



Universidade Federal de Pernambuco (UFPE)  
Mestrado em Ciência da Computação  
Centro de Informática (CIn)

---

# **BALANCEAMENTO DINÂMICO DE JOGOS: UMA ABORDAGEM BASEADA EM APRENDIZAGEM POR REFORÇO**

---

DISSERTAÇÃO DE MESTRADO

Gustavo Danzi de Andrade

Recife, Pernambuco, Brasil  
Agosto de 2006



UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CENTRO DE INFORMÁTICA  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO DANZI DE ANDRADE

“Balanceamento Dinâmico de Jogos: Uma Abordagem  
Baseada em Aprendizagem por Reforço”

*TRABALHO APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE  
FEDERAL DE PERNAMBUCO COMO PARTE DOS REQUISITOS PARA A  
OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.*

ORIENTADOR: PROF. GEBER LISBOA RAMALHO  
(CENTRO DE INFORMÁTICA – UFPE)

CO-ORIENTADORES: PROF. ALEX SANDRO GOMES  
(CENTRO DE INFORMÁTICA – UFPE)

PROF. VINCENT CORRUBLE  
(LABORATOIRE D'INFORMATIQUE DE PARIS VI)

RECIFE, AGOSTO/2006

**Andrade, Gustavo Danzi de**

**Balanceamento dinâmico de jogos: uma abordagem baseada em aprendizagem por reforço / Gustavo Danzi de Andrade. – Recife: O autor, 2006.**

**ix, 112 folhas. : il., fig., tab.**

**Dissertação (mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2006.**

**Inclui bibliografia.**

**1. Inteligência Artificial – Aprendizagem. 2. Jogos por computador - Programação 3. Aprendizagem por reforço. 4. Interfaces Inteligentes. I. Título.**

**006.31**

**CDD (22.ed.)**

**CIN2006-023**

# Resumo

A capacidade de entretenimento de jogos digitais depende de vários fatores, como qualidade gráfica, enredo e jogabilidade. Um dos elementos que afetam a jogabilidade é o nível de desafio enfrentado pelo usuário, que é determinado pelo balanceamento do jogo. Balancear consiste em proporcionar níveis adequados de desafios, evitando os extremos de entediar o jogador com tarefas triviais ou frustrá-lo com tarefas intransponíveis. Jogos possuem uma grande diversidade de usuários, em termos de habilidades e experiências, e cada um evolui em um ritmo distinto. Dessa forma, a dificuldade sentida por cada jogador é influenciada por suas características individuais. A adaptação dos desafios ao perfil de cada um é realizada através do balanceamento dinâmico, que automaticamente avalia cada usuário e propõe desafios adequados a suas habilidades. Este trabalho apresenta um método original de balanceamento dinâmico de jogos baseado em aprendizagem por reforço. A abordagem consiste em dividir o problema em duas dimensões: competência (o conhecimento) e desempenho (a utilização prática do conhecimento). Para a aquisição de competência, são criados agentes inteligentes capazes de descobrir, por meio de aprendizagem por reforço, diferentes estratégias de comportamento. Para adaptar o desempenho, é definida uma política de atuação que escolhe, dentre as estratégias aprendidas, aquela mais adequada a cada usuário. O método proposto e outras abordagens encontradas na literatura são implementados, testados e comparados em um jogo de luta. Os experimentos realizados através de simulações com outros agentes indicam que o método proposto consegue, com sucesso, adaptar-se a oponentes com diferentes perfis. Complementarmente, são realizados testes com jogadores reais, para os quais é definido um método de avaliação de balanceamento de jogos baseado em técnicas de usabilidade. Os resultados obtidos indicam as vantagens e limitações de cada abordagem e fornecem indícios de que um balanceamento adequado efetivamente melhora o entretenimento.

## **Palavras-chave:**

balanceamento de jogos, aprendizagem por reforço, interfaces inteligentes, adaptação ao usuário, testes de usabilidade

# Abstract

The entertaining value of digital games depends on issues like graphical interface, story, and gameplay. One of the elements that affect gameplay is the challenge faced by the user, which is defined by game balancing. Balance means to provide adequate challenges, avoiding the extremes of getting the player bored with trivial tasks or becoming frustrated with unachievable goals. Due to the great diversity among game users, in terms of skills and experiences, the difficulty perceived by each one is influenced by their individual characteristics. The adaptation of challenges to each player is made through dynamic game balancing, the process of automatically evaluating the user profile and proposing challenges adequate to his or her skills. This work presents an original method of dynamic game balancing based on reinforcement learning. Our approach divides the problem in two dimensions: competence (the knowledge) and performance (the actual use of knowledge). In order to obtain competence, we create intelligent agents capable of discovering, through reinforcement learning, different playing strategies. In order to adapt performance, we define and make these agents use an action selection policy that chooses, among the learned strategies, the most suitable for each user. The proposed method and other approaches found in the literature are implemented, tested, and compared in a fighting game. The experiments executed through simulations with other agents show that the proposed method successfully adapt to different opponents' profiles. Additionally, tests with real players are executed, applying a method of evaluating game balancing based on usability techniques. The results reveal the advantages and limitations of each approach and provide evidence that an adequate game balance effectively improves entertainment.

**Keywords:**

Game balancing, reinforcement learning, intelligent interfaces, user adaptation, usability tests

# Agradecimentos

A todos aqueles que, direta ou indiretamente, consciente ou inconscientemente, tornaram este trabalho possível. Em especial, a todos aqueles que orientaram, ensinaram, corrigiram, revisaram, comentaram, testaram, ou simplesmente ajudaram a tornar este trabalho tão gratificante.

# Sumário

<b>1. INTRODUÇÃO .....</b>	<b>1</b>
1.1. Objetivos.....	2
1.2. Abordagem Proposta .....	3
1.3. Estrutura da Dissertação.....	4
<b>2. BALANCEAMENTO DE JOGOS .....</b>	<b>5</b>
2.1. Desenvolvimento de Jogos .....	5
2.2. Jogabilidade e Satisfação .....	7
2.3. Descrição do Problema.....	10
2.4. Abordagem Tradicional .....	13
2.5. Balanceamento Dinâmico .....	15
2.5.1. Função de Desafio .....	16
2.5.2. Requisitos .....	17
2.5.3. Limitações.....	19
2.6. Conclusões .....	20
<b>3. ABORDAGENS DINÂMICAS.....</b>	<b>21</b>
3.1. Manipulação de Parâmetros .....	21
3.1.1. Descrição.....	21
3.1.2. Vantagens e Limitações .....	22
3.2. Balanceamento Baseado em Scripts Dinâmicos .....	23
3.2.1. Scripts Dinâmicos .....	23
3.2.2. Balanceamento com Scripts Dinâmicos .....	24
3.2.3. Vantagens e Limitações .....	26
3.3. Balanceamento Baseado em Algoritmos Genéticos.....	27
3.3.1. Descrição.....	27
3.3.2. Vantagens e Limitações .....	28
3.4. Conclusões .....	29
<b>4. BASES DA ABORDAGEM PROPOSTA.....</b>	<b>30</b>
4.1. Competência vs. Desempenho .....	30
4.1.1. As Idéias de Saussure .....	30
4.1.2. A Abordagem de Chomsky .....	31
4.1.3. Competência e Desempenho em Jogos.....	32
4.2. Aprendizagem por Reforço (AR) .....	34
4.2.1. O Problema de Aprendizagem por Reforço.....	34

4.2.2.	<i>Q-Learning</i> .....	35
4.2.3.	Aprendizagem por Reforço em Jogos .....	38
4.3.	Conclusões .....	40
<b>5.</b>	<b>BALANCEAMENTO BASEADO EM APRENDIZAGEM POR REFORÇO</b> .....	<b>41</b>
5.1.	Balanceamento com AR Tradicional .....	41
5.2.	Balanceamento com AR Baseado em Desafio .....	42
5.2.1.	Adquirindo e Aperfeiçoando a Competência .....	43
5.2.2.	Adaptando o Desempenho .....	46
5.3.	Conclusões .....	51
<b>6.</b>	<b>IMPLEMENTAÇÃO</b> .....	<b>53</b>
6.1.	Knock'em: Descrição do Ambiente .....	53
6.2.	Arquitetura do Ambiente .....	54
6.3.	Criação dos Agentes .....	57
6.3.1.	Codificação do Ambiente .....	57
6.3.2.	Agentes Básicos .....	60
6.3.3.	TRL – <i>Traditional Reinforcement Learning</i> .....	61
6.3.4.	CBRL – <i>Challenge-Based Reinforcement Learning</i> .....	62
6.3.5.	DSTC – <i>Dynamic Scripting Top Culling</i> .....	65
6.3.6.	GA – <i>Genetic Adaptive</i> .....	66
6.4.	Conclusões .....	68
<b>7.</b>	<b>EXPERIMENTOS: AGENTES X AGENTES</b> .....	<b>69</b>
7.1.	Treinamentos Off-line .....	69
7.2.	Avaliação do Balanceamento .....	71
7.2.1.	Metodologia de Avaliação .....	71
7.2.2.	Resultados e Interpretação .....	73
7.2.3.	Análise dos Resultados .....	78
7.3.	Aprendizagem com Atuação Ótima vs Sub-ótima .....	80
7.4.	Extensão a Outros Gêneros .....	82
7.5.	Conclusões .....	83
<b>8.</b>	<b>EXPERIMENTOS: AGENTES X USUÁRIOS</b> .....	<b>84</b>
8.1.	Metodologia de Avaliação .....	84
8.2.	Plano de Testes .....	87
8.3.	Resultados e Interpretação .....	89
8.4.	Análise dos Resultados .....	92
8.5.	Conclusões .....	98



<b>9. CONCLUSÕES .....</b>	<b>99</b>
9.1. Principais Contribuições.....	100
9.2. Trabalhos Futuros.....	100
<b>REFERÊNCIAS.....</b>	<b>103</b>
ANEXOS .....	109
Anexo 1: Questionários e Documentos Usados nos Testes.....	109

