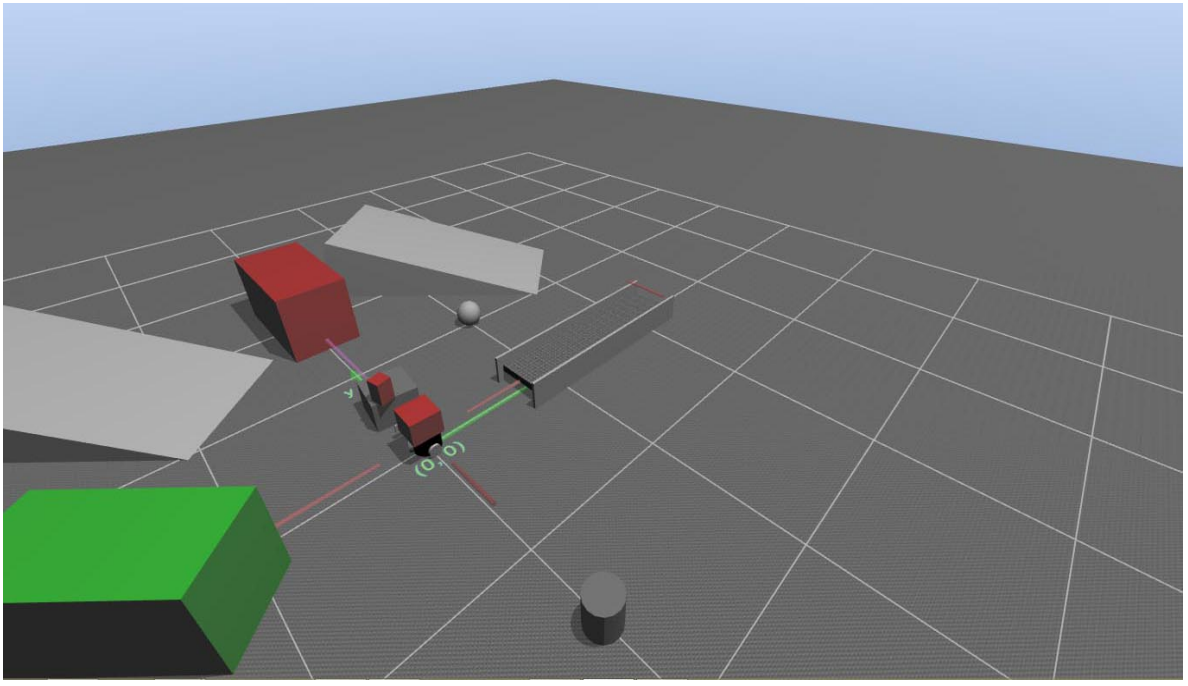

Figura 1.3: Tela principal onde ocorre a simulação no SimTwo.

para mover o robô e as teclas `W, A, S, D` para mover a câmera. Ao executar a simulação abre-se então uma tela com o ambiente virtual já construído, que pode ser vista na Figura 1.4. Os objetos nesta simulação são todos imóveis, com exceção apenas do robô que é controlado diretamente pela pessoa que está testando a simulação.

Esta simulação possui um motor de física já ativo, responsável pelas colisões e pelo movimento, porém como mostrado na Figura 1.5, a colisão não é tão precisa, consequentemente o robô "entra" na caixa vermelha. É possível adicionar diversos objetos utilizando o arquivo fonte da simulação. O MORSE reconhece três tipos de componentes principais, os Robôs, os Sensores, e os Atuadores. Os robôs são em sua maior parte suportes para os atuadores e sensores e eles estão inseridos em um ambiente virtual, que pode ser o padrão ou algum outro criado pelo usuário, e que possuam a física incorporada. O comportamento desses objetos pode ser modificado.

1.3.3 RobotC

O RobotC (ROBOMATTER, 2005b) é uma linguagem de programação de robôs com caráter educacional e competitivo. Ela é uma linguagem baseada em C e possui um ambiente de de-

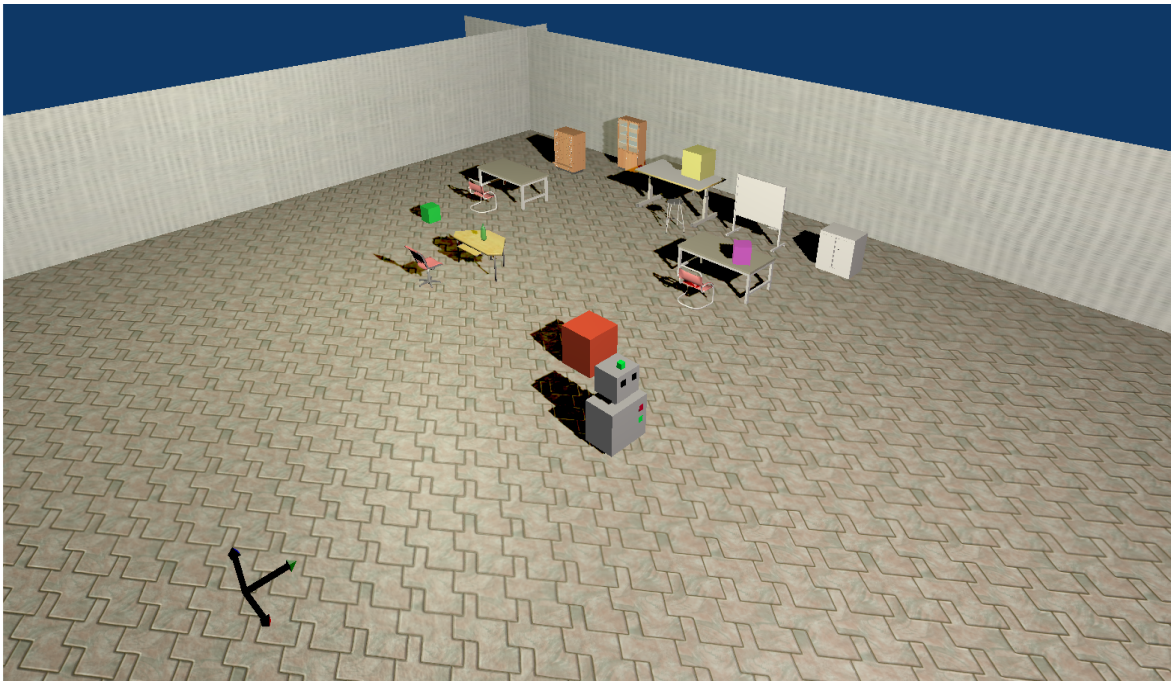

Figura 1.4: Ambiente e robô do MORSE

senvolvimento fácil de utilizar. Este ambiente possui dois tipos de ambiente, o de programação comum e o de desenvolvimento gráfico. Um diferencial desta ferramenta, além do ambiente de desenvolvimento gráfico, é a possibilidade de utilizar Ambientes Virtuais e Robôs virtuais. Ao abrir o programa, cria-se um novo documento para programar o robô, e então pode-se escolher o Robô a ser utilizado e o ambiente virtual a ser simulado (ROBOMATTER, 2005a). Porém, não é possível modificar os robôs LEGO, os ambientes virtuais não podem ser modificados e o programa precisa de licença para ser utilizado. No entanto, é possível baixar ferramentas que

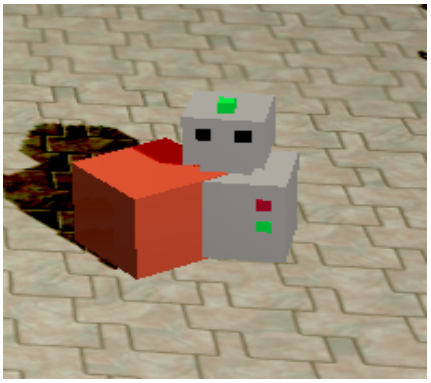

Figura 1.5: Robô colide com uma caixa.

auxiliam na programação e nos testes. Existem quatro destas ferramentas: um emulador virtual do *brick* do LEGO NXT com três opções de linguagens diferentes; um construtor de ambientes virtuais; um importador de modelos 3D para o construtor de ambientes virtuais e uma ferramenta de medidas, permitindo calcular a distância ou ângulos necessários para a movimentação nos ambientes virtuais.

