



UNIVERSIDADE ESTADUAL PAULISTA

Faculdade de Ciências

Departamento de Computação

Campus de Bauru



TRABALHO DE CONCLUSÃO DE CURSO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOGO ELETRÔNICO BASEADO EM RPG DE MESA, COM USO DO MOTOR DE JOGO UNITY PARA A PLATAFORMA DE MULTIPROJEÇÃO MINICAVE

Aluno: Alexandre Salvador Fernandes

Orientador: Prof. Dr. Jose Remo Ferreira Brega

Coorientador: Prof. Me. Mário Popolin Neto

Alexandre Salvador Fernandes

**JOGO ELETRÔNICO BASEADO EM RPG DE MESA, COM USO DO MOTOR DE
JOGO UNITY PARA A PLATAFORMA DE MULTIPROJEÇÃO MINICAVE**

Trabalho de conclusão de curso de graduação
apresentado à Faculdade de Ciências da
Universidade Estadual Paulista “Júlio de
Mesquita Filho” como requisito para obtenção
do título de Bacharel em Ciência da
Computação

Orientador: Prof. Dr. José Remo Ferreira Brega
Coorientador: Prof. Me. Mário Popolin Neto

UNESP

2016

Fernandes, Alexandre Salvador.

Jogo eletrônico baseado em RPG de mesa, com uso do motor de jogo Unity para a plataforma de multiprojeção MiniCAVE / Alexandre Salvador

Fernandes, 2016

45 f. : il.

Orientador: José Remo Ferreira Brega

Monografia (Graduação)-Universidade Estadual Paulista. Faculdade de Ciências, Bauru, 2016

1. MiniCAVE. 2. Unity. 3. RPG. I. Universidade Estadual Paulista. Faculdade de Ciências. II. Título.

Alexandre Salvador Fernandes

**JOGO ELETRÔNICO BASEADO EM RPG DE MESA, COM USO DO MOTOR DE
JOGO UNITY PARA A PLATAFORMA DE MULTIPROJEÇÃO MINICAVE**

Trabalho de conclusão de curso de graduação
apresentado à Faculdade de Ciências da
Universidade Estadual Paulista “Júlio de
Mesquita Filho” como requisito para obtenção
do título de Bacharel em Ciência da
Computação

Banca Examinadora

Prof. Adjunto José Remo Ferreira Brega
Faculdade de Ciências
Universidade Estadual Paulista - Bauru
Presidente da Banca

Prof. Adjunto Antonio Carlos Sementille
Faculdade de Ciências
Universidade Estadual Paulista - Bauru

Prof. Adjunto Aparecido Nilceu Marana
Faculdade de Ciências
Universidade Estadual Paulista - Bauru

UNESP

2016

Dedico este trabalho em memória de meus avós paternos, Netícia Zacaria Fernandes e Manoel Fernandes.

AGRADECIMENTOS

Agradeço aos meus pais Narciso Valdir Fernandes e Eunice Maria Piasentin Salvador Fernandes e irmã Flávia Teresa Salvador Fernandes por sempre me apoiarem incondicionalmente e à minha irmã pela ajuda durante a fase pré-vestibular, na escolha do curso de Ciência da Computação.

Ao Professor José Remo Ferreira Brega e Mário Popolin Neto, pela orientação e ajuda neste projeto, sem os quais a conclusão deste trabalho não seria possível.

Aos meus colegas do LTIA - Laboratório de Tecnologia da Informação Aplicada - pelos mais diversos projetos, desde competições - Imagine Cup, Global Game Jam - à projetos de TCC e pelas amizades feitas.

Ao Professor Eduardo Martins Morgado, responsável pelo LTIA, pela oportunidade de fazer parte de um grande laboratório, que me proporcionou importantes experiências e inúmeros projetos.

Ao PET BSI - Programa de Educação Tutorial - pela oportunidade de participar deste projeto de extensão, auxiliando no aprimoramento dos meus conhecimentos quanto ao desenvolvimento de jogos eletrônicos e pelo apoio financeiro na modalidade bolsa.

Aos meu amigos da turma de 2012 do Bacharelado em Ciência da Computação que sempre estiveram ao meu lado nesta caminhada, em especial a Henrique de Sousa Sumitomo, colega de apartamento nos últimos três anos que sempre pude contar com a ajuda.

Ao meu amigo Caetano Fidalgo de Lima, companheiro de longa data, que mesmo tendo tomado caminhos diferentes sempre foi alguém que pude contar, independente da distância.

À minha namorada, Juliana Montenegro Brasileiro, pelo companheirismo, por sempre estar ali para me ouvir, por me ajudar a manter a calma e por ser a mulher mais especial que já conheci.

À todos colegas, amigos e familiares não citados diretamente, mas que contribuíram e me apoiaram.

“No que diz respeito ao empenho, ao compromisso, ao esforço, à dedicação, não existe meio termo. Ou você faz uma coisa bem feita ou não faz.” - Ayrton Senna da Silva.

RESUMO

Este projeto trata da criação de um protótipo de jogo eletrônico, seguindo o modelo de *Role-playing Game* (RPG), cuja plataforma foi o ambiente de multiprojeção MiniCAVE, desenvolvido com a ajuda do motor de jogo Unity, para que utilizasse esta tecnologia imersiva da melhor forma possível, proporcionando uma experiência única ao jogador. Por meio da análise das características da MiniCAVE, em conjunto com os principais atributos de um jogo de RPG, foi possível aferir que o ambiente de multiprojeção atende aos requisitos de um RPG, sendo possível utilizar todos os seus recursos. Ao final do teste multiprojetado, isto é, execução do protótipo dentro do ambiente da MiniCAVE, foi possível averiguar seu efeito imersivo.

PALAVRAS-CHAVE: MiniCAVE, UNITY, RPG.

ABSTRACT

This project deals with the creation of a game prototype, following the role-playing game model, whose platform was the multiprojection environment MiniCAVE, developed with the support of Unity game engine, using the best that this immersive technology can offer, providing a unique experience to the player. By analyzing the MiniCAVE characteristics, together with the key attributes that make up a role-playing game, it was possible to assess that the multiprojection environment meets the requirements of a RPG, being feasible to use all of its features. At the end of the multiprojection test, where the prototype was executed in the MiniCAVE platform, it was possible to inquire the immersion effect caused by the platform.

KEY-WORDS: MiniCAVE, Unity, RPG.

LISTA DE FIGURAS

Figura 1 - A interface do editor Unity	16
Figura 2 - O componente NetworkView	17
Figura 3 - O Unity Cluster Package	18
Figura 4 - Disposição do Unity Cluster Package para o MiniCAVE	19
Figura 5 - Arquivo de configuração XML do nó escravo Slave #3	20
Figura 6 - O MiniCAVE: AG (sete PCs e um switch), três telas e seis projetores com lentes polarizadas.....	21
Figura 7 - Fluxo do jogo.....	24
Figura 8 - Painel de atributos.....	25
Figura 9 - Cena do jogo	26
Figura 10 - Avatar do jogador	27
Figura 11 - Descrição da classe Player dentro do painel Inspector	28
Figura 12 - Método GetHandInBag, responsável por retornar a mão utilizada pelo jogador para trocar de item	30
Figura 13 - CloseCollider, gatilho criado para detectar a presença de objetos a curto alcance	31
Figura 14 - FarCollider, gatilho criado para detectar objetos distantes.....	31
Figura 15 - Gatilho responsável por detectar qualquer um dos braços do jogador suspenso à frente do corpo (ao centro)	32
Figura 16 - Colliders utilizados para detecção do movimento de troca de item	33
Figura 17 - Conjunto de seis triggers, cuidando da detecção do ataque à curta distância...34	
Figura 18 - Conjunto de três gatilhos, responsáveis pela detecção do movimento de arremesso	36
Figura 19 - Uso do protocolo OSC dentro da interface de programação do GlovePie	38
Figura 20 - Trecho de código da classe OSCReceiver	39
Figura 21 - Emulação de comandos de mouse e teclado via Wii remote com auxílio do GlovePie	39
Figura 22 - Cena apresentada pelo projeto-exemplo do Kinect Wrapper Package for Unity3D	40
Figura 23 - Teste de execução no ambiente multiprojetado.....	42
Figura 24 - Diagrama de classes.....	47

LISTA DE TABELAS

Tabela 1 - Descrição dos painéis do editor unity.....	16
---	----

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Objetivos.....	13
1.1.1	Objetivo Geral	13
1.1.2	Objetivos Específicos	13
1.2	Organização da Monografia.....	14
2	DESCRIÇÃO DAS TECNOLOGIAS	15
2.1	Unity	15
2.1.1	O Editor	15
2.1.2	O Módulo de Rede	17
2.2	Unity Cluster Package.....	18
2.2.1	<i>Multi Projection Camera</i>	19
2.2.2	<i>Node Manager</i>	19
2.3	MiniCAVE	20
2.4	GlovePie.....	21
2.5	Open Sound Control	22
3	CAVE RPG	23
3.1	<i>Role Playing Game - RPG</i>	23
3.2	Fluxo do Jogo	24
3.3	Gerenciamento de jogadores	27
3.4	Características	28
3.5	Mochila e itens	29
3.6	Gerenciamento de Ações.....	29
3.6.1	Pegar e Soltar	32
3.6.2	Troca de item com a mochila	33
3.6.3	Ataque à Curta Distância	33
3.6.4	Arremesso	35
3.6.5	Tiro e uso de poção	36
3.6.6	Andar.....	37
3.7	Comunicação dos controladores	37
3.7.1	Wii Remote	37
3.7.2	Kinect	40
3.8	Gerenciamento de turnos.....	41

3.9	Multiprojeção.....	41
4	TESTE DE FUNCIONAMENTO	41
5	RESULTADOS	43
6	CONCLUSÃO.....	44
6.1	Trabalhos Futuros.....	44
	REFERÊNCIAS	45
	APÊNDICE A - Diagrama de classes.....	47

1 INTRODUÇÃO

A área dos jogos eletrônicos vem ganhando adeptos e tem crescido gradativamente a cada ano, apresentando um aumento de 9% de 2014 a 2015 no mercado global (Newzoo, 2015). No Brasil, este crescimento foi de 60% entre 2011 e 2012 (LANNOY, 2013). Devido a este franco crescimento, e a potencialidade deste mercado, a demanda por tecnologias cada vez mais realistas e imersivas aumenta, resultando em dispositivos como o Oculus Rift, que faz com que o jogador tenha um ponto de vista totalmente imerso dentro do *game*, e o Kinect, que é um dispositivo capaz de detectar os movimentos do corpo do jogador, a fim de criar uma interação mais realista com os jogos.

Neste projeto, foi criado um jogo eletrônico baseado no estilo de jogo conhecido como *Role-playing Game* ou simplesmente RPG, que é um gênero cuja característica principal é a liberdade do jogador quanto ao modo e a variedade com que se podem executar ações.

A tecnologia aplicada neste jogo foi a MiniCAVE, que consiste de um ambiente de multiprojeção com três grandes telas, tendo como dispositivos de interação o sensor Kinect e o controle de Nintendo Wii (Wii Remote), que atuarão de forma complementar, intensificando a imersão no jogo, uma vez que os comando são bastante intuitivos.

1.1 Objetivos

1.1.1 Objetivo Geral

Produzir um protótipo de jogo eletrônico baseado em RPG de mesa para o ambiente de multiprojeção MiniCAVE, permitindo que o jogador edite suas habilidades, locomova-se dentro de um ambiente virtual e confronte outros participantes, interagindo com o Kinect e o Wii Remote.

1.1.2 Objetivos Específicos

São os objetivos específicos deste projeto:

- a. Permitir que o jogador edite pelo menos seis habilidades ao iniciar o jogo.
- b. Criar ao menos quatro ações dentro do gênero RPG que utilizem o Kinect e Wii Remote em conjunto.
- c. Fazer a execução do protótipo dentro do ambiente da MiniCAVE.
- d. Possibilitar a participação de mais de um jogador.

