PÂMELA DE ASSIS BELTRANI

PREDIÇÃO DE MOVIMENTO EM JOGOS DISTRIBUÍDOS BASEADA EM APRENDIZADO DE MÁQUINA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Elias P. Duarte Jr.
Co-Orientadora: Prof. Aurora T. R. Pozo

CURITIBA

2015

B453p Beltrani, Pâmela de Assis
Predição de movimento em jogos distribuídos baseada em aprendizado de máquina/ Pâmela de Assis Beltrani. – Curitiba, 2015.
78 f. : il. color. ; 30 cm.

TeseDissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2015.

Orientador: Elias P. Duarte Jr.Aurora T. R. Pozo.
Bibliografia: p. 70-74.

1. Jogos eletrônicos - Programação. 2. Aprendizado do computador. 3. Algoritmos de computador. I. Universidade Federal do Paraná. II.Duarte Jr., Elias P.. III. PozoAurora T. R. Título.

CDD: 793.93231

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna Pâmela de Assis Beltrani, avaliamos o trabalho intitulado, "Predição de movimento em jogos distribuídos baseada em aprendizado de máquina ", cuja defesa foi realizada no dia 15 de setembro de 2015, às 09:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X)**aprovação** do(a) candidato(a). ( )**reprovação** do(a) candidato(a).

Curitiba, 15 de setembro de 2015.

Prof. Dr. Elias Procópio Duarte Júnior
PPGInf - Orientador

Profª. Dra. Aurora Trinidad Ramirez Pozo
PPGInf – Co-orientadora

Prof. Dr. Júlio Cesar Nievola
PUC/PR – Membro Externo

Prof. Dr. Arioa de Campos Jr.
UEPG – Membro Externo

Prof. Dr. Lucas Ferrari de Oliveira
PPGInf – Membro Interno

MINISTÉRIO DA EDUCAÇÃO · UNIVERSIDADE FEDERAL DO PARANÁ · Setor de Ciências Exatas · Coordenação de Pós-Graduação em Informática

SUMÁRIO

LISTA DE FIGURAS **v**

RESUMO **vi**

ABSTRACT **vii**

1 INTRODUÇÃO **1**

2 DEFINIÇÃO E CLASSIFICAÇÃO DE JOGOS ELETRÔNICOS **4**

2.1 Definição de Jogos Eletrônicos 4

2.2 Gêneros de Jogos Eletrônicos 5

2.3 Quantidade de Jogadores 9

2.4 Avatar *versus* Onipresente 11

2.5 Os *Frames* e o *Game Loop* 13

3 TÉCNICAS DE PREDIÇÃO DE MOVIMENTO EM JOGOS **18**

3.1 Área de Interesse 18

3.2 O Problema da Latência 20

3.3 Dead Reckoning 22

3.4 AntRecknoing 24

3.5 Aprendizado de Máquina 28

3.5.1 Fases do Aprendizado de Máquina 28

3.5.2 Tarefas e Problemas de Aprendizado de Máquina 30

3.5.3 Algoritmos 32

3.5.3.1 *Reduced Error Pruning Tree* 32

3.5.3.2 *Local Weighted Learning* 33

3.5.3.3 *Multilayer Perceptron* 34

3.5.3.4 *Bootstrap Aggregating* 36

3.5.4 Matriz de Confusão 37

4 PREDIÇÃO DE MOVIMENTO NO *WORLD OF WARCRAFT* 39

4.1 Entendimento do Negócio 39

4.1.1 Bases de Dados 40

4.1.2 A Cidade *Ironforge* 42

4.1.3 Estratégia Utilizada 43

4.2 Entendimento dos Dados 45

4.3 Preparação 46

4.3.1 Construção da Nova Base 46

4.3.2 Limpeza dos Dados 50

4.4 Modelagem e Avaliação 54

4.4.1 Primeiro Teste 55

4.4.2 Segundo Teste 59

4.4.3 Terceiro Teste 61

4.4.4 Histograma 64

5 CONCLUSÃO 68

REFERÊNCIAS 74

A APÊNDICE - ATIVIDADES DOS JOGADORES EM WORLD OF WARCRAFT 75

LISTA DE FIGURAS

2.1 Captura de tela do jogo *Portal*. 6
2.2 Captura de tela de um jogo da série *Ace Attoney*. 7
2.3 Captura de tela do jogo *Diablo III*. 7
2.4 Captura de tela do jogo *Age of Empires Online*. 8
2.5 Captura de tela do jogo *Rollller Coaster Tycoon*. 9
2.6 Captura de tela do jogo *Mario Tennis*. 9
2.7 Captura de tela do jogo *Rogue Legacy*. 10
2.8 Captura de tela do jogo *Dota 2*. 10
2.9 Captura de tela do jogo *Mortal Kombat*. 11
2.10 Captura de tela do jogo *Ragnarok* 12
2.11 Captura de tela do jogo *League of Legends*. 12
2.12 Captura de tela do jogo *The Elder Scrolls: Skyrim*. 13
2.13 Captura de tela do jogo *SimCity*. 13
2.14 Fluxo de execução na estrutura do *Game Loop*. 15
2.15 Exemplo de situação que se tornaria impossivel de ser jogada caso o *Game Loop* não existisse. 16

3.1 Exemplos de posicionamento de objetos na área de interesse do jogador. . . 20
3.2 Exemplo de inconsistência gerada pelo atraso na entrega das mensagens. . 21
3.3 Exemplo de execução do algoritmo *AntReckoning*. 27
3.4 Exemplo de uma árvore de decisão. 33
3.5 Exemplo de uma rede neural com duas camadas ocultas. 35
3.6 Exemplo de conjuntos de treinamentos gerados pelo Bagging. 37

4.1 Posição dos NPCs dentro do mapa de *Ironforge* 43
4.2 Fluxograma da nova abordagem adotada. 45

4.3 Exemplos de posicionamento de jogadores dentro do campo de visão de um jogador. 49
4.4 Exemplos de posicionamento de jogadores dentro do campo de visão de um jogador. 49
4.5 Posicionamento dos jogadores presentes na bases 1 e 2. 53
4.6 Casos em que o algoritmo do *Dead Reckoning* acerta, porém o respectivo algoritmo acusa que ele irá errar, ou seja, as saídas Falsas Negativas. . . . 58
4.7 Fluxograma das possibilidades do resultado da nova abordagem. 64
4.8 Histograma construído com os valores dos erros apresentados pelo *Multilayer Perceptron*. 65
4.9 Histograma construído com os valores dos erros apresentados pelo *Bagging*. 66
4.10 Histograma construído com os valores dos erros apresentados pelo *LWL*. . . 67
4.11 Histograma dos erros do *REPTree* 67

RESUMO

Em jogos distribuídos *multiplayer* os jogadores mantém uma visão consistente das posições uns dos outros através da troca periódica de informações sobre a movimentação de seus personagens. As mensagens de atualização, além de representarem uma sobrecarga na rede, podem sofrer atrasos de entrega, podendo causar inconsistências ou ainda saltos abruptos de renderização. Tradicionalmente, o algoritmo *Dead Reckoning* é utilizado para que os jogadores possam prever as movimentações que realizam e, quando acerta, evita a troca de mensagens. Por exemplo, considere um jogador prevendo a posição do outro; ambos executam o algoritmo e a mensagem de atualização é apenas enviada quando o jogador que movimentou constata que o *Dead Reckoning* não previu a nova posição correta. O *Dead Reckoning* utiliza as leis da física para fazer uma previsão, assumindo que a movimentação ocorre em linha reta. Porém, é notória sua baixa taxa de precisão. Neste trabalho, apresentamos uma nova estratégia para a predição de movimentação em jogos distribuídos baseada em aprendizado de máquina. A estratégia consiste de duas fases bem definidas, em que modelos de aprendizado são construídos utilizando os algoritmos: LWL, *Bagging*, *Multilayer Perceptron* e REPTree. Na primeira fase, um modelo de aprendizado classifica se o *Dead Reckoning* acerta ou erra sua predição. Em caso de acerto, o *Dead Reckoning* é utilizado para fazer a previsão. Entretanto, em caso de erro, concretamente quando se conclui que o jogador muda sua direção de movimentação, é utilizado um novo modelo de aprendizado para prever a nova direção. A estratégia proposta foi aplicada para o jogo *World of Warcraft*. Os modelos de aprendizado foram construídos utilizando a ferramenta Weka, com dados de *traces* do jogo extraídos de bases de dados publicamente disponíveis. Resultados mostram que a estratégia proposta obtém uma taxa de acerto médio de 76.60% para a primeira fase; e de 51.02% para a segunda fase. Destaca-se o algoritmo Bagging, que obtém uma taxa de acerto para a primeira e a segunda fases de 81.10% e 73.37%, respectivamente. Esses resultados confirmam o potencial da aplicação de aprendizado de máquina na previsão de movimentação em jogos distribuídos.

ABSTRACT

In distributed multiplayer games, players have to keep a consistent vision of the positions of each other. Usually they exchange periodic messages about avatar movement updates. These messages not only represent an overload, but can also suffer delays that may lead to renderization inconsistencies. Traditionally, the Dead Reckoning algorithm is used by players for predict movement updates and, when the prediction is correct, a message is avoided. For example, consider a player predicting the position of another player; both players execute the algorithm and the message is only sent when the output of the Dead Reckoning is not the new position. The Dead Reckoning algorithm uses physics laws to make predictions, and it assumes that the movement is on a straight line. However, the low precision of this method is notorious. In this work, we present a new strategy to predict movement in distributed multiplayer games based on machine learning. The strategy consists of two well-defined phases, in which models are constructed using the following machine learning algorithms: LWL, Bagging, Multilayer Perceptron and REPTree. In the first phase, a learning model is used to classify whether the Dead Reckoning algorithm will make a correct or incorrect the prediction. If the prediction is right, then Dead Reckoning is executed. Otherwise, if the prediction is classified as incorrect, in other words, if the player changes the movement direction, another learning model is used to predict the new direction. The new strategy was applyed to the popular World of Warcraft game. The learning models were constructed using the Weka tool, using game trace data publicly available. Results show that the proposed strategy has an average success rate of 76.60% for the first phase; the success rate for the second phase is 51.02%. Its stands out The Bagging algorithm produced the best results, the success rate for the first and second phases were of 81.10%and 73.37%, respectively. These results confirms the potential for the application of machine learning to predict player movement in distributed multiplayer games.

CAPÍTULO 1

INTRODUÇÃO

Na última década tem sido notavél a evolução dos jogos eletrônicos, que estão atingindo níveis de popularidade extremamente elevados. A *Entretaiment Software Association* (ESA), aponta que em 2015 são 155 milhões os norte-americanos que jogam video-games freqüentemente [6]. A ESA também aponta que em cada cinco casas norte-americanas em quatro delas existe algum dispositivo eletrônico que é utilizado para jogar. Essa presença cada vez mais marcante dos jogos na vida das pessoas se reflete diretamente na comercialização dos mesmos. Por exemplo, o jogo *Grand Theft Auto 5*, lançado em setembro de 2013, vendeu 11,21 milhões de cópias no dia de seu lançamento [32]. Outro caso de sucesso é o jogo *Call of Duty: Black Ops II* (CODII) que vendeu um total de 13,62 milhões de cópias [42]. O jogo CODII é um jogo digital que apresenta um modo em que vários jogadores podem jogar juntos, através da Internet, chamado de *multiplayer*. Os jogos *multiplayer* são sistemas distribuídos e neste trabalho são chamados de jogos distribuídos.

Em jogos distribuídos, os jogadores devem ter uma visão consistente do estado do mundo do jogo, caso contrário podem ocorrer inconsistências entre o que está sendo visualizado pelos múltiplos jogadores. Para que essa visão seja consistente, as informações sobre o estado do jogo são transmitidas através da troca periódica de mensagens. Dentre as informações que precisam ser trocadas, estão as informações sobre a posição de cada avatar, a entidade que representa um jogador dentro do jogo, e a sua movimentação. Entretanto, estas mensagens de atualização de movimento dos avatares, além de representarem uma sobrecarga na rede, podem sofrer atrasos de entrega, podendo causar inconsistências ou ainda saltos abruptos de renderização.

Tradicionalmente, o algoritmo *Dead Reckoning* é utilizado para prever a movimentação que os avatares realizam e, quando acerta, evita a troca de mensagens [27]. Por exemplo,

considere que um cliente deve prever a posição de um avatar de um outro jogador; em ambos os clientes o *Dead Reckoning* é executado e somente ocorre uma troca de mensagens quando é constatado que a previsão está incorreta. Para fazer essa previsão, o *Dead Reckoning* utiliza as leis da física e assume que esta movimentação será em linha reta. Em outras palavras, o *Dead Reckoning* não considera que o avatar poderá se movimentar em uma nova direção. Como em jogos é desejavel que o jogador tenha muita interação, e é comum que essa interação seja através da movimentação seu avatar, é notória a baixa taxa de precisão do *Dead Reckoning* nestes ambientes.

Uma solução encontrada para o problema da baixa taxa de precisão do *Dead Reckoning* é a utilização do algoritmo *AntReckoning* [43]. O algoritmo *AntReckoning* tem como objetivo incorporar o interesse dos jogadores, assim como os aspectos temporais e espaciais do mundo do jogo. São utilizados pontos de interesse dos jogadores que geram feromônios que, por sua vez, são utilizados para gerar um modelo de forças de atração e repulsão. Por exemplo, um jogador que está próximo a um item é atraído para próximo deste item, enquanto monstros muito mais fortes que o avatar do jogador o repelem. Para incorporar os aspectos temporais e espaciais do jogo, os feromônios gerados pelos pontos de interese, ao longo do tempo se propagam pelo mundo do jogo e sua concentração acaba decaindo. Portanto, naturalmente, os fatores temporais e da geometria do mundo são simulados. O problema desta abordagem é que necessita que um especialista no jogo configure os multiplos parâmetros utilizados pelo *AntReckoning*. Isso acaba dificultando a utilização desta abordagem na prática.

Neste trabalho, apresentamos uma nova estratégia para a predição da movimentação de avatares em jogos distribuídos baseada em aprendizado de máquina. Esta estratégia tem duas fases bem definidas. Durante a primeira fase, um modelo de aprendizado classifica se o *Dead Reckoning* acerta ou erra a predição de um determinado avatar. Em caso de acerto, o *Dead Reckoning* é utilizado pra fazer a previsão. Porém, em caso de erro, a segunda fase é executada. Nessa segunda fase, é utilizado um novo modelo de aprendizado para fazer a previsão da nova direção de movimento. Os modelos de aprendizado são construídos utilizando os algoritmos: *Local Weighted Learning* [7], *Bootstrap Agregating*

[33], *Multilayer Perceptron* [31] e *Reduced Error Pruning Tree* [28].

A estratégia proposta foi aplicada para o jogo *World of Warcraft*. Os modelos de aprendizado foram construídos utilizando uma nova base de dados. Esta nova base de dados foi montada utilizando dados de *traces* que estão disponíveis publicamente [38] [20]. Esta base de dados contém informações sobre o campo de visão de cada um dos jogadores e apresenta informações sobre quais jogadores e NPCs (*Non-Playable Characters*) estão visíveis por determinado jogador em um instante de tempo.

Resultados mostram que a estratégia proposta obtém uma taxa de acerto médio de 76,60% para a primeira fase; e de 51,02% para a segunda fase. Destaca-se o algoritmo *Bagging*, que obtém uma taxa de acerto para a primeira e a segunda fases de 81,10% e 73,37%, respectivamente. Esses resultados confirmam o potencial da aplicação de aprendizado de máquina na previsão de movimentação em jogos distribuídos. Os testes executados mostram também uma taxa de acerto final médio de 66,37%, observa-se que nesta mesma base o algoritmo *Dead Reckoning* obteve a taxa de acerto de 44,54%. Destaca-se novamente o algoritmo *Bagging* obteve uma taxa de acerto final de 78,76%. Estes resultados mostram que a utilização de aprendizado de máquina para a predição da movimentação de avatares jogos distribuídos é uma alternativa viável.

Em comparação com uma outra abordagem relacionada da literatura, o AntReckoning, a taxa de sucesso é semelhante, cerca de 30% de melhoria sobre o *Dead Reckoning*. Entretanto, a grande vantagem da estratégia proposta é que não necessita de um especialista para setar os múltiplos parâmetros utilizados por aquele modelo. Utilizando aprendizado de máquina, os parâmetros são "automaticamente aprendidos".

Essa dissertação está organizada da seguinte forma. No capítulo 2 são apresentados os conceitos de jogos utilizados na atualidade e os conceitos básicos para a construção de um jogo digital. Em seguida, no capítulo 3, o algoritmo *Dead Reckoning* e o *AntReckoning* são apresentados, bem como os algoritmos de aprendizado de máquina utilizados nesta dissertação. A maneira com que este trabalho foi conduzido é descrita no capítulo 4. Por fim, a conclusão deste trabalho e as considerações finais são apresentadas no capítulo 5.

CAPÍTULO 2

DEFINIÇÃO E CLASSIFICAÇÃO DE JOGOS ELETRÔNICOS

Nesse capítulo são apresentados os conceitos e termos necessários para a compreensão do estado atual dos jogos. Para isso, o capítulo está dividido da seguinte maneira. Na seção 2.1 é apresentada uma definição formal de jogos eletrônicos. A partir dessa definição, os jogos podem ser classificados utilizando: gêneros, quantidade jogadores e modo de apresentação. Na seção 2.2 são apresentados os principais gêneros de jogos atuais e alguns exemplos pertencentes a esses gêneros. Na seção 2.3 são apresentadas as diferenças entre os jogos *singleplayer*, *multiplayer* e os *massively multiplayer online*. Em seguida, na seção 2.4, são apresentadas as diferenças entre jogos que utilizam o conceito de avatar e os jogos que utilizam uma visão onipresente. Em seguida, alguns conceitos importantes na construção de jogos são apresentados na seção 2.5.

2.1 Definição de Jogos Eletrônicos

Segundo a definição apresentada em [37] um jogo é uma atividade lúdica composta por uma série de ações e decisões, limitada por regras e pelo universo do jogo, que resultam em uma condição final. As regras e o universo do jogo são apresentados por meios eletrônicos e controlados por um programa digital. As regras e o universo do jogo existem para proporcionar uma estrutura e um contexto para as ações de um jogador. As regras também existem para criar situações interessantes com o objetivo de desafiar e se contrapor ao jogador. As ações do jogador, suas decisões, escolhas e oportunidades - em suma, sua jornada - compõe a alma do jogo. A riqueza do contexto, o desafio, a emoção e a diversão da jornada de um jogador, e não simplesmente a obtenção da condição final é que determinam o sucesso do jogo.

A partir dessa definição, destaca-se a necessidade de que um jogo contenha um ob-

jetivo e regras bem definidas. A diferença entre jogos eletrônicos e os tradicionais é a necessidade de uma interface eletrônica de interação entre o usuário e o jogo. Assim, um videogame é definido como uma plataforma que funciona como interface entre um jogo eletrônico e o jogador. Os comandos são enviados para o jogo através do controle do videogame e este gera um *feedback* visual para o jogador. Jogos eletrônicos são uma forma de entretenimento bastante popular, visto que, segundo a Entertainment Software Association 59% da população norte-americana jogava regularmente jogos eletrônicos em 2013 [5].

2.2 Gêneros de Jogos Eletrônicos

Diversão é uma caraterística inerente aos jogos, porém o que é considerado divertido varia muito de pessoa para pessoa. Por conta disso, os jogos têm características diversas e são classificados em vários gêneros. Uma característica presente nos jogadores é sua insatisfação com convenções genéricas que são exaustivamente repetidas. Assim, existe uma expectativa que sempre existam inovações quanto a jogabilidade, sendo comum aos desenvolvedores combinarem mais de um gênero buscando a diversão. Destaca-se que essas classificações em gêneros tratam apenas da jogabilidade, não se tratando de estética ou ainda da plataforma em que o jogo é executado. Também é necessário destacar que estão sendo apresentadas possíveis classificações para jogos eletrônicos sendo que jogos não eletrônicos utilizam outra categorização. Nessa seção são apresentados os gêneros de ação, aventura, estratégia, *role-playing*, simulação e esporte.

Um jogo de ação exige que o jogador tenha reflexos rápidos e precisão para superar obstáculos. O gênero de ação é considerado o mais básico dos gêneros e certamente um dos mais amplamente explorados. Portanto, é natural que contenha uma ampla lista de subgêneros. Seu principal foco é no combate entre o jogador e seus inimigos. Dentre os subgêneros temos os jogos de lutas tradicionais, os jogos de labirinto, jogos de plataformas, os jogos de tiros, entre vários outros. A figura 2.1 apresenta uma captura de tela retirada do jogo *Portal*[1] que é classificado como um jogo desse gênero.

[1]Portal: http://www.valvesoftware.com/games/portal.html

Figura 2.1: Captura de tela do jogo *Portal*.

Em jogos do gênero de aventura o jogador assume o controle de um protagonista e é guiado através de uma história interativa explorando e buscando a solução de quebra-cabeças. Este gênero procura ter um foco na história e assim acaba herdando muitas características de outras mídias como filmes ou livros. Porém, ao contrário dos livros ou dos filmes, a própria história ou o conteúdo tratado nestes jogos são foco de categorização. Normalmente exigem que o jogador resolva vários enigmas, interaja com os personagens ou com o ambiente em que o jogo é apresentado, em geral de maneira não conflituosa. Dessa forma, os elementos de ação nestes jogos são verdadeiros "minijogos". A figura 2.2 apresenta uma captura de tela retirada de um jogo da série *Ace Attoney*[2] que pertence ao gênero de aventura.