1.4 Organização do Trabalho

Este trabalho está dividido em mais três capítulos. O segundo capítulo apresenta o que são realidade virtual e ambientes virtuais e suas características neste trabalho. O terceiro explica o desenvolvimento do trabalho, falando sobre as ferramentas e como foram utilizadas. E no último são feitas as considerações finais sobre este trabalho e as sugestões de trabalhos futuros.

Capítulo 2

Realidade Virtual e Ambientes Virtuais

2.1 Realidade Virtual

Segundo Tori, Kirner e Siscoutto (2006), a representação da realidade ou da imaginação sempre fizeram parte da vida do ser humano, e são expressas de diversas formas, seja por desenhos, figuras, pinturas, teatro, ópera, ilusionismo, cinema ou outras expressões artísticas. Com a evolução da realidade virtual e o avanço dos recursos computacionais, a representação do imaginário bem como a reprodução do real são mais facilmente obtidas.

A realidade virtual possibilita ao usuário interagir com situações imaginárias, e também moldá-las, como cenários fictícios, utilizando objetos virtuais estáticos e em movimento. Permite também reproduzir com fidelidade ambientes da vida real permitindo a interação com seus recursos de forma natural, com auxílio de aparatos tecnológicos.

Realidade Virtual (RV), segundo Tori, Kirner e Siscoutto (2006) é, basicamente, uma "interface avançada do usuário", para acessar aplicações de computadores, caracterizando principalmente a visualização de, e a movimentação em, ambientes 3D em tempo real, além da interação com seus elementos. Portanto, no contexto da RV, o ambiente tridimensional é descrito pelo usuário para então ser gerado pelo computador, e é possível visualizar este ambiente de qualquer posição escolhida. A interação do usuário com o ambiente virtual é um dos principais aspectos da interface, e sua grande vantagem é a tradução das habilidades e conhecimento intuitivos do usuário para manipular os objetos deste ambiente.

Para complementar essa definição, foi utilizada a de Lowood (2012): RV é o uso de um modelador e de simulação computacional, permitindo a uma pessoa interagir com um mundo tridimensional artificial usando ao menos um dos sentidos. E estas aplicações causam um sen-

timento de imersão no usuário presente neste ambiente virtual por meio de aparatos eletrônicos interativos. Desta forma, RV utiliza um modelador, simulação computacional e uma interface bem elaborada para causar sentimentos de imersão ao utilizador. Ambas caracterizando principalmente ambientes virtuais 3D, e geralmente utilizando dispositivos de entrada e saída não convencionais para a interação.

A realidade virtual pode ser classificada de duas formas (KIRNER; PINHO, 1997): imersiva ou não imersiva. Para que a RV seja considerada imersiva ela necessita que sejam usados dispositivos de entrada e saída não convencionais, como luvas, capacetes, óculos, ou salas de projeção nos tetos e paredes, entre outros, que permitam que o usuário sinta como se estivesse dentro do ambiente, cada dispositivo baseado em outros sentidos, como audição ou tato, melhoram a imersão da aplicação. Já a não imersiva utiliza os métodos convencionais de entrada e saída, como mouse, teclado e monitores. Mas, mesmo com a evolução da tecnologia de realidade virtual, a utilização de um monitor como saída ainda é aceitável como uma aplicação de RV.

Para definir o sistema de realidade virtual, que apresenta interface de hardware e software bem elaboradas e não convencionais, é importante considerar o relacionamento entre o usuário e o ambiente. O sistema de RV consiste de um usuário, uma interface homem-máquina e um computador. O computador gera o ambiente, que pode ser um ambiente real ou um ambiente fictício, no qual o usuário é inserido. A interação do usuário com o ambiente acontece usando dispositivos sensoriais de percepção e controle. Esse sistema pode ser visto na Figura 2.1 (KIRNER; PINHO, 1997).

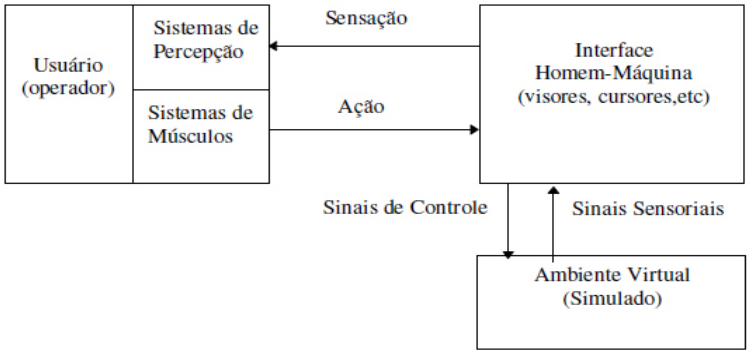

Figura 2.1: Sistema de Realidade Virtual

Nesse sistema, o usuário se comunica com a interface, e ela atua diretamente no computador, que por sua vez atua sobre o mundo virtual ou o mundo real simulado. Alguns exemplos de aplicações de Realidade Virtual são: jogos, simulações e treinamentos de controles de veículos, como aviões, carros, barcos entre outros, treinamentos militares e Medicina. Estes são apenas alguns exemplos das diversas possibilidades de aplicação (KIRNER; PINHO, 1997).

Para os treinamentos militares (KIRNER; PINHO, 1997), existem diversos exemplos, desde simulações de controle de aviões, barcos e submarinos até simuladores para operações militares. As simulações de veículos ocorre utilizando dispositivos de entrada que simulam o controle do veículo escolhido. O planejamento de operações militares ocorre criando uma simulação do local aonde a operação militar ocorrerá, permitindo aos soldados caminhar no local e analisar os caminhos e obstáculos que terão que enfrentar.

Na medicina existem diversas aplicações que utilizam realidade virtual (KIRNER; PINHO, 1997), como por exemplo simulações cirúrgicas e Cirurgias Minimamente Invasivas (CMIs), no planejamento de radioterapia e ensino de Anatomia. As CMIs ocorrem ao fazer um pequeno corte na pele do paciente, e inserindo uma camêra com instrumentos de corte no paciente, a cirurgia ocorre com o médico olhando um monitor, que mostra o que a câmera grava, não podendo olhar para as suas mãos. Para o planejamento de radiologia, é construído um modelo 3D do corpo do paciente e então é sobreposto a representação gráfica da radiação nestes modelos, neste sistema o médico senta em frente a um monitor e manipula a posição do paciente e a direção e intensidade dos raios. Nas Figuras 2.2 e 2.3 são mostrados exemplos de interface de um dos mais comuns tipos de aplicação de RV: jogos.

A figura 2.2 mostra um jogo em terceira pessoa, o personagem do jogador pode ser visto e é possível ver seus movimentos. Na figura 2.3 é mostrado um jogo em primeira pessoa, neste o personagem não pode ser visto pelo próprio jogador, apenas as mãos e os equipamentos segurados por ele. Nesta imagem podem ser vistos diversos personagens, cada um deles são outros jogadores. Ao comparar as duas imagens é possível perceber a diferença entre os métodos de visualização de cada um desses jogos. Ao utilizar a visualização em terceira pessoa, mostrada na figura 2.2, o usuário pode ver completamente o que ele está controlando, e pode mudar a posição da câmera sem mover o personagem. Porém, quando utilizado o método de visualizaçao em primeira pessoa, só é possível ver as partes "importantes" do personagem controlado, e não

Figura 2.2: Interface do jogo Dark Souls 2

é possível mudar a câmera de posição sem mover o personagem junto.