1.2 Organização da Monografia

Este documento está dividido em seis capítulos, iniciando-se a partir deste capítulo de introdução (Capítulo 1) e seguindo-se como é descrito abaixo:

- Capítulo 2, **Descrição das Tecnologias**: apresentação de cada uma das tecnologias utilizadas na execução do trabalho.
- Capítulo 3, **CAVE RPG**: descrição de todo o desenvolvimento do jogo.
- Capítulo 4, **Teste de Funcionamento**: descrição do teste de execução realizado em uma adaptação da MiniCAVE.
- Capítulo 5, **Resultados**: apresentação dos resultados obtidos com o teste de funcionamento.
- Capítulo 6, **Conclusão**: conclusão a respeito dos resultados deste trabalho.

2 DESCRIÇÃO DAS TECNOLOGIAS

Este capítulo apresenta uma descrição de cada uma das tecnologias que foram utilizadas para o desenvolvimento deste trabalho, mostrando suas funcionalidades, principais características e aplicações.

2.1 Unity

“Unity é uma plataforma de desenvolvimento flexível e eficiente, usado para criar jogos e experiências interativas 3D e 2D em multiplataforma. É um ecossistema completo para todos que queiram montar um negócio de criação de conteúdo.” (UNITY, 2015a)

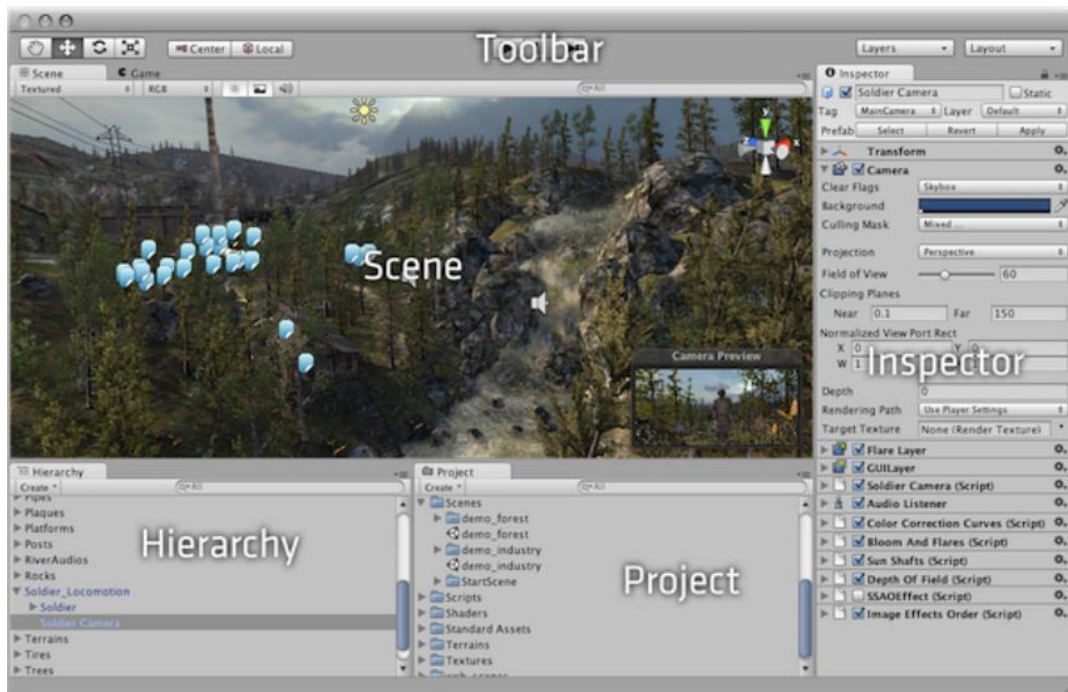
Entre os benefícios de se utilizar o motor de jogo Unity é sua documentação, composta por explicações de codificação, desde nomenclatura até funções, e a comunidade de desenvolvedores, incluindo tutoriais e até projetos prontos para estudo. (EDUARDO, 2015)

Muitos jogos já foram criados utilizando este motor de jogo, tais como *Endless Legend*, que é um jogo de estratégia baseado em turnos, para as plataformas Windows e Mac, e *Crossy Road*, um jogo casual simples onde a mecânica gira em torno de atravessar infinitas ruas com o seu personagem, sendo disponível para as plataformas móveis iOS e Android. (UNITY, 2015b)

2.1.1 O Editor

O motor de jogo Unity possui um grande suporte à construção de ambientes virtuais, por meio do mecanismo *drag-and-drop*. Quanto à sua interface, ela é altamente configurável, pois permite ao usuário modelar seus painéis da maneira que bem entender, ao mesmo tempo em que possui diversos layouts pré-definidos para escolher. (POPOLIN NETO, 2014)

Figura 1 - A interface do editor Unity.



Fonte: UNITY (2015c).

Tabela 1 - Descrição dos painéis do editor Unity.

Painel	Descrição
Project	Por meio deste painel, pode-se acessar e gerenciar os recursos pertencentes ao projeto Unity, localizados no diretório escolhido ao se criar o projeto, apresentando tais recursos e as árvores de diretórios nos quais estes estão armazenados
Scene	Onde são criadas os AVs, dispondo os recursos disponíveis no painel Project por meio do “arrastar e soltar”. Pode-se navegar neste painel de forma livre fazendo uso de teclado e mouse, visualizando e editando o posicionamento dos objetos dentro do AV.
Hierarchy	Apresenta todos os objetos contidos no AV, ou seja, os dispostos no painel Scene. Por meio deste, é possível o agrupamento de objetos, criando um sistema de coordenadas local para os objetos filhos com origem global do objeto do pai.
Game	Permite a visualização e interação com a execução do que vem sendo criado no painel Scene.
Toolbar	Consiste em controles básicos, como opções de navegação e posicionamento dos objetos para o painel Scene, e botões para execução, pausa e quadro por quadro para o painel Game.
Inspector	Ao selecionar qualquer objeto no painel Project, Scene ou Hierarchy, é apresentado no painel Inspector os componentes (malhas, textura, sons, animações, <i>scripts</i> , entre outros) do objeto selecionado. Pode-se alterar as propriedades dos componentes apresentados, visualizando tais alterações na edição do AV no painel Scene, ou em tempo execução no painel Game.

Fonte: POPOLIN NETO (2014).

A Figura 1 mostra todos os painéis disponíveis na interface do motor de jogo Unity, enquanto que na Tabela 1 cada um dos painéis seguem descritos.

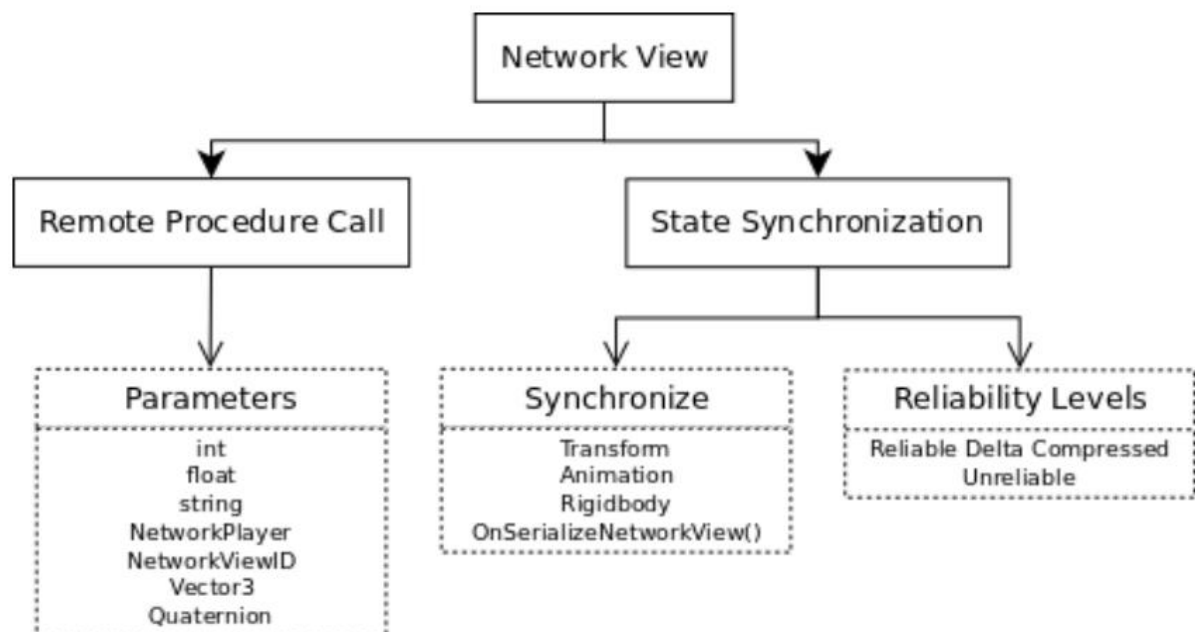
Cada objeto da cena tem sua lógica feita por meio de *scripts*, sendo aceitas atualmente três linguagens: UnityScript (algo próximo a JavaScript), C# e Boo. Todo e qualquer *script* criado na Unity deriva da classe *MonoBehaviour*, não é possível utilizar como componente de um objeto um *script* que não derive desta classe.

O editor também permite que se criem *prefabs*, uma espécie de arquivo que armazena objetos e seus componentes, a fim de reutilizá-los em seu jogo, seja por *drag-and-drop* durante a construção do AV ou através da instanciação de novos objetos em tempo de execução através de *scripts*.

2.1.2 O Módulo de Rede

O módulo de rede do motor de jogo Unity, possui uma classe principal chamada *Network*, que segue um modelo Cliente e Servidor. Esta classe possui métodos para inicialização e conexão com servidor. Para objetos cujos dados necessitam ser compartilhados na rede, há o componente *NetworkView*, onde deve-se decidir que tipo de dado será enviado e de qual forma. (UNITY, 2015c)

Figura 2 - O componente NetworkView.



Fonte: POPOLIN NETO (2014).

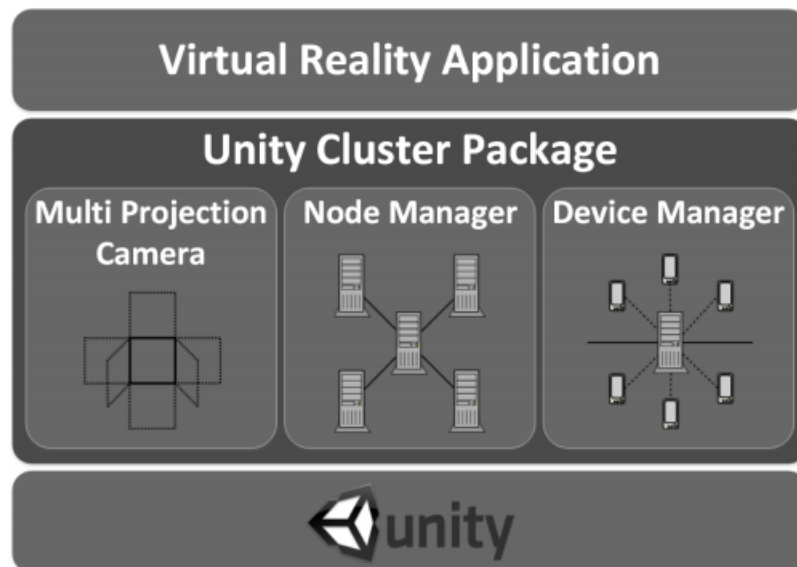
É possível também executar funções em outras instâncias através de Remote Procedure Call (RPC) - chamadas remotas de procedimento, para situações onde não há necessidade dos dados serem enviados diretamente. (UNITY, 2015c).

Ainda dentro da classe NetworkView (Figura 2), quando faz-se o envio de dados por meio de sincronização de estado, tem duas opções: *Reliable Delta Compressed*, onde os pacotes são recebidos sempre na ordem em que são enviados, porém apenas as diferenças de um estado anterior para o outro serão enviados, não havendo transmissão de dados caso não existam mudanças; e *Unreliable*, onde todo o estado é transmitido e há um uso maior da banda. (UNITY, 2015c)

2.2 Unity Cluster Package

Desenvolvido por Popolin Neto, M. (2014), o Unity Cluster Package é um pacote com a implementação de componentes cujo objetivo é facilitar o desenvolvimento de aplicações RV por meio do drag-and-drop, por meio do uso de prefabs.