Jogos pertencentes ao gênero de *role-playing* (RPG - *Role Playing Game*) são jogos que herdam suas principais mecânicas dos jogos tradicionais de *role-playing*. Nesses jogos tradicionais os jogadores assumem os papéis de personagens e criam de maneira colaborativa a narrativa. Tradicionalmente, o jogo utiliza um conjunto de regras predeterminadas, dentro das quais os jogadores podem improvisar livremente. Assim, as escolhas dos jogadores determinam a direção que o jogo irá tomar. Porém, na grande maioria dos jogos eletrônicos deste gênero, o jogador normalmente controla um personagem que é um aventureiro especializado em um conjunto específico de habilidades e é conduzido através de um enredo predeterminado. Originalmente, os jogos pertencentes a esse gênero eram ba-

[2]Ace Attoney: http://www.ace-attorney.com/

Figura 2.2: Captura de tela de um jogo da série *Ace Attoney*.

seados em turnos, porém nos jogos modernos o progresso do jogo é em tempo real. Nessa categoria temos como exemplo o jogo *Diablo III*[3] que é apresentado na figura 2.3.

Figura 2.3: Captura de tela do jogo *Diablo III*.

Em jogos de estratégia o foco é estimular no jogador o desenvolvimento de estratégias e táticas próprias que o permitirão progredir no jogo. É comum também a presença de exploração em jogos pertencentes a esse gênero. Na grande maioria dos jogos de estratégia é dado ao jogador um conjunto de unidades para serem controladas. São os jogos mais

[3]Diablo III: http://us.battle.net/d3

parecidos com os jogos tradicionais de tabuleiro e assim como nos jogos de RPG, muitos jogos de estratégia estão gradualmente deixando de serem baseados em um sistema de turnos para se tornarem sistemas de tempo real. No gênero de estratégia a série *Age Of Empires*[4] é considerada um clássico e uma captura de tela pode ser vista na figura 2.4

Figura 2.4: Captura de tela do jogo *Age of Empires Online*.

A categoria de simulação abrange jogos que procuram simular aspectos reais ou de uma realidade fictícia. Existe uma discussão muito forte sobre alguns jogos classificados nessa categoria serem brinquedos e não jogos propriamente ditos [17]. Isso acontece por conta de que muitos jogos nessa categoria não têm um objetivo bem definido ou ainda uma situação de vitória. Dentro desse grupo existe o jogo *Roller Coaster Tycoon*[5] que pode ser visto em na figura 2.5.

Jogos do gênero de esportes buscam simular os esportes físicos. Alguns jogos buscam enfatizar a experiência do próprio esporte, enquanto outros buscam enfatizar a estratégia por trás do jogo. Ainda existem os jogos que buscam satirizar esses esportes físicos, com um objetivo cômico. Um exemplo pertencente a esse grupo é o jogo *Mario Tennis*[6] que pode ser observado na figura 2.6.

[4]Age of Empires Online: http://ageofempiresonline.com/en/
[5]Roller Coaster Tycoon: http://www.rollercoastertycoon.com/
[6]Mario Tennis:http://mariotennisopen.nintendo.com/

Figura 2.5: Captura de tela do jogo *Rollller Coaster Tycoon*.

Figura 2.6: Captura de tela do jogo *Mario Tennis*.

2.3 Quantidade de Jogadores

Outra classificação comumente utilizada considera com quantos jogadores uma pessoa pode interagir simultaneamente agrupando assim, os jogos em *singleplayer*, *multiplayer* e *massively multiplayer online*. Na primeira categoria temos os jogos *singleplayer* que têm como característica a presença de apenas um jogador que não pode interagir com os outros jogadores. Nessa categoria temos como exemplo o jogo *Rogue Legacy*[7] que pode ser observado na figura 2.7.

Na segunda categoria estão os jogos *multiplayer*, que permitem uma experiência com vários jogadores, podendo ser em forma de disputa, cooperação ou rivalidade, e normal-

[7]Rogue Legacy: http://roguelegacy.com/

Figura 2.7: Captura de tela do jogo *Rogue Legacy*.

mente proporciona uma forma de comunicação social que está quase sempre ausente em jogos *singleplayer*. Os jogos *multiplayer* podem ser jogados tanto em rede quanto localmente. Nesse conjunto podemos citar como exemplo o jogo *Dota 2*[8] que pode ser visto na figura 2.8

Figura 2.8: Captura de tela do jogo *Dota 2*.

É comum jogos com ambos os modos *singleplayer* e *multiplayer*. Tanto jogos *singleplayer* quanto *multiplayer* são extremamente populares, porém jogos que contém um modo multiplayer são muito interessantes para a indústria já que 77% dos jogadores jogaram com outras pessoas pelo menos 1 hora semanalmente em 2013, segundo a ESA [5]. O jogo *Mortal Kombat*[9] contém tanto um modo *singleplayer* quanto um modo *multiplayer* e pode

[8]Dota 2: http://br.dota2.com/
[9]Mortal Kombat:http://www.themortalkombat.com/

ser visto na figura 2.9.

Figura 2.9: Captura de tela do jogo *Mortal Kombat*.

Por último, também existem os *massively multiplayer online games* (MMOG) que são capazes de suportar grandes quantidades de jogadores simultaneamente em uma mesma partida. Porém, são raros os jogos que sejam *massively multiplayer* e contenham um modo *singleplayer* ou ainda *multiplayer*. É importante ressaltar que essa classificação considera o número de jogadores interagindo ao mesmo tempo, portanto, mesmo que existam 7,5 milhões de pessoas jogando simultaneamente o jogo *League of Legends*[10] (LoL)[18] por existir um limite máximo de 10 jogadores em um partida, LoL é considerado um jogo *multiplayer* e não um MMOG. Um característica presente em MMOGs é a necessidade de uma grande quantidade de jogadores já que quanto mais jogadores estão presentes no mundo, mais interativo, complexo e atrativo o ambiente do jogo se torna. Dentre os jogos pertencentes a esse grupo existe o jogo *Ragnarok*[11] que pode ser visto na figura 2.10.

2.4 Avatar *versus* Onipresente

Uma classificação de jogos é apresentada em [13] e utiliza uma divisão entre o modelo de avatar e o modelo onipresente. No modelo de avatar o jogador interage com o jogo ´por meio de um personagem representativo, chamado de avatar, que existe em uma localização do mundo do jogo. Pelo avatar o jogador consegue apenas ver e interagir com o que está presente em suas imediações. Nos jogos com avatar pode existir uma diferença

[10]League of Legends: http://br.leagueoflegends.com/
[11]Ragnarok Online: http://iro.ragnarokonline.com/

Figura 2.10: Captura de tela do jogo *Ragnarok*

de perspectiva, esta pode ser tanto em primeira quanto em terceira pessoa. Existem alguns jogos que permitem inclusive a escolha da visão utilizada. A figura 2.11 apresenta uma captura de tela do jogo *League of Legends*, que utiliza o modelo de avatar com a visão em terceira pessoa. A figura 2.12 apresenta uma tela de um outro exemplo de jogo baseado em avatar o *The Elder Scrolls: Skyrim*[12], com visão em primeira pessoa.

Figura 2.11: Captura de tela do jogo *League of Legends*.

No modelo onipresente o jogador pode ver e influenciar de maneira simultânea vários aspectos do jogo. Apesar do jogador não conseguir sempre controlar ou ainda ver completamente o mundo do jogo, segundo [13], o jogador é dito como onipresente por ter controle

[12]The Elder Scrolls : http://www.elderscrolls.com/skyrim

Figura 2.12: Captura de tela do jogo *The Elder Scrolls: Skyrim*.

total sobre os recursos que estão sobre seu comando. Na figura 2.13 pode ser observada a captura de uma tela do jogo *SimCity*[13] que utiliza o modelo onipresente.

Figura 2.13: Captura de tela do jogo *SimCity*.

2.5 Os *Frames* e o *Game Loop*

É importante que os jogos tenham efeitos gráficos de estética agradável, e que portanto, sejam harmoniosos com o ambiente proposto pelo jogo. Outra restrição importante é que as animações sejam suaves. Assim um *frame* é definido como uma única imagem dentro de uma sequência de imagens que é apresentada de forma rápida, dando uma ilusão de

[13]SimCity: http://www.simcity.com/

movimento [26]. Para a análise do desempenho dos jogos em cada ambiente a métrica *frames* por segundo é comumente utilizada. Se a taxa de *frames* por segundo é alta, significa que a execução do jogo está acontecendo de forma suave e rápida, enquanto se a taxa é baixa a execução pode estar comprometida por lentidão, de maneira que a renderização dos objetos pode parecer saltada. Dois fatores podem influenciar a quantidade de *frames* por segundo. O primeiro fator é a complexidade da atualização e a renderização das entidades. Por exemplo, um jogo que possui uma física complexa envolvendo muitos objetos e gráficos muito detalhados pode precisar de mais tempo para a execução em relação aos jogos mais simplificados. O segundo fator é a plataforma em que este jogo está sendo executado, como por exemplo, computadores potentes com múltiplos núcleos, com ou sem placas gráficas dedicadas, com ou sem placas de som dedicadas, com um ou outro sistema operacional, entre outros fatores. Para garantir que a quantidade de *frames* por segundo seja constante a estrutura do *Game Loop* foi criada [26].

Quando os primeiros jogos foram criados, os desenvolvedores sabiam exatamente em quais plataforma seus jogos seriam executados, assim esses especificavam o código exatamente para esses ambientes. Dessa forma, o único fator que influenciava a variação da quantidade de *frames* por segundo executados era a complexidade da fase de atualização e a renderização. Atualmente, poucos desenvolvedores conseguem saber exatamente em qual ambiente seus jogos irão ser jogados e, dessa forma, são forçados a se adaptarem aos mais diversos ambientes. O *Game Loop* se mostra importante exatamente por garantir que o jogo irá rodar em uma velocidade consistente independente das diferenças de plataforma.

O *Game Loop* é utilizado para a organização das três fases de execução tradicional de um jogo, que são: entrada, atualização e a renderização. Na figura 2.14 é apresentado o loop básico utilizado. Nessa figura temos a apresentação do loop com as fases de entrada, atualização e renderização, além disso é importante ressaltar a existência de um loop interno dentro da fase de atualização.

Na fase de entrada as ações do jogador são capturadas, é importante que esta fase não seja bloqueante, ou seja, caso o jogador não faça nenhuma entrada o loop deve continuar

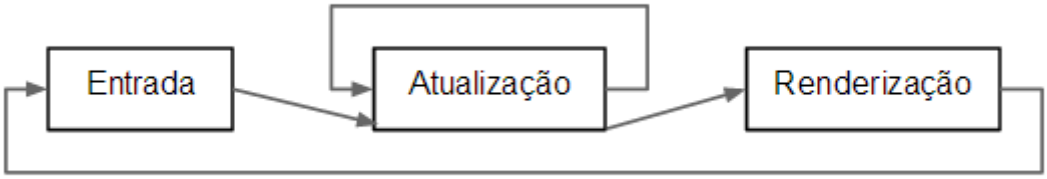


Figura 2.14: Fluxo de execução na estrutura do *Game Loop*.

para a próxima fase. Essa característica da fase de entrada é importante já que na grande maioria dos jogos o jogador não irá fazer uma entrada em todas as iterações do loop. Além disso, mesmo em jogos que exigem uma entrada do jogador para dar andamento, esta fase não será bloqueante já que existem efeitos de animação e som que devem continuar sendo atualizados mesmo que o jogador não faça uma entrada.

Em seguida, na fase de atualização, todas as entidades do jogo são atualizadas. Essa fase ser executada várias vezes em processadores mais potentes. As alterações feitas nas entidades são diretamente relacionadas ao instante de tempo em que estão sendo executadas e não ao número de iterações do *Game Loop* já ocorridas. Caso contrário, os processadores mais potentes que executam o *Game Loop* mais rapidamente correm o risco de deixar o jogo impossível de ser jogado ou ainda desequilibrado. Assim, a saída deste loop interno ocorre quando está no momento de mostrar o estado do jogo ao jogador. Por exemplo, se um jogo está sendo executado a 60 FPS (*Frames Per Second*) e o *Game Loop* começa no instante 0.0, então a renderização precisa ser feita no instante 0.016 segundos (1/60). Ignorando a fase de entrada e supondo que cada execução da fase de atualização exija 0.4 segundos nesta plataforma, a fase de atualização será executada quatro vezes antes do inicio da fase de renderização. Assim jogadores com plataformas mais potentes são recompensados com um jogo mais suave, porém jogadores com computadores menos potentes também conseguem executar esse mesmo jogo. Dessa forma, é garantido que o jogo está executando a 60 FPS. Além disso, jogadores com computadores inferiores também não serão prejudicados, já que as atualizações das entidades são feitas considerando o instante de tempo não o número de atualizações já ocorridas.

Na figura 2.15 [14] é possível observar uma situação em que se a fase de atualização não for implementada corretamente o jogador com um processador mais potente poderia ser prejudicado. Nessa figura temos dois personagens, um goblin que é o avatar do jogador e um cavaleiro que é um inimigo. O cavaleiro faz uma patrulha entre as duas barras vermelhas e fica indo e voltando neste trecho. É possível também observar cada um dos *frames* em que o goblin é desenhado, mostrando assim como o goblin se movimenta ao longo do tempo. Como o jogador é obrigado a pular dentro da área em que o inimigo está patrulhando para prosseguir, um jogador com um processador mais potente talvez não tivesse tempo para reagir antes do cavaleiro atacar o goblin.

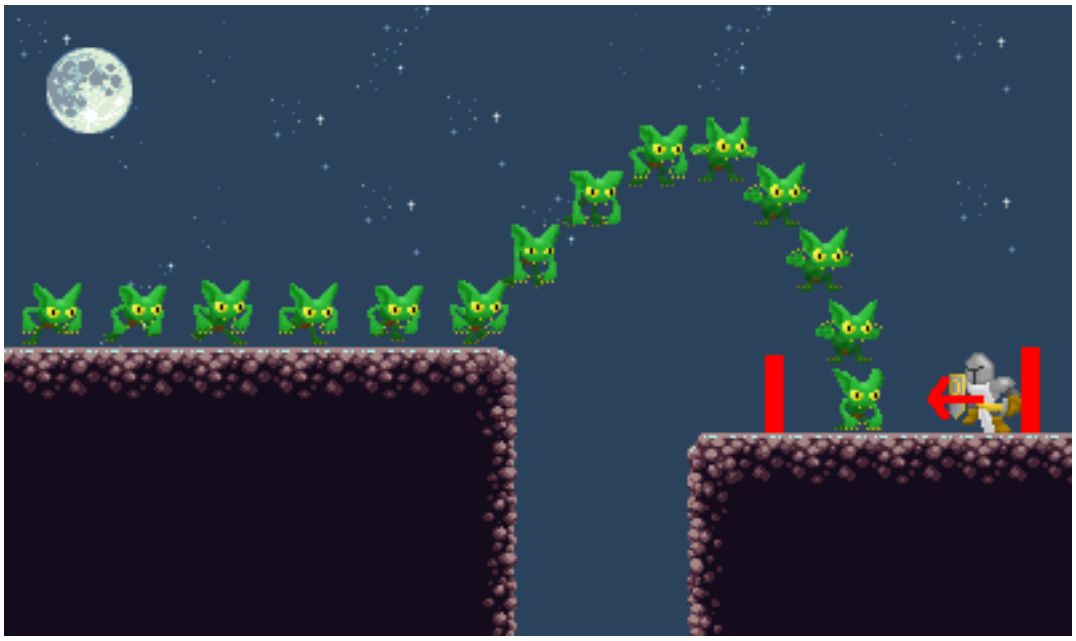


Figura 2.15: Exemplo de situação que se tornaria impossivel de ser jogada caso o *Game Loop* não existisse.

Após a fase de atualização, é executada a fase de renderização. Nesta fase todas as entidades visíveis são mostradas ao jogador de forma visual. Em jogos com duas dimensões as técnicas para a geração de animação utilizam apenas 1 sequência de imagens e é dito que a cena é desenhada. Porém, em jogos que utilizam as três dimensões para a geração da cena são utilizadas técnicas de computação gráfica. É importante ressaltar que atualmente muitas técnicas que eram apenas utilizadas em ambientes 3D estão sendo

[14] Para a construção desta cena exemplo foram utilizadas imagens criadas por Brent Anderson e que podem ser encontradas em: http://www.dumbmanex.com/bynd_ freestuff.html

adaptadas para jogos em 2D também, como iluminação, por exemplo. É utilizado o termo "renderização"para referenciar esta fase do *Game Loop*.

CAPÍTULO 3

TÉCNICAS DE PREDIÇÃO DE MOVIMENTO EM JOGOS

Neste capítulo primeiro discorre-se sobre o ambiente de jogos.Na Seção 3.1 é apresentado o conceito de área de interesse dos jogadores existente em MMOGs que visa reduzir a quantidade de objetos tratados por cada cliente. Em seguida, na Seção 3.2, os problemas causados pela existência da latência na transmissão das mensagens para todos os clientes são apresentados. Na Seção 3.3 o algoritmo de *Dead Reckoning* é apresentado. Este algoritmo é uma solução utilizada para o problema da atualização da posição dos jogadores entre os *frames* de renderização nos jogos *multiplayer*. Em seguida, na Seção 3.4 o algoritmo de *AntReckoning* é apresentado como uma forma de incorporar o interesse dos jogadores ao algoritmo original do *Dead Reckoning*. Por último, na Seção 3.5 são apresentados os conceitos de aprendizado de máquina utilizados para fazer a predição do movimento do jogador.

3.1 Área de Interesse

Em jogos *multiplayer*, os clientes mantêm cópias dos objetos presentes no mundo do jogo em seu computador. Quando um jogador interage com algum desses objetos, todos os outros clientes que são afetados por essa ação devem ter o seu estado interno do jogo atualizado. A maneira mais simples de tratar esse problema é que todos os clientes mantenham uma cópia completa do estado do jogo e que a cada atualização comunique essa alteração para todos os outros clientes. O problema é que essa abordagem se torna inviável em MMOGs devido à grande quantidade de jogadores presentes. Uma solução encontrada para este problema são as áreas de interesse (*Area of Interest* - AOI). O cliente só recebe as informações que são relevantes sob seu ponto de vista, normalmente considerando a posição e o campo de visão do avatar dentro do mundo do jogo. Dessa forma, os clientes apenas mantem réplicas dos objetos que são interessantes para eles.

Apesar da grande quantidade de gêneros e estilos dos jogos atuais, existem apenas quatro tipos de entidades com as quais o jogador pode interagir em jogos *multiplayer*: os objetos imutáveis, os avatares, os objetos mutáveis e os personagens não jogáveis (NPCs - *Non Playable Character*) [44]. Os objetos imutáveis são objetos criados *offline* e que nunca são alterados durante o jogo. Esses objetos normalmente são instalados no cliente e são inicializados no início do jogo como, por exemplo, objetos de cenário, o mapa do jogo, os modelos dos inimigos, dentre outros. Os avatares são personagens que são controlados por um jogador. Os avatares normalmente têm uma posição no mundo do jogo e uma aparência. Os jogadores, em geral, apreciam poder personalizar a aparência de seus avatares, mesmo que isso tenha efeito apenas estético. Já os objetos mutáveis são instâncias de objetos com os quais o jogador pode interagir e usar em si mesmo ou em outros jogadores. Exemplo desses objetos são poções que o jogador pode utilizar para recuperar a saúde de seu avatar. Os personagens não jogáveis são personagens ou ainda avatares que não são controlados por um jogador, mas normalmente através de inteligência artificial. São muito utilizados para contar uma história, entregar missões ao jogador ou simplesmente trazer mais realismo para o ambiente fazendo ações rotineiras.

As informações relevantes para o jogador geralmente correspondem à percepção de seu avatar [44, 9]. Por conta, disso é necessário considerar a posição e o campo de visão dentro do mundo do jogo. Entretanto, a área de interesse pode ser facilmente modelada como uma esfera que envolve o avatar do jogador. Fazendo assim, o jogador pode vir a receber informações sobre objetos que não são visíveis. Na figura 3.1 é possível observar um exemplo dessa situação. Nesta figura temos o círculo maior representando a área de interesse do jogador, enquanto o círculo menor representa a área de atuação do avatar do jogador. Assim, o objeto 1 está dentro da área de interesse do jogador, porém ele não pode nem interagir ou mesmo observar esse objeto, pois tem uma parede obstruindo a visão do jogador. O objeto 2 está tanto dentro da área de interesse quanto dentro da área de atuação do jogador. Enquanto isso o objeto 3 pode ser observado, porém o avatar do jogador não pode interagir com ele, devido a distância.

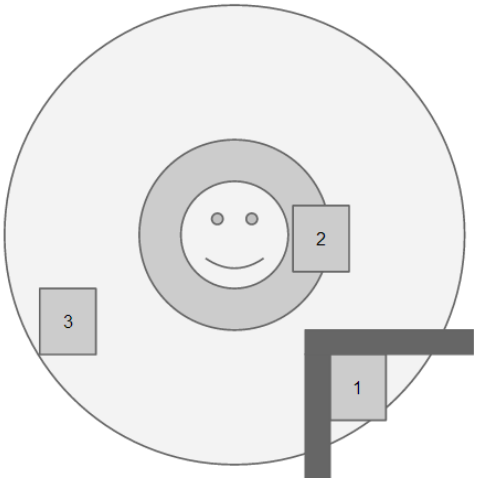


Figura 3.1: Exemplos de posicionamento de objetos na área de interesse do jogador.

3.2 O Problema da Latência

Um dos grandes problemas nos jogos *multiplayer* é causado pelo atraso na transmissão das mensagens. Por conta desse atraso e da necessidade intrínseca de vários jogos serem dinâmicos, muitas informações se tornam desatualizadas rapidamente. Apesar de ser possível reduzir o atraso na transmissão das mensagens, é impossível anulá-lo por completo. De acordo com *survey* [44] os jogadores toleram uma latência entre 100 e 300 milissegundos, dependendo da ação enviada.