2.2 Ambientes Virtuais

Os Ambientes Virtuais (AVs) podem ser utilizados por apenas uma única pessoa, utilizando apenas um computador, ou por um grupo grande de indivíduos acessando um sistema distribuído (TORI; KIRNER; SISCOUTTO, 2006). Um AV é uma representação gráfica tridimensional em tempo real de um mundo real ou imaginário onde os usuários navegam e interagem com os objetos tridimensionais inseridos nele (HAGSAND, 1996).

A caracterização mais comum utilizada para a RV são os jogos, e devido a grande maioria ser multi-jogador, foram definidos os Ambientes Virtuais Distribuídos (AVDs). Estes AVDs diferem dos AVs comuns pela interação entre diversos usuários em diferentes posições geográficas cujos objetivos são a cooperação e o compartilhamento dos recursos em tempo real, melhorando o desempenho coletivo.

Quando os usuários podem compartilhar um mesmo espaço tridimensional virtual, onde eles poderão se ajudar na execução de uma determinada tarefa, usando os princípios de trabalho

Figura 2.3: Interface do jogo Battlefield 4

cooperativo baseado em computador, o AVD pode também ser chamado de Ambiente Virtual Colaborativo (AVC). A principal diferença do AVC para o AVD é que o AVC requer a possibilidade de cooperação entre os usuários, enquanto o AVD exige apenas que múltiplos usuários compartilhem o mesmo ambiente (RODELLO et al., 2007).

Ao utilizar um AV, um usuário utiliza um avatar, que é uma representação gráfica, geralmente de modelos humanos, podendo ser objetos inanimados dependendo do objetivo da aplicação utilizando o ambiente. Nos AVDs é possível a um usuário ver os avatares dos outros participantes localizados no mesmo lugar e também suas ações. Mas nem todos os avatares são controlados por usuários, eles podem ser controlados por um sistema de simulação dirigido a eventos.

Apesar da visualização ser a base de um AVD, é necessário que os usuários tenham a sensação de estarem em tempo real, podendo ver as ações uns dos outros e também que os participantes possam se comunicar de alguma forma.

Para os usuários poderem ver os ambientes virtuais, eles são mostrados, na maioria dos casos, em janelas (ou visualizadores). Cada usuário é associado a um avatar, que é utilizado pelo visualizador para representar a movimentação do usuário e exibir o cenário e seus objetos. A manipulação e movimentação do avatar são feitas, mais comumente, por teclado e *mouse*

mesmo não sendo os métodos mais eficientes.

Figura 2.4: Avatar do jogador lutando contra avatar do jogador inimigo em um barco ancorado em uma caverna no jogo Dark Souls 2

Na figura 2.4 pode ser visto o avatar do jogador enfrentando o avatar de um jogador inimigo. E na figura 2.5 é possível ver um ambiente virtual fictício, representando uma ponte de uma montanha para outra com um castelo no topo dela.

O AVD deve possuir uma rede de comunicação para permitir a troca de informações entre os usuários, como o posicionamento dos seus avatares e a movimentação dos objetos neste ambiente. A rede também deve manter o estado compartilhado no AVD sincronizado. Um esquema de comunicação geral pode ser visto na Figura 2.6 (KIRNER; PINHO, 1997).

A comunicação de um usuário com o ambiente virtual pode ser feita de quatro formas: como espectador, como participante real, com participação simulada ou sem participação ou possível supervisão, e podem ser conferidas observando a Figura 2.7 (KIRNER; PINHO, 1997). Na maioria dos casos, o ambiente virtual representa o mundo real, com excessão da participação simulada, onde o ambiente virtual pode ser fictício ou representar o real. No caso do espectador, o usuário apenas observa as ações de um robô, inserido em um ambiente real ou virtual, sem poder interferir com suas ações; no caso da participação real o usuário interage diretamente

Figura 2.5: Avatar do jogador observando o cenário no jogo Dark Souls 2

com o ambiente real ou virtual utilizando um robô; na participação simulada o usuário interage apenas com o ambiente virtual utilizando um avatar; e o último caso representa o usuário supervisionando um robô, utilizando um ambiente virtual, e suas ações em um ambiente real ou virtual.

Do mesmo modo que a comunicação de um usuário, quando existem múltiplos usuários a interação pode ocorrer de três maneiras: comunicação entre usuários; compartilhar o ambiente virtual; e realizar trabalhos cooperativos no mundo real utilizando o ambiente virtual compartilhado. Esses três métodos podem ser vistos na Figura 2.8 (KIRNER; PINHO, 1997). No caso

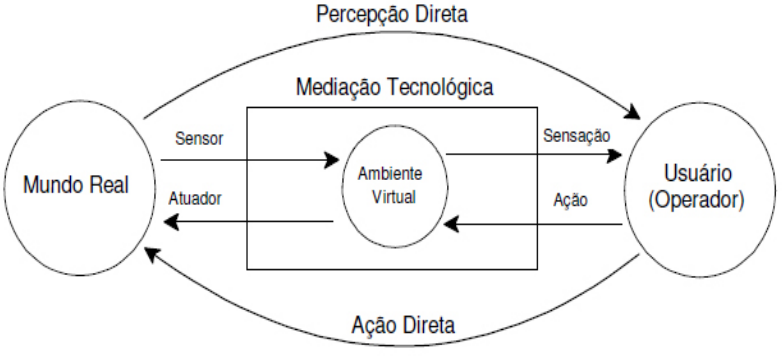

Figura 2.6: Esquema geral de comunicação de um Ambiente Virtual com os Usuários

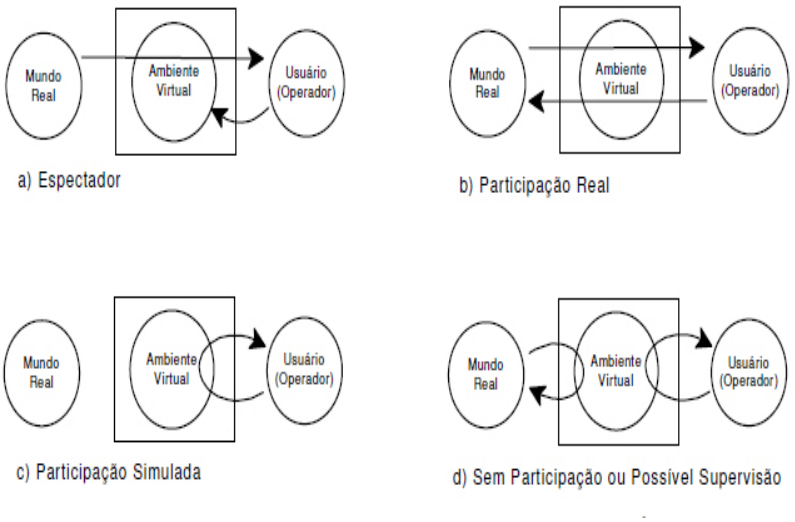

Figura 2.7: Os quatro métodos de comunicação de um único usuário com o Ambiente Virtual

da comunicação, os usuários utilizam um ambiente virtual para troca de informações; no caso de compartilhar o ambiente virtual, os usuários utilizam um ambiente virtual para interagir uns com os outros; e no caso de trabalhos cooperativos, os usuários cooperam entre si no mundo real, através de sua representação do mundo como um AVC. Também é possível essa cooperação utilizando uma representação de mundo imaginária, não tendo relação alguma com o mundo real.