Figura 3 - O Unity Cluster Package.



Fonte: POPOLIN NETO (2014).

Como pode ser visto na Figura 3, dentre os componentes que compõem o Unity Cluster Package, estão o *Multi Projection Camera* e o *Node Manager*, descritos nas seções a seguir. O terceiro componente, *Device Manager*, responsável pela captura dos dados do Kinect e do Wii Remote não foi utilizado neste projeto.

2.2.1 Multi Projection Camera

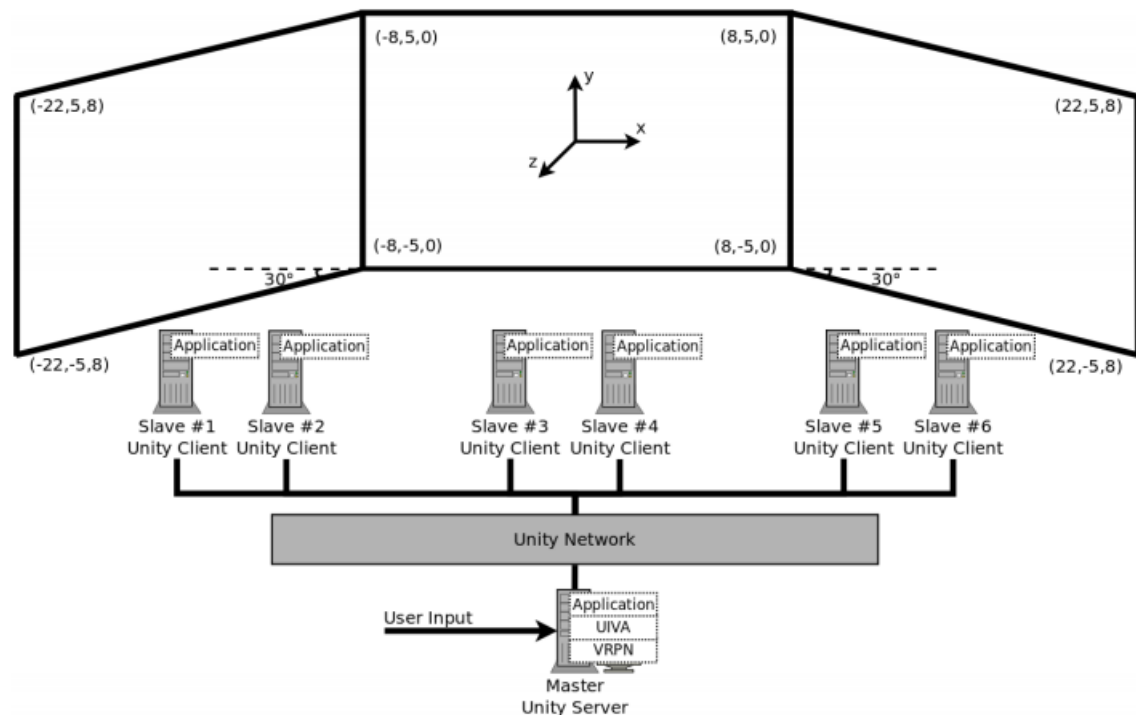
Como é descrito por Popolin Neto, M. (2014), a *Multi Projection Camera* é uma câmera virtual Unity customizada que implementa a função *PreRender()* herdada da classe base *MonoBehaviour*, a matriz de projeção de perspectiva generalizada de Kooima (2015), que para tal utiliza os pontos Pa, Pb, Pc e Pe.

Portanto, esta classe é responsável por situar a cabeça do jogador dentro da cena, possuindo também dois objetos - *ProjectionPlane* e *UserHead* - que utilizam os pontos Pa, Pb, e Pc citados anteriormente para posicionar o plano de projeção e o ponto restante Pe para posicionar a cabeça em relação ao plano.

2.2.2 Node Manager

Como é explicado por Popolin Neto, M. (2014), o Node Manager inicializa a aplicação de acordo com o tipo do nó, onde cada nó escravo (Game Client) é conectado na aplicação de nó mestre (Game Server)..

Figura 4 - Disposição do Unity Cluster Package para o MiniCAVE.



Fonte: POPOLIN NETO (2014).

A Figura 4 ilustra a conexão dos nós escravos com o nó mestre descrito anteriormente.

Ainda de acordo com Popolin Neto, M., é a partir do *Node Manager* que se instancia o prefab *Multi Projection Camera*, através da função *Instantiate()*, que é fornecida pelo módulo de rede do motor de jogo Unity. Por conta disso, cada um dos nós escravo possui uma instância da *Multi Projection Camera*, cuja posição e rotação é sincronizada com as alterações ocorridas no nó mestre, por meio da funcionalidade *StateSynchronization* do componente *NetworkView*.

O Node Manager acompanha também um arquivo de configuração no padrão XML.

Figura 5 - Arquivo de configuração XML do nó escravo Slave #3.

```
<node type="slave">
  <server ip="192.168.1.13" port="25000"/>
  <screen stereo="true" eye="left">
    <pa x="-8" y="-5" z="0"/>
    <pb x="8" y="-5" z="0"/>
    <pc x="-8" y="5" z="0"/>
    <pe x="0" y="0" z="5"/>
  </screen>
</node>
```

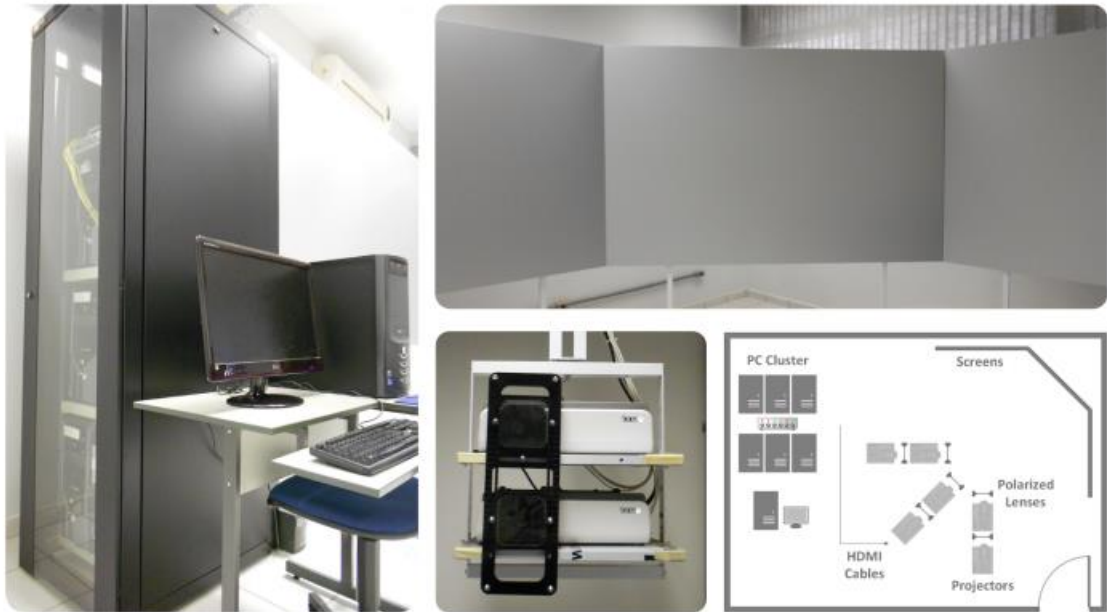
Fonte: POPOLIN NETO (2014).

A Figura 5 mostra o arquivo de configuração do nó Slave #3, que é atribuído à tela central com os pontos $P_a = (-8, -5, 0)$, $P_b = (8, -5, 0)$ e $P_c = (-8, 5, 0)$. (POPOLIN NETO, 2014)

2.3 MiniCAVE

O MiniCAVE é um sistema de multiprojeção de baixo custo utilizado em aplicações de realidade virtual imersiva com três telas (2.5m x 1.5m cada) dispostas em um ângulo de 30°. Para cada uma das telas tem-se dois projetores (BenQ W1000 HD) com lentes polarizadas conectados a dois PCs dedicados (Intel Core i7 8GB RAM e NVIDIA FX 1800), dando suporte a estereoscopia passiva. (DIAS, 2011)..

Figura 6 - O MiniCAVE: AG (sete PCs e um switch), três telas e seis projetores com lentes polarizadas.



Fonte: POPOLIN NETO (2014).

Cada um dos componentes podem ser observados na Figura 6.

Por se tratar de um sistema utilizado para realidade virtual, há a possibilidade (se as aplicações desenvolvidas para esta plataforma permitirem) de que o usuário visualize o ambiente de qualquer ponto de vista, assim como se movimentar e interagir com qualquer objeto virtual, ao modo que o computador detecta e reage às suas ações, modificando o ambiente, tentando se aproximar o máximo possível da realidade. (DIAS, 2011)

2.4 GlovePie

GlovePie (*Glove Programmable Input Emulator*) é um middleware utilizado tanto como um emulador quanto para a criação de macros. Apesar do que o nome indica, não é exclusivo para luvas de Realidade Virtual, apenas começou como um sistema para emular entradas de *Joystic* e teclado usando a *Essential Reality P5 Glove*. Atualmente há suporte para todos os tipos de dispositivos, conseguindo controlar inclusive saídas do tipo MIDI e OSC. (KENNER, 2010)

A interface do GlovePIE permite que o usuário carregue ou crie um código simples. Um exemplo básico para controlar as teclas W, A, S, e D com uma luva seria:

$$W = \textit{glove}.z > -50\textit{cm}$$

$$S = \textit{glove}.z < -70\textit{cm}$$

$$A = \textit{glove}.x < -10\textit{cm}$$

$$D = \textit{glove}.x > 10\textit{cm}$$

2.5 Open Sound Control

Desenvolvido por Matt Wright e Adrian Freed, OSC (Open Sound Control) é um protocolo para controle de transporte de dados. Este sistema define o tipo de dado transportado e gerencia o fluxo desses tipos de dados. Da mesma forma que outros protocolos de transporte, OSC permite a comunicação entre computadores e outros dispositivos de mídia, assim como a comunicação entre programas rodando em uma mesma máquina. Como o nome sugere, este protocolo foi desenvolvido para transportar dados entre aplicações multimídia, mas serve também para outros tipos de programas (PHILLIPS, 2008).

3 CAVE RPG

Neste capítulo será apresentado uma descrição do tema RPG e o desenvolvimento do jogo eletrônico feito, ou seja, a forma como as tecnologias apresentadas no capítulo anterior foram utilizadas, de modo a descrever a dinâmica do jogo e cada um de seus componentes, classes e objetos.

3.1 *Role Playing Game* - RPG

O tema do jogo eletrônico foi baseado em um estilo de jogo conhecido como RPG de mesa, que tem uma estrutura modelada em turnos, cujas regras e ações serão reproduzidas em diversos pontos deste projeto.

Como o RPG de mesa possui diversos formatos, foi usado para consulta o livro de regras do *Dungeons & Dragons*, por ser um dos mais tradicionais e conhecidos. De acordo com o site da empresa, a primeira versão foi publicada 1976 e tem inspirado diversos jogos em diferentes mídias desde então. A empresa disponibiliza gratuitamente dois documentos com as regras do jogador e do mestre. (DUNGEONS & DRAGONS, 2015)

O RPG é um estilo de jogo onde há um aspecto muito forte de imersão, uma vez que o jogador coloca-se no lugar do personagem , criado por ele mesmo.

É importante dizer que esse é um jogo onde os jogadores participantes assumem papéis de personagens criados por eles mesmos (geralmente) e criam uma história narrativa colaborativamente. Assim sendo, pode-se concluir que cada participante de um RPG interpreta, age, pensa, fala, luta e sente o papel de um personagem escolhido. (MARSOLLA, 2014)

Ainda sobre o RPG de mesa, observa-se a liberdade que o jogador possui quanto às ações que pode tomar, tendo um número quase ilimitado de opções, variando conforme a aventura envolvida.