# Lista de Figuras

Figura 1: Curva de aprendizagem padrão dos usuários [Nielsen 1993] .....	11
Figura 2: Tela de escolha do nível de dificuldade do NFS Underground 2 (EA Games) .....	13
Figura 3: Evolução dos níveis de dificuldade por meio de fases .....	14
Figura 4: Tela de escolha do nível de dificuldade do Sid Meier's Civilization IV (Firaxis) .....	15
Figura 5: Representação do balanceamento de jogos (adaptado de [Falstein 2004]) .....	16
Figura 6: Exemplo de regras para o balanceamento baseado em scripts .....	24
Figura 7: Evolução da competência e adaptação do desempenho. ....	43
Figura 8: Um agente atuando no seu segundo melhor nível de desempenho.....	47
Figura 9: Telas do ambiente de testes Knock'em .....	53
Figura 10: Sistemas e módulos do ambiente de testes Knock'em .....	54
Figura 11: Diagrama de classes resumido do estado de luta.....	56
Figura 12: Exemplos de regras do agente SMPlayer escritas em pseudo-linguagem.....	60
Figura 13: Diagrama de classes resumido do agente CBRLPlayer.....	63
Figura 14: Avaliação do treinamento off-line com diferentes professores .....	70
Figura 15: Desempenho do <i>State-Machine</i> (SM) contra outros agentes.....	73
Figura 16: Desempenho do <i>Traditional RL</i> (TRL) contra outros agentes .....	74
Figura 17: Desempenho do <i>Challenge-Based RL</i> (CBRL) contra outros agentes .....	75
Figura 18: Desempenho do <i>Dynamic Scripting TC</i> (DSTC) contra outros agentes .....	76
Figura 19: Desempenho do <i>Genetic Adaptive</i> (GLA) contra outros agentes .....	77
Figura 20: Histograma de ações do agente <i>Challenge-Based RL</i> (CBRL) .....	80
Figura 21: Curva de aprendizagem dos agentes TRL e CBRL contra oponente SM.....	82
Figura 22: Curva de aprendizagem dos agentes TRL e CBRL contra oponente RD.....	82
Figura 23: Divisão das tarefas executadas nos testes com usuários .....	85
Figura 24: Visual dos personagens utilizados nos testes com usuários.....	88
Figura 25: Questionário de satisfação: "Qual o oponente mais fácil?" .....	91
Figura 26: Questionário de satisfação: "Qual o oponente mais difícil?" .....	92

# Lista de Tabelas

Tabela 1: Diferenças entre aplicativos de produtividade e jogos.....	8
Tabela 2: Requisitos de jogabilidade para o balanceamento dinâmico .....	18
Tabela 3: Requisitos de implementação para o balanceamento dinâmico .....	19
Tabela 4: Análise de desempenho do agente <i>State-Machine</i> (SM).....	74
Tabela 5: Análise de desempenho do agente <i>Traditional RL</i> (TRL) .....	75
Tabela 6: Análise de desempenho do agente <i>Challenge-Based RL</i> (CBRL).....	75
Tabela 7: Análise de desempenho do agente <i>Dynamic Scripts Top Culling</i> (DSTC).....	76
Tabela 8: Análise de desempenho do agente <i>Genetic Adaptive</i> (GLA).....	77
Tabela 9: Análise das diferenças de pontos de vida durante a fase de avaliação.....	91

# Lista de Algoritmos

Algoritmo 1: Aquisição e aperfeiçoamento do conhecimento .....	44
Algoritmo 2: Política $\epsilon$ -greedy de escolha de ações baseada em desafio .....	49
Algoritmo 3: Política softmax de escolha de ações baseada em desafio .....	50

# 1. Introdução

Jogos digitais podem ser definidos como sistemas interativos que permitem a um ou mais usuários experimentar, através de algum dispositivo computacional, situações de conflito [Crawford 1982].

Essa definição contempla duas características fundamentais: a interatividade e o conflito. Como **sistemas interativos**, jogos influem e são influenciados pelos usuários. Isso os diferencia de outras formas de entretenimento, como filmes e músicas, nos quais a influência é unidirecional: o sistema (por exemplo, um filme) causa impacto no usuário, mas o usuário não pode alterar o sistema (o final do filme não pode ser modificado por cada um). Como sistemas que criam **situações de conflito**, os jogos precisam constantemente desafiar os usuários, seja fisicamente, seja mentalmente, de modo a incentivá-los a jogar pelo maior tempo possível. Isso os diferencia de aplicativos de produtividade (como planilhas e editores de textos), que são ferramentas para facilitar a execução de uma tarefa.

O sucesso de sistemas interativos é diretamente dependente de sua usabilidade [Maguire 2001]. Usabilidade é uma característica que não depende apenas do sistema projetado, mas principalmente de como os usuários conseguem interagir com ele. Um sistema de sucesso, portanto, não é aquele que oferece mais ou melhores funcionalidades, porém aquele que oferece o que realmente se espera dele. A qualidade do sistema é determinada pelo atendimento das expectativas dos usuários [Crosby 1979].

No domínio de jogos, os usuários esperam enfrentar conflitos adequados [Crawford 1982]. Conflitos, ou desafios no contexto deste trabalho, são situações que estimulam o usuário a se esforçar para superar as dificuldades encontradas. Entretanto, nem todo conflito incentiva o jogador. Por exemplo, diante de uma tarefa impossível, dificilmente uma pessoa se sente estimulada a tentar superá-la. A criação de desafios apropriados corresponde ao problema de balanceamento de jogos. Balancear consiste em criar mecanismos que desafiem adequadamente os jogadores, evitando entediá-los com tarefas triviais ou frustrá-los com tarefas intransponíveis.

A abordagem tradicional de balanceamento consiste em definir, durante o desenvolvimento do jogo, níveis de dificuldade que oferecem desafios pré-definidos. Como todo o processo ocorre durante o desenvolvimento, é necessário realizar um grande esforço de implementação e testes até atingir um resultado satisfatório. Esse problema tende a ser agravar à medida que a complexidade dos jogos cresce.

Uma vez definido os níveis de dificuldade, cada jogador pode escolher aquele que mais se adapta a seu perfil. Entretanto, como a quantidade de níveis é limitada, um usuário pode não encontrar uma alternativa adequada a suas habilidades. Como cada pessoa possui conhecimentos e experiências distintas, além de evoluir em ritmos diferentes, considerar todas as possibilidades de perfis durante o desenvolvimento é inviável.

A evolução natural dessa abordagem é a utilização de um método de balanceamento que se adapte automaticamente e dinamicamente a cada usuário, mais adequado à característica interativa dos jogos. O balanceamento dinâmico consiste em criar mecanismos que avaliem as habilidades de cada jogador e, automaticamente, atualizem a dificuldade dos desafios propostos. Desafios adequados tendem a melhorar a capacidade de entretenimento dos jogos, na medida que evitam situações de aborrecimento e frustração.

O presente trabalho se baseia em duas premissas: (1) o balanceamento está diretamente relacionado à satisfação do usuário; e (2) uma abordagem dinâmica é mais eficiente do que o balanceamento estático. Essas premissas propõem que a capacidade de um jogo atuar no mesmo nível do seu usuário influencia positivamente no entretenimento. Usando essas premissas, podemos focar nosso trabalho no problema de balanceamento, abordando sua relação com a satisfação do usuário apenas de forma colateral. Embora as premissas ainda não tenham sido confirmadas por testes de satisfação detalhados, existem fortes indícios de que sejam verdadeiras [Crawford 1982; Pagulayan *et al.* 2003; Koster 2004].

## 1.1. Objetivos

Considerando que uma abordagem dinâmica melhora o balanceamento, o que por sua vez aumenta a satisfação do usuário, definimos como objetivo principal deste trabalho propor um método original de balanceamento dinâmico de jogos. Para efetivar esse objetivo, é necessário superar os seguintes desafios:

- Avaliar o nível de dificuldade percebido por cada jogador;
- Criar um método de adaptação;
- Definir e aplicar testes de avaliação.

A dificuldade percebida por cada usuário depende dos desafios do jogo e das habilidades de cada um. Relacionar essas duas dimensões (sistema e usuário) é essencial para que o sistema se adapte na direção correta, reduzindo o nível dos desafios quando o jogo estiver difícil e aumentando-o quando estiver fácil. Uma vez conhecido o nível de dificuldade percebido pelo jogador, é preciso adaptá-lo automaticamente quando o jogo não estiver balanceado. Uma abordagem possível para a criação de sistemas adaptativos é a utilização de aprendizagem de máquina [Langley 1997]. Entretanto, como a velocidade da aprendizagem de

computador ainda não é suficientemente rápida e eficiente para satisfazer os requisitos de tempo de uma aplicação interativa como jogos, é preciso estender ou evoluir as atuais técnicas para criar um método de adaptação eficaz. Por fim, é preciso avaliar se os métodos desenvolvidos atuam no mesmo nível de jogadores com diferentes habilidades, propondo sempre desafios adequados.

## 1.2. Abordagem Proposta

Para alcançar o objetivo especificado anteriormente, este trabalho se inspira em conceitos de Psicologia Cognitiva para criar métodos que utilizam e estendem técnicas de Inteligência Artificial (IA) e de Interface Homem-Computador (IHC).

Para avaliar o nível de dificuldade percebido por cada jogador são utilizadas funções heurísticas, denominadas função de desafio, que mapeiam cada situação do jogo a um valor especificando o nível de facilidade ou dificuldade com relação a cada usuário [Demasi 2003].

A criação de um método de adaptação utiliza como base a distinção entre competência e desempenho estabelecida inicialmente nos estudos de lingüística [Chomsky 1965]. Competência se refere ao conhecimento existente em um jogo, inserido explicitamente ou aprendido automaticamente. Desempenho é a utilização desse conhecimento em uma situação real. A partir dessa divisão, utilizamos aprendizagem por reforço [Sutton e Barto 1998] para criar e evoluir a competência. Para adaptar o desempenho, definimos uma política de atuação original que utiliza o conhecimento aprendido da maneira mais adequada a cada usuário. Com essa divisão, a competência é constantemente evoluída, mas o desempenho é limitado pelas características de cada jogador. Essa abordagem se diferencia de outras aplicações de aprendizagem de máquina por não utilizar as melhores estratégias aprendidas. Possivelmente, estratégias sub-ótimas são seguidas para garantir que o nível de desafio seja adequado: nem trivial, nem intransponível.