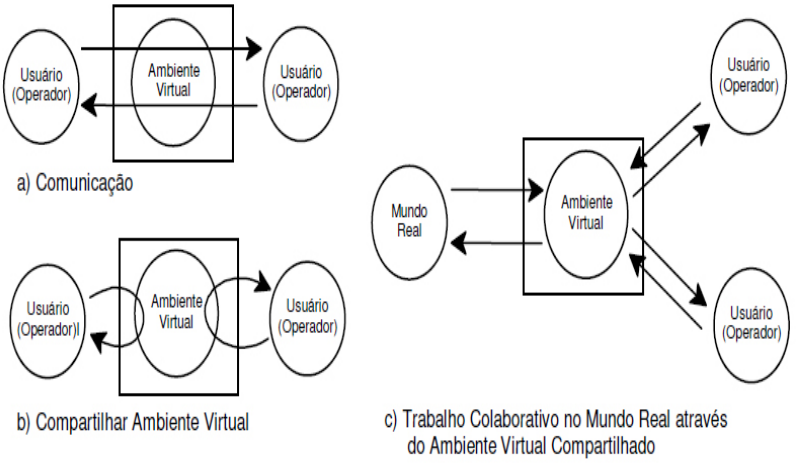

Figura 2.8: Os três métodos de comunicação de múltiplos usuários com o Ambiente Virtual

No próximo capítulo é descrito brevemente sobre as ferramentas que foram utilizadas neste

trabalho, o que foi feito com cada ferramenta e uma breve descrição do tipo de Ambiente Virtual utilizado.

Capítulo 3

Desenvolvimento

Para o desenvolvimento do módulo de simulação do SimBot foi utilizado o Unity 3D como ferramenta. Nele foi construído o ambiente virtual mostrado na Figura 3.1, que será descrito na seção 3.2. O ambiente virtual do SimBot interage com apenas um único usuário, e dessa forma é considerado um ambiente virtual com participação simulada (KIRNER; PINHO, 1997), pois além do ambiente poder representar um mundo fictício, o usuário interage apenas com a cena, não havendo interações entre o ambiente e um objeto existente no mundo real ou o usuário e os objetos reais.

O SimBot segue o Sistema de Realidade Virtual, e é considerado como não imersivo devido a falta de utilização de dispositivos de entrada e saída não convencionais. Para a entrada é utilizado o *mouse* e o teclado, e para saída um monitor. Para a simulação foi escolhido o sistema de visão em terceira pessoa, onde é possível controlar e ver o robô utilizado.

O sistema de física utilizado para a colisão e as interações entre os objetos e a cena é o PhysX. Como o PhysX está imbutido no próprio Unity, a sua utilização ocorre de forma oculta, portanto não é possível identificar exatamente quais os métodos do PhysX são utilizados.

3.1 Ferramentas

As ferramentas utilizadas foram o Unity3D, o Blender e o PhysX. O Unity (UNITY, 2014a) é um ambiente de desenvolvimento de jogos: um motor de renderização poderoso com um conjunto completo de ferramentas intuitivas, permitindo criar rapidamente conteúdos interativos em 2D e 3D. Permite facilmente publicar em diversas plataformas, possui muitos objetos já funcionais em sua loja online e uma comunidade compartilhando informações. Nesta loja é possível

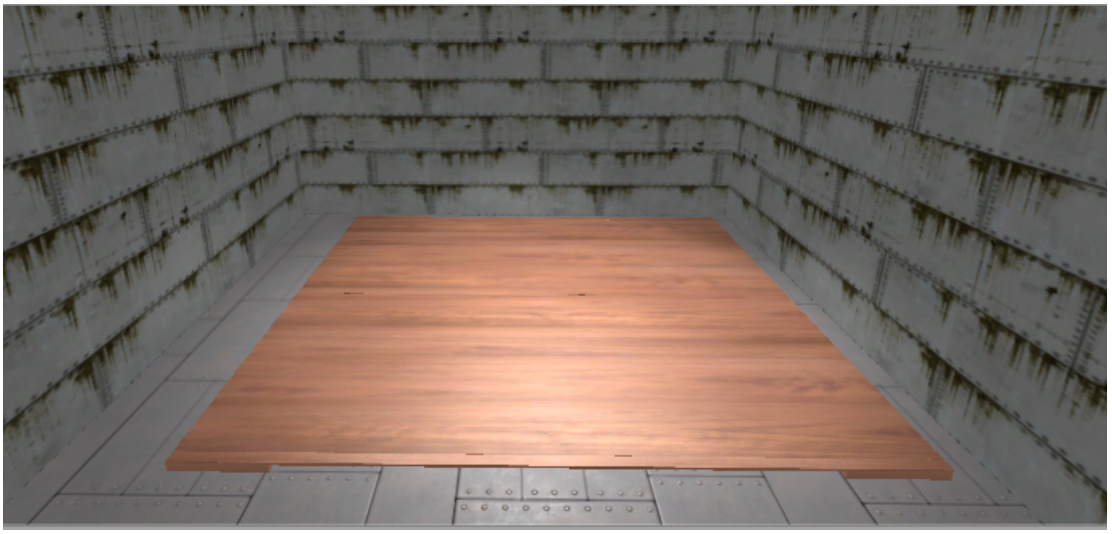

Figura 3.1: SimBot - Ambiente Virtual Laboratório

fazer *download*, pagando ou gratuitamente, de diversos objetos, texturas, sons, animações ou *scripts*, bastando apenas uma conta do utilizador.

O Blender (BLENDER, 2014) é uma ferramenta gratuita e *Open Source* para animações em 3D. Ela dá suporte a todos os passos da produção de animações em 3D, modelagem, criação das regras de movimentação dos modelos (*rigging*), simulação, renderização, composição e percepção de movimentação (*motion tracking*), e permite até edição de vídeo e criação de jogos. Algumas ferramentas são criadas por usuários mais avançados usando a API de codificação em Python, e essas são incluídas nas versões seguintes. O Blender beneficia pequenos estúdios e indivíduos que o utilizam devido aos seus processos de desenvolvimento de animações em 3D unificados e de seus processos responsivos.

PhysX SDK (KUMAR, 2013) é um motor de física que está em desenvolvimento desde 2004, na tentativa de vender PPU(s) (*Physical Processing Units*), placas semelhantes a GPU(s). Em 2008 a Nvidia obteve a licença do PhysX SDK e então o implementou para executar em cima da arquitetura CUDA para GPU(s) da própria Nvidia, e por isso a aceleração de GPU PhysX está disponível apenas para placas Nvidia. As principais características do PhysX são: Dinâmica de Corpos Rígidos, Controle de Personagem, Dinâmica de Veículos, Simulação de Partículas e Fluídos, Simulação de Roupas, Simulação de corpos moles e Detecção de Colisões, uma descrição de cada uma dessas características podem ser vistas no Apêndice A na seção A.1.

O Unity3D e o PhysX foram as ferramentas mais utilizadas neste trabalho. O Unity foi utilizado para criar o Ambiente Virtual, os botões de interface do usuário e a importação e movimentação do robô. O PhysX foi utilizado pelo Unity para cuidar da simulação, da gravidade e da colisão do robô com os objetos e o ambiente. O Blender foi utilizado apenas para modelar o objeto rampa, que então foi importado para o Unity e é usado para criar os obstáculos no modelador de mundos.

3.2 Desenvolvimento

Para a criação do ambiente virtual da Figura 3.1 foram utilizados seis planos, estes representam as paredes, o chão e o telhado, uma luz direcional apontando para baixo, representando uma lâmpada, e uma mesa pré-fabricada, encontrada na loja online do Unity.

Os objetos citados acima são aqueles que formam o ambiente virtual, não considerando os objetos para a interação com o usuário que podem ser vistos na Figura 3.2. Estes novos objetos são utilizados para interagir com o usuário. Os três botões na direita permitem adicionar um cubo, uma rampa ou uma esfera respectivamente.

Das funcionalidadas do PhysX, no SimBot são utilizados a Dinâmica de Corpos Rígidos, a detecção de colisão utilizando o componente *Mesh Collider*, e uma chamada *Raycasting*, descritas no Apêndice A e na seção A.2. A Dinâmica de Corpos Rígidos foi utilizada adicionando o componente de física *RigidBody* do Unity nos cubos, esferas e rampas que podem ser adicionados em cena e na mesa. Ela permite que os objetos tenham massa, que possam ser movidos pela gravidade ou por uma força externa e que possam quicar ao cair. A detecção de colisão impede que os objetos que possuem o componente *Mesh Collider* ocupem a mesma posição que outros objetos que tenham o mesmo componente. O *Raycasting* é um componente do Unity que permite simular um raio laser, e pode ser detectado quando o raio bate em qualquer objeto da cena que possua o componente *Mesh Colider* utilizando outro componente, o *RaycastHit*. Estes componentes são utilizados para adicionar os objetos na posição clicada, e para selecionar os objetos já adicionados em cena.