“Dentro desse sistema [de regras], os jogadores improvisam e decidem livremente o que seus personagens vão fazer, determinando assim, que a história siga um rumo totalmente novo e fora do script, afinal, ele é criado em conjunto com os outros jogadores.” (MARSOLLA, 2014)

Ao fazer um paralelo entre a estrutura do projeto e o RPG de mesa, teríamos o mestre englobando a MiniCAVE, os dispositivos e a rede, porque é ele quem recebe as ações dos jogadores, interpreta e retorna o resultado.

Esse narrador, mestre ou GM (Game Master, em inglês) é a pessoa que cria a história do mundo, julga as ações dos personagens, interpreta os monstros, vilões, aliados e outros heróis, cria desafios e determina o que acontece. No geral, quando um jogador decide fazer alguma coisa ou ação, o narrador decide e narra para ele o resultado. Quando é uma ação complicada e/ou com grande chance de erro [...] o narrador pode exigir um teste, que é feito com uma jogada de dados. (MARSOLLA, 2014)

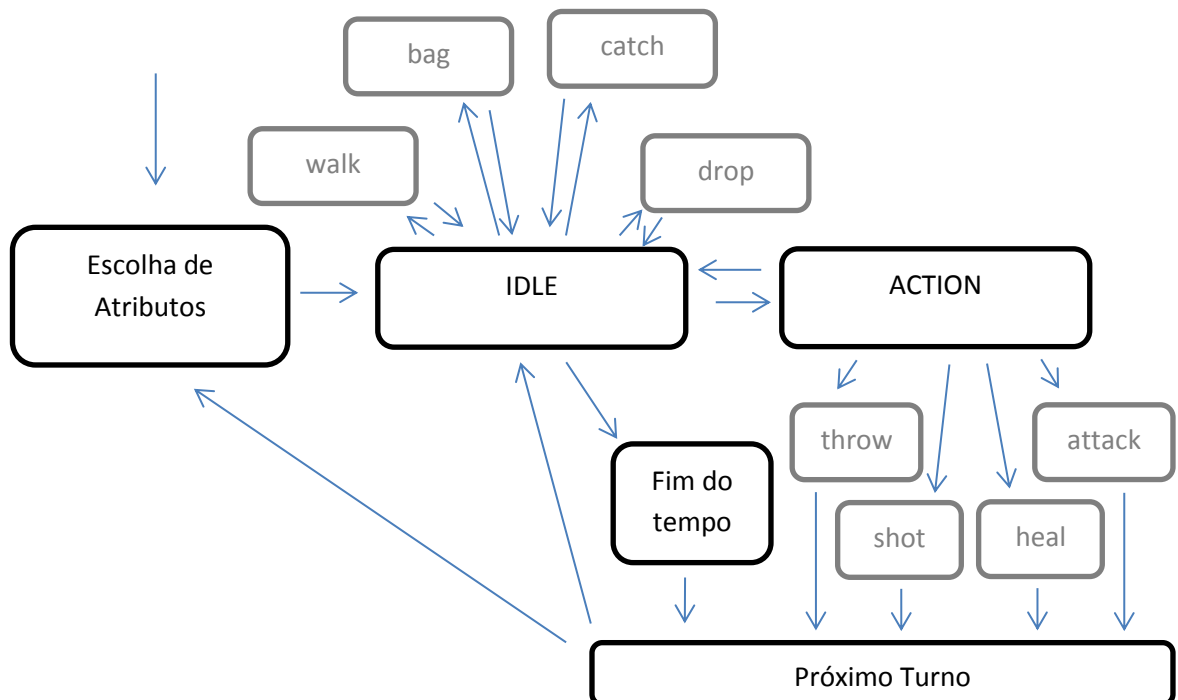
Entre as características do RPG que não foram aplicadas no jogo ou pelo menos em parte, é importante citar a construção do personagem do jogador, pois no RPG ela é muito bem trabalhada, possuindo muitos detalhes, desde a escolha do nome até sua personalidade. Neste projeto, por ser algo importante para o decorrer do jogo, o jogador pode apenas editar suas habilidades. O armamento e itens são escolhidos de forma aleatória.

“(...) o Jogador poderá definir qual será a raça do personagem, sua classe / ocupação, seus atributos físicos, mentais, emocionais, suas perícias, suas vantagens, desvantagens, personalidade, aparência e até mesmo sua história de vida.” (MARSOLLA, 2014)

Outro ponto é que jogos de RPG não possuem um final definido - “O RPG é mais um jogo cooperativo e colaborativo entre jogadores e narrador, do que uma disputa” (MARSOLLA, 2014) E como este trabalho permite o combate, haverá um final, baseado nas mortes dos personagens dos jogadores. Porém este último fato não é explícito, para justamente manter a liberdade dos jogadores.

3.2 Fluxo do Jogo

Figura 7 - Fluxo do jogo.



Fonte: elaborado pelo autor.

Figura 8 - Painel de atributos.



Fonte: elaborado pelo autor

A Figura 7 exibe o fluxo do jogo. Como pode ser observado, inicia-se na escolha de atributos (Figura 8), sendo estes: agilidade, força, resistência, vida, habilidade de longa distância, habilidade de curta distância. Dada uma quantidade de pontos de característica, o usuário deverá distribuí-los entre cada uma das habilidades disponíveis da forma que desejar.

Uma vez encerrada esta etapa, o jogo define quem será o primeiro a jogar (sendo sempre o avatar que possuir maior agilidade), avançando para o campo *IDLE* (campo de espera), uma vez que o jogador não está executando nenhuma ação. Como é apresentado acima, qualquer ação feita neste campo, seja ela andar (walk), pegar um item (catch), soltar um item (drop) ou trocar de item com a mochila (bag) não determina o final do turno, sendo possível ficar neste estado até que se esgote o tempo (Fim do tempo). Caso o jogador alterne para o modo *ACTION*, suas ações irão determinar o final do turno, porém estas serão mais efetivas, pois envolvem principalmente movimentos de ataque, tais como: arremesso (throw), ataque à curta distância (attack), tiro (shot) e uso de poção (heal).

Todas as quatro últimas ações citadas influenciam diretamente o decorrer do jogo, uma vez que todas podem alterar a quantidade de pontos de vida dos jogadores, de forma positiva (poção) ou negativa (throw, shot e attack).

Cada um dos jogadores possui uma quantidade de pontos de vida de acordo com valor do atributo vida, estes pontos são determinantes para o final do jogo, encerrando no momento em que apenas um dos *players* possuir mais de zero pontos de vida.

Figura 9 - Cena do jogo.



Fonte: elaborado pelo autor.

Figura 10 - Avatar do jogador.



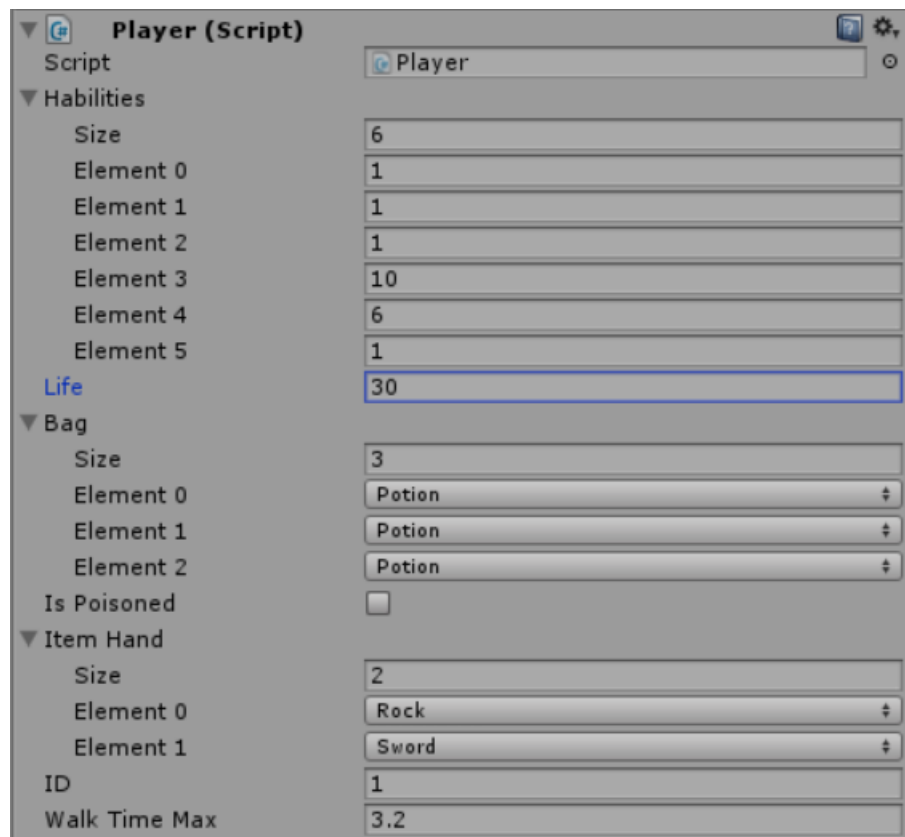
Fonte: elaborado pelo autor.

Quanto ao visual, o avatar e o campo do jogo (Figuras 9 e 10) possuem uma estrutura fixa, não sendo possível fazer alterações dentro do jogo.

3.3 Gerenciamento dos jogadores

Como é característico do gênero RPG, cada jogador possui atributos, cujos valores são atribuídos pelos próprios jogadores, respeitando um limite máximo de pontos a serem distribuídos entre as características disponíveis. Cada um dos personagens também possui alguns itens a serem utilizados, dois destes presentes em cada uma de suas mãos e o restante guardado em uma mochila.

Figura 11 - Descrição da classe Player dentro do painel Inspector



Fonte: elaborado pelo autor.

A Figura 11 exibe a estrutura da classe Player, mostrado pelo painel Inspector da Unity, onde o vetor *Habilities* possui as habilidades editadas pelo jogador, sendo elas (iniciando-se no Element 0): Agilidade, Força, Resistência, Vida, Habilidade de Longa Distância e Habilidade de Curta Distância.

3.4 Características

As características são inicialmente atribuídas pelo jogador em uma tela inicial, onde permite-se que distribua vinte pontos entre seis atributos: agilidade, força, resistência, vida, habilidade à longa distância e habilidade à curta distância. Uma vez que o jogador esteja satisfeito com a sua configuração, o jogo avança para a próxima cena e os dados são registrados em um objeto *Player*. Com exceção de um, todos os atributos têm seus valores mantidos. No caso especial da vida, esta é usada para calcular a vida real do avatar no jogo, somada à um valor base, para que caso o jogador não atribua nenhum ponto nesta característica, o mesmo não perca a partida logo de início.

Cada uma das características afeta um aspecto diferente do jogo. A agilidade define a ordem dos turnos: o avatar mais ágil irá sempre ser o primeiro a jogar. A força influencia no cálculo de dano, a vida indica a o quanto de dano o jogador pode receber até perder, a resistência define o tempo que o

jogador poderá se movimentar durante seu turno, enquanto que as ultimas duas habilidades irão influenciar no sucesso de uma ação de longo ou curto alcance.

3.5 Mochila e itens

A mochila é responsável por armazenar os itens dos jogadores. Ela funciona como uma lista de itens e é inicializada de forma aleatória, ao mesmo tempo em que são registradas as características dos jogadores. Não é levado em conta nenhuma das habilidades dos jogadores, havendo a possibilidade de um jogador que gastou seus pontos em habilidade à longa distância começar o jogo com uma espada.

Funcionando como uma lista, o acesso à mochila é dado de forma simples, onde ao efetuar o movimento de troca de item, o primeiro objeto presente na lista é repassado à mão do jogador, enquanto que o que estava em sua mão retorna para o final da lista.

3.6 Gerenciamento de ações

As ações dos jogadores foram detectadas através dos controlares *Kinect* e *Wii Remote*, sendo que a maior parte dos movimentos envolvem ambos os periféricos. Dado isso, as ações foram divididas em dois blocos: detecção do movimento via *Kinect* e posteriormente reconhecimento da entrada do *Wii Remote*. Cada um dos movimentos do *Kinect* foram separados em métodos dentro do script principal *GameManager*, que retornam a mão utilizada na ação (esquerda ou direita).