O termo “sub-ótimo” utilizado neste trabalho está sempre associado ao objetivo do jogo. Por exemplo, em um jogo de corrida, ser primeiro lugar é o objetivo ótimo. Um piloto que termine em segundo ou terceiro está obtendo um resultado sub-ótimo. Como o foco do nosso trabalho é a adaptação ao usuário, em alguns casos, a melhor estratégia é obter, propositalmente, um resultado sub-ótimo (e mais próximo do usuário). Logo, a melhor estratégia do ponto de vista *da adaptação* não necessariamente é a melhor do ponto de vista *do jogo*. Uma estratégia de adaptação ótima pode ser sub-ótima em relação ao objetivo do jogo.

Para validar o método proposto, foram realizados experimentos com agentes inteligentes que simulam diferentes perfis de jogadores humanos. Esses experimentos foram

efetuados em um jogo de luta, no qual foram implementadas, testadas e comparadas diferentes abordagens de balanceamento encontradas na literatura.

Como atividade complementar, foram realizados testes com jogadores humanos para coletar dados concretos e subjetivos (como satisfação e entretenimento) em um ambiente real de jogo, considerando todas as questões e técnicas envolvidas na realização de testes de usabilidade. Através destes testes, foi possível identificar o impacto de diferentes métodos no entretenimento do jogo, relacionando empiricamente e preliminarmente o balanceamento com a satisfação sentida pelos usuários.

As principais contribuições deste trabalho podem ser assim resumidas: (1) definir uma abordagem inovadora e geral de balanceamento dinâmico baseada na distinção entre competência e desempenho, (2) definir uma técnica original de implementação dessa abordagem, utilizando aprendizagem por reforço, (3) adaptar e aplicar metodologias de IHM para avaliação do balanceamento, e (4) implementar e comparar diferentes abordagens existentes na literatura, consolidando-as em um único trabalho. A próxima seção descreve como esses pontos são apresentados no restante deste documento.

### 1.3. Estrutura da Dissertação

Este trabalho está dividido em nove capítulos. No Capítulo 2, definimos e detalhamos o balanceamento de jogos, descrevendo a abordagem tradicional para o problema e analisando como uma abordagem dinâmica pode ser utilizada. No Capítulo 3, são expostas as principais propostas de balanceamento dinâmico encontradas na literatura, com as vantagens e limitações de cada uma. No Capítulo 4, expomos o referencial teórico de nossa abordagem: a distinção entre competência e desempenho e o problema de aprendizagem por reforço.

O Capítulo 5 expõe nossa proposta original de balanceamento dinâmico e apresenta como aprendizagem por reforço pode ser utilizada para criar competência e adaptar o desempenho. No Capítulo 6, explicamos os aspectos relacionados à implementação da proposta em um jogo de luta e no Capítulo 7 apresentamos os resultados obtidos nos experimentos realizados por meio de simulações. O Capítulo 8 apresenta testes complementares realizados com jogadores humanos e, por fim, o Capítulo 9 resume nossas principais contribuições e analisa algumas direções de trabalhos futuros.



## 2. Balanceamento de Jogos

Um dos componentes que influenciam na capacidade de entretenimento de jogos é o nível dos desafios propostos aos usuários, que é determinado pelo balanceamento. Este capítulo detalha esse problema e a sua relação com o conceito mais geral de jogabilidade, e expõe suas duas categorias de solução: balanceamento estático (a abordagem tradicional) e balanceamento dinâmico. Primeiramente, no entanto, é apresentada uma discussão sobre o processo de desenvolvimento de jogos como um todo e suas diferenças em relação ao desenvolvimento de aplicativos de produtividade.

### 2.1. Desenvolvimento de Jogos

O desenvolvimento de jogos é um processo multidisciplinar que envolve programação, arte e design. Tipicamente, o primeiro passo para a criação de um jogo é a escrita do documento de Game Design [Crawford 1982]. O Game Design define toda a estrutura do jogo a ser implementado, como as regras, os critérios de pontuação e vitória, o resumo do enredo, a identidade gráfica dos personagens e do ambiente, etc. Este documento serve como um guia para conduzir toda a equipe durante a elaboração do jogo. Embora existam profissionais especializados na criação do Game Design, em geral todos os participantes (programadores, artistas e gerentes) colaboram na produção do documento.

Uma vez finalizado o Game Design, é iniciada a implementação do jogo. Além das etapas tradicionais de desenvolvimento de software (elaboração de requisitos, definição da arquitetura, codificação e testes), nesta etapa também se realiza o refinamento do enredo, o desenho e modelagem gráfica dos personagens e a criação da trilha e efeitos sonoros. Todas essas áreas precisam estar integradas para que o jogo resultante forneça uma experiência unificada aos usuários.

A arquitetura básica de um jogo pode ser assim resumida. Primeiramente, são criados os personagens que atuarão no ambiente. Parte dos personagens é controlada por usuários e outra parte é controlada pelo computador. Os personagens controlados por computador são denominados NPCs (*Non-player characters*). A cada momento, são executadas ações que, de alguma forma, modificam o ambiente ou os próprios personagens. No caso de jogos de turno, cada um atua em um momento distinto (enquanto um personagem joga, os demais esperam); no caso de jogos em tempo real, todos podem sempre executar novas ações. Os efeitos de cada ação no jogo são determinados pelas regras definidas no Game Design.

Além dessa arquitetura geral, existem diferentes gêneros que agrupam os jogos com características semelhantes. Embora não exista um consenso sobre a quantidade e as características dos gêneros existentes, podemos destacar as seguintes categorias, que incluem os principais jogos desenvolvidos atualmente:

- *Ação*: são aqueles jogos que enfatizam a perícia do jogador na utilização dos controles, geralmente em tempo real. Podem ser subdivididos em FPS (*First Person Shooters*) e aventura;
- *Esportes*: caracterizado por simulações de partidas e competições esportivas, incluindo corridas de carro;
- *Estratégia*: caracterizado pela simulação de mundos complexos, nos quais os jogadores tomam decisões em elaboradas para prosperar no mundo virtual;
- *Luta*: caracterizado pela simulação de uma arena na qual lutadores se enfrentam para derrotar o adversário;
- *Puzzles*: caracterizado por colocar desafios mentais (matemáticos, lógicos, etc.) para o jogador;
- *Role-Playing*: caracterizado pela criação de ambientes fantásticos e com grande riqueza de detalhes no enredo, e;
- *Simulação*: caracterizado pela replicação de ambientes reais, como a pilotagem de aviões ou a economia de cidades.

Outras categorias de jogos poderiam ser listadas, como jogos educacionais e jogos de terror. Entretanto, especificamos apenas esses gêneros principais porque não é o foco deste trabalho produzir uma taxonomia detalhada e definitiva de jogos. Além disso, a tendência atual dos desenvolvedores é, cada vez mais, produzir jogos “híbridos”, que compartilham as características de mais de um gênero, o que dificulta ainda mais um consenso sobre as categorias de jogos existentes.

No caso de jogos, o Game Design substitui o documento de requisitos como o artefato que estabelece, similarmente a um contrato, o escopo e as funcionalidades do sistema a ser desenvolvido [Robertson 2001]. O game design é comumente elaborado por pessoas da própria equipe de desenvolvimento, sem a incorporação direta da perspectiva do usuário, uma vez que a interação com jogadores nessa fase não é uma tarefa fácil. Como o objetivo do sistema é o entretenimento, não há um conjunto de tarefas bem definidas a serem executadas. Existem várias possibilidades que podem atender a esse objetivo, de modo que a participação do usuário no início do projeto pode gerar muitas possibilidades distintas (e sem relação entre si) de soluções. Além disso, os próprios desenvolvedores também são usuários (e geralmente possuem um conhecimento profundo do domínio), o que possibilita a definição de um projeto

sem a participação direta de colaboradores externos. Recentemente, a incorporação do usuário ao projeto do jogo vem sendo feita por ferramentas extras fornecidas pelos desenvolvedores, como os editores de cenários ou as linguagens de programação do comportamento de NPCs.

Com relação aos testes, em geral os estúdios desenvolvedores de jogos realizam várias atividades durante a fase de desenvolvimento, de modo a identificar problemas técnicos e recolher sugestões de melhorias para o projeto. Algumas instituições diferenciam os testes em 3 categorias [MGSUserResearch.com 2006]: testes de usabilidade (que buscam problemas e possibilidades de melhoria), *playtests* (que analisam o comportamento e as preferências dos usuários), e *reviews* (que consistem no feedback de profissionais da indústria).

Os principais testes realizados durante o desenvolvimento de jogos visam o balanceamento. Analisando usuários com diferentes perfis, os cenários, parâmetros e comportamentos do jogo são alterados para contemplar as sugestões obtidas (geralmente através de grupos de estudo – *focus groups* – ou de *beta* testes) e proporcionar uma experiência de jogo adequada, nem muito fácil, nem muito difícil.

## 2.2. Jogabilidade e Satisfação

Quando se analisa a experiência do usuário diante de um sistema, tradicionalmente se considera a sua usabilidade. Usabilidade é uma propriedade com múltiplos componentes, que podem ser agrupados em cinco atributos básicos [Nielsen 1993]: aprendizagem (*learnability*), eficiência, memorização (*memorability*), taxa de erros e satisfação. Um sistema usável é aquele que tem como propriedades uma baixa necessidade de aprendizagem, alta eficiência, alta capacidade de memorização, baixa taxa de erros e alto grau de satisfação.

Em aplicativos de produtividade (tais como processadores de texto e planilhas), o usuário é, de alguma forma, induzido a usar o sistema como forma de aumentar sua eficiência na realização de tarefas bem definidas. Dessa forma, busca-se remover desafios e manter a consistência, para que o usuário execute suas tarefas o mais rapidamente possível.

No caso de jogos, o objetivo principal é o entretenimento. Nesse caso, o usuário utiliza o sistema por iniciativa própria, com objetivo de se divertir pelo maior tempo possível. Busca-se então impor desafios e introduzir inovações, para que o usuário seja estimulado a jogar pelo maior tempo possível [Pagulayan *et al.* 2003]. As diferenças entre aplicativos e jogos são resumidas na Tabela 1.

Essas diferenças essenciais tornam necessária a distinção entre usabilidade no contexto de aplicativos de produtividade e usabilidade no contexto de jogos. Nesse último caso, a semântica do termo se confunde com outra propriedade amplamente citada na área de jogo: a **jogabilidade**.