Nos cantos inferiores existem dois botões para a rotação da câmera, o da direita move a câmera para a direita e a rotaciona para poder ver o ambiente, o da esquerda faz o mesmo, porém para o lado contrário. Na Figura 3.3 é possível ver um objeto selecionado, em vermelho,

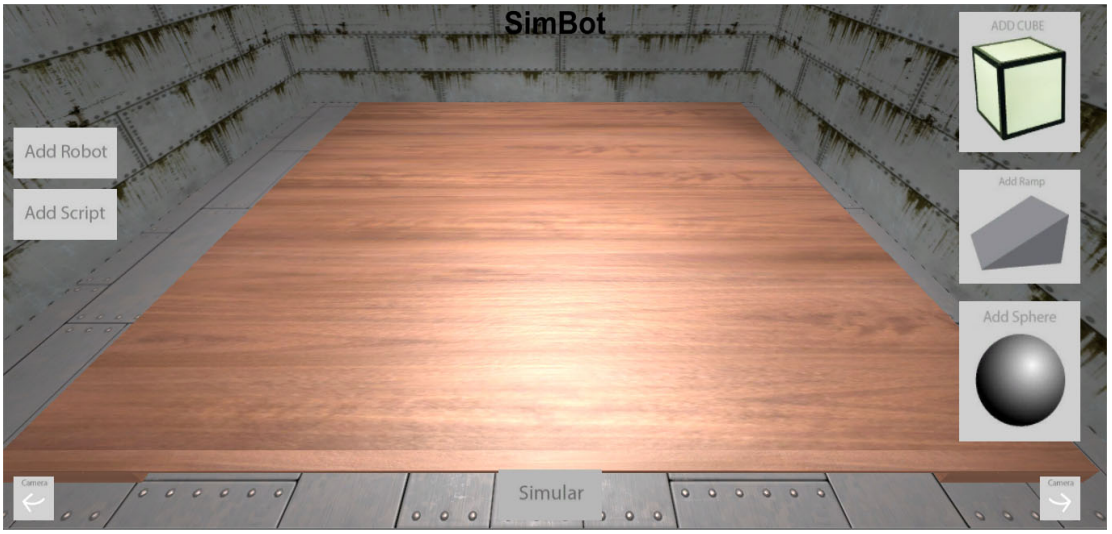

Figura 3.2: SimBot - Tela Inicial do Simulador de Mundos

e os outros em branco, e enquanto um objeto está selecionado, a interface do canto superior esquerdo aparece e permite que o utilizador mude o tamanho das três dimensões dos objetos e rotacione os objetos em apenas dois dos seus eixos. A rotação no eixo X foi ignorada pois quando ela ocorria, os objetos acabavam entrando no chão ou simplesmente sumindo de cena.

O botão "Simular" adiciona o robô na cena, que apenas por questões de teste utiliza o controlador de personagens em terceira pessoa nativo do Unity. Após adicionar o robô, começa então a simulação, retirando todos os botões da tela. Os botões da esquerda não estão sendo utilizados por enquanto. Futuramente, o botão "adicionar robô" permitirá importar um robô do modelador de Robôs para o criador de mundos e o utilizar na simulação, e o botão "adicionar *script*" fará com que um *script* com instruções de movimento para o robô seja importado e utilizado para movimentá-lo durante a simulação.

Quando a simulação começa, o usuário ganha controle do robô, e pode mover-se livremente pelo ambiente virtual. A sua movimentação ocorre utilizando as setas do teclado ou as teclas `W`, `A`, `S`, `D` para andar para frente, para a esquerda, para baixo e para a direita respectivamente. Na Figura 3.4 é mostrada a tela durante a simulação. Nela pode ser visto o controlador nativo do Unity e os objetos adicionados ao clicar nos botões de interação da Figura 3.2.

Para que este módulo não dependa do Unity para ser executado, foi necessário criar uma *build*, uma versão independente do Unity e construída para ser executada em uma plataforma

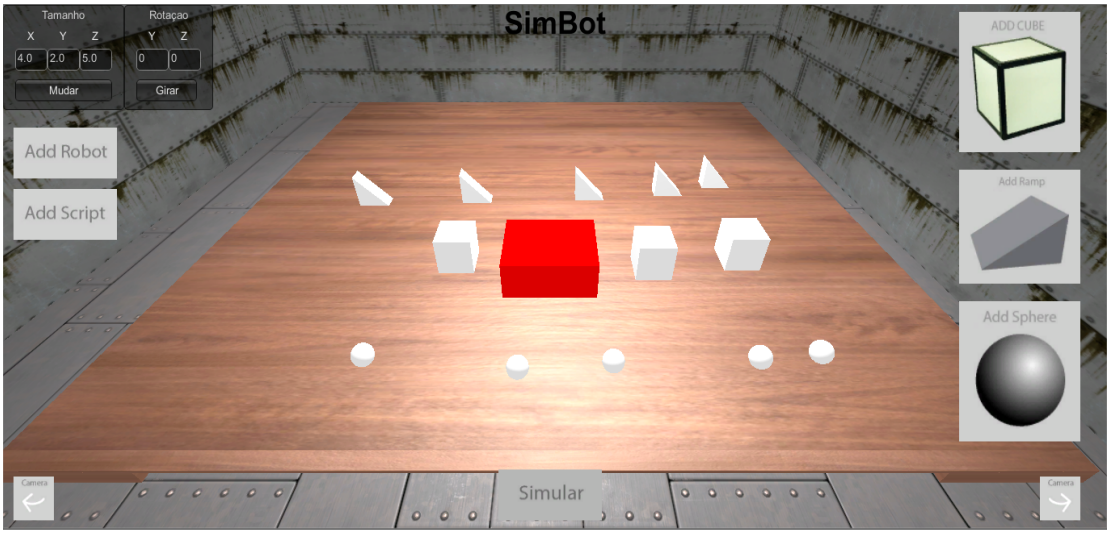

Figura 3.3: SimBot - Objeto selecionado, interface de mudança de tamanho

específica, que neste caso foi Windows. Porém, é possível fazer *builds* novas para plataformas diferentes, como Linux, Mac OS X, iOs, Android, Blackberry, Windows Phone, entre outros. Para criar estas *builds* basta acessar no Unity o menu *File*: *Build Settings*, ao abrir estas opções será então possível escolher para qual plataforma será criada a *build*, escolher quais cenas do projeto estarão disponíveis e escolher se a *build* será versão de desenvolvedor ou não.