Dentro da classe principal do jogo, quatro métodos ficaram responsáveis por gerenciar o deslocamento horizontal, vertical, mão atrás da cabeça e a suspensão do braço à frente do corpo. Cada uma destas quatro ações foram detectadas por meio de gatilhos posicionados na cena que ao serem ativados na ordem correta, determinam se o jogador executou ou não o movimento.

Figura 12 - Método GetHandInBag, responsável por retornar a mão utilizada pelo jogador para trocar de item.

```
Player.Hand GetHandInBag(int playerID) //Return which player's hand is in the bag
{
    if (headCollisionHandler[playerID].handInBag[0])
    {
        return Player.Hand.Right;
    }
    else if (headCollisionHandler[playerID].handInBag[1])
    {
        return Player.Hand.Left;
    }
    return 0;
}
```

Fonte: elaborado pelo autor.

Na Figura 12, um exemplo de um método que identifica a mão utilizada no movimento de troca de item, interação que envolve a mochila do jogador. E ainda sobre estes métodos, cada um deles dependem de um script que recebe as informações dos gatilhos e que analisam se a ação foi de fato executada. No exemplo acima, a variável que representa uma dessas classes é a *headCollisionHandler*.

Todos os *Colliders* foram posicionados ao redor do objeto *rainbowMan*, uma vez que este reproduz todos os movimentos do esqueleto.

Para detectar a presença de itens e outros objetos no raio de visão do jogador, dois longos gatilhos foram posicionados à frente do *Game Object* deste jogador, acompanhando a rotação do mesmo. Um gatilho ficou responsável pelas ações de longa distância (*Far Collider*), tal como um arremesso e o outro por movimentos de curta distância (*Close Collider*), como por exemplo, ao tentar pegar algum item próximo.

Como foi indicado anteriormente, há um script para todos os objetos que possuem um *Collider*, uma vez que é necessário para que se identifique qual objeto ativou o gatilho. Mais especificamente no caso de ambos os gatilhos de distância, foi criado um script responsável principalmente por registrar dentro de uma lista todos os objetos encontrados.

Figura 13 - CloseCollider, gatilho criado para detectar a presença de objetos a curto alcance.

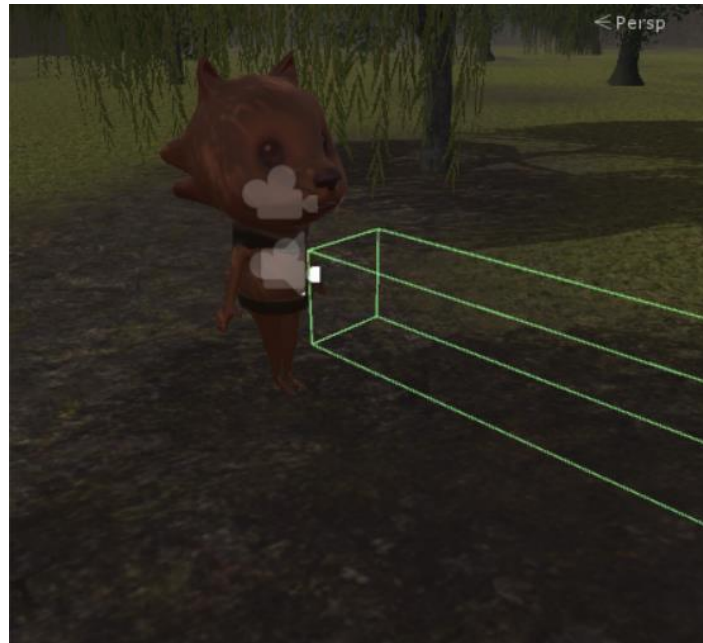
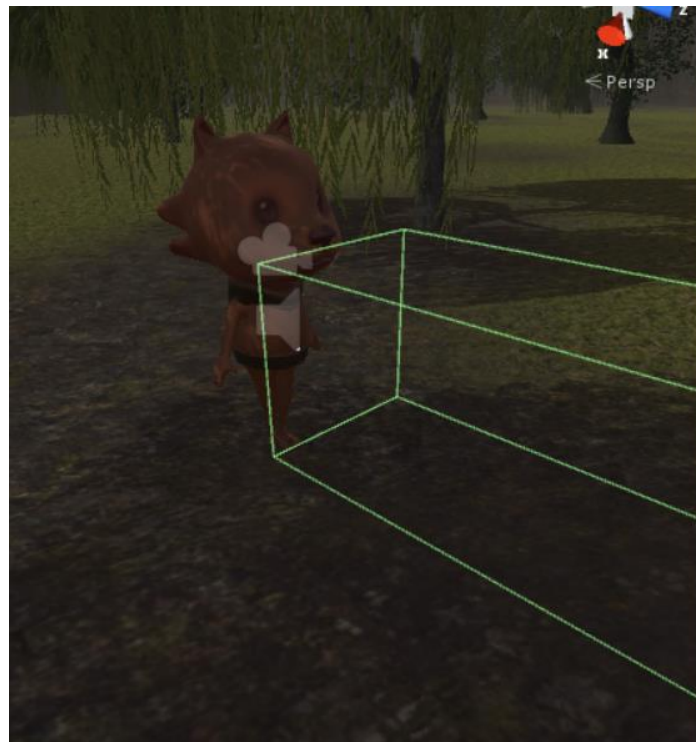


Figura 16: elaborado pelo autor.

A Figura 13 mostra o *Close Collider*. Menor e mais curto, tem seu gatilho ativado apenas por objetos próximos.

Figura 14 - FarCollider, gatilho criado para detectar objetos distantes.



Fonte: elaborado pelo autor.

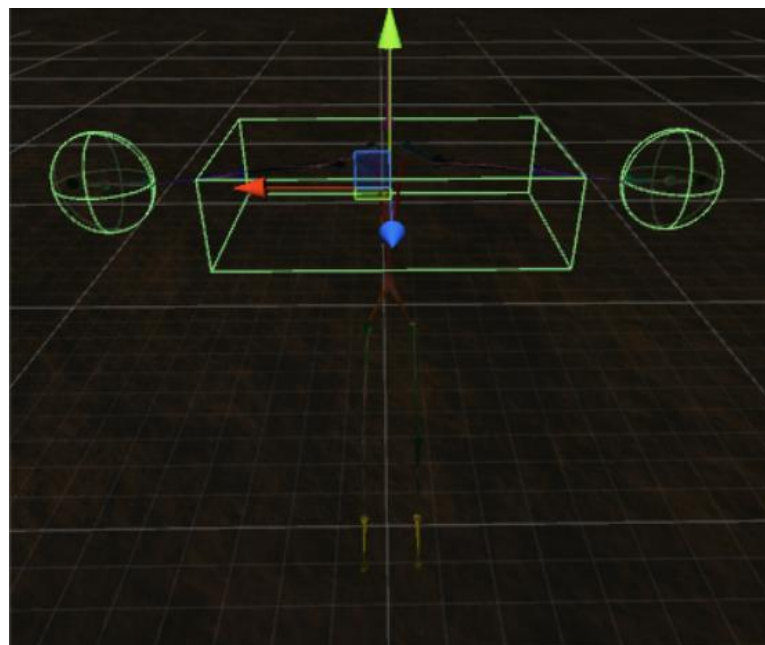
Com praticamente o dobro do tamanho do *Close Collider* (Figura 15) em todas as dimensões, o *Far Collider* (Figura 14) detecta os objetos mais distantes.

3.6.1 Pegar e Soltar

Ao pegar ou soltar um item, o jogador coloca-o na mochila ou o solta de sua mão, respectivamente. Ambas as ações são muito parecidas e tem como principal diferença a presença ou não de um item dentro do alcance do gatilho *Close Collider*.

A detecção de ambos os movimentos se inicia assim que uma das mãos do jogador for detectada como levantada (suspensão do braço à frente do corpo). A partir deste momento ele, o jogador, poderá apertar o botão B do Wii Remote e, se na lista de objetos capturados pelo *Close Collider* houver a presença de algum item, o jogador terá efetivamente efetuado a ação de pegar um objeto da cena e colocado em sua respectiva mochila. Caso contrário, a ação executada será a de soltar. Nesta última situação, onde um item é liberado na cena do jogo, o mesmo é posicionado próximo ao jogador e a mão que perde o objeto é a mesma que foi suspensa. Se ambas as mãos estiverem levantadas, aquela que foi detectada por último é a que será responsável pela ação.

Figura 15 - Gatilho responsável por detectar qualquer um dos braços do jogador suspenso à frente do corpo (ao centro)



Fonte: elaborado pelo autor

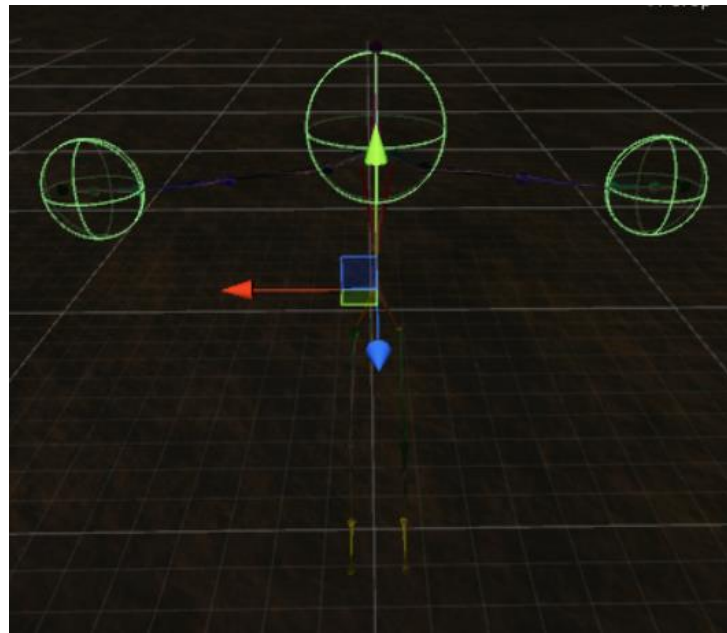
A Figura 15 mostra o gatilho responsável por detectar qualquer uma das mãos do jogador elevadas (ao centro), posicionado à frente do *rainbowMan*.

3.6.2 Troca de item com a mochila

Ao trocar de item com a mochila, o jogador troca o objeto que estava em uma das mãos com um dos que estavam na mochila. Esta ação tem início a partir do momento em que o *trigger* da cabeça do jogador é acionado, detectando a mão do mesmo. Após este momento, o jogador poderá trocar o item presente na mesma mão que levantou, ao pressionar o botão A do Wii Remote. O objeto será trocado com o próximo que estiver na mochila.

Como a mochila foi estruturada como uma lista circular, o jogador poderá pressionar o botão A do Wii Remote tantas vezes quantas forem necessárias, rodando a lista de itens da mochila até que encontre o objeto desejado. Uma vez que sua mão saia do gatilho, o último item selecionado estará presente em sua mão.

Figura 16 - Colliders utilizados para detecção do movimento de troca de item.



Fonte: elaborado pelo autor.

O script responsável por detectar o movimento está presente no objeto que contém o *Collider* central na Figura 16.

3.6.3 Ataque à Curta Distância

Ao executar um ataque à curta distância, o jogador terá a oportunidade de acertar um de seus oponentes, o sucesso e o grau de efetividade desta ação dependerão de quatro componentes: a presença de um jogador dentro do gatilho de curta distância, isto é, o alvo estar próximo o suficiente; a sua

habilidade de curta distância; a sua força; e por fim, o item utilizado para tal ação, uma vez que qualquer objeto pode ser utilizado para isto, variando apenas o dano que cada um deles pode infligir.

Impondo que o jogador esteja na situação de ação, uma vez que um movimento horizontal for detectado com qualquer um dos braços for percebido, este terá efetuado um ataque à curta distância. Se este movimento for executado, inicialmente a lista de detecção do *Close Collider* será verificada para checar se havia algum oponente próximo o suficiente. Em caso negativo, o movimento terá falhado e o turno encerrado sem que o oponente tenha sido afetado. Caso contrário, a habilidade de curta distância somada com um valor inteiro aleatório será utilizada para definir se a ação teve sucesso ou não (1).