Essa latência pode ocasionar vários tipos de problemas de consistência. Na figura 3.2 é possível observar um caso de inconsistência causado por conta da latência. Nessa figura temos a visão de dois jogadores e o jogador 1 utiliza uma habilidade com o objetivo de acertar o jogador 2. Considere que a última mensagem atualizando a posição do jogador 2 ainda não foi entregue ao jogador 1. Dessa forma, na visão do jogador 1 essa habilidade acerta o jogador 2, porém na visão do jogador 2 isso não acontece.

Esse tipo de problema pode ser resolvido utilizando uma das três abordagens : técnicas preditivas, técnicas de atraso das entradas (*delayed input techniques*) ou ainda a técnicas de compensação de tempo (*time-offsetting techniques*) [35]. As técnicas preditivas normalmente envolvem dois componentes. O primeiro utiliza funções de extrapolação para estimar o estado atual dos outros jogadores. O segundo mecanismo é necessário para fazer correções caso a predição esteja errada. Um exemplo de técnica preditiva é o *Dead*

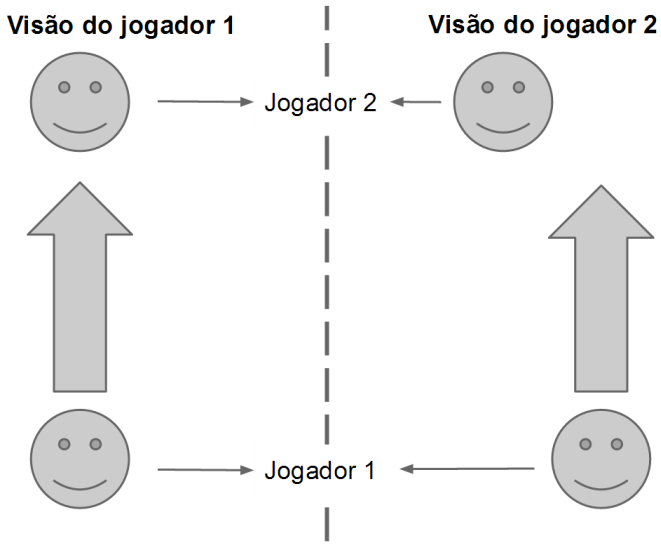


Figura 3.2: Exemplo de inconsistência gerada pelo atraso na entrega das mensagens.

Reckoning. Técnicas de atraso de entradas requerem um mecanismo para atrasar as conseqüências das entradas feitas pelo jogador de maneira que permita a execução síncrona por todos os clientes. Para utilizar esse tipo de abordagem é necessário considerar o tempo necessário para que cada mensagem seja entregue à todos jogadores. Já as técnicas de compensação de tempo fazem a renderização das entidades do jogo em diferentes tempos nos diferentes clientes considerando um atraso fixo e maior que a latência. Portanto, é apresentada uma versão antiga dos objetos e jogadores remotos, dessa forma cada jogador tem sua percepção única. Essa abordagem é muito utilizada quando os desenvolvedores acreditam que é imprescindível dar uma representação exata das ações dos outros jogadores, mesmo que essas ações sejam apresentadas atrasadas.

É importante ressaltar que esse problema da latência não está presente em jogos com turnos bem definidos, já que a troca de turnos normalmente permite a sincronização do estado do jogo em todos os clientes antes que o jogador da vez possa fazer sua jogada. Além disso, como os mecanismos necessários para a sincronização desses eventos normalmente são caros computacionalmente, apenas eventos de grande impacto no estado do mundo do jogo utilizam algum desses mecanismos.

3.3 Dead Reckoning

O objetivo do jogo pode variar, mas em muitos jogos existem missões como coletar objetos, enfrentar e vencer outras entidades ou ainda interagir com NPCs. Todas essas ações podem acabar exigindo que o jogador se desloque de um ponto a outro do mapa e essas atualizações costumam ser grande parte do volume de mensagens geradas nesses jogos. Originalmente o *Dead Reckoning* foi proposto para a redução da quantidade de mensagens necessárias para a comunicação das atualizações de posição dos objetos. Assim, em [43], o problema do *Dead Reckoning* foi modelado como segue:

Considere um jogo que é executado em ciclos discretos de eventos chamados de *frames*. Em cada *frame* o cliente precisa saber a posição atual de todos os avatares visíveis. O cliente recebe a atualização da posição de cada avatar por meio de mensagens. Considere um cliente Q que conhece a posição x_t do avatar P no tempo t. Considere agora que se passou um intervalo de tempo δ_t o cliente Q precisa estimar a posição avatar em $x_{t+\delta t}$. A última atualização que Q recebeu continha a posição x_t, uma velocidade estimada v_t e uma aceleração a_t também estimada do avatar de P no instante de tempo t.

Em [27] é apresentado o *Dead Reckoning* como um algoritmo para prever a posição e orientação de outros objetos entre duas atualizações. Para isso o *Dead Reckoning* estima a posição de um objeto utilizando as atualizações anteriores e equações de movimento. Assim, só é necessário o envio dessa atualização caso o erro seja superior ao aceitável. Em [27] são definidas que as variáveis mais importantes para o *Dead reckoning* são: a última posição conhecida x_t da entidade, a velocidade $v_t = \dot{x}_t$, a aceleração $a_t = \ddot{x}_t$ e t que representa o instante de tempo. Outras informações que ajudam a estimar as forças que atuam nessa entidade também podem ser incluídas. Para objetos representados em um espaço tridimensional também são considerados a orientação, a velocidade angular e possivelmente a aceleração angular. A partir da definição dessas variáveis a trajetória de um objeto pode ser descrita utilizando as das equações de movimento, mais especificamente pela da segunda lei de Newton, que utiliza a aceleração a_t de um objeto, a massa m e as forças F que esse objeto está submetido. Na prática, uma aproximação polinomial derivada das séries de Taylor é utilizada para prever a posição e é apresentada na equação

3.1. É importante notar que esse preditor tem precisão apenas para valores pequenos de δt.

$$x_{t+\delta t} = x_t + v_t\delta + \frac{1}{2}a_t\delta t^2 \tag{3.1}$$

A estimativa da velocidade e da aceleração é comumente feita por meio de das observações anteriores utilizando média móveis exponencialmente ponderadas (*Exponential Moving Average* - EMA). A equação 3.2 apresenta a estimativa para a v_t e a equação 3.3 apresenta o cálculo para a_t. As estimativas da velocidade e da aceleração são feitas utilizando uma soma ponderada considerando o valor atual e os respectivos valores anteriores. Os pesos dos valores anteriores são reduzidos de maneira exponencial ao longo do tempo.

$$v_t = \alpha_v \frac{x_t - x_{x_\delta t}}{\delta_t} + (1 - \alpha_v)v_{t-\delta t} \tag{3.2}$$

$$a_t = \alpha_a \frac{x_t - x_{x_\delta t}}{\delta_t} + (1 - \alpha_a)a_{t-\delta t} \tag{3.3}$$

Utilizando essas estimativas cada cliente deve manter um modelo de si mesmo e um modelo com a posição estimada pelo *Dead Reckoning*. Um limiar deve ser estabelecido comparando a posição e a orientação reais e as previstas. Quando os valores estimados e os reais estiverem com uma diferença maior que o limiar aceitável, o cliente deve comunicar aos outros jogadores sua posição atual. Cada cliente também deve manter um modelo de posição/orientação para todas as outras entidades que estiverem dentro de sua área de interesse. Essas posições/orientações estimadas das outras entidades são utilizadas para renderizar a posição/orientação das entidades no jogo.

Quando uma nova mensagem é recebida de um outro cliente, normalmente a correção da posição é necessária para manter tal entidade com sua posição mais precisa o possível. Entretanto, ao colocar esse objeto nessa nova posição recebida é possível que sejam causados pulos entre a posição renderizada entre um *frame* e outro. Dessa forma é necessária a utilização de uma técnica de suavização de movimento.

O *Dead Reckoning* se mostra extremamente eficiente em ambientes que respeitam as

regras da física. Porém, em jogos é muito comum que as regras físicas sejam relaxadas e mudanças drásticas de movimento são permitidas. É importante ressaltar que os clientes só recebem informações sobre os objetos que estão dentro de sua AOI em MMOGs.

3.4 AntRecknoing

Para conquistar objetivos maiores no jogo, como derrotar um grande vilão, encontrar uma relíquia perdida ou apenas matar o maior número de jogadores do outro time, existem objetivos menores que devem ser realizados. Em [43] é apresentado o algoritmo preditivo *AntReckoning* que é inspirado em colônia de formigas e tem como objetivo a utilização de modelos de interesse para prever a movimentação dos jogadores. As principais contribuições do *AntReckoning* são a incorporação do interesse do jogador na equação de movimento utilizada pelo *Dead Reckoning* e prover um arcabouço que utiliza feromônios para a modelagem considerando aspectos temporais e espaciais do interesse dos jogadores.

Para a modelagem do interesse dos jogadores é necessário considerar: (1) itens no jogo que atraem os jogadores. Isso é especialmente verdadeiro se os itens forem valiosos ou caso o jogador precise deles urgentemente. Como exemplo dessa situação podemos citar uma arma poderosa ou um kit-médico, caso o jogador esteja ferido. (2) Os jogadores também são atraídos por avatares que estão seguindo e repelidos por avatares que estão os perseguem. Como exemplo, podemos citar jogadores que estão seguindo o líder do clã durante o ataque a um castelo inimigo, ou ainda um jogador que está tentando fugir de outros jogadores durante uma caçada. (3) Locais populares no jogo atraem os jogadores. Como exemplo temos as cidades, o topo de uma colina, locais que dão alguma vantagem estratégica, entre outros. (4) Existem lugares também que repelem os jogadores, como por exemplo, locais que não são seguros ao jogador, ou ainda locais em que existam inimigos muito fortes comparados ao nível do avatar.

Para incorporar o interesse dos jogadores na estimativa da posição desses avatares foi criado um modelo de forças de atração e repulsão. A intensidade das forças exercidas pelos pontos de atração (*Point of Interest* - POI) dependem de sua atratividade que pode ser determinada ou ainda aprendida. O *AntReckoning* trata os POIs como formigas que geram

feromônios para modelar a atração dos jogadores. Ao longo do tempo esses feromônios se propagam pelo mundo do jogo e a concentração acaba decaindo. Assim, naturalmente, fatores temporais e a geometria do mundo do jogo são simulados. Na tabela 3.1 é possível observar os principais parâmetros utilizados no *AntReckoning* e que serão referenciados ao longo deste trabalho.

Parâmetro	Descrição	Valor
δ	Quantidade de *frames* desde a última atualização	Variável
α	Coeficiente de aceleração versus forças	0.5
α_v	Coeficiente da velocidade no cálculo da EMA	0.8
α_a	Coeficiente da aceleração no cálculo da EMA	0.8
R	Região de atração	Variável
ε	Coeficiente de evaporação (% remanescente)	Variável
γ	Coeficiente de disseminação (% disseminação)	Variável
k	Redução do poder de atração ao longo do tempo	2
$atratividade$	Atratividade dos objetos dos avatares	Variável
ρ	Quantidade de feromônio gerada	40
ph_{max}	Máximo de feromônio depositado em uma célula	100
C	Tamanho da célula	Variável
η	Limiar	Variável

Tabela 3.1: Parâmetros importantes para o *AntReckoning*.

A incorporação do interesse do jogador na equação de movimento do *Dead Reckoning* feita pelo *AntReckoning* é feita pela da equação 3.4. Nessa equação temos que δt é a quantidade de *frames* que passaram desde a última atualização recebida, F_t é a soma das forças de atração exercidas pelos feromônios em P e pelo ambiente, como por exemplo a gravidade, enquanto v_t é a velocidade estimada e a_t é a aceleração respectiva. O *AntReckoning* considera que as estimativas da velocidade e aceleração são feitas utilizando as equações do EMA, descritas na Seção anterior.

$$x_{t+\delta t} = x_t + v_t \delta t + \frac{1}{2}\left(\alpha \frac{1}{m} F_t + (1-\alpha) a_t\right) \delta t^2 \tag{3.4}$$

Assim como é comum em muitos jogos, o *AntReckoning* assume que o mundo do jogo é divido em células que não se sobrepõem. Essas células são construídas utilizando algoritmos, como por exemplo, o algoritmo de triangulação de Delaunay, o algoritmo de Voronoi ou ainda utilizam grades. Essas células normalmente já existem nos jogos e são utilizadas de diversas formas como a localização de caminhos, detecção de colisão

ou ainda renderização gráfica. Em [43] os exemplos apresentados utilizam uma grade de células quadradas. Denota-se por C o tamanho da célula na unidade do mundo do jogo. A gerência dos feromônios e as forças de atração exercidas por eles dependem da granularidade da célula: para cada avatar P, o cliente Q executa o algoritmo de *Dead Reckoning*, Q calcula a concentração de feromônio em cada célula e a respectiva soma das forças exercidas. Por questão de escalabilidade, apenas as células de uma região limitada em volta de P, chamada de zona de atração e denotada por R, é considerada no cálculo. É importante ressaltar que como os feromônios se propagam, mesmo POIs que não estão dentro de R são considerados. Outra questão é o fato de que mesmo no *AntReckoning* o cliente também precisa executar o algoritmo para si mesmo, já que precisa saber quando deve se comunicar com os outros clientes sobre seu posicionamento, considerando o limiar η escolhido.

Na figura 3.3 é possível observar um exemplo em que o *AntReckoning* é executado para um avatar P. A figura temos o avatar P, um outro avatar e um objeto de interesse, representados por um quadrado, um círculo e um triângulo respectivamente. Além disso, a tonalidade de cinza representa a concentração de feromônios em uma célula e o vetor de força de atração/repulsão de cada uma dessas células pode ser observado pelas linhas tracejadas. A nova posição estimada considera o vetor de deslocamento anterior e o somatório das forças de atração divido pela massa do avatar. A trajetória percorrida anteriormente pelo avatar P pode ser observada na linha pontilhada. Destaca-se que apenas as células dentro do quadrado tracejado estão sendo consideradas para o cálculo da nova posição.

A equação 3.5, apresentada em [43], trata da variação da concentração de feromônio em cada célula do mundo jogo. Para isso, quatro termos são considerados: a geração, a evaporação, a disseminação de entrada (*incoming dissemination*) e a disseminação de saída (*outcoming dissemination*). O primeiro termo, o termo de geração, representa a quantidade de feromônio gerada pelos POIs nessa célula. Os feromônios evaporam ao longo do tempo, assim o segundo termo descreve esse comportamento, se isso não acontecesse a quantidade de feromônios no mundo do jogo cresceria indefinidamente. O terceiro

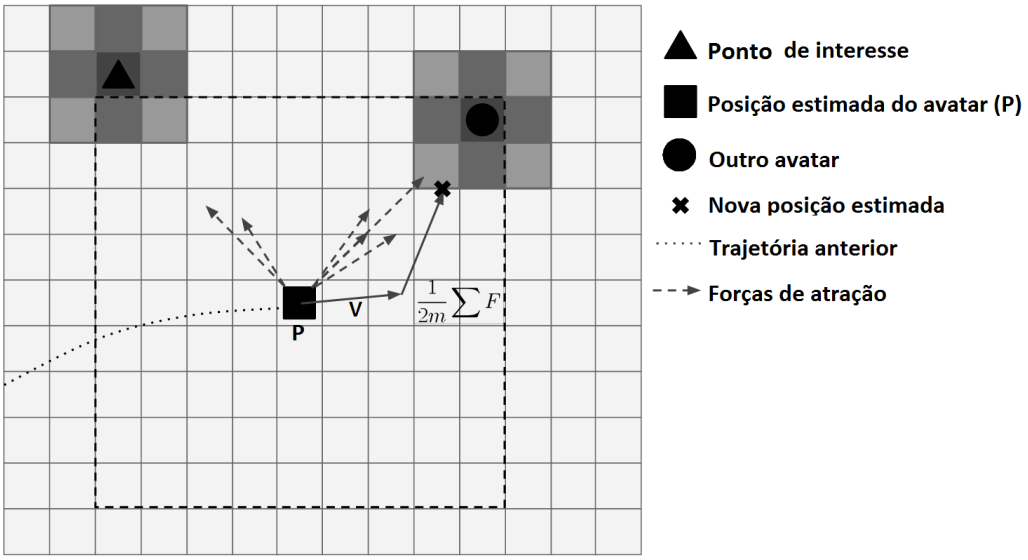


Figura 3.3: Exemplo de execução do algoritmo *AntReckoning*.

e o quarto termos são referentes à disseminação que ocorre quando os feromônios se espalham para as células vizinhas. Ao longo do tempo, uma certa quantidade de ferormônio, definida através do coeficiente γ de disseminação, é realocado de uma célula para sua vizinhança, essa vizinhança é representada pelo conjunto $N(\cdot)$. Destaca-se que um avatar nunca é considerado como um POI para si mesmo. Além disso, por questões de desempenho, quando a concentração de feromônio de uma determinada célula é menor que um limiar, essa concentração é ignorada já que a força de atração é considerada desprezível.

$$ph_t(celula) = \overbrace{\sum_{POI \epsilon celula} atratividade(POI, P)}^{geracao} + \overbrace{\sum_{\epsilon} \varepsilon ph_{t-\delta t}(celula)}^{evaporacao} + \underbrace{\sum_{c \epsilon N(celula)} \frac{\varepsilon \cdot \gamma}{|N(c)|} ph_{t-\delta t}(c)}_{Disseminacao_de_entrada} - \underbrace{\sum_{\epsilon} \varepsilon \cdot \gamma ph_{t-\delta t}(celula)}_{Disseminacao_de_saida} \tag{3.5}$$

No *AntReckoning* as forças de atração exercidas por uma célula em um avatar são representadas por um vetor. Esse vetor tem a direção definida pelo ângulo feito entre a posição do avatar e o centro da célula. A intensidade desse vetor é proporcional à concentração de ferormonio nessa célula dividida pela distância entre o avatar e a célula elevado ao parâmetro k, como pode ser observado na equação 3.6. Este parâmetro k é

definido como um parâmetro do sistema e deve ser ajustado conforme a necessidade do jogo.

$$\|f_t(celula, x_t)\| = \frac{ph_t(celula)}{d(celula, x_t)^k} \tag{3.6}$$

É importante ressaltar que apesar de somente terem sidos citados exemplos em que os POIs atraem os jogadores, para a simulação de POIs que façam repulsão é necessário considerar apenas que o POI gera um valor negativo de feromônios. Dessa forma, a força gerada por esses feromônios em um avatar P será negativa.

3.5 Aprendizado de Máquina

Esta Seção apresenta os conceitos básicos de aprendizado de máquina e está organizada em Subseções. A Subseção 3.5.1 apresenta as fases do aprendizado de máquina. Em seguida, na Subseção 3.5.2, são apresentadas que tipo de tarefas que podem ser resolvidas utilizando aprendizado de máquina. Após, são apresentados os algoritmos utilizados nesta dissertação. Por último, é apresentado o conceito de matriz de confusão que pode ser utilizado para auxiliar na avaliação dos modelos gerados por algoritmos de aprendizado de máquina.

3.5.1 Fases do Aprendizado de Máquina

Aprendizado de máquina é um ramo da inteligência artificial que faz intersecção entre a computação e a estatística, que busca compreender quais são as leis fundamentais que regem os processos de aprendizado com o objetivo de construir sistemas que aprendam para melhorar seus resultados de forma automática [24].

Atualmente as técnicas de aprendizado de máquina têm sido aplicadas com o objetivo de filtrar e-mails [3], em sistema de recomendações [29], para detecção de fraudes [39], dentre muitas outras aplicações [41]. De acordo com [24], para a construção desses sistemas existem cinco fases distintas: Entendimento do Negócio, Entendimento dos Dados, Preparação dos Dados, Modelagem e a Avaliação. Estas fases são descritas a seguir.

A fase de Entendimento do Negócio tem foco na compreensão do objetivo que se deseja atingir com o aprendizado.

Na próxima fase, durante o Entendimento dos Dados, como as fontes fornecedoras dos dados podem ser diversas, é necessário entender os dados e as suas relações. Este entendimento dos dados deve procurar uma descrição clara do problema, identificar os dados relevantes e certificar que as variáveis relevantes para o aprendizado são interdependentes. É importante ressaltar que os dados trabalhados podem ser categorizados em dois grupos: quantitativos de qualitativos. Os dados quantitativos são representados por valores numéricos e podem ser ainda discretos ou contínuos. A outra categoria é a dos dados qualitativos que podem ter valores nominais ou ordinais. Assim que os dados são conhecidos, é necessária uma compreensão maior, inclusive identificando os valores que podem comprometer a qualidade do modelo construído. Esses valores podem ser brancos ou nulos, viciados, ou ainda valores duplicados e são tratados na fase de Preparação.

A fase de Preparação também é chamada de Pré-Processamento por alguns autores e comumente é a fase mais longa de todo o processo. É comum que os dados encontrados contenham várias inconsistências, como registros incompletos ou valores errados. Existem técnicas de limpeza de dados que visam eliminar estes problemas de forma que não tenham influência nos resultados obtidos pelos algoritmos. Essas técnicas podem ser baseadas em remoção de registros com problemas, atribuição de valores padrões, aplicação de técnicas de agrupamento para auxiliar na descoberta dos melhores valores, dentre outras. Além dos valores inconsistentes, é possível que os dados venham de diversas fontes: bancos de dados, arquivos texto, planilhas, *data warehouses*, vídeos, imagens, entre outras fontes. Com isso, surge então a necessidade da integração destes dados em um repositório único e consistente. Para a construção deste repositório é necessária uma análise aprofundada dos dados observando redundâncias, dependências entre variáveis e valores conflitantes. Além disso, alguns algoritmos apenas trabalham com valores numéricos enquanto outros apenas com valores qualitativos. Portanto, em alguns casos, é necessário transformar valores numéricos em qualitativos e vice-versa. Como não existe um critério único para essa transformação dos dados, diversas técnicas são empregadas como a remoção dos

valores errados, agrupamento de valores em faixas, a conversão de valores específicos para valores genéricos, a normalização e ainda é possível que novos atributos sejam criados a partir dos outros já existentes. Por último é comum que o volume dos dados utilizados seja grande. Em alguns casos, o processo de análise dos dados e o próprio aprendizado se torna impraticável. Assim, é comum que técnicas de redução de dados sejam aplicadas com o objetivo de converter a massa de dados original em uma massa de dados menor buscando não perder a representatividade dos dados originais, construindo-se assim uma amostra do repositório.