3.2.1 Algoritmos

Nesta seção são explicados os principais algoritmos implementados durante o desenvolvimento do trabalho, os códigos podem ser vistos no Apêndice B. O código mostrado no Algoritmo 1 mostra como ocorre a seleção dos objetos. É criado um objeto *Raycast* e um *RaycastHit*. O objeto *Raycast* gera um raio da posição indicada, neste caso é escolhido o ponteiro do *mouse* como ponto de origem do raio. É verificado se a adição de objetos bases está ativada, caso não esteja é verificado se o objeto que é atingido pelo raio, indicado pelo *RaycastHit*, possui a *tag "Select"*, usada para identificar objetos selecionáveis. Se o objeto atingido possuir esta *tag*, sua cor será mudada para vermelha para mostrar que está selecionado como visto na Figura 3.3, então é ativada a variável showGUI para que a interface construída no Algoritmo 2 seja mostrada e mude a cor de todos os outros objetos de volta para branco. Também são coletadas as informações de tamanho e rotação do objeto selecionado e guardado numa variável para ser alterada pelo Algoritmo 2. Caso o *RaycastHit* não mostre um objeto com a *tag "Select"* e se

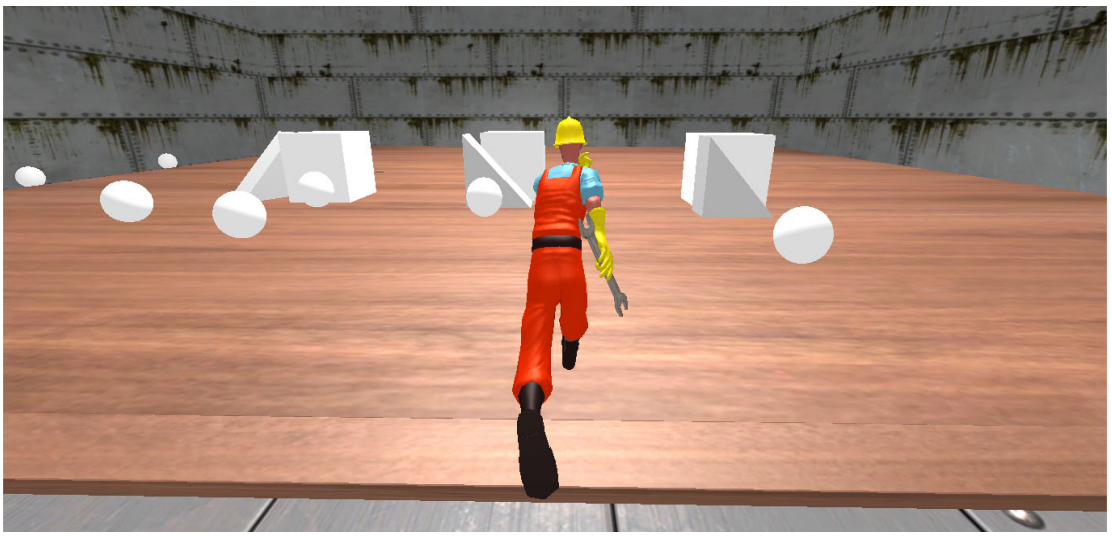

Figura 3.4: SimBot - Tela durante a simulação

o ponteiro não clicar em nenhum objeto de interação, então será desselecionado o objeto e o código do Algoritmo 2 não será mais mostrado e as variáveis de controle serão zeradas para evitar problemas.

O Algoritmo 2 adiciona a interface que permite modificar o tamanho e a rotação do objeto selecionado. Ele recebe os dados do Algoritmo 1 e são mostrados na interface, dois *containers* distintos, um contendo as informações do tamanho e o outro com as informações da rotação. A caixa do tamanho permite modificar as três dimensões e a rotação em apenas duas dimensões, e ao modificar os valores deve-se clicar nos botões de "Mudar" e "Girar" para efetuar as respectivas mudanças, esta interface pode ser vista na Figura 3.3.

O Algoritmo 3 coloca cada um dos objetos na cena. Devido aos objetos terem suas peculiaridades, foi necessário criar três algoritmos parecidos para adicioná-los e todos seguem a mesma ideia do Algoritmo 3 que adiciona um objeto. Este algoritmo utiliza o *Raycast* e o *RaycastHit* do mesmo modo como o Algoritmo 1, porém este requer que o botão "Add (Object)" seja clicado uma vez para então poder ser efetuada a adição dos objetos.

3.2.2 Construindo uma Cena

Ao iniciar o programa abre-se uma tela semelhante a mostrada na Figura 3.2. Para colocar os objetos em cena, é necessário clicar no botão "Add (Object)" sendo *Object* o objeto que queira colocar em cena. Ao clicar neste botão, é ativado o modo de colocar objetos. Neste

Algoritmo 1 : Algoritmo de seleção dos objetos em C#

```
void selection(){
 Cria Raycast do ponteiro até os objetos
 se clicar com o botão esquerdo do mouse
 {
   se Raycast atingiu algum objeto com Mesh Collider
   {
     se não está colocando objetos{
       se clicar em um objeto selecionavel{
       {
          Seleciona Objeto
          Mostra interface do algoritmo 2
          Desseleciona todos os outros objetos
          Obtem tamanho e rotação do objeto selecionado
          guarda o objeto selecionado numa variável extra

        } se clicar fora de algum objeto da interface{

          Desseleciona todos os outros objetos
          Esconde interface do algoritmo 2
          Zera todas as variáveis.
        }
      }
    }
  }
}
```

Algoritmo 2 : Algoritmo que adiciona a interface para modificar o tamanho e a rotação dos objetos em C#

```
void OnGUI () {
  se é pra mostrar a interface{
    Cria Container mais externo (invisivel)
      Cria conteiner externo visível: tamanho
      Cria o texto acima dos campos
      Cria os campos de texto e garante que a entrada só possa ser
      numérica com alguns caracteres especiais
      Cria Botão, e se ele for clicado, executa a função de mudança de
      tamanho

      Cria conteiner externo visível: rotaçao
      Cria o texto acima dos campos
      Cria os campos de texto e garante que a entrada só possa ser
      numérica com alguns caracteres especiais
      Cria Botão, e se ele for clicado, executa a função de rotação
    fecha o container mais externo
  }
}
```

Algoritmo 3 : Algoritmo que adiciona os objetos na posição indicada pelo *mouse* em C#

```
void Update () {
  Guarda posição do mouse em um vetor de 3 posições, setando a 3
  posição como 0.
    Se esse objeto não foi clicado antes{
      se clicar com o botão esquerdo do mouse e não foi a primeira vez
       que clicou no botão{
        Cria raio do ponteiro do mouse para o mundo.
        se até uma profundidade de 1000 houve algum acerto{
          atualiza posição guardada
        } se não {
          atualiza posição guardada para a posição da câmera.
        }
        cria objeto no mundo.
      }
      variável de controle para saber se foi a primeira vez que o botão
      foi clicado
  }
}
```

modo basta clicar em qualquer posição do ambiente virtual que não tenha nenhum botão da interface e o objeto será posicionado neste lugar, para parar, basta clicar no botão novamente. Não há necessidade de parar de colocar um objeto específico antes de mudar para o outro, ao mudar o objeto que deseja criar, o objeto antigo não será mais colocado em cena. Para facilitar a inserção destes objetos, foi criado um sistema que muda a câmera de posição, rotacionando-a na sala. Isso facilita a movimentação e a inserção de objetos atrás de outros objetos maiores. Essa rotação ocorre na posição da câmera, ela é jogada para um lado, e rotacionada para o lado contrário para que a visão esteja voltada para a mesa.

Após inserir objetos em cena, será então possível movê-los de lugar e selecioná-los. Para movê-los basta segurar o botão esquerdo do *mouse* no objeto e movê-lo para o lugar desejado, isto também selecionará o objeto. Para selecionar basta clicar com o botão esquerdo do *mouse* no objeto. Ao selecionar um objeto, ele é pintado de vermelho e no canto superior esquerdo aparecerá a interface para modificação do tamanho e rotação do objeto selecionado. Ao selecionar um segundo objeto, o primeiro voltará a ser branco, e todas as modificações vão afetar o segundo objeto. Os valores de tamanho mostrados na interface são os atuais do objeto, porém os valores de rotação serão sempre o valor padrão, 0.

Para desselecionar os objetos, deve-se clicar no cenário, que então fará o objeto voltar a ser branco e desativará a interface. É possível inserir novos objetos enquanto exista um objeto selecionado. Ao clicar no botão "Simular" o controlador em terceira pessoa padrão do Unity será colocado em cena sempre na mesma posição, e toda a interface desaparecerá, o objeto selecionado será desselecionado e o utilizador poderá controlar livremente as ações do controlador. Para a movimentação do controlador estão sendo usado os comandos padrões: as teclas `W, A, S, D` ou as setas do teclado para mover respectivamente para cima, para a esquerda, para trás e para a direita.