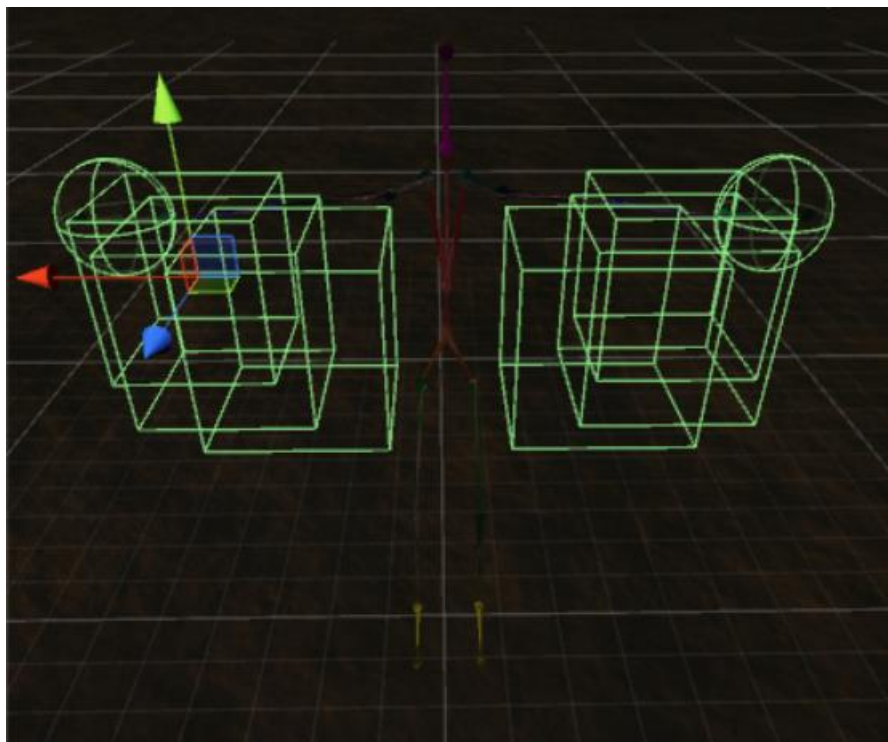
$$y = \text{hab. Curta Distância} + x \quad (1)$$

Se o valor de y na equação acimar for acima de quinze, então o jogador obteve sucesso, e então será calculado o dano infligido ao oponente como é mostrado na equação (2).

$$\text{dano} = x + \text{hab. Força}/2 \quad (2)$$

Acima, x simboliza um valor aleatório, cujo intervalo varia de acordo com o item utilizado na ação. Calculado o dano, este será deduzido do jogador atingido, finalizando o turno.

Figura 17 - Conjunto de seis triggers, cuidando da detecção do ataque à curta distância.



Fonte: elaborado pelo autor.

A Figura 17 mostra como foram posicionados os gatilhos do movimento à curta distância. Apesar de haver seis cubos, trata-se de apenas três objetos com dois *box colliders* cada, posicionados de forma simétrica. O movimento possui um aspecto mais horizontal, começando no primeiro *collider* da direita ou da esquerda, indo até o centro.

3.6.4 Arremesso

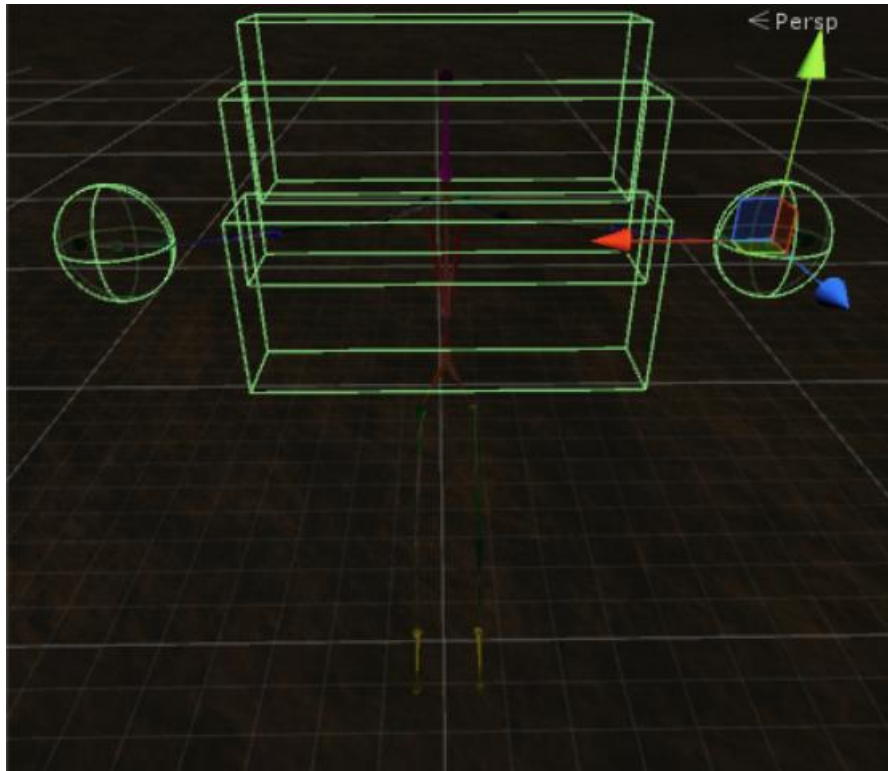
O movimento de arremesso será feito quando o jogador desejar atingir algum de seus adversários à longa distância, tendo como maior desvantagem o fato de que irá perder o objeto arremessado. É uma ação de grande risco, uma vez que a chance de errar é mais elevada, havendo desta forma, uma chance maior de se terminar o turno sem ter feito nada.

Esta ação será executada uma vez que o jogador tenha descrito um movimento vertical com qualquer um dos braços (com o pré-requisito de estar na situação de ação). A partir deste ponto, segue-se a mesma sequência lógica do ataque à curta distância: primeiro verifica-se a existência de algum oponente no *Far Collider*, se verdadeiro, utiliza-se da habilidade de longa distância somada com um valor aleatório e confirma o sucesso do movimento; em caso afirmativo, calcula-se o dano da mesma forma que o ataque à curta distância, com a diferença de que a força tem uma participação menor no cálculo, como é mostrado na equação (3).

$$\text{dano} = x + \text{hab. Força}/4 \quad (3)$$

Da mesma forma que a ação apresentada no último tópico, se antes do cálculo de dano em alguma das condições o resultado for negativo, a ação será dada como falha.

Figura 18 - Conjunto de três gatilhos, responsáveis pela detecção do movimento de arremesso.



Fonte: elaborado pelo autor.

Diferentemente do ataque à curta distância, são utilizados apenas três gatilhos (Figura 18), cada um pertencendo à um objeto diferente. O movimento tem um aspecto vertical, iniciando no *collider* mais alto.

3.6.5 Tiro e uso de poção

Apesar de terem propósitos diferentes, ambas as ações são executadas da mesma forma. O que define se o jogador irá atirar ou usar alguma poção de vida é o item presente em sua mão. Caso o movimento seja o uso de poção, esta será removida do jogo assim que for utilizada, ao contrário da arma, que continua em posse do usuário depois do tiro.

Desde que o jogador esteja no campo de ação, os dois movimentos são iniciados a partir da suspensão de uma das mãos à frente do corpo. Dada esta situação, assim que o usuário pressionar o botão B do controle, uma das duas ações tem início: se a mão suspensa conter uma poção, o jogador irá recuperar parte de seus pontos de vida, ter sua poção removida e o próximo item de sua mochila colocado em seu lugar. Se o item for uma arma de fogo, são executados os mesmos testes do arremesso para testar o seu sucesso. Em caso afirmativo, calcula-se o dano utilizando apenas a força da arma (4).

$$\text{dano} = x \quad (4)$$

Como este movimento é exclusivo de um item, o valor aleatório de x será sempre dentro de um mesmo intervalo (4 a 12).

3.6.6 Andar

Este é o movimento feito para que o jogador possa se deslocar no mapa do jogo, seja para se aproximar de algum adversário ou para simplesmente fugir.

Esta ação é feita através dos botões direcionais do Wii Remote, e é limitada por um contador de tempo, que incrementa sempre que o jogador se deslocar. Ao atingir o limite de tempo, a velocidade do jogador é zerada até que o turno termine. O limite de tempo é definido pelo atributo de resistência.

$$\text{tempo limite} = 3 + \text{resistencia}/5 \quad (5)$$

Na equação (5) está representada a fórmula para calcular o limite de tempo de locomoção de um jogador, em segundos.

3.7 Comunicação dos controladores

3.7.1 Wii Remote

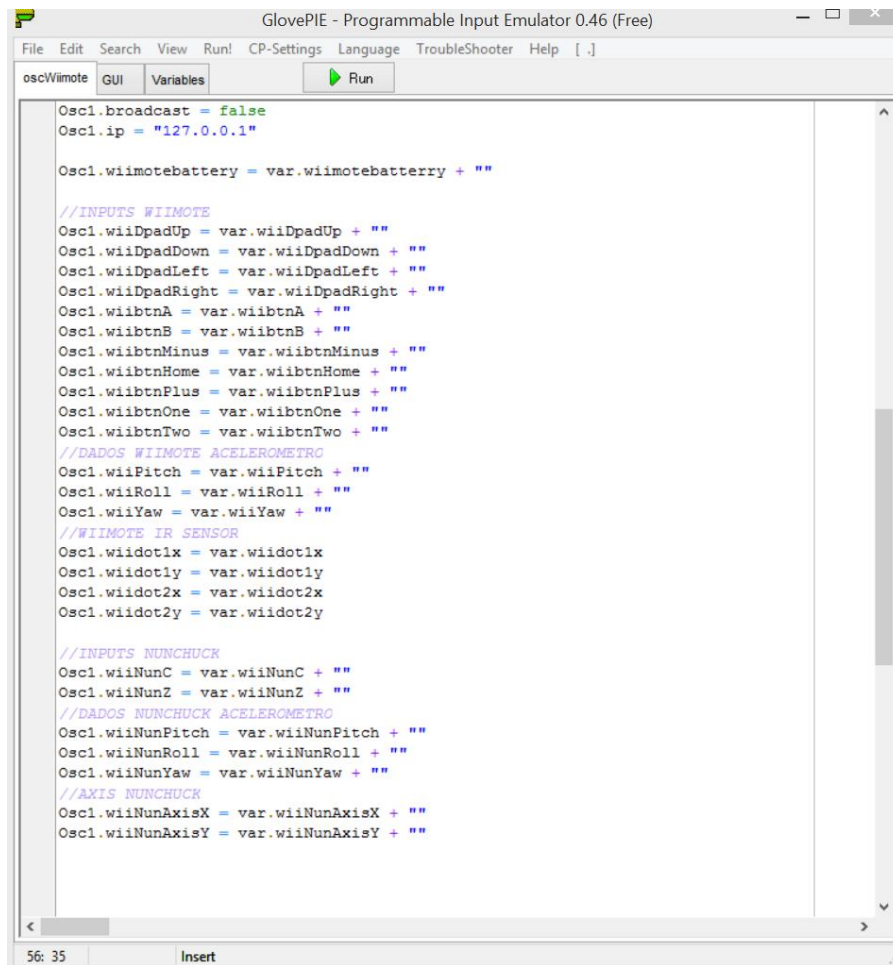
A conexão do dispositivo de entrada Wiimote com o computador foi feita via *bluetooth*, que pode ser ativada por dois modos: pressionando-se os botões “1” e “2” de forma simultânea ou ao apertar o botão “sync”, localizado dentro do controle, no compartimento de pilhas.

O programa utilizado para capturar as informações via *bluetooth* foi o middleware GlovePie.

Para auxiliar na interpretação dos dados OSC dentro da Unity foi utilizado um pacote disponível no fórum da Unity¹ (WIIMOTE + UNITY 3D FREE DEMO). Dentro deste Unity package existem um total de quatro classes (Osc, OSCReceiver, UDPPacketIO, InfosManagerBehaviour) que capturam os pacotes enviados através do protocolo OSC e os interpretam, repassando para métodos que cuidam de cada botão separadamente (Figura 23), até uma classe final (InfosManagerBehaviour) que apenas trabalha com o dado booleano de cada variável (verdadeiro se pressionado, falso se solto). Para o GlovePie, o pacote também inclui códigos exemplo, que para este projeto, foram feitas algumas alterações, permitindo o suporte a mais de um controle, seguindo o padrão mostrado na Figura 22.