Em seguida, na fase de Modelagem, as técnicas de aprendizado são aplicadas. Para a construção do modelo é comum que a amostra seja dividida em três conjuntos: o Conjunto de Treinamento, o Conjunto de Testes, o Conjunto de Validação. O Conjunto de Treinamento contém os registros que serão utilizados para a construção do modelo, já o Conjunto de Testes contém os registros usados para testar o modelo e o Conjunto de Validação apresenta os registros utilizados para validar o modelo. Essa divisão em grupos é necessária para que o modelo não fique dependente de um conjunto de dados específico, e quando submetido a outros conjuntos, apresente resultados insatisfatórios.

Durante a etapa de Avaliação são utilizadas ferramentas gráficas com o objetivo de visualizar e analisar os modelos construídos. Para isso, são feitos testes e validações com o objetivo de avaliar a confiabilidade dos modelos construídos durante a fase de Modelagem. Ressalta-se que foi construído um conjunto de dados para a validação do modelo durante a etapa anterior, assim são obtidos indicadores com o objetivo de auxiliar a análise dos resultados.

3.5.2 Tarefas e Problemas de Aprendizado de Máquina

Os problemas que podem ser resolvidos utilizando de aprendizado de máquina podem ser classificados em dois grupos: os que utilizam aprendizado supervisionado e os de aprendizado não supervisionado [12]. Os problemas que podem ser resolvidos através de aprendizado supervisionado utilizam um conjunto de dados de entrada como treinamento. Enquanto, os de aprendizado não supervisionado não utilizam um conjunto de dados de

treinamento.

Os algoritmos de aprendizado de máquina podem realizar tarefas tais como: Classificação, Predição, Clusterização e Associação [11]. As tarefas de Classificação e Predição são vistas como tarefas supervisionadas, enquanto as tarefas de Clusterização e Associação são tarefas não supervisionadas.

As tarefas de Classificação têm como objetivo catalogar valores de variáveis qualitativas. Por exemplo, pode ser construído um modelo que classifica os clientes de um banco como especiais ou de risco. Em outro exemplo, um laboratório químico pode usar sua base histórica de voluntários e verificar em quais indivíduos uma droga pode ser melhor ministrada. Em ambos os cenários um modelo é construído para classificar qual a categoria de um determinado indivíduo. Quando se busca estimar um valor numérico de uma variável a tarefa pode ser chamada de Regressão. Um exemplo de problema que pode ser resolvido por técnicas de Regressão é a estimativa da pressão arterial ideal para um paciente baseando-se na idade, sexo e massa corporal.

Já as tarefas de Predição têm como objetivo prever um valor possível para uma variável no futuro. As predições numéricas tem como objetivo final a previsão para variáveis contínuas; no caso de variáveis discretas, devem ser utilizadas técnicas de classificação.

As tarefas de Agrupamento funcionam como descrito a seguir. Dado um conjunto de dados de entradas, são construídos agrupamentos, também chamados de *clusters*, que contém os registros mais semelhantes. Portanto, um dado é considerado similar aos elementos pertencentes ao mesmo *cluster* e, normalmente, para o cálculo da similaridade entre as entradas são utilizadas medidas de distâncias tradicionais. Como utiliza o conceito de distância entre os registros, geralmente é necessário realizar a transformação dos diferentes tipos de dados para uma escala comum. As técnicas de Agrupamento são todas consideradas como não-supervisionadas.

Por último, as tarefas de Associação são as mais conhecidas devido ao problema conhecido como Análise da Cesta de Compras. Esse problema consiste em identificar o relacionamento dos ítens mais freqüentes em um determinado conjunto de dados. Um exemplo de resultado que pode ser obtido é : *se* compra leite e pão, *também* compra

manteiga. Este tipo de construção recebe o nome de Regra de Associação. Assim, os métodos desta categoria também são métodos não-supervisionados.

3.5.3 Algoritmos

Nesta Subseção são descritos os algoritmos de aprendizado de máquina utilizados nesta dissertação. Todos os algoritmos apresentados nessa Subseção podem ser utilizados para tarefas de Classificação e de Predição e são algoritmos de aprendizagem supervisionada.

Dentre os algoritmos disponíveis foram escolhidos quatro algoritmos para fazer a predição de movimento: *REPTree*, *LWL*, *Bagging* e o *Multilayer Perceptron*. Cada um destes algoritmos é descrito a seguir.

3.5.3.1 *Reduced Error Pruning Tree*

O algoritmo *Reduced Error Pruning Tree* (REPTRee) , apresentado em [31], inicialmente constrói uma árvore de decisão. Árvores de decisão são estruturas similares às árvores tradicionais e são muito utilizadas para a tarefa de classificação [34]. A classificação feita por uma árvore de decisão é feita pela execução de uma sequência de testes condicionais. Em uma árvore de decisão os nós internos representam as condições que devem ser testadas e os nós folhas são o resultado da classificação, ou seja a classe dessa instância. Assim, a partir do nó raiz, o primeiro nó do topo da árvore, são executados testes até que se encontre um nó folha. A figura 3.4 mostra um exemplo de árvore de decisão para a situação de um aluno em uma matéria. Esse aluno pode ser aprovado, reprovado ou ter que fazer a prova final dependendo de sua nota.

Para reduzir o tamanho de uma árvore de decisão é utilizada uma técnica chamada de poda. Essa técnica busca reduzir o tamanho da árvore sem reduzir a precisão das classificações feitas por essa árvore. O algoritmo REPTree utiliza um método para a poda de árvores de decisão. O algoritmo percorre a árvore a partir das folhas, verificando se pode sustituir cada nó interno sem interferir na precisão dessa árvore de classificação. O processo prossegue até que não seja possível fazer nenhuma substituição que melhore o resultado obtido.

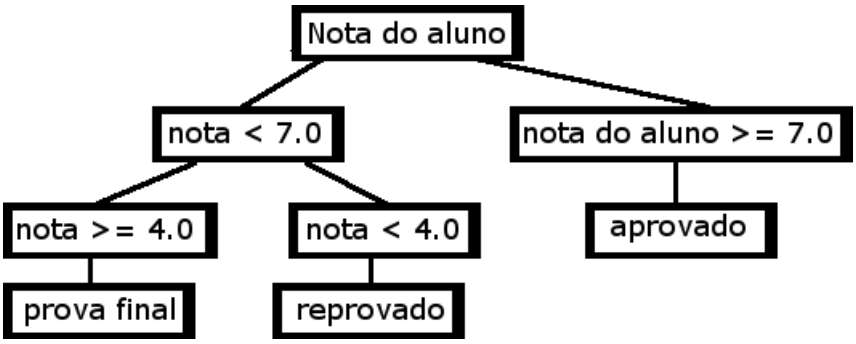


Figura 3.4: Exemplo de uma árvore de decisão.

O algoritmo REPTree tem diversas aplicações. Já foi utilizado para auxiliar na previsão de tempo em conjunto a técnica *Bootstrap Aggregating* [45]. Também foi utilizado para determinar a idade gestacional de recém-nascidos [4], assim como para detecção de intrusão em redes [23]. Dessa forma, é possível observar que o algoritmo REPTree é uma ferramenta que pode ser utilizado em diversos campos do conhecimento humano.

3.5.3.2 *Local Weighted Learning*

Métodos preguiçosos de aprendizado (*Lazy Learning Methods*) [1], também conhecidos como métodos de aprendizagem baseados em memória, se diferem dos métodos tradicionais de aprendizagem, pois adiam o processamento das instâncias de treinamento até que uma consulta seja feita [7]. Isso geralmente envolve armazenar os dados de treinamento em memória e encontrar as instâncias de treinamento relevantes para responder uma consulta. Normalmente, a relevância de cada instância pertencente ao conjunto de treinamento é dada pela distância entre esta instância e a que se deseja classificar. É comum selecionar um conjunto de instâncias próximas da instância que se deseja classificar ou prever e utilizá-las em um processo de votação que tem como objetivo definir o valor da instância consultada. Em [7], é apresentado o algoritmo *Local Weighted Learning* (LWL) que faz um sistema de votação de maneira ponderada considerando o peso de cada uma das instâncias selecionadas na votação. Cada um dos componentes deste sistema de votação é explicado a seguir, seguindo a descrição apresentada em [14].

É definido que para cada instância x_i que pertence ao conjunto de treinamento $D =$

$\{(x_i, y_i)|i = 1, 2, ..., n\}$ existe uma saída correspondente de valor y_i. Deseja-se fazer uma predição $\hat{y}_q$ para uma instância x_q. O peso de uma instância, w_i, é determinado pela função K que considera a distância, $d(x_i, x_q)$, entre x_i e x_q. Essa função K, que também pode ser chamada de função peso, pode convergir para zero e essa propriedade pode ser utilizada para diminuir o custo computacional, já que é possível ignorar todas as instâncias com peso zero. Além disso, é possível que a função K seja trocada, tornando assim o algoritmo LWL mais flexível. Assim, o objetivo é encontrar um valor do coeficiente de regressão β_q para minimizar a equação 3.7 para a instância x_q.

$$J = \frac{1}{2}\sum_{i=1}^{n} w_i(x_q)(y_i - x_i\beta_q)^2 \tag{3.7}$$

Em [36] é demonstrado que o algoritmo LWL pode ser utilizado para aprendizado em tempo real de robôs. Nesse artigo, o algoritmo LWL é apresentado como sendo capaz de solucionar problemas complexos e com uma grande quantidade de dados.

3.5.3.3 *Multilayer Perceptron*

Redes neurais artificiais são modelos estatísticos que foram inspirados no sistema nervoso central dos animais, ou seja no cérebro, e são utilizadas para estimar ou ainda aproximar funções com um grande número de entradas [33]. Uma rede neural artificial é uma rede de nós, chamados neurônios artificiais, que estão interconectados e enviam mensagens entre si e são capazes de computar valores de entrada. Um neurônio é considerado como ativado quando recebe uma mensagem para computar um valor de entrada. Quando um neurônio é ativo o dado de entrada então pode ser ponderado, transformado e então repassado para outros neurônios até que um neurônio de saída seja ativado. A função que faz a ponderação, transformação e que repassa para outros neurônios é definida pelo *designer* de rede neural. Esse processamento feito por cada um dos neurônios ativos pode ser feito de forma paralela, otimizando assim o desempenho da rede neural artificial.

Dentre as redes neurais artificiais existem duas topologias principais: as redes realimentadas (*recurrent*) e as de propagação para frente (*feedforward*). Redes neurais artifi-

ciais realimentadas são redes que têm ligações entre os neurônios sem qualquer tipo de restrição. No caso das redes de propagação para frente o fluxo de informação é unidirecional. Nessas redes neurais artificias com topologia de propagação para frente, os neurônios que recebem a informação simultaneamente podem ser agrupados em camada de entrada, camada de saída e camadas ocultas. Os neurônios presentes na camada de entrada são os neurônios que recebem os dados que são fornecidos para a solução do problema, enquanto os neurônios presentes na camada de saída são neurônios que apresentam a resposta do processamento feito pela rede neural. As camadas de neurônios presentes que não são nem uma camada de entrada e nem camada de saída são chamadas de camadas ocultas.

A figura 3.5 mostra um exemplo de rede neural de propagação para frente com duas camadas ocultas. Nessa figura temos os círculos representando os neurônios enquanto as setas representam as conexões entre eles, definindo assim quais neurônios podem ser ativados a partir de um neurônio da camada anterior.

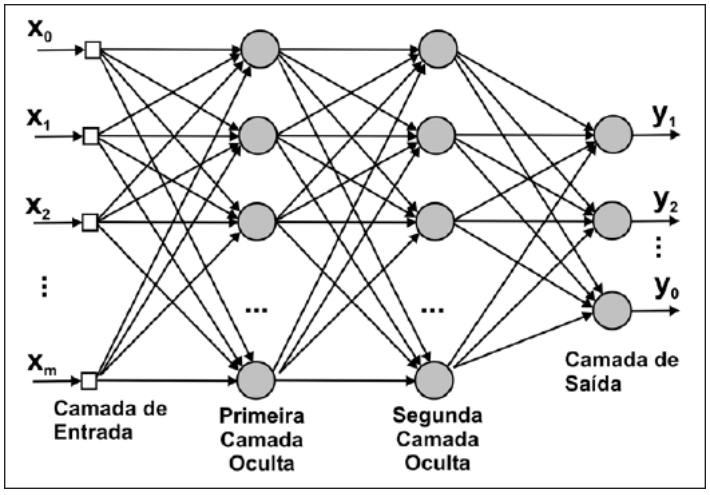


Figura 3.5: Exemplo de uma rede neural com duas camadas ocultas.

Área de redes neurais artificiais de propagação para frente são também chamadas de perceptrons. O caso mais simples de um perceptron é o que contém apenas uma única camada. E são capazes de aprender apenas certos tipos de problemas de classificação. Por outro lado, já perceptrons em multi-camadas, *Multilayer Perceptron*, são utilizados em diversas áreas de conhecimento humano com o objetivo de auxiliar na classificação. Em [25] os perceptrons multi-camadas são utilizados para classificar células para diagnosticar

câncer. Em [21] essas redes são utilizadas para reconhecimento do código de endereço escritos a mão. Já em [8] são usadas para o reconhecimento facial.

3.5.3.4 *Bootstrap Aggregating*

Os conjunto de classificadores (*Ensembles Methods*) usam múltiplos algoritmos de aprendizado de máquina de forma combinada com o objetivo de obter um melhor desempenho na previsão [28]. Dentre os classificadores agregados existe o *Bootstrap Aggregating* [10], também conhecido como *Bagging*. A estratégia utilizada pelo *Bagging* é a de gerar várias versões de um mesmo classificador utilizando diferentes conjuntos de treinamento gerados a partir do conjunto de treinamento original. Dessa maneira, são geradas várias versões de um mesmo modelo. O conjunto de treinamento de cada classificador é gerado de forma aleatória e tem o mesmo tamanho do conjunto de treinamento original. Por conta disso, muitas instâncias do conjunto de treinamento original são repetidas dentro do novo conjunto de treinamento, enquanto outras talvez sejam deixadas de fora. A figura 3.6 apresenta um exemplo de como este método funciona para a geração desses conjuntos de treinamento. Observe que o primeiro conjunto de treinamento tem as instâncias 5 e 1 duplicadas, porém não há nenhuma instância 3 e 2. Os classificadores construídos com esses conjuntos de treinamento obterão individualmente um resultado inferior do que se fosse treinado com o conjunto original. Entretanto, como os classificadores estão sendo combinado, o resultado é superior ao do classificador treinado com o conjunto original de dados.

Após a construção das diferentes versões de um mesmo classificador, é possível utilizá-los para realizar a tarefa de classificação. Para classificar uma instância utilizando o *bagging* é feito um processo de votação para definir à qual classe essa instância pertence. Para essa votação cada versão do classificador deve classificar essa instância e a classe de saída deste classificador é computada como um voto para a mesma. A classe que receber mais votos é a classe a qual esta instância pertence, segundo o *bagging*.

De acordo [10], é mostrado que essa técnica é muito eficiente em algoritmos de aprendizagem em que pequenas mudanças no conjunto de treinamento alteram muito os resul-

Um conjunto de dados imaginários e simples para treinamento	
Conjunto de Treinamento original	1,2,3,4,5,6,7,8,9

Exemplos de conjuntos gerados pelo Bagging do conjunto de treinamento original	
Conjunto de treinamento 1	1,5,7,8,9,4,5,6,1
Conjunto de treinamento 2	2,9,8,3,4,5,7,1,1
Conjunto de treinamento 3	5,3,4,3,1,5,6,7,9
Conjunto de treinamento 4	4,5,8,7,6,7,9,1,2
Conjunto de treinamento 5	3,2,1,3,4,6,9,4,8

Figura 3.6: Exemplo de conjuntos de treinamentos gerados pelo Bagging.

tados obtidos pelos classificadores. É destacado que esse método é muito eficiente quando utiliza internamente redes neurais ou ainda com árvores de decisão.

3.5.4 Matriz de Confusão

A partir da obtenção dos resultados apresentados por um algoritmo de aprendizado de máquina, é necessário fazer um teste para verificar o quão confiável é o modelo. Quando são utilizados algoritmos de Classificação é possível construir uma matriz de confusão. A matriz de confusão, apresentada em [30], contém informações sobre como as entradas foram classificadas, determinando se o valor previsto corresponde ao valor correto. Para obter uma matriz de confusão, é apresentado um conjunto de dados com suas classificações e para cada dado apresentado se obtém a classificação predita e compara-se com a classificação real. Todos os dados são contabilizados e os totais são apresentados em uma matriz. Um exemplo de matriz de confusão para um classificador binário é apresentado na tabela 3.2. Nessa matriz temos as colunas apresentando em quais categorias os valores foram classificados pelo algoritmo, enquanto nas linhas temos as categorias que essas instâncias realmente pertencem. É possível observar que existem duas categorias em que as instâncias podem estar e existem quatro grupos em que uma instância pode ser representada nessa matriz. As instâncias que pertencem aos grupos a e d foram corretamente classificadas. Já as instâncias pertencentes aos grupos b e c estão sendo classificadas de

forma errônea.

	Classificado: Negativo	Classificado: Positivo
Realidade: Negativo	TN = a	FP = b
Realidade: Positivo	FN = c	TP = d

Tabela 3.2: Matriz de confusão exemplo.

As instâncias pertencentes ao grupo a são chamadas de Verdadeiras Negativas (*True Negative* ou TN), são entradas que são negativas e estão sendo classificadas corretamente. As instâncias pertencentes ao grupo b são chamadas de Falsa Positivas (*False Positives* ou FP), são instâncias que são negativas, mas estão sendo classificadas como positivas. As entradas do grupo c são conhecidas como Falsas Negativas (*False Negatives* ou FN), são instâncias que são positivas e estão sendo classificadas como negativas. E por último as entradas que estão dentro do grupo d são chamadas Verdadeiras Positivas (*True Positives* ou TP), são instâncias que estão sendo corretamente classificadas como positivas. A proporção entre esses valores pode ser calculada como apresentados nas equações 3.8 até 3.11

$$TP = \frac{d}{c+d} \tag{3.8}$$

$$FP = \frac{b}{a+b} \tag{3.9}$$

$$TN = \frac{a}{a+b} \tag{3.10}$$

$$FN = \frac{c}{c+d} \tag{3.11}$$

As equações mostram que para calcular a proporção das falsas negativas e verdadeiras positivas é necessário considerar todas as instâncias positivas. Enquanto, para calcular as falsas negativas e as verdadeiras negativas se considera todas as instâncias negativas. Portanto, somando-se as falsas positivas e as verdadeiras positivas, ou somando-se as falsas negativas e as verdadeiras negativas, é obtido o valor total de instâncias positivas e negativas.

CAPÍTULO 4

PREDIÇÃO DE MOVIMENTO NO *WORLD OF WARCRAFT*

Nesse capítulo são apresentadas as etapas da solução baseada em aprendizado de máquina proposta com o objetivo de prever a movimentação de jogadores no jogo *World of Warcraft*. Para isso, este capítulo está estruturado seguindo as fases apresentadas no Capítulo 3 para a construção de um sistema de aprendizado de máquina. Na primeira Seção é descrito o Entendimento do Negócio. Em seguida, na Seção 4.2, em Entendimento dos Dados, os dados utilizados para a solução do problema são apresentados. A construção da nova base e seu pré-processamento são apresentados na Seção 4.3. Por fim, na última, Seção é apresentada a fase de Modelagem e a Avaliação.

4.1 Entendimento do Negócio

Este trabalho apresenta uma solução baseada em aprendizado de máquina com o objetivo de prever a movimentação dos jogadores utilizando algoritmos de aprendizado de máquina. Para realizar esse trabalho foi escolhido o jogo *World of Warcraft*[1] (WoW). Este jogo pertence ao gênero de MMORPGs e constitui uma grande parcela do mercado deste gênero com 100 milhões de contas criadas, sendo jogado em 224 países e territórios, segundo [15]. Esse gênero tem algumas características intrínsecas: a necessidade de uma maneira de medir quantitativamente o progresso do jogador, a interação social com outros jogadores, a possibilidade de personalização dos avatares e uma arquitetura do sistema construída focando a presença de um grande volume de jogadores. Em WoW os jogadores podem observar o seu progresso no jogo por meio do avanço do nível do avatar, nos pontos gastos em habilidades, em equipamentos adquiridos, na riqueza obtida dentro do jogo ou ainda pelas de conquistas obtidas fazendo certas tarefas. Para progredir nesse jogo, os avatares

[1] http://us.battle.net/wow/pt/

batalham com monstros ou ainda fazem missões que são dadas por NPCs. Ambas ações podem ser feitas em grupo ou de forma solitária e normalmente provêm recompensas como pontos de experiência que são utilizados para que os avatares subam de nível. Mais informações sobre o jogo e as ações dos jogadores estão disponíveis no apendice A.

A partir da escolha do jogo foram levantadas quais bases de dados estão disponíveis e a estratégia que deveria ser utilizada. As bases encontradas são apresentadas em 4.1.1. A base de dados escolhida utiliza dados que foram coletados na cidade do jogo chamada de IronForge, mais informações sobre essa cidade são apresentadas em 4.1.2. Em seguida, a estratégia é apresentada em 4.1.3.

4.1.1 Bases de Dados

Em [19] são reunidas várias bases de dados de jogos. Dentre as bases de dados disponíveis existem três construídas com dados de *traces* do jogo *World of Warcraft*. Essas bases de dados estão disponíveis e são chamadas de *WoWAH* [22], *WoWSession* [40] e a *WoWPosition* [38].