Capítulo 4

Considerações Finais

4.1 Problemas e Soluções

Durante o desenvolvimento do modelador de ambientes virtuais do SimBot ocorreram diversos problemas. O principal foi a mudança de ferramenta de desenvolvimento, anteriormente seria utilizada o Blender devido a facilidade de modelagem com ele, porém optou-se em mudar para o Unity pois facilitaria a junção dos três módulos do SimBot. O tempo levado para construir o ambiente virtual foi maior do que o esperado, devido a utilização de objetos que não existiam nas ferramentas nativas do Unity. Estes objetos foram a mesa e a rampa. Ao utilizar a loja do Unity, foi encontrado uma mesa gratuita que poderia ser utilizada no ambiente virtual, porém o mesmo não ocorreu para a rampa, portanto foi decidido modelar. Tentou-se utilizar o Unity para modelar o objeto rampa, mas ele não possui ferramentas de modelagem de objetos. Para contornar este problema, decidiu-se utilizar o Blender, ferramenta previamente estudada, para esta modelagem.

O desenvolvimento dos *scripts* utilizados ocorreu sem muitos problemas, porém não descobriu-se como fazer a adição dos *scripts* para movimentação do robô. Para não prejudicar o desenvolvimento foi decidido definir os comandos que o robô poderá executar durante a simulação, e então foi entregue o controle do robô diretamente ao usuário do projeto. Este mesmo problema aconteceu na tentativa de adicionar novos modelos de robôs ao ambiente, e para resolvê-lo foi utilizado um componente nativo do Unity, o controlador em terceira pessoa. Ambos os problemas ocorreram devido aos módulos Modelador de Robôs e Tradutor de comandos NXC estarem sendo desenvolvidos ao mesmo tempo por pessoas diferentes.

4.2 Trabalhos Futuros

Para trabalhos futuros, é interessante o aperfeiçoamento do sistema de inserir os obstáculos no ambiente virtual, permitindo mais facilmente modificar suas posições, seus tamanhos e suas rotações. Além disso, a criação de novos objetos para serem adicionados como obstáculos é uma funcionalidade que, eventualmente será necessária.

O módulo simulador atual possui apenas um ambiente virtual criado, portanto seria interessante a criação de novos ambientes virtuais de teste e um sistema que permita a troca entre estes ambientes. Não foi descoberto como adicionar robôs novos e *scripts* com instruções de movimentação para o robô na simulação no tempo disponível, porém o Unity possui uma função chamada `EditorUtility.OpenFilePanel` que permite abrir arquivos do computador do usuário e que pode ser vista no Manual do Unity (2014b) indicando que é possível fazê-lo, porém deve ser analisado como pode ser feito para cada plataforma alvo do SimBot. Também será necessária a implementação de uma interface que integre os três módulos do SimBot, citados na sessão 1.1. O Unity se mostrou uma plataforma complexa e muito poderosa para criação de aplicações em 3D, sendo uma ótima opção para desenvolver este sistema.

Apêndice A

Ferramentas

Neste apêndice, estão descritas com mais detalhes as funções do PhysX e do Unity utilizadas durante o desenvolvimento deste trabalho.

A.1 PhysX

Nesta seção são explicadas as principais características do PhysX: Dinâmica de Corpos Rígidos, Controle de Personagem, Dinâmica de Veículos, Simulação de Partículas e Fluídos, Simulação de Roupas, Simulação de corpos moles e Detecção de Colisões (NVIDIA, 2014).

Dinâmica de Corpos Rígidos: É o componente mais importante da simulação da física, é ela que utiliza os conceitos de posição, velocidade, aceleração, forças, momento, impulso, fricção, colisão, restrições e gravidade.

Controle de Personagem: É um tipo especial de controle, que é utilizado para controle de personagens em primeira ou terceira pessoa, mas pode ser usado para qualquer outro corpo cinético que se beneficie dessas propriedades.

Dinâmica de Veículos: Oferece a capacidade de simular a física de veículos utilizando formas esféricas que simulam modelos de fricção de pneus, e é usado um sistema de juntas para a suspensão dos veículos.

Simulação de Partículas e Fluidos: Simulação de Partículas permite criar diversos efeitos cinematográficos, partículas para fogo, escombros e descargas elétricas, enquanto a de fluidos é para simular líquidos, gases e fumaças.

Simulação de Roupas: É capaz de simular roupas de personagens ou qualquer objeto baseado em tecidos como bandeiras e cortinas, esses objetos podem ser esticados, destruídos e aplicados em outros corpos físicos.

Simulação de Corpos Moles: Permite simular a deformação volumétrica dos objetos.

Detecção de Colisões: Possibilita que dois objetos colidam e caso a cena seja controlada pela simulação física, impossibilita que dois objetos estejam no mesmo lugar ao mesmo tempo.

A.2 Unity

Aqui estão descritas as funções e os componentes do Unity que foram utilizados no desenvolvimento, estes são: *RigidBody*, *Mesh Collider*, *Raycast* e *Raycast Hit* (UNITY, 2014b).

Mesh Collider: Este componente pode ser adicionado a qualquer objeto, e possibilita a detecção de colisões entre objetos. Pode possuir diversas formas como representadas pelos componentes: *Box Collider*, *Capsule Collider*, *Wheel Collider*, *Sphere Collider* entre outros. Também pode utilizar um objeto modelado como colisor, por exemplo a rampa utilizada neste trabalho.

RigidBody: Este componente pode ser adicionado a qualquer objeto, e mesmo sem adicionar código algum, o objeto com este componente será puxado para baixo pela gravidade e reagirá às colisões com objetos em movimento se o *collider* certo estiver adicionado. Também permite aplicar forças aos objetos e controlá-los de forma realista.

Raycast: É uma função que cria um raio, utilizado para obter informações dos objetos atingidos por ele. Pode ser decidido aonde este raio será criado, a direção para onde ele vai ser gerado, a distância máxima até onde o raio irá e uma máscara que é utilizada para definir os objetos a serem ignorados pelo raio.

Raycast Hit: É uma função utilizada para receber as informações obtidas pelo *Raycast* ao atingir algum objeto. Por exemplo: qual é o objeto atingido, a distância entre o ponto de origem do raio e o objeto, o ponto onde o raio atingiu o colisor, entre outros.

Apêndice B

Códigos

Neste apêndice, estão os principais códigos utilizados neste trabalho. As variáveis *cube*, *ramp* e *sphere* utilizados nos Algoritmos 6, 7 e 8 contém as informações dos objetos respectivos.

Algoritmo 4 : Algoritmo de seleção dos objetos em C#

```
void selection(){
 ray = Camera.main.ScreenPointToRay(Input.mousePosition);
 if(Input.GetMouseButtonDown(0))
 {
   if(Physics.Raycast(ray,out hit,100))
   {
     if (!IsPlacing){
       if(hit.transform.tag == "Select")
       {
          hit.transform.tag = "Untagged";
          hit.transform.renderer.material.color = Color.red;
          showGUI = true;
          GameObject[]go = GameObject.FindGameObjectsWithTag("Select");
          foreach(GameObject choose in go){
            choose.renderer.material.color = Color.white;
          }
          hit.transform.tag = "Select";
          sizeX = ((int)hit.transform.localScale.x).ToString("F2");
          sizeY = ((int)hit.transform.localScale.y).ToString("F2");
          sizeZ = ((int)hit.transform.localScale.z).ToString("F2");
          rotateY = ((int)hit.transform.localRotation.y).ToString();
          rotateZ = ((int)hit.transform.localRotation.z).ToString();
          oldhit = hit;
        } else if (GUIUtility.hotControl == 0){
          GameObject[]go = GameObject.FindGameObjectsWithTag("Select");
          foreach(GameObject choose in go){
            choose.renderer.material.color = Color.white;
          }
          showGUI = false;
          hit = new RaycastHit();
          sizeX = "0";
          sizeY = "0";
          sizeZ = "0";
          rotateY = "0";
          rotateZ = "0";
        }
      }
    }
  }
}
```