¹ <http://www.mediafire.com/download/8tmr27hmb4k8q0e/WIIMOTE+%2B+UNITY+3D+FREE+DEMO.rar>

Figura 19 - Uso do protocolo OSC dentro da interface de programação do GlovePie.



```

Osc1.broadcast = false
Osc1.ip = "127.0.0.1"

Osc1.wiimotebattery = var.wiimotebattery + ""

//INPUTS WIIMOTE
Osc1.wiiDpadUp = var.wiiDpadUp + ""
Osc1.wiiDpadDown = var.wiiDpadDown + ""
Osc1.wiiDpadLeft = var.wiiDpadLeft + ""
Osc1.wiiDpadRight = var.wiiDpadRight + ""
Osc1.wiibtnA = var.wiibtnA + ""
Osc1.wiibtnB = var.wiibtnB + ""
Osc1.wiibtnMinus = var.wiibtnMinus + ""
Osc1.wiibtnHome = var.wiibtnHome + ""
Osc1.wiibtnPlus = var.wiibtnPlus + ""
Osc1.wiibtnOne = var.wiibtnOne + ""
Osc1.wiibtnTwo = var.wiibtnTwo + ""

//DADOS WIIMOTE ACELEROMETRO
Osc1.wiiPitch = var.wiiPitch + ""
Osc1.wiiRoll = var.wiiRoll + ""
Osc1.wiiYaw = var.wiiYaw + ""

//WIIMOTE IR SENSOR
Osc1.wiidot1x = var.wiidot1x
Osc1.wiidot1y = var.wiidot1y
Osc1.wiidot2x = var.wiidot2x
Osc1.wiidot2y = var.wiidot2y

//INPUTS NUNCHUCK
Osc1.wiiNunC = var.wiiNunC + ""
Osc1.wiiNunZ = var.wiiNunZ + ""

//DADOS NUNCHUCK ACELEROMETRO
Osc1.wiiNunPitch = var.wiiNunPitch + ""
Osc1.wiiNunRoll = var.wiiNunRoll + ""
Osc1.wiiNunYaw = var.wiiNunYaw + ""

//AXIS NUNCHUCK
Osc1.wiiNunAxisX = var.wiiNunAxisX + ""
Osc1.wiiNunAxisY = var.wiiNunAxisY + ""

```

Fonte: elaborado pelo autor.

Como é mostrado na Figura 19 acima, cada uma das variáveis de entrada do Wii Remote foi repassada para sua respectiva variável no OSC, para que pudesse ser acessada dentro do motor de Jogo Unity.

Figura 20 - Trecho de código da classe OSCReceiver.

```
//INPUTS WIIMOTE
handler.SetAddressHandler("/wiiDpadUp", WiiDpadUp);
handler.SetAddressHandler("/wiiDpadDown", WiiDpadDown);
handler.SetAddressHandler("/wiiDpadLeft", WiiDpadLeft);
handler.SetAddressHandler("/wiiDpadRight", WiiDpadRight);
handler.SetAddressHandler("/wiibtnA", WiibtnA);
handler.SetAddressHandler("/wiibtnB", WiibtnB);
handler.SetAddressHandler("/wiibtnMinus", WiibtnMinus);
handler.SetAddressHandler("/wiibtnHome", WiibtnHome);
handler.SetAddressHandler("/wiibtnPlus", WiibtnPlus);
handler.SetAddressHandler("/wiibtnOne", WiibtnOne);
handler.SetAddressHandler("/wiibtnTwo", WiibtnTwo);
//DADOS WIIMOTE ACELEROMETRO
handler.SetAddressHandler("/wiiPitch", WiiPitch);
handler.SetAddressHandler("/wiiRoll", WiiRoll);
handler.SetAddressHandler("/wiiYaw", WiiYaw);
//WIIMOTE IR SENSOR
handler.SetAddressHandler("/wiidot1x", Wiidot1X);
handler.SetAddressHandler("/wiidot1y", Wiidot1Y);
handler.SetAddressHandler("/wiidot2x", Wiidot2X);
handler.SetAddressHandler("/wiidot2y", Wiidot2Y);

//INPUTS NUNCHUCK
handler.SetAddressHandler("/wiiNunC", WiiNunC);
handler.SetAddressHandler("/wiiNunZ", WiiNunZ);
```

Fonte: elaborado pelo autor.

A Figura 20 mostra o trecho do código que delega a cada método a identificação de cada entrada, baseado na mensagem recebida via OSC, atividade que é feita pela classe *OSCReceiver*.

Figura 21 - Emulação de comandos de mouse e teclado via Wii remote com auxílio do GlovePie.

```
Key.Up = Wiimote1.Up
Key.Down = Wiimote1.Down
Key.Left = Wiimote1.Left
Key.Right = Wiimote1.Right

var.speed = nunchuck1.Joy
var.speed = var.speed * 30
end if
mouse.DirectInput2D += var.speed
```

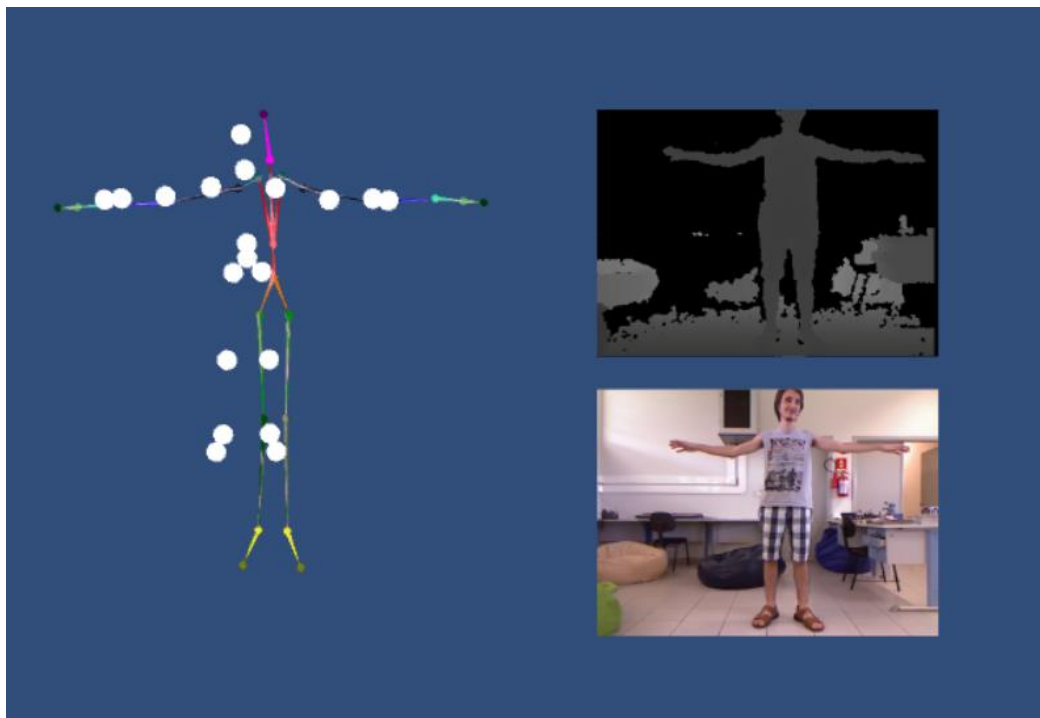
Fonte: elaborado pelo autor.

O GlovePie também foi utilizado para emular através dos direcionais e do botão analógico do Wiimote as setas direcionais do teclado e o mouse, respectivamente, como mostra a Figura 21.

3.7.2 Kinect

Para trabalhar com o Kinect foi necessário o uso do *Kinect for Windows SDK v1.8* (MICROSOFT, 2013) em conjunto com o *Kinect Wrapper Package for Unity3D* que contém todos os scripts necessários para receber os dados capturados pelo Kinect.

Figura 22 - Cena apresentada pelo projeto-exemplo do Kinect Wrapper Package for Unity3D



Fonte: elaborado pelo autor

A Figura 22 mostra um dos projetos-exemplo disponíveis *Kinect Wrapper Package for Unity3D* para facilitar a compreensão dos scripts. Para este projeto foi utilizado o modelo do *rainbowMan* localizado sob os pontos brancos, à esquerda.

Na cena do jogo foi colocado um *KinectPrefab* contendo quatro principais scripts: *KinectSensor*, *SkeletonWrapper*, *DepthWrapper* e *DeviceOrEmulator*. O primeiro é responsável por capturar os dados do sensor; o segundo script recolhe as informações referentes ao esqueleto do jogador; o terceiro trabalha com os dados da imagem do Kinect; o quarto e último define se será utilizado o dispositivo Kinect ou um emulador.

Independentemente do número de jogadores, apenas uma instância do objeto citado acima foi necessário para o funcionamento, uma vez que apenas um Kinect foi utilizado neste projeto.

Os movimentos do jogador foram repassados para um esqueleto virtual identificado como *rainbowMan*, um prefab que utiliza as informações do *KinectPrefab*, sendo necessário neste caso uma instância para cada jogador, fazendo uso do mesmo *KinectPrefab* como referência.

3.8 Gerenciamento de turnos

Os turnos do jogo foram controlados pela classe principal *GameManager*, que dividiu cada turno em dois campos: *action* e *idle*. *Action* envolve todas as ações que determinam o final do turno que em geral são ataque, com exceção do uso de poção. *Idle* possui todos os movimentos mais simples que por afetarem menos o jogo, podem ser feitos inúmeras vezes. Para determinar o final de uma rodada, o *Game Manager* também possui um contador de tempo que encerra o turno ao atingir o limite.

A transição entre os campos *Idle* e *Action* ocorre toda vez que é detectado que o jogador pressionou o botão A do Wiimote. Não há um limite de turnos: o final do jogo só é dado quando resta apenas um jogador em cena.

3.9 Multiprojeção

Para a multiprojeção foi utilizado o pacote Unity Cluster Package em conjunto com o componente Network View. Como para controlar a rotação da câmera do jogador foi utilizado o componente *FirstPersonController* (AssetStore), foi necessário uma pequena adaptação no prefab *Multi Projection Camera* colocando-o como objeto filho de um objeto vazio (*Multi Projection Camera FATHER*), uma vez que a rotação nos eixos x e y no *FirstPersonController* separadamente para o objeto filho e pai, respectivamente.

Para que a posição e a rotação dos objetos da cena fossem sincronizadas com as instâncias dos clientes, todos os objetos passíveis de movimentação receberam a inclusão do componente Network View, incluindo o *Multi Projection Camera FATHER*.

4 TESTE DE FUNCIONAMENTO

Para testar o funcionamento do jogo no ambiente de multiprojeção, foi feito um teste adaptado, onde ao invés de serem utilizados todos os seis projetores e os seis computadores dedicados, apenas três projetores foram usados, conectados a três outros computadores, tendo um notebook Windows como servidor (nó mestre). Esta opção de execução foi usada devido ao Sistema Operacional de cada uma das máquinas da MiniCAVE estar desatualizado, impedindo a execução do jogo.

Figura 23 - Teste de execução no ambiente multiprojetado.



Fonte: elaborado pelo autor.

A Figura 23 mostra o jogo em execução. A imagem projetada é a cena principal, após a tela de configuração de atributos, onde os jogadores farão suas ações. À direita na tela central é possível observar um personagem, correspondente a um segundo jogador.

5 RESULTADOS

O teste realizado mostrou que o jogo funciona na multiprojeção, uma vez que o mesmo não apresentou qualquer problema quanto ao atraso na imagem. Foi possível se locomover pelo mapa usando os direcionais do Wii remote, assim como a rotação da câmera.

Apesar dos feedbacks serem poucos, foi possível também averiguar a detecção das ações por meio do console, no servidor (nó mestre), tais como andar, atirar, trocar item com a mochila e ataque à curta distância. Duas instâncias de jogadores estavam rodando, mostrando que o jogo funciona para mais de um jogador. O suporte oferecido pelo GlovePie, que apesar de ser um middleware e, desta forma, um processo extra rodando juntamente com o jogo, não trouxe problema quanto ao desempenho da aplicação.