A base mais antiga e com maior tempo de coleta de informações é a *WoWAH* [22], o qual contém informações de mais de 90.000 avatares do jogo. Essa base contém dados dos jogadores relativas a tempo de jogo deste avatar, a guilda a que esse avatar pertence, o nível, a raça, a classe, e por fim, em qual zona do jogo este jogador estava naquele momento. Essa base de dados foi construída com o objetivo de fazer a previsão de quando um jogador cancelará sua assinatura no jogo. Infelizmente, essa base, apesar de identificar os avatares e apresentar informações sobre em qual mapa os jogadores estavam, não apresentavam a localização exata dos mesmos dentro dos mapas. Foram apresentados trabalhos que utilizam essa base de dados com o objetivo de prever qual seria a próxima área que um jogador iria.

A base *WoWSession* [40] apresenta informações sobre as sessões dos jogadores no jogo e foram recolhidas com o objetivo de determinar padrões das atividades dos jogadores. As informações coletadas são referentes às ações dos jogadores, entretanto, não há informações exatas sobre a posição dos avatares. Nela é possível observar, por exemplo,

que determinado avatar conversou com um NPC em certo horário e em seguida saiu para caçar um determinado inimigo, porém não é possível determinar qual a rota exata que ele teve entre um evento e outro.

A base de dados *WoWPosition* [38] é uma das bases que foi utilizadas neste trabalho. Essa base de dados apresenta informações sobre a posição dos jogadores. Entretanto, apresenta unicamente esta informação, não tendo qualquer outro tipo de informação sobre nível, raça ou ainda classe de um determinado avatar. Mais detalhes sobre as informações sobre os campos disponíveis são apresentadas na Seção 4.2. Destaca-se que essa base de dados não contém qualquer informação que também possa ser utilizada para identificar o avatar. Dentre os mapas que foram analisados para a construção desta base foi escolhida os dados do arquivo "Ironforge-1d". Essa base contém informações de 1302 jogadores, os quais foram coletadas na cidade de *Ironforge*.

Outra base de dados que foi utilizada neste trabalho é a base *WowHead* [20]. No jogo *World of Warcraft* é possível que os usuários criem extensões com o objetivo de melhorar e personalizar a experiência do jogo. Esses plugins são desenvolvidos utilizando a linguagem de programação LUA e têm seu uso difundido dentre a comunidade de jogadores. O *Thottbot* era um extensão que foi desenvolvida com o objetivo de coletar informações e estatísticas do jogo e foi descontinuado em novembro de 2010. As informações que eram coletadas através deste plugin eram enviadas para um servidor com o objetivo de atualizar uma base de dados conhecida, também, como *Thottbot*. Essa base de dados continha informações como, por exemplo, a taxa de queda de um item, a localização de certos inimigos, a localização de personagens não jogáveis (NPCs) e de missões disponíveis aos jogadores, além de outras informações. Em março de 2008, a empresa Alexa [2] reportou que o site thottbot.com era o 250° site mais visitado de toda Internet. Em 2012, a base de dados do *Thottbot* foi fundida com a base de dados *Wowhead* [20]. A partir dessa base de dados é possível extrair informações sobre os NPCs presentes no mapa da cidade de *Ironforge*. Neste trabalho, essas informações foram utilizadas para construir uma base com informações sobre localização e funções dos NPCs presentes em *IronForge*. A Seção 4.2 traz mais informações sobre os dados utilizados neste trabalho.

4.1.2 A Cidade *Ironforge*

A cidade de *Ironforge* é considerada uma cidade capital da Aliança no jogo *World of Warcraft*. Os jogadores de *World of Warcraft* podem participar de uma das duas facções presentes, a Aliança ou a Horda. Dependendo do servidor em que esses jogadores estejam jogando é possível fazer batalhas do tipo jogador-versus-jogador entre jogadores de facções rivais. É importante ressaltar que como a cidade analisada é uma das capitais da Aliança, é improvável a presença de jogadores com avatares da Horda, e portanto geralmente o comportamento dos jogadores não assume formas agressivas. O que garante isso é que inclusive os NPCs da cidade irão atacar avatares da Horda caso sejam vistos, se tornando assim um ambiente extremamente agressivo aos avatares da Horda. Outra informação importante sobre *Ironforge* é que ela é a cidade inicial para todos os jogadores da raça anã, portanto, é uma cidade que é visitada tanto por avatares poderosos quanto pelos avatares mais fracos. O mapa da cidade é apresentado na figura 4.1; nessa mesma figura é possível visualizar a distribuição dos NPCs na cidade. A cor de cada ponto representa qual a função que este NPC desempenha, as funções são: *Auctioneer, Banker, Battlemaster, Collector, Innkeeper, Merchant, Quartermaster, Quest Giver, Stable Master, Trainer, Weapon Master*.

Há algumas informações importantes que podem ser extraídas deste mapa da cidade. A primeira informação é que somente existe um NPC com a função de *Innkeeper*. Assim todos os jogadores que quiserem definir a cidade de *Ironforge* como ponto de retorno devem ir até esse NPC. Outra informação importante é que os NPCs com a função de *Banker*, *Battlemaster* e os *Collector* estão todos agrupados. OS NPCs que dão missões aos jogadores, os *Quest Givers*, estão espalhados por toda a cidade, assim como os *Merchants* e os *Trainers*. O interesse por qualquer um desses NPCs é muito individual de um jogador, pois um jogador que tenha a profissão de cozinheiro não se interessará pelo mercante que vende ítens para forja ou forneça treinamento em alfaiataria, por exemplo. Além disso, um jogador com um nível muito alto, não se interessará por missões para iniciantes.

Em *World of Warcraft*, a partir do nível 20, os jogadores podem utilizar montarias com uma velocidade maior do que o avatar caminha e acima do nível 60 os jogadores podem

Figura 4.1: Posição dos NPCs dentro do mapa de *Ironforge*

inclusive utilizar montarias que voam. Em alguns mapas mais antigos os jogadores não podem utilizar essas montarias voadoras, mas este não é o caso da cidade de Ironforge, dessa maneira os avatares podem fazer parte ou até mesmo todo o seu trajeto voando.

4.1.3 Estratégia Utilizada

Após a escolha do jogo, foi definida a estratégia para a predição de movimento. A estratégia é utilizada sempre que é necessário renderizar para um determinado jogador um avatar de outro jogador que esteja presente na sua área de interesse. A estratégia proposta tem duas fases bem definidas. Na primeira fase, algoritmos de aprendizado de máquina são utilizados para gerar um modelo que classifica se o *Dead Reckoning* acerta ou erra na predição de movimento. O *Dead Reckoning*, conforme apresentado no Capítulo 3, con-

sidera que um jogador se movimenta em linha reta, utilizando regras físicas para prever a nova posição. Na primeira fase da estratégia proposta, o modelo aprendido deve ser capaz de indicar se apenas com o *Dead Reckoning* é possível fazer a previsão correta. Caso contrário, isto é, quando o jogador muda de direção, ou seja, o ângulo de movimentação é alterado e, portanto, o jogador não se movimenta em linha reta, é executada a segunda fase da estratégia proposta. Na segunda fase, algoritmos de aprendizado de máquina são utilizados para gerar um modelo que faz a predição do ângulo de movimento do avatar. A estratégia foi definida assim, pois não se sabia qual a taxa de acerto do *Dead Reckoning* para o jogo *World of Warcraft*. Assim, em um momento inicial não deveria ser descartada a possibilidade de que o *Dead Reckoning* obtivesse um desempenho alto.

O fluxograma apresentado na figura 4.2 mostra a estratégia utilizada. Nesse fluxograma é possível observar que, para toda nova entrada, é feito um teste que verifica, através do modelo aprendido, se o jogador irá mudar o ângulo de movimentação. Caso o ângulo de movimentação do jogador não seja alterado, deve-se utilizar o *Dead Reckoning*. Entretanto, caso o jogador mude o ângulo da sua movimentação, deve-se utilizar o modelo aprendido da segunda fase para prever qual será o novo ângulo de movimentação. Em ambos os casos, será apresentada uma nova posição do jogador.

A partir da estratégia proposta, foi possível definir também a estratégia de avaliação. A primeira fase consiste de uma tarefa de classificação, portanto, a matriz de confusão é uma métrica relevante para a compreensão de como os algoritmos se comportam. Já a segunda fase consiste de uma tarefa de predição. Para isso serão utilizados histogramas para a compreensão da distribuição de erro dos algoritmos de aprendizado.

Para executar essas tarefas foi utilizado o software Weka [2] (Waikato Environment for Knowledge Analysis) em sua versão 3.6.12. Este software permite a execução tanto de algoritmos supervisionados quanto não supervisionados para todas as tarefas apresentadas em 3.5.

[2]http://www.cs.waikato.ac.nz/ml/weka/

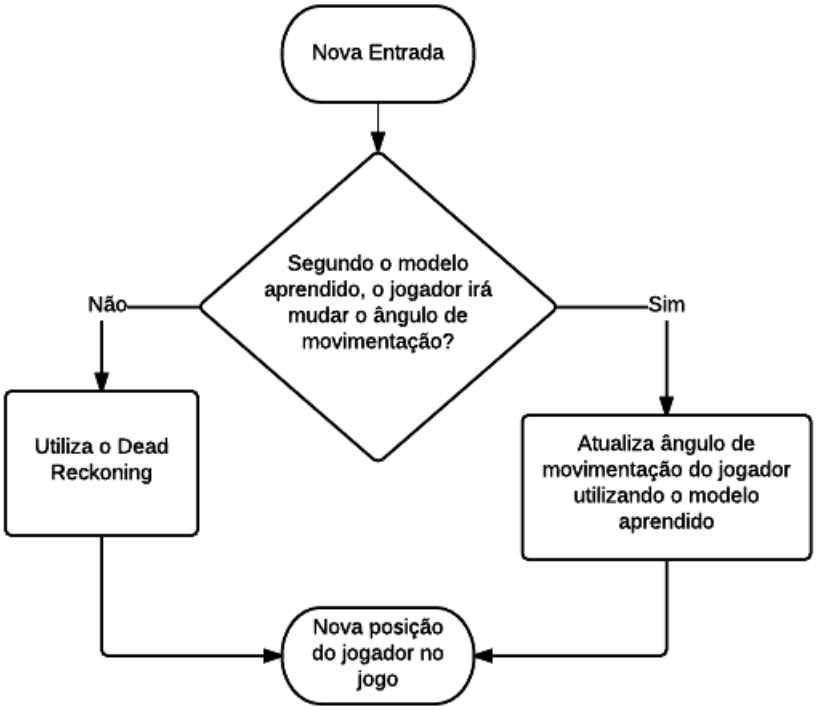


Figura 4.2: Fluxograma da nova abordagem adotada.

4.2 Entendimento dos Dados

Para este trabalho foi construída uma nova base de dados, utilizando os dados disponíveis na base *WoWPosition* e na base *WowHead*. Essas bases de dados são descritas a seguir. A base de dados *WoWPosition* contém mais de 30.000 entradas e é composta por uma tabela com os seguintes campos: *timestamp, id, x, y, z e ângulo de visão*, descritos a seguir. O campo *timestamp* é a quantidade de segundos que se passaram desde do primeiro segundo do primeiro dia do ano de 1970. O campo *id* apresenta um número de identificação utilizado internamente para identificar um jogador. Os jogadores foram anonimizados e, portanto, não é possível extrair mais informações, além das fornecidas na própria base, sobre os mesmos. Os campos *x, y e z* são os dados da posição do jogador dentro do jogo e o *ângulo de visão* representa para qual direção o jogador está olhando.

A base de dados *Wowhead* foi utilizada para construção de uma base de dados dos NPCs. Como descrito na Seção 4.1.1, a base *WowHead* incorporou dados do sistema *Thottbot*, um *plugin* que coletava informações e estatísticas do jogo *World of Warcraft*. Essa base de dados é composta por uma tabela com 4 campos: *Nome, ThottbotXPosition,*

ThottbotYPosition e FunçãoNPC. O primeiro campo é composto pelo nome do NPC, o segundo e o terceiro campos são responsáveis pelo armazenamento da posição do NPC no mapa dentro do sistema de coordenadas do *Thottbot* e o último campo descreve a função deste NPC. Foram criadas as seguintes categorias para descrever a função de um NPC dentro do jogo : *Auctioneer, Banker, Battlemaster, Collector, Innkeeper, Merchant, Quartermaster, Quest Giver, Stable Master, Trainer, Weapon Master*.

O jogo *World of Warcraft* é um jogo em 3 dimensões, entretanto o *Thottbot* faz uma conversão da posição tridimensional de todos o NPCs para um mapa com 2 dimensões [16]. Por conta dessa característica do *Thottbot*, a coordenada que representa a altura é ignorada. O sistema de coordenadas utilizado pelo *Thottbot* consiste uma tupla (x, y) com os valores variando de 0 até 100 e representam uma localização dentro deste mapa com duas dimensões. Nesse sistema, o canto superior esquerdo representa a posição $(0, 0)$ e o canto inferior direito representa a posição $(100, 100)$, dessa maneira, o centro do mapa é representado pela posição $(50, 50)$.

4.3 Preparação

A etapa de preparação neste trabalho foi feita para a construção de uma nova base de dados e do pré-processamento desta nova base. Portanto, em 4.3.1 é apresentada a maneira como a nova base de dados foi construída. Em 4.3.2 é apresentado o pré-processamento que foi aplicado sobre a nova base de dados.

4.3.1 Construção da Nova Base

Utilizando a base de dados criada por [38], a *WoWPosition*, em conjunto com a base de NPCs, foi possível construir uma nova base de dados. A nova base contém os campos apresentados na tabela 4.1. Ela é composta pelos seguintes campos: *ThottbotX*, *ThottbotY*, o *AnguloVisual*, *AnguloMovimentacaoAnterior*, os dados dos outros jogadores e NPCs presentes no campo de visão, o *AnguloMovimentacao* e *DeadReckoning*. Os campos *ThottbotX* e *ThottbotY* indicam a posição do jogador no sistema de coordenadas

do *Thottbot*, enquanto o campo *AnguloVisual* é responsável por indicar para qual direção que o jogador que está sendo análisado está observando. Os campos com as informações dos outros jogadores e dos NPCs são apresentados na tabela 4.2 e serão descritos no próximo parágrafo. Os dois últimos campos presentes são o *AnguloMovimentacao* e o *DeadReckoning*. O campo *AnguloMovimentacao* descreve qual o ângulo em que o jogador se movimentou no respectivo *timestamp* e o campo *DeadReckoning* indica se o algoritmo *Dead Reckoning* acerta aquele caso ou não. Em outras palavras, o campo *DeadReckoning* deve ser utilizado para o treinamento dos modelos de aprendizagem na primeira fase da nova estratégia. Enquanto, o campo *ÂnguloMovimentacao* é utilizado durante a segunda fase da estratégia proposta. Portanto, quando se está na etapa de o aprendizado para determinar se o *Dead Reckoning* irá acertar ou não, durante a primeira fase, o campo *AnguloMovimentacao* é ignorado. Já quando está sendo feito o aprendizado do modelo da segunda fase, quando de fato é determinado o novo ângulo de movimentação do jogador, o campo *DeadReckoning* deverá ser ignorado. Por conta disso, o campo *AnguloMovimentacao* e o campo *DeadReckoning* são mutuamente exclusivos, quando um está sendo considerado o outro é ignorado.

Nome do Campo	Tipo de dado
ThottbotX	Real
ThottbotY	Real
AnguloVisual	Real
AnguloMovimentacaoAnterior	Real
Dados dos Jogadores	Múltiplos dados
Dados dos NPCs	Múltiplos dados
AnguloMovimentacao	Real
DeadReckoning	Booleano

Tabela 4.1: Campos presentes na nova base de dados.

Os campos dos jogadores e NPCs, presentes na tabela 4.2, são: *X*, *Y*, *Distância*, *Angulo* e *Função*. Os campos *X* e *Y* apresentam a posição destes jogadores ou NPCs no sistema de coordenadas do *Thottbot*. O campo *Distância* mostra a distância entre o jogador analisado e um outro jogador ou NPC, enquanto o campo *Angulo* indica o ângulo entre os dois. O campo *Função* é exclusivo dos NPCs e mostra qual a função de um

determinado NPC no jogo.

Campos dos Jogadores e NPCS	Tipo de dado
X	Real
Y	Real
Distância	Real
Ângulo	Real
Função (Exclusivo dos NPCS)	Categoria

Tabela 4.2: Campos para cada um dos N jogadores e NPCs.

Como a quantidade de campos presentes na base de dados precisa ser fixa, foi utilizado um valor N que determina quantos jogadores e NPCs serão tratados dentro do campo de visão do jogador analisado. Portanto, para cada entrada presente na base de dados, existem no máximo N jogadores e N NPCs que estão sendo considerados dentro do campo de visão de um jogador. Caso existam menos jogadores ou NPCs visíveis que N, os valores destes são indefinidos. Caso existam mais que N jogadores ou NPCs no campo de visão, apenas os N mais próximos são tratados.

Para determinar se um avatar ou NPC está dentro do campo de visão de um jogador é necessário verificar o ângulo entre eles e qual é o ângulo que o jogador está observando. Os dados fornecidos pela base *WoWPosition* estão no sistema de posicionamento real utilizado pelo jogo, entretanto todos os NPCs têm o posicionamento apresentado pelo sistema *Thottbot*. Portanto, para poder ser feita a análise do campo de visão de um jogador, todos os avatares tiveram suas coordenadas de posição convertidas para o sistema *Thottbot*. Destaca-se que esta decisão foi tomada considerando que no sistema *Thottbot* não existe uma das coordenadas e portanto essa informação não existe na base. Como conseqüência não há como saber em qual altura um NPC está. Portanto, não havia uma maneira de ser feito o contrário. Outra informação relevante é que não temos qualquer informação sobre o posicionamento da câmera de cada jogador ou ainda sua direção de observação. O que temos é a informação sobre qual o ângulo de determinado avatar está rotacionado. Na figura 4.3 é possível observar 4 jogadores. O jogador 1 tem seu campo de visão analisado. Nesse exemplo, apenas o jogador 3 está dentro do campo de visão do jogador 1. Isso acontece, pois apesar do jogador 2 estar a uma distância menor que d, o

ângulo feito pelo vetor entre o jogador 1 e 2 está fora dos limites observáveis pelo jogador. Já o jogador 4 está muito longe do jogador 1 para poder ser visto.

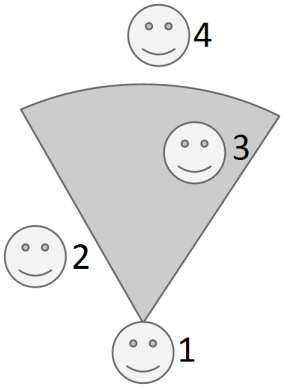


Figura 4.3: Exemplos de posicionamento de jogadores dentro do campo de visão de um jogador.

A figura 4.4 explicita as conseqüências das diferenças no cálculo para verificar se um objeto está presente ou não no campo de visão. Nessa figura existem três exemplos, descritos nos próximos parágrafos.

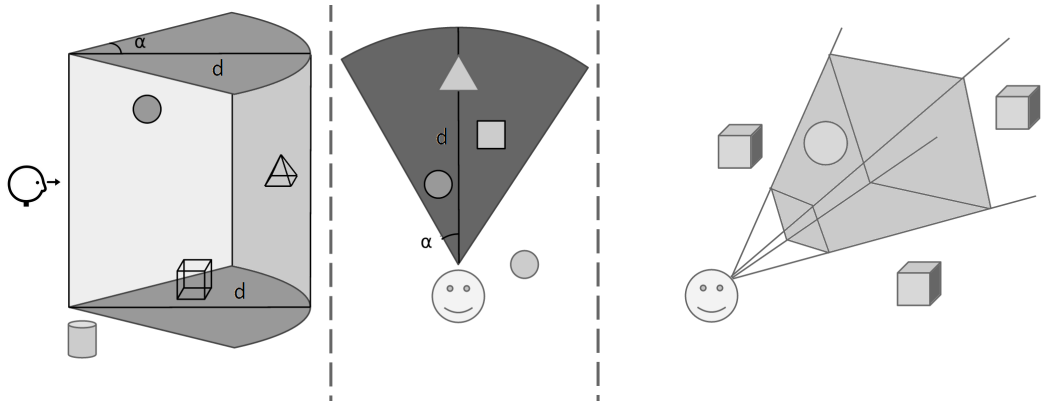


Figura 4.4: Exemplos de posicionamento de jogadores dentro do campo de visão de um jogador.

O primeiro exemplo mostra como um campo de visão poderia ser interpretado caso estivesse disponível a informação sobre o posicionamento real de todos os objetos. Neste exemplo, temos um observador, que consegue ver todos os objetos que estão a uma distancia menor que d e estão dentro de seu campo de visão que tem abertura de α. Esta implementação seria facilmente feita por um desenvolvedor de jogos, já que todas as

informações de posicionamento de todos os objetos estariam disponíveis para que a renderização fosse correta. Observa-se neste exemplo que é considerado que o jogador está observando todos os objetos na tela, menos o cilindro.

O segundo exemplo mostra como está sendo tratado o campo de visão considerando as informações disponíveis. Neste segundo exemplo, temos a mesma cena apresentada no exemplo anterior, entretanto o ambiente que está sendo apresentado é bidimensional. Neste trabalho está é a forma utilizada para calcular se um NPC ou jogador está dentro campo de visão. Destaca-se que nessa interpretação não é considerada a informação de altura dos objetos. E essa forma de interpretação foi escolhida devido à falta de informação sobre a altura dos NPCs no sistema *Thottbot*. Em [38] é apresentado que a área de interesse dos jogadores é representada por um círculo de raio de 100 unidades do jogo, essa medida está sendo utilizada para o valor de d. Foi utilizado o valor de 80^{o} para α por ser próximo ao valor do ângulo de abertura do campo de visão de seres humanos.