Algoritmo 5 : Algoritmo que adiciona a interface para modificar o tamanho e a rotação dos objetos em C#

```
void OnGUI () {
  if (showGUI == true) {
    GUI.BeginGroup (new Rect (0, 0, 400, 400));
      GUI.Box (new Rect (0, 0, 150, 120), "Tamanho");
      GUI.Label (new Rect (30, 25, 25, 25), "X");
      GUI.Label (new Rect (70, 25, 25, 25), "Y");
      GUI.Label (new Rect (110, 25, 25, 25), "Z");
      sizeX = GUI.TextField (new Rect (15, 50, 40, 25), sizeX, 3);
      sizeX = Regex.Replace (sizeX, "[^0-9.]", "");
      sizeY = GUI.TextField (new Rect (55, 50, 40, 25), sizeY, 3);
      sizeY = Regex.Replace (sizeY, "[^0-9.]", "");
      sizeZ = GUI.TextField (new Rect (95, 50, 40, 25), sizeZ, 3);
      sizeZ = Regex.Replace (sizeZ, "[^0-9.]", "");
      if (GUI.Button (new Rect (15, 85, 120, 25), "Mudar")) {
        Vector3 changesize = new Vector3
        (float.Parse(sizeX), float.Parse(sizeY), float.Parse(sizeZ));
        ChangeSize(changesize);
      }
      GUI.Box (new Rect (150, 0, 110, 120), "Rotaçao");
      GUI.Label (new Rect (180, 25, 25, 25), "Y");
      GUI.Label (new Rect (220, 25, 25, 25), "Z");
      rotateY = GUI.TextField (new Rect (165, 50, 40, 25), rotateY, 3);
      rotateY = Regex.Replace (rotateY, "[^0-9]", "");
      rotateZ = GUI.TextField (new Rect (205, 50, 40, 25), rotateZ, 3);
      rotateZ = Regex.Replace (rotateZ, "[^0-9]", "");
      if (GUI.Button (new Rect (165, 85, 75, 25), "Girar")) {
        Vector3 changerotation = new Vector3
        (270 , int.Parse(rotateY) , int.Parse(rotateZ));
        Rotate(changerotation);
      }
    GUI.EndGroup ();
  }
}
```

Algoritmo 6 : Algoritmo que adiciona um cubo na posição indicada pelo *mouse* em C#

```
void Update () {
  Vector3 mousePos=new Vector3(Input.mousePosition.x,
   Input.mousePosition.y, 0f);
    if (temp == 1){
      if((Input.GetMouseButtonDown(0)) && (control == 1)) {
        Vector3 wordPos;
        Ray ray=Camera.main.ScreenPointToRay(mousePos);
        RaycastHit hit;
        if(Physics.Raycast(ray,out hit,1000f)) {
          wordPos=hit.point+nobounce;
        } else {
          wordPos=Camera.main.ScreenToWorldPoint(mousePos);
        }
        Instantiate(cube,wordPos,Quaternion.identity);
      }
      control = 1;
  }
}
```

Algoritmo 7 : Algoritmo que adiciona uma rampa na posição indicada pelo *mouse* em C#

```
void Update () {
  Vector3 mousePos=new Vector3(Input.mousePosition.x,
   Input.mousePosition.y, 0f);
    if (temp == 1){
      if((Input.GetMouseButtonDown(0)) && (control == 1)) {
        Vector3 wordPos;
        Ray ray=Camera.main.ScreenPointToRay(mousePos);
        RaycastHit hit;
        if(Physics.Raycast(ray,out hit,1000f)) {
          wordPos=hit.point+nobounce;
        } else {
          wordPos=Camera.main.ScreenToWorldPoint(mousePos);
        }
        Instantiate(ramp,wordPos,Quaternion.identity);
      }
      control = 1;
  }
}
```

Algoritmo 8 : Algoritmo que adiciona uma esfera na posição indicada pelo *mouse* em C#

```
void Update () {
  Vector3 mousePos=new Vector3(Input.mousePosition.x,
   Input.mousePosition.y, 0f);
    if (temp == 1){
      if((Input.GetMouseButtonDown(0)) && (control == 1)) {
        Vector3 wordPos;
        Ray ray=Camera.main.ScreenPointToRay(mousePos);
        RaycastHit hit;
        if(Physics.Raycast(ray,out hit,1000f)) {
          wordPos=hit.point+nobounce;
        } else {
          wordPos=Camera.main.ScreenToWorldPoint(mousePos);
        }
        Instantiate(sphere,wordPos,Quaternion.identity);
      }
      control = 1;
  }
}
```

Referências Bibliográficas

BLENDER. *Blender. The Software*. Julho 2014. Acessado em 10 out. 2014. Disponível em: <http://www.blender.org/about/>.

CHWIF, L.; MEDINA, A. C. *Modelagem e Simulação de Eventos Discretos: Teorias e Aplicações*. [S.l.]: São Paulo, 2010.

COSTA, P. J. C. G. da. *SimTwo - A Realistic Simulator for Robotics*. Setembro 2012. Acessado em 10 out. 2014. Disponível em: <http://paginas.fe.up.pt/~paco/pmwiki/index.php?n=SimTwo.SimTwo>.

HAGSAND, O. Interactive multiuser ves in the dive system. *IEEE Multimedia*, v. 3, n. 1, p. 30 – 39, Spring 1996.

HANSEN, J. *NXC Programmer's Guide.* Março 2013. Acessado em 10 out. 2014. Disponível em: <http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html>.

HOUAISS, I. A. *Dicionário Houaiss da Língua Portuguesa.* [S.l.]: Editora Objetiva, 2009.

KIRNER, C.; PINHO, M. S. Introdução à realidade virtual. *1º Workshop de Realidade Virtual São Carlos, SP*, Novembro 1997.

KUMAR, K. *Learning Physics Modeling with Physx: Master the PhysX 3 Physics Engine and learn how to program your very own physics simulation.* [S.l.]: Packt Publishing, 2013.

LAAS-CNRS. *Create your first simulation*. 2010. Acessado em 10 out. 2014. Disponível em: <http://www.openrobots.org/morse/doc/stable/user/beginner_tutorials/tutorial.html>.

LEGO. *LEGO Mindstorms*. março 2014. Acessado em 10 out. 2014. Disponível em: <http://mindstorms.lego.com>.

LOWOOD, H. E. *Virtual reality (VR)*. Novembro 2012. Acessado em 10 out. 2014. Disponível em: <http://global.britannica.com/EBchecked/topic/630181/virtual-reality-VR>.

NVIDIA. *Nvidia PhysX*. Março 2014. Acessado em 10 out. 2014. Disponível em: <http://www.nvidia.com.br/object/physx_faq_br.html>.

ROBOMATTER. *Robot Virtual World*. 2005. Acessado em 10 out. 2014. Disponível em: <http://www.robotvirtualworlds.com/>.

ROBOMATTER. *RobotC a C Programming Language for Robotics*. 2005. Acessado em 10 out. 2014. Disponível em: <http://www.robotc.net/>.

RODELLO, I. A. et al. Sistemas distribuídos de realidade virtual e aumentada. In: SISCOUTTO, C. K. R. (Ed.). *Realidade Virtual e Aumentada: Conceitos, Projetos e Aplicações*. [S.l.]: Editora SBC - Sociedade Brasileira de Computação, 2007. cap. Capítulo 7, p. 129 – 150.

TORI, R.; KIRNER, C.; SISCOUTTO, R. A. *Fundamentos e tecnologia de realidade virtual e aumentada*. [S.l.]: Editora SBC, 2006.

UNITY. *Unity Game Engine*. Março 2014. Acessado em 10 out. 2014. Disponível em: <http://portuguese.unity3d.com>.

UNITY. *Unity Manual*. Setembro 2014. Acessado em 20 set. 2014. Disponível em: <http://docs.unity3d.com/Manual/>.