**Tabela 1: Diferenças entre aplicativos de produtividade e jogos**

<b>Aplicativos de produtividade</b>	<b>Jogos digitais</b>
Utilização obrigatória	Iniciativa própria do usuário
Consistência	Inovação
Remover desafios	Impor desafios
Objetivos são tarefas bem definidas	Objetivo básico é o entretenimento

No contexto deste trabalho, o termo jogabilidade se refere a toda a experiência do usuário (interface, enredo, satisfação, etc.) diante do sistema [Rollings e Morris 2003]. Em outras palavras, são as propriedades do jogo que influenciam na satisfação do usuário, podendo ser associada à facilidade e à motivação do usuário para jogar. A jogabilidade, e, por consequência, a satisfação, é influenciada por vários fatores, dentre os quais o balanceamento, no qual este trabalho se concentra. Logo, embora tenhamos como finalidade geral o aumento da satisfação, outros elementos não abordados influenciam na realização deste objetivo.

Para relacionar satisfação do usuário com balanceamento é necessário escolher cuidadosamente as variáveis a serem medidas, assim como os melhores métodos para coletá-las. Simplesmente deixar os usuários jogarem e perguntar se gostaram ou não da experiência provê apenas informações superficiais. Respostas a esse tipo de pergunta são profundamente influenciadas por fatores como design gráfico, controles de entrada, enredo e todas as dimensões que compõem um jogo [Pagulayan *et al.* 2003].

Como o balanceamento é uma propriedade relacionada ao desafio enfrentado pelo usuário, pode-se tentar inferir o seu nível por meio de algumas variáveis concretas. Uma abordagem natural é utilizar a evolução da pontuação (*score*) do jogador. *Scores* são baseados em métricas objetivas, como número de peças ganhas e perdidas, evolução nos pontos de vida, ou porcentagem de disparos certos, e podem ser automaticamente calculadas no decorrer do jogo. A questão consiste em como relacionar essas métricas com a satisfação do usuário, permitindo que seja checado se um balanceamento adequado, expresso por um *score* justo, é realmente mais divertido.

O problema de relacionar métricas objetivas com satisfação já foi abordado por outros autores. Yannakakis e Hallam [2005] se propuseram a criar medidas objetivas para o interesse do usuário em um jogo. Os autores definem que o interesse (equivalente à satisfação) é determinado por três critérios:

- Se o jogo não é muito fácil nem muito difícil (isto é, se é balanceado);
- Se há diversidade de comportamento dos oponentes NPCs, e;
- Se os oponentes NPCs são agressivos (e não estáticos).

Esses critérios valem, segundo os autores, para todos os jogos, mas o peso que cada um possui na satisfação geral depende do tipo de jogo em questão. Por exemplo, em jogos de guerra, a agressividade possui um peso maior do que em jogos de simulação. Os critérios são definidos de modo a serem suficientemente genéricos e objetivos, para serem aplicados em diferentes gêneros de jogos e ainda poderem ser mapeados em métricas bem definidas. No trabalho apresentado, os autores utilizam o modelo em uma versão do tradicional *Pac-Man*, produzido pela Namco (Japão) em 1980. Nesse jogo, os critérios são mapeados em variáveis como tempo de cada partida, número de visitas a cada célula do mapa, ou distâncias dos oponentes ao jogador. Quando a partida é demorada, temos um indicativo de que o jogo está balanceado, porque nem o jogador nem o oponente conseguem vencer rapidamente. Quando a porcentagem de células do mapa visitadas é grande, há um indicativo de que o oponente diversifica seu comportamento. Por fim, quando as distâncias entre os personagens são pequenas, há um indicativo de que o oponente está perseguindo o jogador de forma agressiva.

Os experimentos realizados com agentes inteligentes que simulam diferentes estratégias sugerem que essas métricas são suficientemente genéricas para serem aplicados em diferentes tipos de jogos (além de serem bastante objetivas). Entretanto, a proposta ainda não foi validada com jogadores humanos, para que seja analisado se os critérios definidos efetivamente se relacionam com o interesse dos usuários.

Outros autores definem um modelo completo de avaliação da satisfação em jogos [Sweetser e Wyeth 2005]. Neste trabalho, são integradas diferentes heurísticas encontradas na literatura para consolidar um modelo de satisfação, baseando-se no conceito de *Flow*:

*(Flow is an experience) “so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it” [Csikszentmihalyi 1990]*

O conceito de *Flow* foi desenvolvido pelo psicólogo Csikszentmihalyi [1990] através de extensas pesquisas com pessoas de diferentes perfis. O objetivo era identificar os elementos que influenciam as atividades que proporcionam grande satisfação a essas pessoas. Alguns exemplos desses elementos são a necessidade de concentração, a capacidade de atingir metas bem definidas, e o sentimento de envolvimento com outra realidade (diferente da experimentada no dia-a-dia). Segundo Csikszentmihalyi, são elementos como esses que

incentivam uma pessoa a ler um livro, compor uma música, ou escalar uma montanha. Todas essas atividades são identificadas como prazerosas por seus executores, independentemente do esforço despendido para realizá-las e da aparente ausência de recompensa.

Sweetser e Wyeth [2005] adaptaram o conceito de *Flow* e desenvolveram um modelo para avaliar a satisfação do usuário em jogos. O modelo possui oito elementos que determinam como o usuário é entretido:

- Concentração: é necessário requerer o máximo de sua concentração;
- Desafio: deve-se desafiá-lo suficientemente, de acordo com seu perfil;
- Habilidades: deve-se permitir / incentivar a evolução de suas habilidades;
- Controles: o usuário deve sentir que influencia nos resultados do jogo;
- Objetivos: deve-se prover metas claras e possíveis de serem atingidas;
- *Feedback*: deve-se recompensá-lo e informá-lo periodicamente sobre suas metas;
- Imersão: deve-se mantê-lo profundamente envolvido no ambiente;
- Interação social: deve-se permitir a interação social entre os jogadores.

Este modelo reconhece a influência do balanceamento (elemento desafio) no entretenimento. O modelo proposto é validado, com sucesso, por meio de análise com especialistas de jogos (*expert reviews*). Embora este universo não represente adequadamente os perfis de jogadores existentes, temos fortes indícios de que a satisfação está diretamente relacionada ao balanceamento.

Neste trabalho, utilizamos como premissa que o balanceamento influi positivamente na satisfação. A análise detalhada da relação entre esses conceitos está fora do nosso escopo, uma vez que requer o estudo de diversos outros componentes de um jogo, conforme indicado pelos trabalhos apresentados anteriormente. Dessa forma, no restante deste trabalho focamos os estudos no balanceamento e nas questões envolvidas com esse problema.

## 2.3. Descrição do Problema

O balanceamento de jogos (*game balancing* ou *difficulty scaling*) consiste em modificar parâmetros, cenários, ou comportamentos com o objetivo de garantir um nível adequado de desafio ao usuário, evitando os extremos de frustrá-lo porque o jogo é muito difícil ou entediá-lo porque o jogo é muito fácil [Koster 2004]. Essa questão é reconhecida pela comunidade de desenvolvimento de jogos como uma característica fundamental para o sucesso [Falstein 2004].

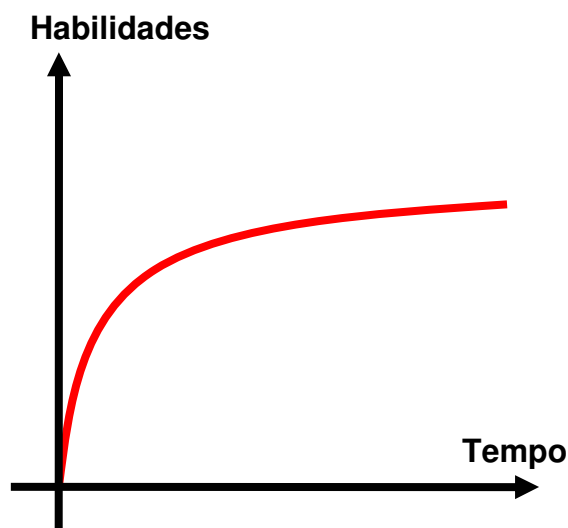
A função do balanceamento é garantir que os conflitos colocados para os jogadores sejam adequados. Conflitos insuficientes ou triviais prejudicam a qualidade na medida em que

não desafiam o usuário a interagir com o sistema. Desafios simples, repetitivos, e previsíveis entediam o jogador, que não se sente estimulado para utilizar o sistema por muito tempo.

Por outro lado, conflitos excessivos ou intransponíveis também prejudicam a qualidade, na medida em que limitam a evolução do usuário. Nesses casos, ele pode se sentir frustrado por sempre enfrentar o mesmo desafio e nunca conseguir superá-lo, sendo também desestimulado para jogar por muito tempo.

Para evitar essas situações, é necessário realizar um balanceamento adequado. No entanto, balancear não se limita a encontrar um nível mediano de desafio. É preciso considerar a evolução do usuário no decorrer do tempo, isto é o aumento de suas habilidades. À medida que interage com um sistema, seja ele um jogo ou não, as habilidades de uma pessoa são melhoradas, como resultado do processo natural de aprendizagem humana. Com isso, o nível dos desafios também precisa acompanhar essa evolução, adaptando-se continuamente a essas novas habilidades. Essa questão é crítica nas primeiras interações com o jogo, quando as habilidades crescem rapidamente, conforme ilustra a Figura 1.

O início de um jogo é um momento crítico para o balanceamento. Além de ser a fase que apresenta maior variação das habilidades, é nesse momento que os usuários são mais sensíveis a problemas. Se após as primeiras tentativas o jogador perceber que está enfrentando desafios triviais ou intransponíveis, dificilmente voltará a jogar.



**Figura 1: Curva de aprendizagem padrão dos usuários [Nielsen 1993]**

Embora a Figura 1 ilustre a aprendizagem padrão dos usuários, na qual as habilidades sempre aumentam, em alguns casos pode haver uma regressão. As habilidades podem diminuir quando o usuário passa um longo período sem jogar. No caso do balanceamento, as regressões do jogador tornam necessárias uma regressão equivalente no nível de desafio.

Dessa forma, um método eficiente de balanceamento deve acompanhar não apenas a evolução, mas também as eventuais regressões de cada usuário.