6 CONCLUSÃO

Através dos resultados obtidos no teste de execução, conclui-se que a MiniCAVE é uma plataforma com grande potencial para desenvolvimento de jogos do gênero Role-playing game, apresentando um resultado satisfatório quanto ao seu efeito de imersão, isto devido ao fato de que a conexão do servidor com os clientes não apresentou atrasos, da mesma forma que a comunicação controladores não afetaram a velocidade da comunicação de forma significativa.

6.1 Trabalhos Futuros

Dando continuidade neste projeto, propõe-se completar o protótipo de jogo com animações e mais formas de *feedback* para o usuário, uma vez que a captura das ações mostraram-se funcionais no teste. Pretende-se também executar uma versão do jogo na MiniCAVE sem adaptações, fazendo um *gameplay* com vários usuários, para poder coletar as suas experiências, que uma vez conhecidas, será possível fazer uma análise mais profunda sobre o jogo e suas implicações.

REFERÊNCIAS

- DIAS, D. R. C. Sistema avançado de realidade virtual para visualização de estruturas odontológicas. Bauru, 2011. 119 p.
- DUNGEONS & DRAGONS. Forty years of adventure. Disponível em: <<http://dnd.wizards.com/dungeons-and-dragons/what-dd/history/history-forty-years-adventure>>. Acesso em: 25 mai. 2015.
- DUNGEONS & DRAGONS. Basic Rules: Player's Basic Rules Vversion 0.2. Disponível em: <http://media.wizards.com/2014/downloads/dnd/PlayerDnDBasicRules_v0.2.pdf>. Acesso em: 23 mai. 2015.
- DUNGEONS & DRAGONS. Basic Rules: Dungeon Master's Basic Rules Version 0.3. Disponível em: <<http://media.wizards.com/2014/downloads/dnd/DMBasicRulesv.0.3.pdf>>. Acesso em: 23 mai. 2015.
- EDUARDO, D. Porque utilizar o Unity?. Disponível em: <<http://www.eng.com.br/novosite/artigos/detalhes.cfm?id=17>>. Acesso em: 10 nov. 2015.
- JAMERSON, Mark. The Wiimote and Csound. Disponível em: <<http://www.csounds.com/journal/issue7/wiimoteCsound.html>>. Acesso em: 18 dez. 2015.
- KENNER, Carl. GlovePIE. 2010. Disponível em: <<http://glovepie.org/>>. Acesso em: 10 nov. 2015.
- KOOIMA, R. *Generalized Perspective Projection*. 2015. Disponível em: <<http://aoeu.snth.net/static/gen-perspective.pdf>>. Acesso em: 10 nov. 2015.
- LANNOY, C. Brasil lidera crescimento do mercado de jogos eletrônicos em 2012. **Jornal da Globo**, Mai. 2013. Disponível em: <<http://g1.globo.com/jornal-da-globo/noticia/2013/05/brasil-lidera-crescimento-do-mercado-de-jogos-eletronicos-em-2012.html>>. Acesso em: 24 mai. 2015.
- MARSOLLA, G. Falando sobre RPG #1 - Introdução Básica para RPG de Mesa. Disponível em: <http://www.cogumelando.com.br/falando-sobre-rpg-1-introducao-basica-para-rpg-de-mesa/>. Acesso em: 12 nov. 2015.
- METACRITIC. Games releases by score. Disponível em: <<http://www.metacritic.com/browse/games/score/metascore/all/xbox360?hardware=kinect&view=condensed>>. Acesso em: 24 mai. 2015.
- POPOLIN NETO, M. Multiprojeção em ambientes virtuais gerados pelo motor de jogo Unity. Bauru, 2014. 89 p.

NEWZOO. Global Games Market Will Grow 9,4% to \$91.5Bn in 2015. Disponível em: <http://www.newzoo.com/insights/global-games-market-will-grow-9-4-to-91-5bn-in-2015/>.

Acesso em: 23 nov. 2015.

PHILLIPS, D. An Introduction To OSC. **Linux Journal**, Nov. 2008. Disponível em: <http://www.linuxjournal.com/content/introduction-osc>. Acesso em: 14 dez. 2015.

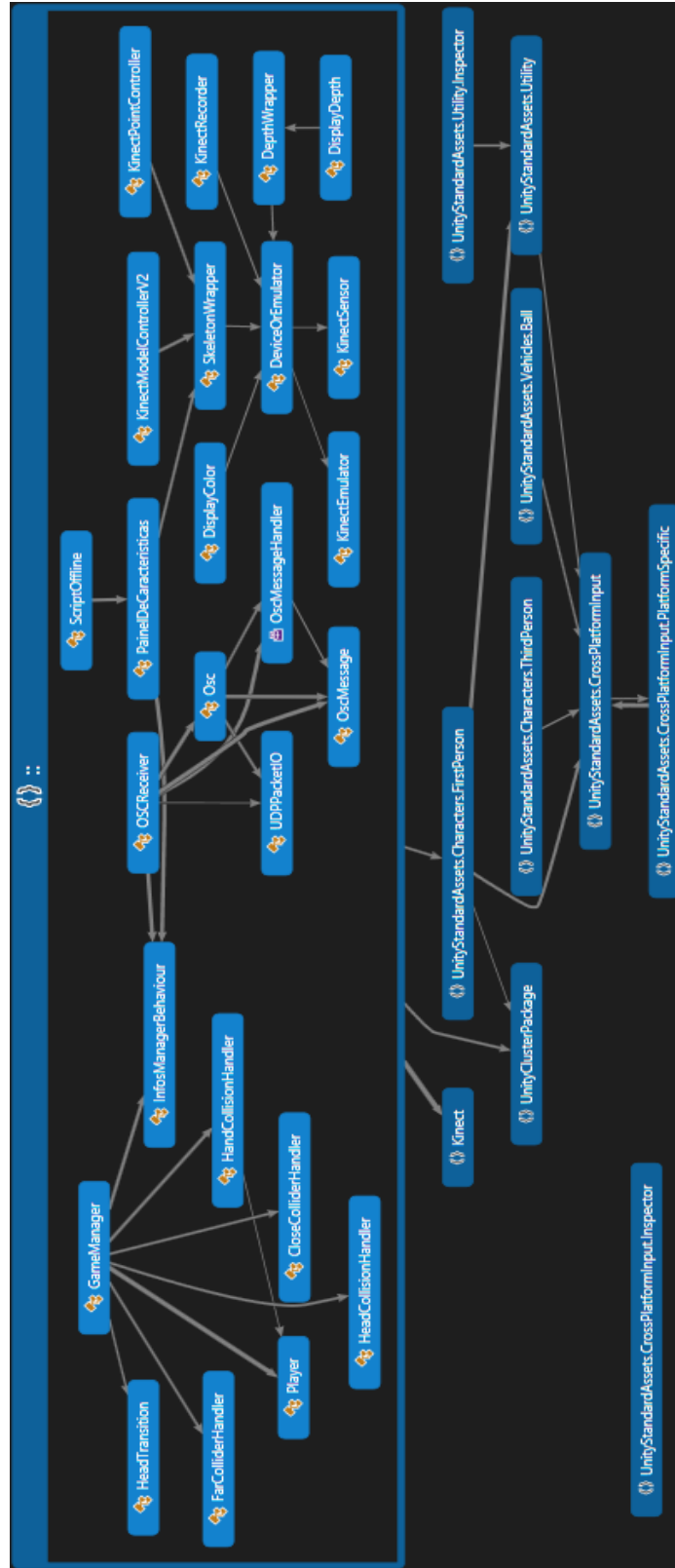
UNITY. A melhor plataforma de desenvolvimento para criar jogos. 2015a. Disponível em: <https://unity3d.com/pt/unity>. Acesso em: 23 mai. 2015.

UNITY. Feito com o Unity. 2015b. Disponível em: <https://unity3d.com/pt/showcase/gallery>. Acesso em: 23 mai. 2015.

UNITY. *Unity Manual*. 2015c. Disponível em: <http://docs.unity3d.com/Manual>. Acesso em: 20 ago. 2015.

APÊNDICE A - Diagrama de classes

Figura 24 - Diagrama de classes



Fonte: elaborado pelo autor

A Figura 24 apresenta o diagrama das classes utilizadas neste projeto, composto de duas cenas diferentes, a primeira onde os jogadores configuram as características de seus personagens e uma segunda, contendo o *gameplay*.

Iniciando-se com a cena inicial, duas classes foram escritas especificamente para esta cena: *ScriptOffline* e *PainelDeCaracteristicas*.

O *PainelDeCaracteristicas* é quem gerencia a navegação do jogador dentro das opções de atributos disponíveis, através da leitura das entradas do Wii Remote, lida por meio da classe *InfosManagerBehaviour*, a qual possui uma variável pública (booleana) para cada um dos botões do Wii Remote, permitindo que sejam acessadas de qualquer outra classe.

São quatro as classes que enviam os dados de entrada à *InfosManagerBehaviour*: *Osc*, que oferece métodos para enviar e receber mensagens via OSC, *OSCReceiver*, que faz o tratamento das mensagens OSC recebidas, *UDPPacketIO*, responsável por configurar a porta e o endereço da comunicação OSC, e *OscMessage*, que armazena o tipo e o endereço da mensagem.

Ainda dentro da cena inicial, o *ScriptOffline* grava os atributos já editados pelos jogadores na própria máquina para serem acessados pelas classes da próxima cena, que corresponde ao *gameplay*.

Sobre a segunda cena, foram criadas as seguintes classes (localizadas dentro do painel na Figura 24): *GameManager*, *CloseColliderHandler*, *HeadTransition*, *HandCollisionHandler*, *HeadCollisionHandler*, *Player* e *FarColliderHandler*.

GameManager é a classe responsável por centralizar as ações do jogo, fazendo a comunicação dos movimentos do jogador com o cenário. É quem inicia a classe *Player*, pelo método *StartPlayer()* que recupera os atributos gravados pelo *ScriptOffline* e sorteia quais serão os itens que estarão disponíveis ao jogador tanto em suas mãos quanto na mochila. A *GameManager* também gerencia a passagem dos turnos e o final do jogo (métodos *Turn()* e *IsGameOver()*).

As quatro classes cujos nomes terminam em *Handler* são as responsáveis por interpretar os movimentos do esqueleto do jogador, identificando se determinado movimento foi executado ou não. Para cada um deles há um método diferente dentro do *GameManager* que recebe estas informações - *GetHandInBag()*, *GetThrow()*, *GetHandUp()*, *GetSword()* - e retorna a mão utilizada para cada movimento. Como já foi indicada, a classe *Player* armazena os dados de cada jogador, como suas habilidades e os itens presentes tanto na mochila quanto em sua mão. Possui também métodos como *GetNextItem()*, que troca um objeto presente na mão do jogador com outro que esteja na mochila.

HeadTransition é uma classe onde uma de suas tarefas é fazer a transição da câmera virtual entre os jogadores ao modo que os turnos terminam, através do método *ChangeCameraPosition()*, que recebe como entrada um identificador com o próximo jogador e troca a posição do objeto câmera para a mesma localização do jogador. A classe *HeadTransition* também é responsável por mover a câmera de acordo com a movimentação do jogar, mantendo-os sempre alinhados.

Sobre as demais classes presentes dentro do painel na Figura 33, todas fazem parte do *Kinect Wrapper Package for Unity3D* que recebem os dados do Kinect e para serem posteriormente acessados através da classe *KinectModelControllerV2*.

Fora do painel da Figura 33 estão as classes provenientes do pacote *UnityStandardAssets*, sendo utilizada a classe *FirstPerson*, responsável pela movimentação do jogador dentro do jogo, assim como a rotação da câmera, esta feita pelo método *MouseLook()* que faz a rotação por meio de dois objetos diferentes, um para o eixo x e outro para o eixo y.

Também fora do painel está situado o pacote *UnityClusterPackage*, que possui uma classe - *NodeManager* - que invoca a câmera virtual para multiprojeção na MiniCave, de acordo com um arquivo de configuração no formato XML dentro da pasta do projeto.