O terceiro e último exemplo mostra como é considerado se um objeto será renderizado ou não. Esta seria a melhor dentre todas as implementações apresentadas para ambientes tridimensionais, já que considera todos os objetos que de fato são renderizados para cada jogador. Entretanto, é necessário mais informações além das informações da posição de todos os objetos, além disso, o custo desta implementação pode ser muito alto. Para a renderização, apenas os objetos que estão dentro do *view frustrum* são renderizados, esse processo de seleção de quais objetos serão renderizados é chamado de *culling*. Um *view frustrum* normalmente é definido como uma pirâmide de visão truncada por dois planos paralelos, esses dois planos são paralelos entre si e perpendiculares à direção de observação.

4.3.2 Limpeza dos Dados

Depois que foi determinada a estrutura das bases, foi feito um estudo inicial com o objetivo de determinar quais seriam os campos que efetivamente poderiam ser utilizados para a construção dos modelos. Desta forma, foi feito um pré-processamento utilizando o software Weka em campos com dados com muito ruídos ou que precisavam ser interpreta-

dos em conjuntos com outros campos foram eliminados. Os campos eliminados foram: a posição do jogador no espaço bimensional dada pelos campos *ThottbotX* e *ThottbotY*, além das posição dos outros jogadores e NPCs dada por *X* e *Y*. Portanto, a base de dados ficou com os seguintes campos: *AnguloVisual*, *AnguloMovimentaçãoAnterior*, *AnguloMovimentacao* e *DeadReckoning* e dados dos jogadores e dos NPCS são compostos por *Distância*, *Angulo* e *Funcao*. As tabelas 4.3 e 4.4 apresentam os campos que continuaram presentes na nova base de dados.

Nome do Campo	Tipo de dado
AnguloVisual	Real
AnguloMovimentacaoAnterior	Real
Dados dos Jogadores	Múltiplos dados
Dados dos NPCs	Múltiplos dados
AnguloMovimentacao	Real
DeadReckoning	Booleano

Tabela 4.3: Campos presentes na nova base de dados após a remoção dos campos com problemas.

Campos dos Jogadores e NPCS	Tipo de dado
Distancia	Real
Ângulo	Real
Função (Exclusivo dos NPCS)	Categoria

Tabela 4.4: Campos para cada um dos N jogadores e NPCs a remoção dos campos com problemas.

Inicialmente foram criadas duas bases de dados com os dados de um período de tempo a 1 hora. A primeira base tratou da primeira hora e a segunda base tratou dos dados da 8ª hora da base de dados. Para a construção dessas bases foi utilizado o valor N igual a 7, N é a quantidade de NPCs e jogadores que estão sendo considerados presentes no campo de visão do jogador analisado.

Em seguida, essas bases foram pré-processadas de forma que somente os campos descritos nas tabelas 4.3 e 4.4 foram mantidos. As informações dessas bases foram sintetizadas na tabela 4.5. Essa tabela é composta por sete campos. O primeiro campo identifica a base e o segundo campo indica o total de entradas presente nesta base. O segundo e o terceiro campo indicam respectivamente o total de acertos e erros do algoritmo *Dead*

Reckoning nesta base. O quarto campo indica quantos avatares distintos estão sendo retratados nesta base de dados. Os últimos dois campos indicam o *Timestamp* inicial e final que foram utilizados para a construção desta base de dados. Observa-se que na última linha é apresentado o somatório das informações fornecidas. Assim é possível observar o total de entradas e a quantidade de avatares distintos, enquanto nos campos total de acertos e erros é apresentado tanto o somatório quanto a porcentagem da distribuição dos mesmos.

Identificação da base	Total de Entradas	Total de acertos	Total de erros	Avatares distintos	Timestamp Inicial	Timestamp Final
1	19870	12154(61,17%)	7716 (38,83%)	164	63445137060	63445140659
2	4885	2176 (44,54%)	2709 (55,46%)	43	63445165860	63445169459
Somatório	24755	14330(57,89%)	10425(42,11%)	200	-	-

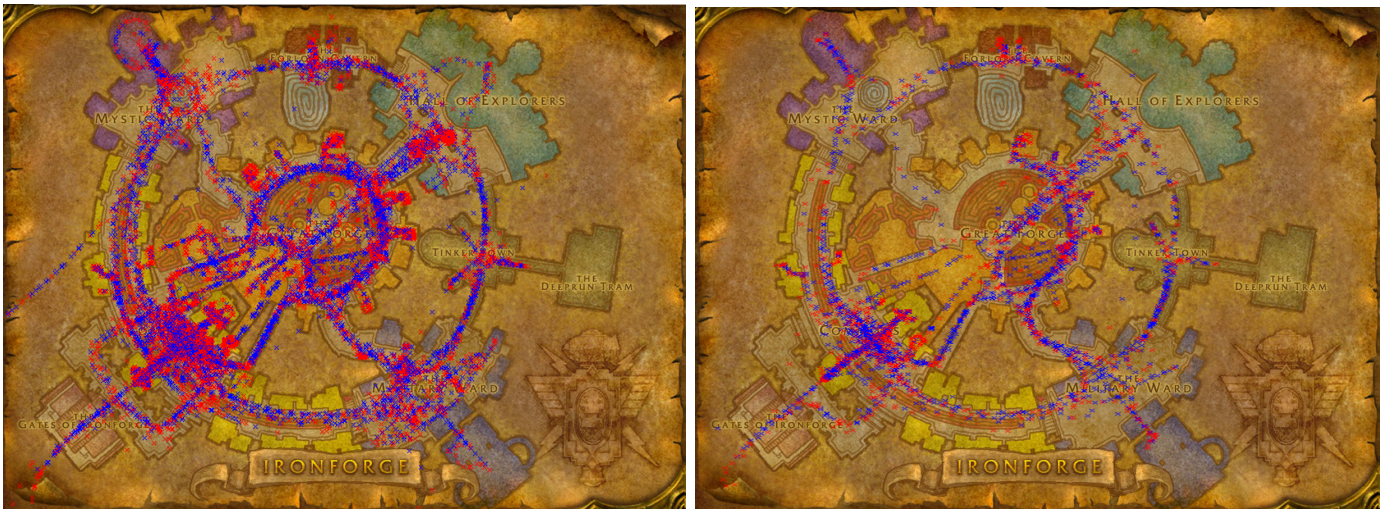
Tabela 4.5: Informações sobre as bases de dados geradas.

Através da tabela 4.5 é possível retirar algumas informações sobre o comportamento do *Dead Reckoning*. Nessas bases a taxa de acerto médio do algoritmo *Dead Reckoning* é de 57,89%, e , portanto, a taxa de erro é de 42,11%. É importante ressaltar que cada uma das entradas presentes na base de dados equivale ao registro da posição de um avatar dentro do jogo *World of Warcraft*. Como apresentado anteriormente, as situações em que o *Dead Reckoning* erra sua previsão, são as situações em que devem haver troca de mensagens comunicando a nova direção. Portanto, nessas bases de dados em 42.11% entre um segundo e outro houve ao menos uma troca de mensagens para comunicar a mudança da movimentação de jogadores.

Além do comportamento do algoritmo *Dead Reckoning*, é possível também observar que os jogadores tem um comportamento diferente dependendo do horário. Observa-se que na tabela 4.5 é significativa a diferença entre a quantidade de avatares distintos entre a primeira e a oitava hora apesar de ter sido utilizada a mesma quantidade de timestamps. Na primeira hora, que foi utilizada para construir a base de dados 1, existem 164 avatares distintos. Enquanto na oitava hora, que foi utilizada para a construção da base 2, existem apenas 43 avatares distintos. Como havia menos jogadores é esperado que existam menos dados de movimentação na base 2 do que na 1.

Nas figuras 4.5a e na 4.5b são apresentados os posicionamentos aproximados dos joga-

dores no mapa da cidade de *Ironforge* nas respectivas bases. Nessas duas figuras os pontos em vermelho marcam os lugares onde o algoritmo *Dead Reckoning* errou, enquanto os pontos azuis marcam onde o algoritmo tradicional acertou. Também nota-se que há muito mais jogadores na cidade durante a primeira hora se comparado com a oitava hora.



(a) Base 1 (b) Base 2

Figura 4.5: Posicionamento dos jogadores presentes na bases 1 e 2.

Nessas duas figuras é possível observar que existe uma grande concentração de jogadores na entrada da cidade de *IronForge*. Outro local em que os jogadores se concentram é a região central, próximos aos NPCs vendedores. Em ambas as bases, existem poucos jogadores que visitam o *Hall of Explorers*, essa região como foi apresentada anteriormente contém apenas NPCs que entregam missões aos jogadores. Isso também ocorre com a sala *The Deeprun Tram*, entretanto, esta sala não contém nenhum NPC. Na base 2 observa-se que uma parte da região central, onde os vendedores se encontram, em poucos jogadores foram. Acredita-se que por conta do comportamento diferente observado entre os jogadores presentes nas duas bases, havia poucos jogadores naquele horário que tinham interesse em negociar com aqueles NPCs, ou ainda aquela região era menos interessante naquele horário do que durante a primeira hora. Através dos dados disponíveis na base *WoWPosition* não foi possível concluir exatamente qual o motivo disso.

Observa-se também na primeira base que há uma rota que foi utilizada para saída ou entrada da cidade que teoricamente não é válida. Acredita-se que esta rota foi feita por algum funcionário que faz manutenção do jogo e estava utilizando um avatar *Game Master*. Esses avatares, chamados de *Game Master*, contrário dos jogadores pode ultrapassar paredes. Para não causar estranhamento aos outros jogadores quando este avatar atravessou as paredes, provavelmente este avatar estava invisível e não estava sendo renderizado nos clientes dos jogadores. Entretanto, como a base de dados *WoWPosition* foi construída utilizando *traces* ela registrou essa movimentação incomum. Essas informações sobre a rota incomum não foram removidas da base de dados 1, já que não é possível ter certeza sobre o que exatamente aconteceu para que estas posições tenham sido observadas.

4.4 Modelagem e Avaliação

Inicialmente foram feitos três testes utilizando as bases construídas durante a fase de preparação. Estes testes tinham como objetivo determinar se algoritmos de aprendizado de máquina poderiam ser utilizados efetivamente durante as duas fases do processo. Portanto, o primeiro teste tinha como objetivo verificar se os algoritmos seriam capazes de determinar em quais casos o algoritmo *Dead Reckoning* erra ou acerta. Enquanto o segundo teste tinha como objetivo determinar, nos casos em que o *Dead Reckoning* erra, qual será o novo ângulo de movimentação. Para ambos os testes foram escolhidos os algoritmos *Multilayer Perceptron*, *LWL*, o *Bagging* utilizando o *REPTree* e o próprio *REPTree*. Como eram testes iniciais, os algoritmos utilizados não tiveram seus parâmetros de entrada alterados. Esses algoritmos foram escolhidos por terem apresentado os melhores resultados nos testes preliminares feitos durante a construção do programa que gera as bases de dados. Os dois testes executados motivaram um terceiro teste, em que a segunda fase foi executada apenas utilizando os dados que foram classificados como Verdadeiros Positivos e Falsos Negativos na primeira fase. Em seguida, foram construídos histogramas para compreensão da distribuição de erro de cada um dos algoritmos de aprendizado.

Dessa maneira, esta Seção está organizada com o primeiro teste sendo apresentado em 4.4.1. O segundo teste é apresentado em seguida em 4.4.2, em seqüência o terceiro e último

teste é apresentado em 4.4.3. Por último, os histogramas construídos são apresentados na Subseção 4.4.4

4.4.1 Primeiro Teste

O primeiro teste tinha como objetivo verificar se é possível utilizar algoritmos de aprendizado de máquina para classificar quando o algoritmo *Dead Reckoning* não é suficiente para prever a movimentação de um jogador. Dessa maneira, para o primeiro teste, as bases de dados tiveram o campo *AnguloMovimentacao* retirado, já que essa informação é utilizada apenas para o treinamento na segunda fase. Por outro lado, a coluna *DeadReckoning*, responsável por indicar se o *Dead Reckoning* acerta ou erra a previsão para aquela entrada, foi mantida.

Os algoritmos do Weka, quando utilizados para a classificar, apresentam como saída a matriz de confusão daquela execução. Como descrito em [30], na matriz de confusão são apresentados respectivamente os valores de Falsos Positivos, Falsos Negativos, Verdadeiros Positivos, Verdadeiros Negativos. Neste primeiro teste, os casos de Verdadeiros Positivos representam a situação na qual o algoritmo de aprendizado classificou que o algoritmo do *Dead Reckoning* iria errar e, de fato, o algoritmo tradicional erra. Da mesma forma, os casos de Verdadeiros Negativos são os casos em que o algoritmo de aprendizado classificou corretamente quando o *Dead Reckoning* acerta. Os casos de Falsos Positivos são os casos em que tanto o *Dead Reckoning* quanto o algoritmo de aprendizado erram. Já os casos de Falsos Negativos são os casos em que o algoritmo de aprendizado indica que o algoritmo tradicional irá errar, porém ele acerta.

Na tabela 4.6 as informações do desempenho de cada um dos algoritmos são apresentadas. A tabela 4.6 é composta por quatro campos. No primeiro campo é indicado o algoritmo, no segundo campo qual base foi utilizada para o treinamento, no terceiro campo foi indicada qual base foi utilizada para o teste e no último campo é apresentada a matriz de confusão. Nessas matrizes de confusão apresentadas na tabela, as colunas indicam qual o valor da classificação, enquanto as linhas indicam qual era o valor real. O valor a indica quando o *Dead Reckoning* erra, enquanto b é quando o *Dead Reckoning*

acerta. Dessa maneira, a matriz de confusão do algoritmo *Multilayer Perceptron* pode ser interpretada a seguinte maneira. Houve 1669 casos Verdadeiros Positivos e 507 Falsos Positivos. Enquanto, houve 498 casos de Falsos Negativos e 2211 casos Verdadeiros Positivos. Totalizando assim, 2176 casos em que o *Dead Reckoning* erra e 2709 casos em que ele acerta a previsão, como foi apresentado na tabela 4.5.

Algoritmo	Treino	Teste	Matriz de Confusão
MultilayerPerceptron	1	2	a b 1669 507 — a 498 2211 — b
Bagging com REPTree	1	2	a b 1723 453 — a 470 2239 — b
LWL	1	2	a b 1390 786 — a 882 1827 — b
REPTRee	1	2	a b 1719 457 — a 518 2191 — b

Tabela 4.6: Resultados para aprendizagem do campo DeadReckoning.

A partir das informações apresentadas na tabela 4.6 foi construída uma nova tabela com informações mais detalhadas sobre o desempenho dos algoritmos neste teste. Essa tabela é composta por oitos campos. O primeiro campo indica o algoritmo, no segundo campo indica a base que foi utilizada para treinamento, enquanto no terceiro campo indica a base de teste. No quarto, quinto, sexto e sétimo campos são apresentados respectivamente as porcentagens de Verdadeiros Positivos, Falsos Positivos, Verdadeiros Negativos e Falsos Negativos. No oitavo campo é apresentada a taxa de acerto de cada um dos algoritmos. Observa-se que na última linha é apresentada a média dos valores dos Verdadeiros Positivos, Falsos Positivos, Verdadeiros Negativos, Falsos Negativos e da taxa de acerto.

Algoritmo	Treinamento	Teste	TP	FP	TN	FN	Taxa de acerto
MultilayerPerceptron	1	2	81,35%	22,98%	77,02%	18,65%	79,42%
Bagging com REPTree	1	2	83,17%	21,43%	78,57%	16,83%	81,10%
LWL	1	2	69,92%	39,82%	61,18%	30,08%	65,85%
REPTree	1	2	82,74%	23,16%	76,84%	17,26%	80,04%
Média	-	-	79,29%	26,59%	73,40%	20,70%	76,60%

Tabela 4.7: Resultados interpretados para aprendizagem do campo DeadReckoning.

Na tabela 4.7 é possível observar que, em média os algoritmos obtiveram uma taxa de acerto de 79,29%. Além disso, é possível observar que o algoritmo *Bagging* obteve o melhor resultado ao obter uma taxa de acerto de 83,17%. Ressalta-se que os algoritmos, em média, classificam 26,59% das entradas em que o *Dead Reckoning* acertaria de forma errada, ou seja, 26,59% das entradas são Falsos Positivos. Dessa forma, 26,59% dos casos em que o *Dead Reckoning* acertaria na predição são enviados para a segunda fase da estratégia proposta de forma incorreta. Também observa-se que em 20,70% das vezes em que o *Dead Reckoning* erra a predição, os algoritmos de aprendizado apontam que *Dead Reckoning* irá acertar. Assim, para todas essas entradas a predição será errada tanto utilizando o *Dead Reckoning* quanto na nova estratégia.

Para auxiliar o entendimento dos resultados no Weka, é possível selecionar uma opção para mostrar junto ao relatório todas as classificações feitas pelo algoritmo. A partir deste relatório foi construído um programa que selecionava quais entradas eram classificadas como Falsas Negativas, Falsas Positivas, Verdadeiras Positivas e Verdadeiras Negativas. Dessa forma, as entradas foram agrupadas e novas bases foram geradas para cada uma das situações de classificação. As entradas Falsas Negativas de cada um dos algoritmos foram utilizadas para gerar as imagens apresentadas na figura 4.6. Ressalta-se que nessas imagens foram utilizados filtros para a ampliação do tamanho dos pontos para facilitar a visualização. Nestas imagens, todos os pontos vermelhos são pontos que foram classificados como se o algoritmo tradicional errasse, porém na realidade ele acerta.

Nessas imagens apresentadas na figura 4.6 é possível observar que os algoritmos de aprendizado de máquina erram mais quando o avatar está próximo da entrada da cidade e a região central onde há uma concentração maior de vendedores. Ambas as situações são facilmente compreensíveis. A primeira situação ocorre quando um avatar está na entrada da cidade não há como saber ainda quais serão os interesses do jogador. A segunda situação pode ser explicada da seguinte maneira cada um dos NPCs daquela região é interessante apenas para determinados jogadores, como foi descrito anteriormente na Seção 4.1.2. Por exemplo, um jogador que tenha a classe ferreiro não irá se interessar por itens de culinária. Entretanto, o nosso modelo não tem essas informações, pois não

(a) MultilayerPerceptron (b) Bagging com REPTree

(c) LWL (d) REPTree

Figura 4.6: Casos em que o algoritmo do *Dead Reckoning* acerta, porém o respectivo algoritmo acusa que ele irá errar, ou seja, as saídas Falsas Negativas.

estão disponíveis nos *traces* utilizados para a construção da base de dados.

A conclusão obtida neste teste é que com a média da taxa de acerto entre os algoritmos sendo 76.60% verificou-se que é viável utilizar aprendizado de máquina para a primeira fase do modelo. Considerando-se também que como não houve um refinamento entre as variáveis de entrada de cada um destes algoritmos, existe a possibilidade de melhorar ainda mais os resultados obtidos. Apesar disso, foi observado que a taxa de Falsos Negativos média foi de 24.41%, que pode ser considerada elevada, dando margem a trabalhos futuros na área.

4.4.2 Segundo Teste

O segundo teste tinha como objetivo determinar se é possível utilizar algoritmos de aprendizado de máquina para a segunda fase da estratégia proposta. Ou seja, se é possível prever qual será o novo ângulo de movimentação de um jogador quando foi previsto durante a primeira fase que o algoritmo *Dead Reckoning* erra. Para isso, em ambas as bases 1 e 2 foi removida a coluna *DeadReckoning* e mantida a coluna *AnguloMovimentacao*, que é a coluna com o valor que se deseja prever. Além da remoção destas colunas, também foram removidas todas as entradas em que o *Dead Reckoning* faz a previsão correta. As informações sobre as bases podem ser observadas na tabela 4.8.

A tabela 4.8 é composta por três campos. No primeiro campo é indicada a base de dados, enquanto no segundo campo é indicada a quantidade de entradas presentes na base dados. No terceiro campo é indicado quantos avatares distintos estão presentes dentre todas as entradas. Na última linha é apresentado o somatório do total de entradas e de avatares distintos presentes nas duas bases.

Identificação da base	Total de Entradas	Avatares distintos
1	7716	154
2	2176	37
Somatório	7932	184

Tabela 4.8: Informações sobre as bases de dados geradas.

Observa-se que estas bases de dados contém apenas as entradas em que o *Dead Reckoning* erra a previsão. Portanto, nas bases originais existem 200 avatares distintos, enquanto as bases utilizadas no segundo teste têm apenas 184 avatares distintos. Assim é possível concluir que para 16 avatares o *Dead Reckoning* conseguiu prever de maneira correta toda a rota destes avatares durante o período observado por essas bases. Destes 16 avatares, 10 avatares pertenciam à base 1, enquanto 6 avatares pertenciam à base 2.

Os resultados obtidos pelos algoritmos para a predição do novo ângulo de movimento podem ser observados na tabela 4.9. Essa tabela é composta por sete campos distintos. O primeiro campo apresenta o algoritmo utilizado, enquanto o segundo e o terceiro apresentam, respectivamente, qual base foi utilizada para treinamento e a base de teste. O quarto

campo indica qual o erro absoluto do algoritmo neste teste. Enquanto o quinto campo indica o coeficiente de correlação. A quantidade de entradas que foram classificadas corretamente é apresentada no sexto campo. No sétimo campo é apresentada a quantidade de entradas que foram classificadas erroneamente pelos algoritmos. Na última linha desta tabela é apresenta a porcentagem média dos valores do erro absoluto, do coeficiente de correlação, das saídas corretas e saídas erradas.