O balanceamento é uma atividade essencialmente de Game Design [Crawford 1982] e é definido em termos dos três componentes especificados anteriormente: cenários, parâmetros, e comportamentos. Cenários são situações e ambientes com os quais o usuário pode interagir. Em jogos, assim como em outros sistemas interativos, não há um cenário fixo com o qual todos interagem, mas sim um conjunto de opções. Durante a interação, cada jogador escolhe que opções seguir. Tipicamente, são definidos cenários de diferentes níveis de dificuldade. Inicialmente, apenas os mais fáceis são disponibilizados aos jogadores, e no decorrer do tempo os mais difíceis são apresentados. A modificação dos cenários e sua evolução é um dos fatores que influenciam no balanceamento.

Outro componente que exerce influência é o conjunto de parâmetros do jogo. Variáveis como número de inimigos, força das armas, e intervalo entre combates, podem facilitar ou dificultar o jogo. Em geral, os parâmetros são especificados heurísticamente pelos desenvolvedores ou através de testes de jogabilidade (*playtests*) [Fullerton *et al.* 2004]. Nesse último caso, são recrutados jogadores (representativos do mercado-alvo) que testam versões preliminares do sistema. Com base nas suas reações e sugestões, os parâmetros são modificados para melhorar o balanceamento, e conseqüentemente, a jogabilidade. Esse processo é realizado iterativamente: após cada teste, é produzida uma nova versão que volta a ser testada pelos jogadores. Por exemplo, durante o desenvolvimento do jogo Ratchet & Clank: Up Your Arsenal (produzido pela Insomniac), foram realizados 9 testes de jogabilidade, 3 testes localizados (com jogadores dos Estados Unidos, da Europa, e da Coreia do Sul), além das tradicionais versões alfa, beta e *gold* (a versão final) [Hastings 2005]. Esses dados mostram o esforço necessário para obter um bom balanceamento.

Por fim, um outro fator que exerce influência é o comportamento dos personagens que simulam jogadores humanos, denominados de NPCs (*Non-player characters*) e controlados pelo computador. Esses personagens são tipicamente implementados como agentes inteligentes [Russel e Norvig 2004] utilizando algum mecanismo de Inteligência Artificial (IA) [Game.AI 2006]. O comportamento dos NPCs pode facilitar ou dificultar o nível de desafio sentido pelos usuários. Por exemplo, em um jogo do tipo FPS (*First Person Shooter*), um oponente NPC que se movimenta pouco e não toma iniciativa do ataque é mais fácil do que um que persegue o jogador agressivamente.

A definição dos cenários, parâmetros e comportamento de um jogo é tipicamente executada pelo game designer. As decisões do game designer possuem um grande impacto sobre o balanceamento, uma vez que é a partir dela que são definidas as possibilidades de



interação. Na próxima seção, é detalhada a implementação utilizada pela maioria dos jogos comerciais.

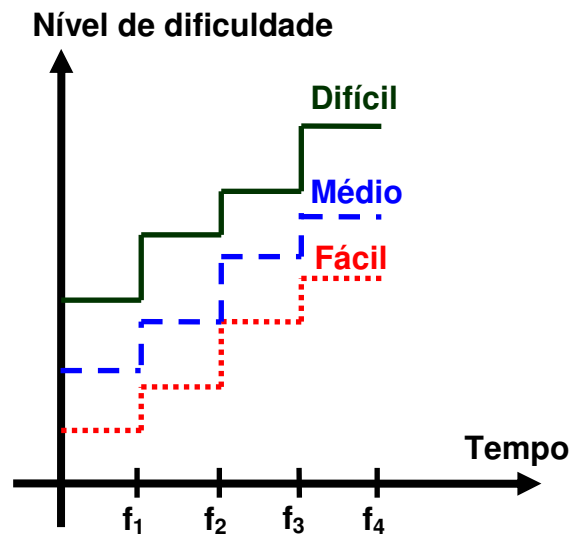
## 2.4. Abordagem Tradicional

A abordagem tradicional de balanceamento é baseada em níveis de dificuldade (como fácil, médio, difícil, e insano), definidos durante o desenvolvimento. Cada jogador escolhe aquele nível que mais se adapta a seu perfil e enfrenta os desafios associados a ele. Por exemplo, em um jogo do tipo FPS, o nível fácil em geral equivale a enfrentar oponentes com atributos fracos e em locais com muitas armas e munições disponíveis. Em jogos de corrida, por outro lado, esse nível equivale a enfrentar oponentes com carros piores e que periodicamente erram nas curvas. A Figura 2 exibe a tela de escolha do nível de dificuldade de um jogo clássico, no qual são disponibilizados os níveis fácil, médio e difícil.



**Figura 2: Tela de escolha do nível de dificuldade do NFS Underground 2 (EA Games)**

Na abordagem tradicional, cada nível de dificuldade costuma aumentar progressivamente os desafios propostos, através do conceito de *fases* (ou níveis). As fases são conjuntos seqüenciais de tarefas que os jogadores devem realizar. A cada fase avançada, a dificuldade das tarefas aumenta (não linearmente). Dessa forma, busca-se considerar a evolução do jogador no decorrer do tempo. Esses conceitos são ilustrados na Figura 3.



**Figura 3: Evolução dos níveis de dificuldade por meio de fases**

As fases atuam como barreiras de nivelamento, que servem para garantir que todos os jogadores que passaram por ela tenham um nível mínimo de habilidade. A idéia é que os jogadores sejam desafiados a evoluir para superar essas barreiras, evitando a situação simplista em que seja possível chegar ao final do jogo sem realizar algum esforço.

Embora seja amplamente utilizado na indústria de jogos, esse tipo de balanceamento apresenta a limitação de se basear em modelos pré-definidos de usuários. Como os níveis de dificuldade e suas respectivas fases são definidos durante o desenvolvimento, jogadores que tenham um perfil diferente do padrão podem ter sua experiência de jogo prejudicada. De fato, essa situação pode ocorrer com frequência, uma vez que os perfis de jogadores são tipicamente representados por poucos modelos (em geral, 3 ou 4). Para atenuar esse problema, alguns jogos oferecem mais níveis do que os clássicos fácil, médio e difícil. Jogos como o Sid Meier's Civilization IV (Firaxis) oferecem 9 níveis de dificuldade, como mostra a Figura 4, o que aumenta a possibilidade do usuário conseguir escolher um nível adaptado a seu perfil.

A limitação dessa abordagem é que, ao criar níveis pré-definidos, o jogo deixa de considerar as habilidades específicas de cada usuário. Cada jogador tem habilidades diferentes, sendo bom em alguns aspectos e ruim em outros. A evolução de cada um também é distinta, pois alguns podem aprender mais rápido do que outros. Desconsiderar essas diferenças pode resultar em um balanceamento frustrante. Além disso, agrupar os usuários em um limitado conjunto de categorias equivalentes aos níveis pré-definidos requer um extenso

trabalho de desenvolvimento, pois é necessário implementar manualmente todos os níveis necessários.



**Figura 4: Tela de escolha do nível de dificuldade do Sid Meier's Civilization IV (Firaxis)**

No caso de jogos, essa dificuldade é intensa porque uma de suas principais características é a grande diversidade de usuários, que possuem níveis de habilidade e estilos distintos. Para utilizar uma classificação mais específica, o balanceamento pode ser abordado de forma dinâmica, sendo automaticamente definido durante a interação com o jogo e considerando as características individuais de cada usuário.

## 2.5. Balanceamento Dinâmico

O balanceamento dinâmico consiste em prover mecanismos que, periodicamente, identifiquem o nível de habilidade do jogador e, automaticamente, atualizem o nível de dificuldade do jogo para mantê-lo próximo à capacidade do usuário.

Esse conceito é ilustrado na Figura 5, que mostra as possibilidades de balanceamento de um jogo: muito difícil (quando a dificuldade é muito maior do que a habilidade do usuário), muito fácil (quando a dificuldade é muito menor do que a habilidade) e balanceado, quando a dificuldade e a habilidade são equivalentes. O objetivo é manter-se na região balanceada independentemente do nível de habilidade do jogador.



**Figura 5: Representação do balanceamento de jogos (adaptado de [Falstein 2004])**

Entretanto, além do nível de dificuldade rígido, um outro componente pode frustrar os usuários: a previsibilidade. Em geral, os jogadores preferem enfrentar momentos de tensão (quando o jogo oferece desafios difíceis) intercalados com momentos de relaxamento (com desafios um pouco mais fáceis) [Falstein 2004]. Essa alternância no nível de dificuldade pode ser implementada em jogos de diferentes maneiras, como a disponibilização de novas armas (momento de relaxamento) após a superação de um chefe-de-fase (momento de tensão). Dessa forma, conforme ilustra a Figura 5, o objetivo não é acompanhar a linha tracejada de jogo perfeitamente balanceado (Balanceamento linear, quando a dificuldade do jogo é exatamente igual à habilidade do jogador). A idéia é intercalar momentos mais difíceis, com momentos menos desafiadores, de modo que o usuário sinta-se estimulado a evoluir e mantenha o contínuo interesse pelo jogo. Esse objetivo é representado pela linha contínua (Balanceamento não previsível) da Figura 5.

Há diferentes abordagens possíveis de balanceamento dinâmico. Em todos os casos, é necessário medir, implícita ou explicitamente, o nível de dificuldade que o usuário está enfrentando em um dado momento. Essa medida pode ser realizada por uma função heurística, que alguns autores denominam função de facilidade (ou **função de desafio**) [Demasi 2003].

### 2.5.1. Função de Desafio

Uma função de desafio mapeia cada estado do jogo em um valor informando quão fácil ou difícil ele está sendo percebido pelo usuário naquele momento. Essa função pode especificar apenas se o jogo está fácil, médio ou difícil, ou pode retornar um valor numérico indicando o nível de dificuldade (ou facilidade) naquele momento. Por exemplo, pode-se definir uma função

que retorna um valor no intervalo  $[-1;1]$ : quanto mais próximo do limite -1, mais fácil está o jogo; quanto mais próximo de 1, mais difícil; valores próximos a 0 indicam balanceamento adequado.

A função de desafio é usada periodicamente no decorrer do jogo. De acordo com o resultado, algum mecanismo de adaptação é utilizado para aumentar ou diminuir o nível de dificuldade. Como a função é usada constantemente, sua implementação deve ser simples, de modo a não degradar o desempenho geral do jogo [Demasi 2003].