O Weka não apresenta a taxa de acerto dos algoritmos de predição, portanto, foi construído um programa com o objetivo de extrair essa informação. Este programa recebe como entrada a saída do Weka que apresenta o ângulo real e o ângulo previsto para todas as entradas. Este programa basicamente verifica para cada saída fornecida pelo Weka se tem um erro menor que 15º, portanto menor que 5%, e classifica essa saída como correta. Caso contrário a saída é classificada como errada. A seguir, este programa gera uma nova base que pode ser colocada novamente dentro do Weka com as entradas corretas. O resultado obtido por esse processo pode ser observado na tabela 4.9 nas coluna Saídas Corretas e Saídas Erradas.

Algoritmo	Treinamento	Teste	Erro Absoluto	Coef. de Cor.	Saídas Corretas	Saídas Erradas
MultilayerPerceptron	1	2	33,44 %	0,81	1067(49,04%)	1109(50,96%)
Bagging com REPTree	1	2	15,35 %	0,92	1823(83,78%)	353(16,22%)
LWL	1	2	52,50 %	0,79	230 (10,56%)	1946(89,44%)
REPTRee	1	2	17,20 %	0,91	1799(82,67%)	377(17,33%)
Média	-	-	29,62 %	0,85	56,50%	43,48%

Tabela 4.9: Resultados para aprendizagem do campo *AnguloMovimento*.

Como apresentado na tabela 4.9, foram utilizados os mesmos algoritmos do teste anterior. Destaca-se que esses algoritmos foram utilizados neste teste com o objetivo de predição e não de classificação. Nessa tabela também é possível observar que os algoritmos erraram 43,48% de todas as entradas que deveriam ir para segunda fase. Em outras palavras, os algoritmos de aprendizado erraram na predição de qual será o novo ângulo de movimentação em 43,48% dos casos. Como os algoritmos estão sendo utilizados para fazer uma predição, é possível observar que a média do erro absoluto foi de 29,62%. Novamente, como foi no teste anterior, o algoritmo de *Bagging* obteve o melhor desempenho, tendo um erro médio absoluto de 15,35% e ele conseguiu prever corretamente para 83,78% das

entradas o novo ângulo de movimento. Estes resultados demonstram a viabilidade do aprendizado do novo ângulo de movimentação dos jogadores no jogo *World of Warcraft*.

4.4.3 Terceiro Teste

Como foi observada uma alta taxa de Falsos Negativos no primeiro teste, foi proposto um terceiro teste. A hipótese é a de que mesmo que a primeira fase erre ao classificar instâncias que seriam previstas de forma correta pelo algoritmo *Dead Reckoning*, se a previsão na segunda fase estiver correta, a saída estará correta. Seguindo esse raciocínio, este teste foi executado utilizando bases de dados geradas especificamente para cada um dos algoritmos. Essas bases de dados foram construídas utilizando todas as entradas em que cada um dos algoritmos classificou como entradas que deveriam ir para a segunda fase. Ou seja, para cada um dos algoritmos foi construída uma base utilizando as entradas Falsas Negativas quanto Verdadeiras Positivas indicadas pelos próprios modelos da primeira fase. Exemplificando, o algoritmo *Multilayer Perceptron* utilizado neste teste, utiliza como base de teste todas as entradas que foram classificadas que deveriam ir para a segunda fase pelo modelo que foi gerado pelo *Multilayer Perceptron* da primeira fase. Ressalta-se também que neste terceiro teste os parâmetros dos algoritmos não foram alterados, assim como no segundo teste, de forma que a comparação entre o desempenho dos algoritmos pode ser feita de maneira direta. Para todos os algoritmos foi utilizada a mesma base de treinamento, a base 1.

Os resultados obtidos estão apresentados na tabela 4.10 que é composta pelos mesmos campos da tabela 4.9, apresentada durante o segundo teste. Além destes campos foi adicionado o campo Total de Entradas, que indica quantas entradas foram fornecidas para cada um dos algoritmos.

Algoritmo	Total de Entradas	Erro Absoluto	Coef. de Cor.	Saídas Corretas	Saídas Erradas
MultilayerPerceptron	2167	36,75 %	0,76	1021 (47,12%)	1146 (52,88%)
Bagging com REPTree	2193	21,86 %	0,87	1609 (73,37%)	584 (26,63%)
LWL	2272	55,79 %	0,67	312 (13,73%)	1960 (86,26%)
REPTRee	2237	25,12 %	0,85	1563 (69,87%)	674 (30,13%)
Média	2717.25	34,88 %	0,78	51,02%	48,97%

Tabela 4.10: Resultados obtidos no terceiro teste.

Algoritmo	1a Fase		2a Fase	
	Saídas Corretas	Saídas Erradas	Saídas Corretas	Saídas Erradas
MultilayerPerceptron	79,42%	20,58 %	47,12%	52,88%
Bagging com REPTree	81,10%	18,90 %	73,37%	26,63%
LWL	65,85%	34,15 %	13,73%	86,26%
REPTRee	80,04%	19,96 %	69,87%	30,13%
Média	76,60%	23,40 %	51,02%	48,97%

Tabela 4.11: Informações compiladas dos três testes.

Observa-se que nesta tabela que os algoritmos erram em média em 48,97% de todas as entradas que foram para segunda fase. No teste anterior, foi observado que eles erravam em 43,48% dos casos. Portanto, houve uma variação de 5,49% na quantidade de saídas erroneamente classificadas. Observa-se também que o erro absoluto médio obtido durante o terceiro teste é de 34,88%, enquanto no segundo teste foi de 29,62%. Assim a variação foi de 5,26% entre o segundo e o terceiro teste.

Com os dados colhidos durante o terceiro e o primeiro teste foi possível construir a tabela 4.11. Nessa tabela é possível observar o desempenho dos algoritmos nas duas fases. Essa tabela é composta por cinco campos. O primeiro indica o algoritmo sendo tratado. O segundo e o terceiro campo são responsáveis por apresentar a porcentagem de como as entradas foram classificadas na primeira fase. Já o quarto e o quinto campo, apresentam a porcentagem a quantidade de saídas que foram corretamente ou erroneamente previstas na segunda fase.

É possível observar da tabela 4.11 que de todas as entradas fornecidas para o algoritmo de aprendizagem Bagging, 81,10% eram classificadas na primeira fase de forma correta. Portanto, na primeira fase ele errava a classificação sobre o acerto do *Dead Reckoning* em 18.90% das entradas. Para todas as entradas que o modelo construído pelo Bagging na primeira fase classificava que deveriam ir para segunda fase, o modelo construído pelo Bagging na segunda fase acertava em 73.37% , e , portanto, errava em 26.63% dos casos. É apresentado na tabela o algoritmo de Bagging obteve o melhor resultado dentre todos os algoritmos. Observa-se que o algoritmo LWL obteve um resultado muito inferior se comparado aos outros algoritmos na segunda fase, obtendo apenas 13.73% de acerto. Outras informações colhidas durante esses três testes foram compiladas na tabela 4.12.

Algoritmo	1a Fase		2a Fase	
	Caso 1	Caso 2	Caso 3	Caso 4
MultilayerPerceptron	2211 (45,26%)	507 (10,37%)	1021 (20,90%)	1146 (23,45%)
Bagging com REPTree	2239 (45,83%)	453 (9,27%)	1609 (32,93%)	584 (11,95%)
LWL	1827 (37,40%)	786 (16,09%)	312 (6,38%)	1960 (40,12%)
REPTRee	2191 (44,85%)	457 (9,35%)	1563 (31,95%)	674 (13,79%)
Média	43,33%	11,27%	23,04%	22,32%

Tabela 4.12: Contagem dos resultados obtidos para as entradas presentes na base 2.

A tabela 4.12 é composta por cinco colunas. A primeira coluna indica o algoritmo, enquanto as outras colunas apresentam a contagem dos resultados obtidos para a predição final de todas as entradas presentes na base 2. Como na estratégia proposta existem duas fases bem defenidas, foi construído o fluxograma mostrado na figura 4.7. Este fluxograma apresenta as quatro possibilidades da predição de movimento desta nova proposta. Nesse fluxograma temos uma nova entrada que é fornecida para a estratégia proposta. Na primeira fase o modelo verifica se o algoritmo *Dead Reckoning* irá acertar ou não para a entrada, esse teste gera três possibilidades, sendo que duas dessas são a predição final da movimentação do jogador. O primeiro caso considera que o modelo indica que o *Dead Reckoning* irá acertar, e de fato ele acerta, estes casos estão contabilizados como Caso 1. Já o segundo caso considera que o *Dead Reckoning* irá acertar, porém na verdade ele erra, estes casos estão contabilizados como Caso 2. Por último o modelo pode prever que o *Dead Reckoning* irá errar, independente deste resultado estar correto, essas entradas são enviadas para a segunda fase da proposta. Durante a segunda fase existem duas possibilidades: o modelo prevê a nova movimentação da forma correta ou não. Os casos em que o modelo prevê da forma correta o novo ângulo são apresentados na como Caso 3, enquanto, os casos em que o modelo prevê incorretamente foram contabilizados como Caso 4.

Desta forma, é possível observar na tabela 4.12, que o algoritmo *Bagging* com o algoritmo REPTree obteve o melhor resultado se comparado a todos os outros algoritmos. De todas as entradas presentes na base 2, o algoritmo *Bagging* previu corretamente, na primeira fase, que 45.83% das entradas o algoritmo *Dead Reckoning* seria suficiente. Além disso, de todas as entradas na base 2, o algoritmo *Bagging* previu corretamente o novo

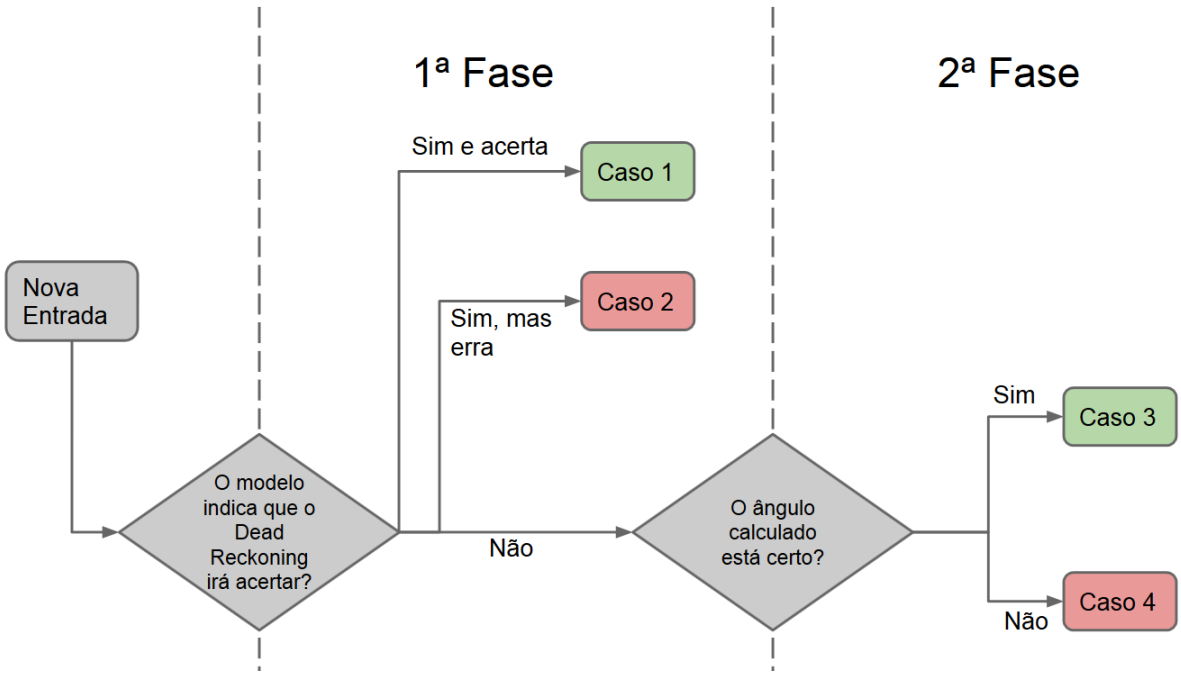


Figura 4.7: Fluxograma das possibilidades do resultado da nova abordagem.

ângulo de movimentação para 32,93% de todas as entradas fornecidas. Desta maneira, somando a porcentagem de casos 1 e 3 temos uma taxa de acerto de 78,76% de todas as entradas presentes na base 2. Destaca-se que nesta base, como foi apresentado anteriormente na tabela 4.5, o algoritmo *Dead Reckoning* obteve uma taxa de acerto de 44,54%.

O segundo algoritmo com melhor desempenho foi o REPTree, que obteve uma taxa de acerto final, na base 2, de 76,78%. Assim a diferença entre a taxa de acerto do *Bagging* com o REPTree sendo utilizado internamente em comparação com o REPTree é de apenas 1,96%. O algoritmo *Multilayer Perceptron* tem uma taxa de acerto final de 66,16%, enquanto o LWL obteve 43,78%, tendo, portanto, um resultado inferior ao resultado obtido pelo *Dead Reckoning* na mesma base.

4.4.4 Histograma

Um histograma, também conhecido como distribuição de freqüências, é uma representação gráfica da distribuição de um valor numérico e foi construído para cada um dos algoritmos de aprendizado utilizados, com o objetivo de compreender o comportamento da distri-

buição do erro. Para a construção desses histogramas foi utilizado o erro de cada uma das entradas fornecidas para os algoritmos durante o terceiro teste. Nos histogramas o eixo horizontal representa a classe, nesse caso o valor do erro, enquanto o eixo vertical representa a freqüência com que um valor se encaixou nessa classe.

Na figura 4.8 é apresentado o histograma gerado pelo erro apresentado pelo algoritmo *Multilayer Perceptron*. Na figura 4.9 é possível observar o histograma gerado pelo erro do algoritmo *Bagging* utilizando o algoritmo *REPTree*. Tanto este histograma como o histograma do *Multilayer Perceptron* apresenta apenas um valor modal. Observa-se também que os valores do erro desses dois histogramas se concentra entre -50 e 50 graus.

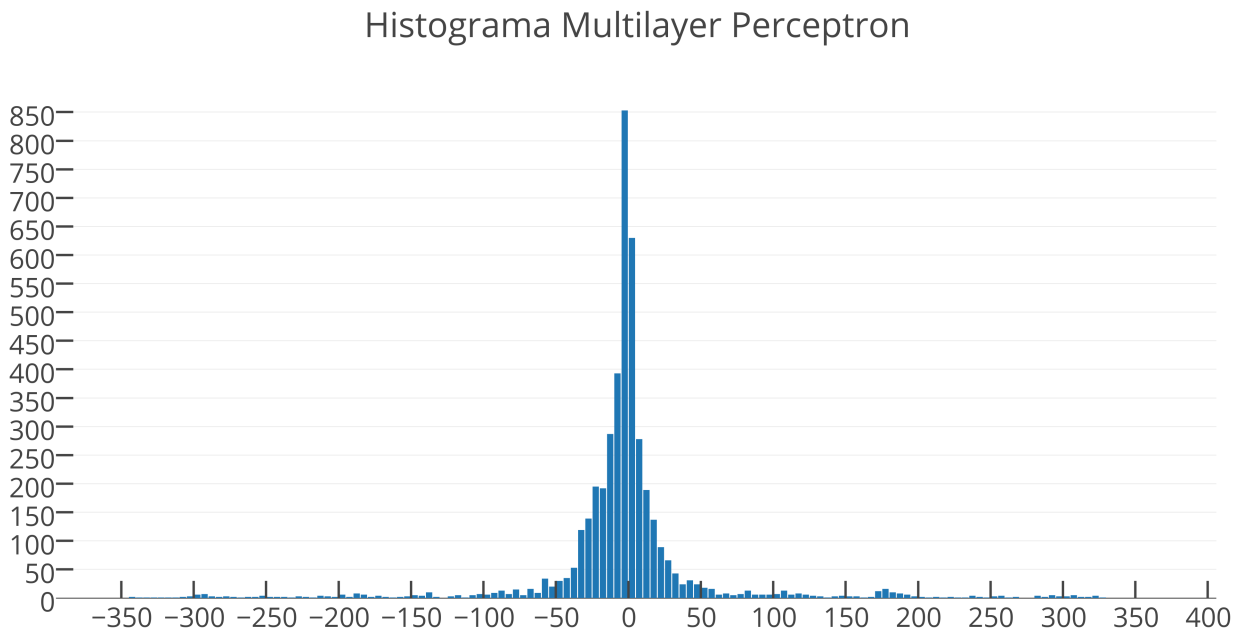


Figura 4.8: Histograma construído com os valores dos erros apresentados pelo *Multilayer Perceptron*.

Na figura 4.10 observa-se o histograma para o LWL. Esse é o histograma é significativamente diferentes dos demais. Isso se deve ao fato de que os valores do erro apresentados pelo algoritmo LWL não são próximos ao 0.

Por último, na figura 4.11 é possível observar o histograma gerado pelo erro apresentado pelo algoritmo *REPTree*. Observa-se nesse histograma que a maior parte do valor do erro se concentra entre -50 e 50 graus. É importante ressaltar que o formato do histo-

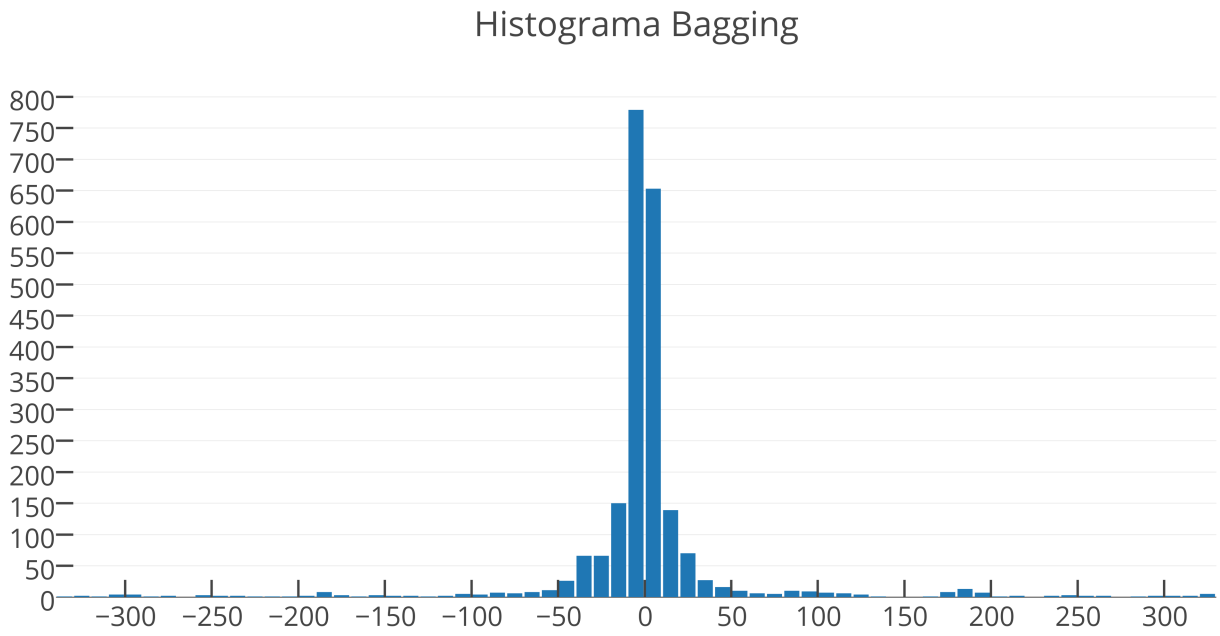


Figura 4.9: Histograma construído com os valores dos erros apresentados pelo *Bagging*.

grama apresentado é o mesmo formato do histograma do *Bagging*. Isso se deve ao fato de que o Bagging utiliza internamente o *REPTree*, utilizando cópias levemente modificadas da árvore de decisão construída pelo REPTree.

Observa-se que todos os histogramas, exceto o do LWL, apresentam os valores do erro concentrados ao centro e apresentam esse como valor moda. Já o histograma do LWL apresenta dois valores como moda: o -45 e o 55.

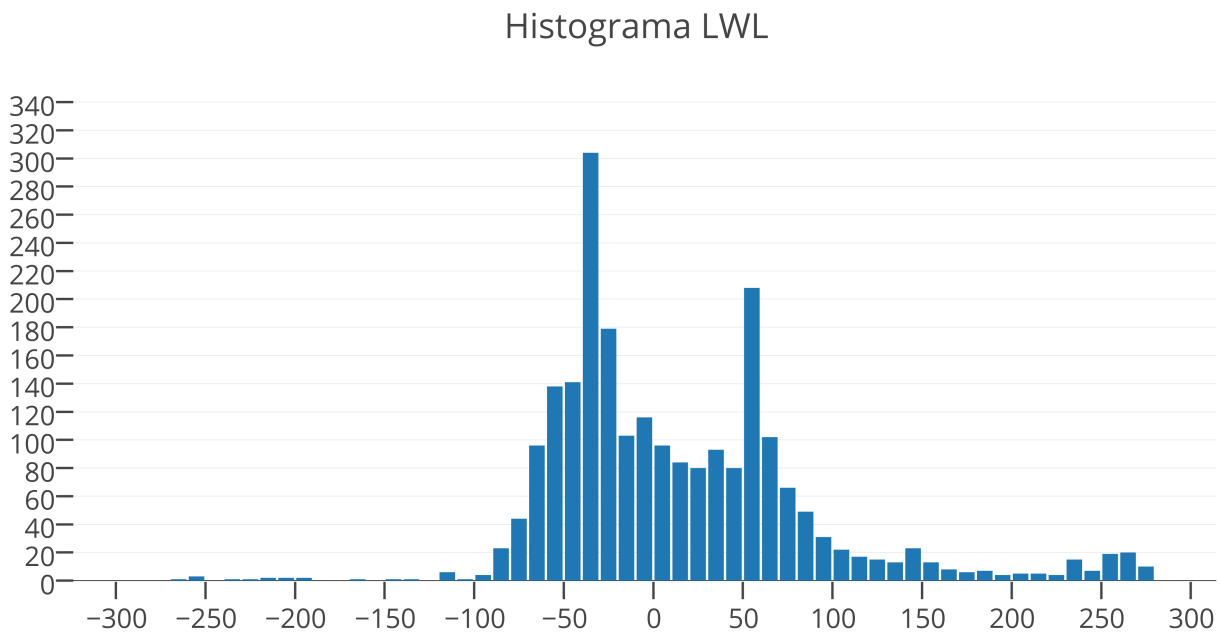


Figura 4.10: Histograma construído com os valores dos erros apresentados pelo *LWL*.