Essas funções podem ser baseadas em heurísticas. Alguns exemplos de implementação são: taxa de golpes ou tiros certos, número de peças ganhas e perdidas, evolução dos pontos de vida, tempo para completar alguma tarefa, ou qualquer outra métrica usada para calcular a pontuação (*score*) de um jogo. Quanto maior a pontuação, mais fácil ele está. A grande questão consiste em definir que intervalos de pontuação correspondem a um balanceamento adequado, o que depende profundamente do jogo analisado.

Uma vez conhecido o nível de dificuldade percebido pelo usuário, é necessário definir algum mecanismo de adaptação, que controle os desafios com base nas características de cada jogador. A seguir, são apresentados os requisitos necessários para esses mecanismos.

## 2.5.2. Requisitos

Para que o balanceamento dinâmico seja efetivo, identificamos que devem ser satisfeitos alguns requisitos básicos, que podem ser divididos em dois grupos: requisitos de jogabilidade e de implementação.

### ***Requisitos de Jogabilidade***

Os requisitos de jogabilidade são aqueles que influenciam diretamente na percepção dos usuários em relação aos jogos. As principais dimensões que afetam essa percepção são a velocidade da adaptação ao usuário (*adaptation*) e a credibilidade das mudanças de comportamento (*believability*). Os requisitos identificados são especificados na Tabela 2.

O requisito [R/JOG 1.A] é importante para evitar a frustração do usuário nas interações iniciais, quando o nível de habilidade pode variar muito entre um iniciante e um jogador avançado. O [R/JOG 1.B] é necessário porque um jogador tipicamente não tem disposição para esperar por muito tempo pela adaptação do balanceamento. Após passar algum tempo experimentando uma situação muito fácil ou muito difícil, provavelmente desistirá de utilizar o jogo.

O requisito [R/JOG 2.A] é essencial para que o balanceamento não seja manipulado pelos usuários. Caso não seja satisfeito, um jogador pode fingir ser pouco habilidoso (executando ações fracas), esperar o jogo baixar o nível de dificuldade, e superar os desafios

impostos. Repetindo esse comportamento periodicamente, teríamos a situação indesejada de um usuário que finaliza o jogo mesmo jogando mal. O requisito [R/JOG 2.B] é importante para que não se perceba que o computador está de alguma forma facilitando o jogo (reduzindo os atributos físicos dos personagens ou executando ações autodestruídas e incoerentes) [Scott 2002]. Por exemplo, em um jogo FPS, não é razoável que o oponente permaneça parado esperando o ataque do usuário (quando o jogo está muito difícil) ou que os oponentes obtenham um escudo indestrutível e armas superpoderosas (quando o jogo está muito fácil).

**Tabela 2: Requisitos de jogabilidade para o balanceamento dinâmico**

Adaptação ( <i>Adaption</i> ):
<b>[R/JOG 1.A] Rápida adaptação inicial</b> ( <i>fast bootstrapping</i> ): O jogo deve perceber e se adaptar rapidamente ao nível inicial do usuário.
<b>[R/JOG 1.B] Rápida adaptação contínua</b> ( <i>fast following</i> ): O jogo deve acompanhar rapidamente as evoluções e regressões do usuário.
Credibilidade ( <i>Believability</i> ):
<b>[R/JOG 2.A] Inércia</b> ( <i>inertia</i> ): As variações do jogo devem ocorrer de forma suave, isto é, não deve haver alterações bruscas entre momentos de muita facilidade e outros de muita dificuldade.
<b>[R/JOG 2.B] Racionalidade</b> ( <i>rationality</i> ): O jogo não deve realizar trapaças ou ações incompatíveis.

### **Requisitos de Implementação**

Os requisitos de implementação são aqueles que influenciam no processo de desenvolvimento e nas características técnicas do jogo, e são especificados na Tabela 3.

O requisito [R/IMP 3] é importante porque o custo de aplicação de um método (codificação e configuração) não deve ser elevado, sob pena de inviabilizar sua aplicação a jogos complexos. O [R/IMP 4] é relevante porque os recursos computacionais utilizados pelas rotinas de processamento gráfico são tipicamente elevados, restando pouca disponibilidade para as rotinas de balanceamento.

Os requisitos especificados servirão de base para a análise de diferentes estratégias de balanceamento encontradas na literatura. Tipicamente, as estratégias possuem pontos fortes e pontos fracos, de modo que é possível identificar alguns requisitos claramente satisfeitos e outros não contemplados.

**Tabela 3: Requisitos de implementação para o balanceamento dinâmico**

Processo
<b>[R/IMP 3] Desenvolvimento:</b> O esforço de desenvolvimento necessário para a aplicação do balanceamento dinâmico ao um jogo deve ser simples.
Desempenho:
<b>[R/IMP 4] Execução</b> Os algoritmos de balanceamento utilizados devem consumir poucos recursos de processamento e de memória.

### 2.5.3. Limitações

O balanceamento dinâmico de jogos possui algumas limitações diante da abordagem tradicional. O maior problema identificado é a dificuldade em incentivar a aprendizagem do jogador, levando-o à acomodação. O conceito tradicional de fases, como ilustrado na Figura 3, estabelece barreiras de nivelamento ao longo do jogo que obrigam os usuários a atingirem um determinado nível de habilidade antes de avançar para a próxima etapa.

Com o balanceamento dinâmico, essas barreiras podem ser diminuídas para se adequar a um jogador iniciante, não induzindo a sua aprendizagem. Dessa forma, jogadores pouco habilidosos podem superar todos os obstáculos (e terminar o jogo) sem terem evoluído substancialmente. Além disso, essa limitação dificulta substancialmente o trabalho do *game designer*, que passa a ter de projetar ambientes para jogadores com habilidades indefinidas. Na abordagem tradicional, o game designer sabe, por exemplo, que quando um usuário atinge a última fase, seu nível de habilidade é suficientemente alto (em virtude das barreiras que precisou superar) pra lidar com elementos complexos do jogo. Com o balanceamento dinâmico, um usuário pouco habilidoso pode atingir a mesma fase e não conseguir entender e manipular os elementos disponíveis. Em geral, toda inserção de componentes dinâmicos, se por um lado flexibiliza, por outro pode dificultar o projeto e os testes de jogos, na medida que torna possível a ocorrência de comportamentos ou situações imprevisíveis [Barnes e Hutchens 2002].

A abordagem mais promissora para aprimorar o balanceamento de jogos é combinar os métodos tradicionais com mecanismos adaptativos. Nesse contexto, as barreiras de nivelamento seriam intercaladas com desafios dinâmicos (no mesmo nível, ligeiramente acima, ou muito acima do nível do usuário), de forma a incentivar a aprendizagem dos jogadores sem

frustrar ou entediar aqueles que não têm um comportamento comum. O balanceamento dinâmico poderia também ser unicamente usado nos níveis (fases) iniciais do jogo, ou no início de cada nível, onde a frustração do usuário pode ser determinante para que ele abandone o jogo. Enfim, pode-se transferir mais controle do processo de adaptação ao game designer (ou ao *level designer*) para que ele possa definir o quão mais fácil ou difícil ele quer que o jogo seja em cada situação. Em outras palavras, quão acima ou abaixo o desempenho dos agentes deve estar em relação ao do jogador.

## 2.6. Conclusões

A satisfação dos usuários diante de um sistema é influenciada não apenas por características técnicas, mas também pelas diferenças e características individuais de cada um [Nielsen 1993]. Isso significa que para satisfazer os usuários de um jogo, é necessário analisar os seus perfis, em termos de habilidades, conhecimento do domínio, e capacidade de aprendizagem ou adaptação no decorrer do tempo. Essa é uma questão crítica em aplicações que possuem uma grande diversidade de usuários, quando se torna essencial desenvolver mecanismos de adaptação para permitir que o produto desenvolvido seja satisfatoriamente utilizável por todos.

Esse capítulo apresentou o problema de balanceamento de jogos e como ele tem sido abordado por jogos tradicionais. A abordagem tradicional é baseada em níveis pré-definidos de dificuldade e apresenta limitações diante da heterogeneidade dos usuários de um jogo. Embora tenha se mostrada efetiva para muitos jogos, o aumento da complexidade observado nos últimos sistemas desenvolvidos tem apontado para a necessidade de desenvolvimento de novas técnicas [Rabin 2003]. Uma abordagem dinâmica para o balanceamento, é portanto uma direção promissora para a evolução da qualidade dos jogos digitais. A seguir, são apresentadas algumas propostas existentes na literatura que seguem essa direção.



## 3. Abordagens Dinâmicas

Nas próximas seções, são detalhadas as abordagens existentes na literatura para o balanceamento dinâmico de jogos. A primeira abordagem é baseada na modificação de parâmetros do jogo. As demais são baseadas na modificação do comportamento dos personagens NPCs (*Non-player characters*). É importante ressaltar que essas duas estratégias (modificação de parâmetros ou comportamentos) são complementares, e podem ser utilizadas simultaneamente em um mesmo jogo.

### 3.1. Manipulação de Parâmetros

O balanceamento dinâmico pode ser realizado através da manipulação automática de parâmetros do jogo. Hunicke e Chapman [2004] defendem a manipulação de variáveis do ambiente para criar dinamicamente desafios mais fáceis ou mais difíceis ao usuário. Por exemplo, se o jogo estiver muito difícil (o que pode ser inferido através da função de desafio), o usuário pode ganhar mais armas, recuperar pontos de vida mais rapidamente ou enfrentar menos oponentes. Uma pequena variação consiste em modificar dinamicamente os parâmetros dos NPCs, isto é, sua força, resistência, ou qualquer outro atributo do personagem, sem alterar o seu comportamento.

#### 3.1.1. Descrição

O trabalho de Hunicke e Chapman utiliza uma visão econômica dos jogos, focando nos mecanismos que determinam a oferta e a demanda no ambiente. Segundo esta visão, a intervenção do jogador corresponde à oferta e os desafios propostos pelo sistema, à demanda. O balanceamento é obtido através de aumentos ou diminuições nas ofertas e demandas, de forma que ambas se mantenham equilibradas.

Do lado da oferta, são manipulados os itens disponibilizados para o jogador (armas, munições, pacotes de saúde) ou os parâmetros do seu personagem (pontos de vida, força, resistência), de forma a facilitar ou dificultar suas intervenções no ambiente. Do lado da demanda, são modificados os parâmetros dos inimigos (força, localização, armas usadas) ou seu comportamento. Nesse caso, é preciso que tenham sido codificadas diferentes estratégias de atuação dos NPCs (com diferentes níveis de dificuldade), para que o jogo possa escolher, dinamicamente, que estratégia utilizar.

Essa abordagem pode utilizar métodos estatísticos para prever as “necessidades” de cada usuário, isto é, como os parâmetros devem ser adaptados para que o jogo mantenha-se

balanceado. Manipulando as variáveis de oferta e demanda na direção correta, podemos facilitar ou dificultar o jogo para cada usuário.

### 3.1.2. Vantagens e Limitações

Essa abordagem apresenta a vantagem de poder ser facilmente integrada à abordagem tradicional. Por exemplo, como os jogos já implementam diferentes níveis de dificuldade (fácil, médio, e difícil), esses níveis pré-definidos pode ser automaticamente escolhidos pelo jogo, de acordo com a função de desafio.

Para modificar outros parâmetros (como força, pontos de vida, número de inimigos), é preciso realizar um esforço extra para determinar os intervalos em que essas variáveis são válidas. Caso contrário, pode ocorrer situações não críveis, como um personagem que derrota todos com um único soco. Essa tarefa exige, particularmente, um esforço do game designer para identificar e testar as modificações que mantenham a coerência do jogo.

A desvantagem dessa abordagem é que, embora possa ser efetiva em alguns gêneros (como FPS – *First Person Shooter*), sua aplicação é limitada àqueles em que as manipulações de parâmetros são possíveis. Em um jogo de tabuleiro, por exemplo, onde as peças têm características estáticas e quantidade pré-definida, essa abordagem se torna inviável: não é possível modificar a “força” de um peão no xadrez, nem adicionar ou remover peças dos jogadores. Além disso, a manipulação de parâmetros pode não ser percebida pelos usuários como uma melhoria do entretenimento, como indica testes preliminares realizados com jogadores [Hunicke 2005]. Em suma, essa abordagem pode não ser suficiente para garantir um balanceamento dinâmico.

Com relação aos requisitos especificados anteriormente, a abordagem satisfaz potencialmente tanto o [R/JOG 1.A] quanto o [R/JOG 1.B], uma vez que a adaptação dos parâmetros pode ser feita instantaneamente. O requisito de desempenho [R/IMP 4] também é provavelmente satisfeito, porque não se exige processamentos complexos. Os requisitos de credibilidade [R/JOG 2.A] e [R/JOG 2.B] podem não ser plenamente atendidos, pois é necessário realizar um cuidadoso trabalho de definição das manipulações possíveis, sob pena de ocorrer situações incoerentes como citado anteriormente. Por fim, o requisito de processo [R/IMP 3] fica provavelmente comprometido, em virtude do esforço extra exigido do game designer para a codificação dos parâmetros.

Para melhorar o balanceamento e reduzir essas limitações, a manipulação de parâmetros pode ser combinada com a modificação de comportamento dos NPCs. A seguir, são apresentadas algumas estratégias nessa direção.

## 3.2. Balanceamento Baseado em Scripts Dinâmicos

A implementação tradicional do comportamento dos NPCs é feita por meio de agentes inteligentes. A cada momento, o agente percebe o *estado* atual do ambiente (através de um conjunto de sensores), e executa uma *ação* (através de um conjunto de atuadores). O estado percebido corresponde a todas as informações atuais do ambiente utilizadas na tomada de decisão. A tomada de decisão corresponde ao processo de analisar os dados atuais (junto com possíveis dados históricos mantidos pelo agente) e definir uma ação a ser executada. Após realizar esta ação, o agente volta a perceber o novo estado e tomar uma nova decisão, em um processo contínuo e iterativo [Russel e Norvig 2004].

Os mecanismos de implementação de agentes se diferenciam pela maneira como é realizado o processo de tomada de decisão. A abordagem clássica em jogos consiste em criar scripts de atuação. Scripts são conjuntos de regras de comportamento, em geral na forma **condição-ação**, que especificam a ação que o agente deve tomar em cada situação possível. Um exemplo de regra em um jogo de luta é “*se o oponente estiver perto, deferir um soco*”. Nesse caso, a condição é “oponente estar perto” e a ação associada é “deferir um soco”. Criando várias regras semelhantes a essa, é possível simular satisfatoriamente um comportamento humano.

### 3.2.1. Scripts Dinâmicos

Essa implementação tradicional pode ser estendida através de scripts dinâmicos (*dynamic scripting*) [Spronck *et al.* 2004a], uma técnica de aprendizagem de máquina. Esse método utiliza uma base de regras, construída manualmente durante o desenvolvimento. A cada uma das regras é associado um peso que indica sua qualidade. O conjunto de regras não deve ser equivalente a uma função que mapeia um estado em uma única ação possível: é preciso haver alternativas de escolha.

Nesse caso, como em cada estado mais de uma regra têm sua condição satisfeita, é preciso definir uma política de escolha. A política é baseada no peso das regras: a cada estado percebido, é associada a cada regra satisfeita uma probabilidade de ser escolhida. Essa probabilidade depende do peso de cada uma, de forma que as regras de maior peso têm maiores probabilidades. Essencialmente, essa abordagem é uma simplificação dos sistemas classificadores (*classifiers systems*) [Kovacs 2002], porque não implementa mecanismos para a geração de regras novas, apenas atualiza o peso das regras existentes. A criação de regras pode ser feita com base em algoritmos genéticos, de modo que o agente resultante seja capaz de atuar em situações não previstas no desenvolvimento do jogo [Robert e Guillot 2005].

A Figura 6 ilustra uma implementação básica dessa situação para um jogo de luta. No exemplo, a propriedade “Perto” significa uma distância fixa e pré-definida, como 150 pixels. Quando o agente percebe que a sua distância para o oponente é pequena, 3 regras são satisfeitas e podem ser executadas. Estas regras e seus pesos são definidos durante o desenvolvimento do jogo. Como a primeira regra possui peso zero, sua probabilidade de ser escolhida também é zero. De acordo com os demais pesos, a segunda regra pode ser escolhida com probabilidade 0.67 e a terceira com probabilidade 0.33.

```
SE (distancia = Perto)
ENTÃO ação = Afastar-se
COM PESO 0

SE (distancia = Perto)
ENTÃO ação = Soco-Forte
COM PESO 4

SE (distancia = Perto)
ENTÃO ação = Chute-Rasteiro
COM PESO 2
```

**Figura 6: Exemplo de regras para o balanceamento baseado em scripts**

Embora os pesos iniciais sejam definidos durante o desenvolvimento, eles são atualizados dinamicamente, durante a execução, de acordo com o resultado de cada ação: quando ocorre sucesso (seguindo uma lógica de aprendizagem por reforço), o peso da regra utilizada é aumentado, quando ocorre falha, é diminuído. Dessa forma, eventuais distorções nos pesos são automaticamente corrigidas, pois regras que causam ações eficientes passam a ter maior probabilidade de serem escolhidas.

Esse método também resulta em agentes que conseguem melhorar seu desempenho ao longo do tempo. Embora não sejam criadas regras novas, é possível identificar aquelas mais adequadas a cada oponente específico e aumentar a sua probabilidade de escolha.

### 3.2.2. Balanceamento com Scripts Dinâmicos

O *dynamic scripting* pode ser modificado para suportar balanceamento dinâmico [Spronck *et al.* 2004b]. Há 3 estratégias possíveis identificadas na literatura: adaptação de regras (*high-fitness penalising*), limitação dos pesos (*weight clipping*), e eliminação de regras fortes (*top culling*). A primeira técnica consiste em atualizar os pesos das regras de acordo com sua adaptação ao

usuário: ao invés de recompensar as regras mais eficientes, são recompensadas aquelas mais adequadas a cada jogador. Com esse mecanismo, o agente, após um período de adaptação, passa a escolher não as melhores regras, mas aquelas mais “justas” para cada usuário. Por exemplo, ainda considerando as regras da Figura 6, quando é enfrentado um oponente muito fraco, o peso da primeira regra (afastar-se) aumenta. Dessa forma, o agente executa menos golpes (porque as probabilidades de escolha das regras 2 e 3 são diminuídas) e passa a se afastar do oponente (porque a probabilidade da regra 1 aumenta), de modo que, no geral, seu comportamento seja menos desafiador.

A segunda técnica, *weight clipping*, consiste em limitar o crescimento dos pesos das regras, de acordo com o nível de habilidade do usuário. O objetivo é evitar que regras muito boas sejam muito executadas diante de jogadores iniciantes. Por exemplo, considerando um limite de 4, a regra 2 da Figura 6 não tem seu peso atualizado, mesmo que produza bons resultados. Com isso, as regras se mantêm com uma probabilidade semelhante, induzindo a execução de ações não ótimas periodicamente. A adaptação ao usuário é obtida através da modificação desse limite, de acordo com a dificuldade enfrentada por cada um. Através de uma função de desafio, o nível de dificuldade do jogo é identificado. Quando o jogo está fácil, o limite é diminuído; quando está difícil, é aumentado.

A terceira técnica, *top culling*, consiste em criar limites superiores para o peso das regras que podem ser executadas. Embora não haja restrições quanto ao crescimento dos pesos, as regras que tenham o peso acima desse limite, não são consideradas na tomada de decisão. Por exemplo, no caso da Figura 6 um limite superior de 3 impede a segunda regra de ser escolhida. Nesse caso, seriam consideradas na tomada de decisão apenas a primeira e a terceira regras. O balanceamento do jogo é obtido, semelhantemente à técnica anterior, através da adaptação desse limite a cada usuário. Quando o limite superior é diminuído, as regras muito eficientes não são escolhidas, e sim regras não ótimas. Quando o jogo se torna fácil, o limite é aumentado, permitindo que o agente volte a escolher as melhores regras.

Segundo alguns experimentos realizados, a terceira técnica (*top culling*) se mostrou mais eficiente do que as demais [Spronck *et al.* 2004b]. Enquanto a primeira não consegue se adaptar rapidamente a cada usuário, a segunda pode degradar para uma estratégia de escolha aleatória, pois quando o limite é muito baixo as probabilidades de escolha se tornam equivalentes. A terceira técnica mostrou-se mais apta a lidar com inimigos fracos e apresentou a menor variação de desempenho.