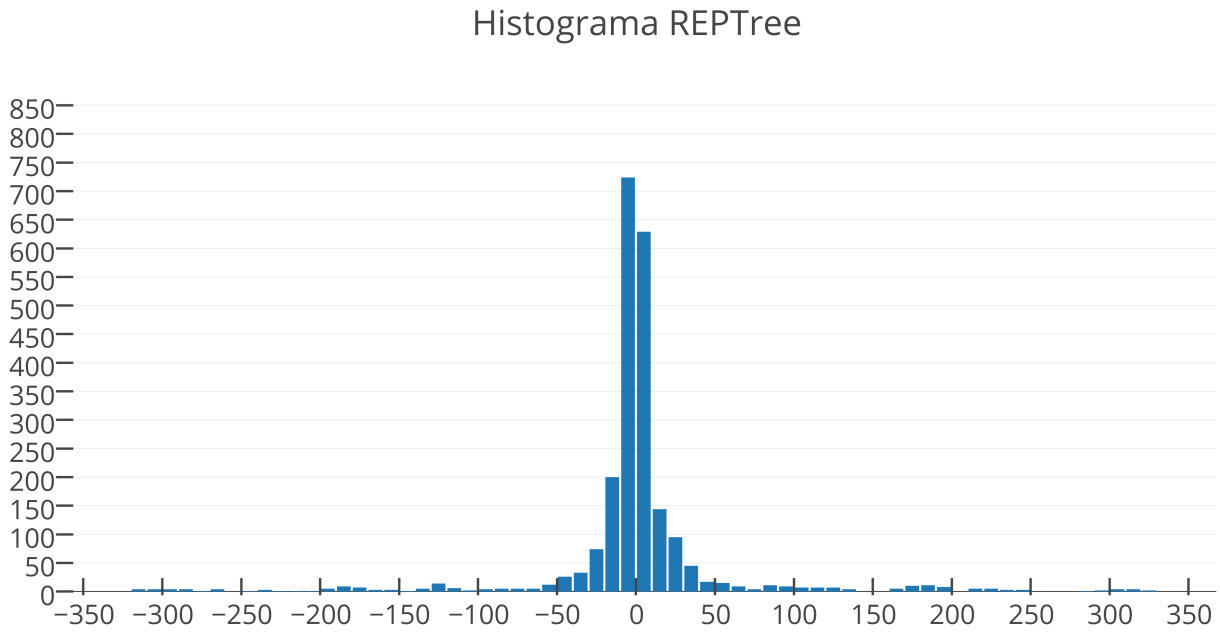


Figura 4.11: Histograma dos erros do *REPTree*

CAPÍTULO 5

CONCLUSÃO

Neste trabalho foi apresentada uma proposta de predição da movimentação de avatares em jogos distribuídos *multiplayer* baseada em aprendizado de máquina. A estratégia proposta é composta por duas fases bem distintas. Na primeira fase, um modelo de aprendizado classifica se a estratégia tradicional de predição, o *Dead Reckoning*, acerta ou erra na predição do movimento. Concretamente, o *Dead Reckoning* erra quando o movimento não ocorre em linha reta; mudando o ângulo de movimentação. Caso o *Dead Reckoning* acerte, ele deve ser utilizado para fazer a previsão. Caso contrário, ou seja, quando o modelo indica que o *Dead Reckoning* irá errar, é executada a segunda fase da estratégia. Na segunda fase, os algoritmos de aprendizado de máquina são utilizados para gerar um modelo que faz a predição do novo ângulo de movimento do avatar. Para a construção dos modelos de aprendizado foram utilizados os seguintes algoritmos de aprendizado de máquina: *Bagging*, LWL, *Multilayer Perceptron* e o REPTree.

Para a validação da estratégia proposta foi construída uma base de dados, baseada em traces do jogo *World of Warcraft*, a partir das bases disponíveis publicamente *WoWPosition* e *WoWHead*. A nova base de dados construída contém informações sobre jogadores e NPCs que estão presentes no campo de visão de um determinado jogador. Além destas informações, também estão disponíveis dados sobre o posicionamento deste jogador.

Observou-se, que a estratégia proposta obtém uma taxa de acerto médio de 76,60% durante a primeira fase. Na segunda fase obtém uma taxa de acerto de 51,02%. O algoritmo de aprendizagem de máquina que obtém o melhor desempenho é o Bagging. Esse algoritmo obtém uma taxa de acerto durante a primeira e a segunda fase respectivamente de 81,10% e 73,37%. O algoritmo Bagging estava utilizando o algoritmo REPTree, que apresentou o segundo melhor resultado dentre os quatro algoritmos utilizados. Estes resultados mostram que a utilização de aprendizado de máquina para a predição da

movimentação de avatares jogos distribuídos é uma alternativa viável.

Em comparação com uma outra abordagem relacionada da literatura, o AntReckoning, a taxa de sucesso é semelhante, cerca de 30% de melhoria sobre o *Dead Reckoning*. Entretanto, a grande vantagem da estratégia proposta é que não necessita de um especialista para configurar os múltiplos parâmetros utilizados por aquele modelo. Utilizando aprendizado de máquina, os parâmetros são "automaticamente aprendidos".

Existem várias possibilidades de trabalhos futuros que podem ser desenvolvidos a partir do trabalho realizado. Uma das possibilidades é extrair novos dados de *traces* de lugares diferentes do jogo *World of Warcraft* para a construção de novas bases mais ricas. Na verdade, existem vários caminhos para a construção de bases com informações diversas também coletadas de *traces* mais detalhados. Para a construção da base utilizada neste trabalho, usamos os dados disponibilizados na base *WoWPosition* que apresenta apenas os dados de posicionamento dos jogadores. Uma base com mais informações certamente refletiria na qualidade da previsão dos modelos de aprendizado construídos. Outra melhoria que seria possível de ser feita na base atual é a utilização das posições reais dos NPCs e não as posições convertidas para o sistema *Thottbot*. Além de trazer as informações da posição dos NPCs utilizando um sistema de tridimensional, possivelmente esta alteração aumentaria em muito a precisão dos modelos.

Existem vários gêneros de jogos, portanto o caminho natural é testar a nova abordagem de previsão de movimento em jogos de outros gêneros. A grande dificuldade é a obtenção de dados de traces públicos, relevantes e completos o suficiente para permitir o aprendizado.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] David Aha. *Lazy learning*. Springer Science & Business Media, 2013.

[2] Alexa. How popular is thottbot.com? Disponível em :http://www.alexa.com/siteinfo/thottbot.com . Acesso em Jul/2015.

[3] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D Spyropoulos, e Panagiotis Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

[4] Anderson V Araújo, Olga RP Bellon, Luciano Silva, Everton V Vieira, e Mônica Cat. Aplicando mineração de imagens para auxiliar na determinação da idade gestacional em recém-nascidos. *Anais do X Congresso Brasileiro de Informática na Saúde, Florianópolis, SC-Brasil*, páginas 986–991, 2005.

[5] Entertainment Software Association. Essential facts about the computer and video game industry. Disponível em: http://www.theesa.com/facts/pdfs/esa_ef_2013.pdf. Acesso em Jul/2014.

[6] Entertainment Software Association. Essential facts about the computer and video game industry 2014. Disponível em: http://www.theesa.com/facts/pdfs/esa_ef_2014.pdf. Acesso em Set/2014.

[7] Christopher G Atkeson, Andrew W Moore, e Stefan Schaal. Locally weighted learning for control. *Lazy learning*, páginas 75–113. Springer, 1997.

[8] Al-Amin Bhuiyan e Chang Hong Liu. On face recognition using gabor filters. *World academy of science, engineering and technology*, 28:51–56, 2007.

[9] Jean Sébastien Boulanger, Jörg Kienzle, e Clark Verbrugge. Comparing interest management algorithms for massively multiplayer games. *NetGames '06 Proceedings*

of 5th ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2006.

[10] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[11] Cássio Oliveira Camilo. Mineração de dados: Conceitos, tarefas, métodos e ferramentas. 2009.

[12] Vladimir Cherkassky e Filip M Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

[13] Mark Claypool e Kajal Claypool. Latency Can Kill: Precision and Deadline in Online games. *Proceeding MMSys '10 Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, 2010.

[14] Peter Englert. Locally weighted learning. 2012.

[15] Blizzard Entertainment. Dados sobre World of Warcraft. Disponível em: http://us.battle.net/wow/pt/blog/12346804/world-of-warcraft-azeroth-by-the-numbers-1-28-2014. Acesso em Jul/2014.

[16] Blizzard Entertainment. Map coordinates. Disponível em: http://www.wowwiki.com/Mapcoordinates . Acesso em Jul/2015.

[17] John Feil e Marc Scattergood. *Beginning Game Design: Level Design*, volume 1, capítulo Chapter 1 - The Basics of Game Design, páginas 6–14. Thomson Course Technology, Boston, MA - USA, 2005.

[18] Riot Games. Site da Riot Games. Disponível em: http://www.riotgames.com. Acesso em Jul/2014.

[19] Yong Guo e A. Iosup. The game trace archive. *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, páginas 1–6, Nov de 2012.

[20] WoW Head. Thottbot merged with wowhead framework. Disponível em: http://www.wowhead.com/news=175371/thottbot-merged-with-wowhead-framework . Acesso em Jul/2015.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, e Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Yeng-Ting Lee, Kuan-Ta Chen, Yun-Maw Cheng, e Chin-Laung Lei. World of warcraft avatar history dataset. *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, páginas 123–128, New York, NY, USA, 2011. ACM.

[23] G MeeraGandhi, Kumaravel Appavoo, e SK Srivasta. Effective network intrusion detection using classifiers decision trees and decision rules. *Int. J. Advanced network and application, Vol2*, 2010.

[24] Tom Michael Mitchell. *The discipline of machine learning*, volume 17. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[25] Ciamac Moallemi. Classifying cells for cancer diagnosis using neural networks. *IEEE Expert*, 6(6):8–12, 1991.

[26] Bob Nystrom. "game programming patterns". Disponível em:http://gameprogrammingpatterns.com/. Acesso em Jul/2014.

[27] Standards Committee on Interactive Simulation. IEEE Standard for Distributed Interactive Simulation - Application Protocols . *Proceedings of IEEE Standard 1278.1-1995*, New York, NY, USA, 2012.

[28] David Opitz e Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, páginas 169–198, 1999.

[29] Michael J Pazzani e Daniel Billsus. Content-based recommendation systems. *The adaptive web*, páginas 325–341. Springer, 2007.

[30] Foster Provost, Tom Fawcett, e Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. *In Proceedings of the Fifteenth International Conference on Machine Learning*, páginas 445–453. Morgan Kaufmann, 1997.

[31] J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

[32] Guinness World Records. CONFIRMED: GRAND THEFT AUTO 5 BREAKS 6 SALES WORLD RECORDS. Disponível em: http://www.guinnessworldrecords.com/news/2013/10/confirmed-grand-theft-auto-breaks-six-sales-world-records-51900/. Acesso em Set/2014.

[33] Brian D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 1996.

[34] Stuart Russell e Peter Norvig. Artificial intelligence: a modern approach. 1995.

[35] Cheryl Savert e Nicholas T. C. Graham. Timelines: symplifying the programming of lag compensation for the next generation of networked games. *Multimedia Systems*, páginas 271–287, 2013.

[36] Stefan Schaal, Christopher G Atkeson, e Sethu Vijayakumar. Real-time robot learning with locally weighted statistical learning. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, páginas 288–293. IEEE, 2000.

[37] Paul Schuytema. *Design de Games: Uma abordagem Prática*. Cengage, São Paulo, São Paulo, Brasil, 1 edition, 2009.

[38] Siqi Shen, Niels Brouwers, Alexandru Iosup, e Dick Epema. Characterization of human mobility in networked virtual environments. *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, páginas 13. ACM, 2014.

[39] Salvatore Stolfo, David W Fan, Wenke Lee, Andreas Prodromidis, e P Chan. Credit card fraud detection using meta-learning: Issues and initial results. *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997.

[40] Mirko Suznjevic, Ognjen Dobrijevic, e Maja Matijasevic. Mmorpg player actions: Network performance, session patterns and latency requirements analysis. *Multimedia Tools Appl.*, 45(1-3):191–214, outubro de 2009.

[41] Sebastian Thrun e Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[42] VGChartz. Call of Duty: Black Ops II. Disponível em: http://www.vgchartz.com/game/70716/call-of-duty-black-ops-ii/. Acesso em Ago/2015.

[43] Amir Yahyavi, Kévin Huguenin, e Bettina Kemme. Interest modeling in games: the case of dead reckoning. *Multimedia Systems*, páginas 255–270, 2012.

[44] Amir Yahyavi e Bettina Kemme. Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey. *ACM Computing Surveys*, 46 Issue 1:Article 9, 2013.

[45] Metin Zontul, Fatih Aydın, Gökhan Doğan, Selçuk Şener, e Oğuz Kaynar. Wind speed forecasting using reptree and bagging methods in kirklareli-turkey. 2013.

APÊNDICE A

APÊNDICE - ATIVIDADES DOS JOGADORES EM WORLD OF WARCRAFT

Neste apêndice será apresentado de forma mais detalhada a classificação das atividades proposta em [40]. Segundo os autores as atividades dos jogadores podem ser classificadas em *Trading, Professions, Dungeons, Raiding, Questing* e *Player versus Player*.

Em MMORPGs o comércio é considerado uma parte importante do jogo e os jogadores podem participar dele através de trocas que são classificadas como atividades de *Trading*. Em WoW as atividades de *Trading* que os jogadores podem fazer são as trocas com NPCs, com outros jogadores ou através das casas de leilão. Nas trocas feitas com outros jogadores é necessário que os jogadores se "encontrem"no mundo virtual e utilizem uma interface em que podem fazer a troca do item pelo dinheiro virtual, este mesmo processo é feito quando um jogador quer trocar com um NPC. Nas casas de leilão é permitido que os jogadores leiloem itens virtuais ou ainda comprem itens de outros jogadores. Essas atividades de *Trading* junto com as atividades de *Profession* são a base de toda a economia virtual do jogo. As atividades de *Profession* costumam ser atividades de criação ou coleta de itens. No total existem doze profissões que um avatar pode aprender, porém o jogador deve escolher apenas duas profissões para serem as principais e outras três como secundárias. Por exemplo, o jogador pode ter a profissão de alquimista como principal e fazer poções que recuperam os pontos de vida do avatar [1], como profissão secundária ele pode ser um herborista e coletar as ervas que serão utilizadas para fazer essas poções.

As *Dungeons* são instancias de áreas confinadas em que apenas um número limitado de jogadores pode entrar. Nessas *Dungeons* os jogadores podem se unir em um grupo com o

[1]Definição de pontos de vida: É um valor numérico que representa a vida do avatar. Quando os pontos de vida de um avatar são zerados o avatar desse jogador morre. Em WoW quando um avatar morre este é transformado em espirito, enviado ao cemitério mais próximo e deve retornar ao um ponto próximo do local de sua morte para ser ressuscitado. Os pontos de vida podem ser recuperados das seguintes maneiras: através de habilidades, de itens de recuperação (poções, comidas e equipamentos) ou naturalmente enquanto este avatar estiver fora de combate.

objetivo de alcançar uma meta. Para alcançar essa meta é necessário que esse grupo seja extremamente organizado e coordenado, em que cada um dos jogadores tem uma função a desempenhar, normalmente exigindo uma comunicação adicional. Cada *Dungeon* é uma instancia criada para apenas esse conjunto de jogadores que pode batalhar com os inimigos sem serem interrompidos ou ainda ajudados por outros jogadores de fora do grupo. Em WoW os inimigos são um tipo especial de NPC que tem como função primaria de ser caçado e morto pelos jogadores, recompensando o jogador através de experiencia, por objetivos cumpridos em missões, ou ainda pelo chamado *loot* que são os itens que podem ser saqueados dos inimigos "mortos". Enquanto as *dungeons* são planejadas para um grupo de até cinco jogadores nas *Raids* as instancias são planejadas para grupos grandes de até quarenta jogadores. As *Raids* são uma das tarefas mais complexas em um MMORPG, devido a necessidade de que os avatares dos jogadores sejam de um nível especifico, tenham os equipamentos necessários, desempenhem papeis pré-definidos e tenham uma coordenação e uma cooperação. Como exemplo da dificuldade que uma *Raid* pode apresentar em [40] é apresentada a chefe inimiga *Lady Vashj* [2] da primeira expansão do WoW a *The Burning Crusade*. Nessa *Raid* são necessários vinte e cinco jogadores que devem ser extremamente organizados, já que a *Lady Vashj* possui três fases com um comportamento diferenciado. Na primeira e na terceira fase o comportamento é igual, exceto pelo fato que na terceira fase *Lady Vashj* convoca morcegos que soltam esporos que reduzem os pontos de vida dos avatares dos jogadores. Na segunda fase o comportamento em equipe do grupo é testado, já que *Lady Vashj* convoca vários escudos, sendo assim imune aos ataques dos jogadores. Para diminuir os pontos de vida da *Lady Vashj* é necessário desligar quatro núcleos de força que mantêm os escudos ativos. Além disso, durante essa fase ela também convoca vários inimigos que devem ser derrotados pelos jogadores responsáveis e todos jogadores precisam ficar posicionados de forma que reduzam o dano recebido pelos ataques da própria chefe. Devido a complexidade do comportamento desses chefes foram criados diversos guias de estratégias que são utilizados pelos jogadores para ajudar na montagem dos grupos.

[2]Lady Vashj: $http://www.wowwiki.com/Lady\_Vashj$

As atividades de *Questing* são relacionadas as missões que são dadas aos jogadores pelos NPCs. As missões variam de natureza e dificuldade, mas em geral recompensam o jogador com uma certa quantia de pontos de experiência. Por exemplo, na missão Comida Crocante [3] um NPC cozinheiro tem o seguinte dialogo com o jogador:

> "Isso é inaceitável! Não tenho nada com o que trabalhar! Milady me pediu para vir a Luaprata. Ela merece uma culinária de classe, condizente com a sua importância ! Você não acha? Pois eu chego aqui e não tem comida! Você precisa me trazer alguma caça, < classe escolhida pelo jogador >, para que eu possa preparar algo! Aurífera prefere uma textura crocante na comida. Há muitas tocaieiras fusaracnas a sudoeste daqui, do outro lado da Trilha da Morte. É repugnante, eu sei, mas é o que temos à mão."

Neste dialogo o cozinheiro pede que o jogador consiga o *loot* de certo monstro para que ele e como recompensa é dado ao jogador, quando a missão for cumprida, uma certa quantidade de experiência, cinco itens chamados de "Escondidinho crocante de Aranha"e a receita para a criação do "Escondidinho crocante de aranha". Assim após completar essa missão o avatar do jogador será capaz de fazer essa receita no futuro caso tenha a profissão cozinheiro. As atividades de *questing* são a maior parte das atividades do jogadores, principalmente antes de alcançarem o nível máximo nesse avatar. São muito utilizadas para o ganho de pontos de experiência, ouro, equipamento, conquistas e pela familiarização dos jogadores com a história apresentada no jogo. As características, segundo [40] das atividades quando os jogadores estão fazendo missões é que são atividades repetitivas, simples e com um alternância entre ações muito dinâmicas e pouco dinâmicas, além de ter uma mobilidade média do jogador.

Além das atividades de *Player versus Environment* (PvE) , que são as atividades de *Raids, Dungeons e Questing*, os jogadores também podem batalhar entre si no modo *Player versus Player* (PvP). Em WoW o PvP é feito de três maneiras diferentes. A primeira forma é o PvP dentro do próprio mundo do jogo, em servidores específicos (chamados de *PvP servers*) os jogadores podem iniciar batalhas contra outros jogadores da

[3]Missão Comida Crocante: http://pt.wowhead.com/quest=9171/comida-crocante

facção oposta em grande parte dos mapas. A segunda forma é conhecido como o modo *Battlegrounds*, nesse modo são instanciados campos de batalha em que grupos de jogadores podem batalhar contra os membros da outra facção com objetivo de cumprir certas metas: capturar a bandeira, conseguir uma quantidade de pontos ou apenas matar os inimigos. O terceiro tipo de PvP é o modo Arena. No modo Arena formam-se times de dois, três ou cinco jogadores que batalham numa arena instanciada apenas para eles. Nesse modo Arena a coordenação dos jogadores também pode variar muito, existem grupos extremamente bem coordenados utilizando comunicação de voz, porém em outras situações existem jogadores que jogam "sozinhos dentro do grupo"sem uma comunicação e coordenação com os outros membros do time. Já nos *Battlegrounds* os jogadores são selecionados de forma aleatória e tem o mesmo conjunto de objetivos e regras, porém não necessariamente agem em um conjunto coordenado para alcançar esses objetivos. As atividades de PvP são bastante dinâmicas e os jogadores constantemente se deslocam. A progressão dessas atividades é dada pelas conquistas obtidas e por equipamentos que os jogadores podem comprar por pontos de honra. Um jogador vitorioso no PvP também sobe de posição em um ranking dos jogadores.

PÂMELA DE ASSIS BELTRANI

PREDIÇÃO DE MOVIMENTO EM JOGOS DISTRIBUÍDOS BASEADA EM APRENDIZADO DE MÁQUINA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Elias P. Duarte Jr.
Co-Orientadora: Prof. Aurora T. R. Pozo

CURITIBA

2015