### 3.2.3. Vantagens e Limitações

A abordagem de balanceamento baseado em scripts apresenta duas características positivas claras: a possibilidade de inserção de conhecimento explícito e o alto controle sobre a adaptação ao usuário. A inserção de conhecimento explícito (através da criação das regras e da inicialização de seus pesos) é importante porque muitas vezes os desenvolvedores de jogos conseguem identificar a melhor ação possível a ser executada em um dado momento. Inserir esse conhecimento explicitamente é muito útil para garantir um comportamento satisfatório e evitar ações incoerentes. Além disso, ao prover uma capacidade de adaptação apenas através da modificação dos pesos das regras, essa abordagem consegue impedir que os comportamentos se tornem imprevisíveis, uma vez que todas as ações são baseadas nas regras pré-definidas, ao mesmo tempo em que mantém a possibilidade de exploração das fraquezas específicas de cada oponente. Por fim, ao se basear em scripts de atuação, essa proposta se aproxima da abordagem tradicional (amplamente usada pela indústria), o que facilita sua assimilação pelos desenvolvedores.

Entretanto, algumas limitações também podem ser identificadas. À medida que a complexidade do jogo aumenta, esse método necessita de várias regras de comportamento. Como essas regras são criadas manualmente, em um processo de difícil construção e manutenção, elas se tornam sujeitas a erros [Tozour 2002]. Nesses casos, surgem “buracos” nas regras, isto é, situações não previstas na fase de desenvolvimento, nas quais o agente apresenta uma atuação muito fraca, e que, quando descobertas, podem ser continuamente exploradas pelos usuários. Essa limitação é agravada pela necessidade de inserção de regras adicionais, para que o comportamento também seja adequado a usuários iniciantes. Se todas as regras forem muito boas, o agente pode não conseguir diminuir seu desempenho quando perceber que está atuando em um nível muito difícil. Além disso, embora o processo de adaptação consiga descobrir as melhores regras existentes para cada usuário, não há a criação de regras novas, o que limita a atuação do agente às melhores regras concebidas durante a fase de desenvolvimento. Em suma, inserir conhecimento a priori pode ser uma vantagem em casos simples, mas um problema em casos complexos.

Com relação aos requisitos de balanceamento dinâmico, essa abordagem deve satisfazer o critério de racionalidade [R/JOG 2.B], uma vez que todo o comportamento é determinado por regras pré-definidas. Codificando regras consistentes, haveria uma certa garantia de que não serão executadas ações incoerentes. O requisito de desempenho [R/IMP 4] também seria provavelmente satisfeito, pois é sabido que os algoritmos de escolha e atualização de regras não consomem recursos excessivos. Para cumprir os requisitos de rápida adaptação, é preciso que os parâmetros de modificação dos pesos das regras sejam

corretamente definidos, de forma a evitar variações abruptas no desempenho e manter a velocidade de adaptação. Esse processo influencia diretamente na credibilidade, por meio do requisito de inércia, e juntamente com a necessidade de codificação explícita das regras, pode tornar a abordagem baseada em scripts dinâmicos altamente custosa. Por isso, os requisitos [R/JOG 1.A], [R/JOG 1.B] e [R/JOG 2.A] são, em princípio, apenas parcialmente atendidos e o requisito de desenvolvimento [R/IMP 3] provavelmente não é satisfeito. A próxima seção apresenta uma alternativa para o balanceamento dinâmico que não depende da inserção manual de regras de comportamento.

### 3.3. Balanceamento Baseado em Algoritmos Genéticos

Uma abordagem alternativa para o problema de balanceamento dinâmico é utilizar algum mecanismo de aprendizagem de máquina que não depende da codificação de regras [Langley 1997] para a construção e evolução do comportamento dos agentes. Demasi e Cruz [2002] propõem a utilização de algoritmos genéticos nessa tarefa. Na aprendizagem baseada em algoritmos genéticos, os agentes aprendizes codificam características de comportamento como genes. Por meio da utilização de operações genéticas (como cruzamento e mutação), são constantemente criados novos indivíduos. Aqueles que apresentam maior aptidão (inferido por uma função heurística) para atuar no ambiente são preservados e utilizados como pais na criação de novos indivíduos, de modo que os filhos evoluam a cada nova geração [Eiben e Smith 2003].

#### 3.3.1. Descrição

O método proposto por Demasi e Cruz consiste em preservar os indivíduos que mais se adaptam ao nível do usuário, e não necessariamente os melhores. O método é baseado na Coevolução em Tempo Real, na qual algoritmos genéticos coevolucionários [Wiegand *et al.* 2002] são modificados para lidar com a restrição de tempo dos jogos. Algoritmos coevolucionários analisam a aptidão dos indivíduos por meio do sucesso atingido por toda a população, o que em jogos pode ser facilmente mapeado ao *score* do agente. De maneira geral, o método consiste em evoluir ou manter o atual nível dos agentes quando estes são destruídos ou extrapolam um limite de tempo pré-definido. A decisão de evoluir ou não depende de uma função de desafio.

A evolução é baseada em indivíduos-objetivo, que correspondem a *modelos* que o agente deve alcançar em um determinado instante, servindo como guia para aumentar a velocidade da evolução. Mais especificamente, a evolução do nível de desafio é feita considerando-se o agente objetivo e um indivíduo da população atual como pais para a criação de um novo indivíduo (através de cruzamento); para manter o mesmo nível, a reposição é feita

por meio de simples replicação ou cruzamento entre indivíduos da mesma população. Os indivíduos-objetivo podem ser criados manualmente, usando-se informações do jogo, ou através de simulações off-line, com outros agentes (caso existam) e com humanos.

O balanceamento do jogo é obtido pela utilização de modelos com diferentes aptidões. No começo, enquanto o usuário ainda está aprendendo a jogar, é usado um indivíduo-objetivo com pouca aptidão. À medida que esse modelo intermediário se torna fácil para o usuário, passa-se a considerar o próximo indivíduo-objetivo (com aptidão um pouco maior). Esse processo continua até que o jogador atinja o melhor modelo pré-definido.

Embora os indivíduos-objetivo sejam importantes para garantir a velocidade da aprendizagem, é possível não utilizá-los, o que resulta na evolução convencional de algoritmos genéticos. Nesse caso, é necessário definir uma função de avaliação que determine a aptidão de cada agente. A partir dela, pode-se usar os melhores indivíduos de cada subespécie como pais para a geração de novos indivíduos. Por fim, também é possível realizar uma abordagem híbrida, com uma primeira fase usando indivíduos-objetivo e uma segunda de evolução livre. Essa abordagem possibilita a definição prévia de alguns modelos (o que facilita a obtenção de um nível mínimo de qualidade), mas mantém a capacidade de aprendizagem ao possibilitar que o agente evolua livremente, sem necessitar de padrões pré-definidos.

### 3.3.2. Vantagens e Limitações

A abordagem genética é uma idéia inovadora que pode resolver o problema de balanceamento com um baixo esforço de desenvolvimento, na medida que não requer a codificação de extensas bases de regras de comportamento. Além disso, a possibilidade de evolução livre (sem a utilização dos indivíduos-objetivo) permite que sejam descobertas estratégias de atuação adequadas a cada oponente, usufruindo suas fraquezas e evitando suas aptidões.

Por outro lado, algumas limitações podem ser identificadas. Um dos problemas dessa abordagem é a criação de indivíduos indesejados. O cruzamento genético pode gerar indivíduos melhores do que o objetivo intermediário desejado dependendo da forma como os genes são definidos e interpretados. Esse problema acaba inserindo ruído na evolução dos agentes, o que pode ser percebido pelas variações entre oponentes muito fracos e muito fortes. Além disso, a utilização de modelos pré-definidos (os indivíduos-objetivo) limita a evolução eficiente dos agentes a esses modelos, prejudicando a aplicação do método contra usuários muito bons ou com comportamento incomum. Como esses usuários não possuem um modelo para agilizar a adaptação dos agentes, pode-se demorar muito tempo até que seus níveis sejam alcançados. Por fim, embora os agentes genéticos sejam capazes de evoluir, não há um mecanismo de regressão explícita na proposta de Demasi e Cruz. A limitação de não



permitir regressões no comportamento induz os agentes a começarem sempre do nível mais fácil. Enquanto essa abordagem pode ser satisfatória para usuários iniciantes, pode ser frustrante para usuários mais experientes, que precisarão esperar muito tempo pela evolução dos agentes.

Com relação aos requisitos anteriores, a abordagem genética tem o potencial de satisfazer o requisito de rápida adaptação inicial [R/JOG 1.A], uma vez que os modelos de evolução podem ser suficientes para lidar com as primeiras interações dos usuários. Entretanto, a eficiência do balanceamento após os momentos iniciais é profundamente dependente da codificação dos genes. Como a evolução genética não tem garantia de que cada geração é melhor do que a anterior, pode ser preciso realizar várias tentativas até que o agente obtenha um comportamento adequado, o que dificulta o acompanhamento do desenvolvimento dos usuários ao longo do tempo. Essa limitação torna-se ainda mais crítica quando não há modelos para guiar a aprendizagem, o que torna todo o processo ainda mais imprevisível. Com relação aos critérios de credibilidade, essa abordagem pode também apresentar limitações, porque podem ser geradas estratégias de atuação incoerentes pelas operações genéticas, caso não seja explicitamente limitada a execução de ações inconsistentes. Dessa forma, os requisitos [R/JOG 1.B], [R/JOG 2.A], e [R/JOG 2.B] podem não ser atendidos por essa abordagem. O requisito de desenvolvimento [R/IMP 3] é, em princípio, parcialmente satisfeito, pois embora não seja necessária a codificação de regras ou de complexos algoritmos, é preciso realizar um cuidadoso trabalho de parametrização das funções de aprendizagem.

### 3.4. Conclusões

Embora o problema de balanceamento de jogos possa ser resolvido por meio de uma abordagem estática, a utilização de métodos que possibilitem um balanceamento dinâmico apresenta-se como uma alternativa promissora. Dentre as possibilidades de balanceamento dinâmico, os métodos baseados em aprendizagem destacam-se pela sua inerente capacidade de adaptação ao usuário. Este capítulo apresentou 3 propostas de balanceamento dinâmico: manipulação do ambiente, adaptação de scripts dinâmicos e aprendizagem genética. Esses métodos não são alternativas disjuntas, mas sim complementares, de modo que um mesmo jogo pode utilizar mais de uma técnica para garantir um balanceamento adequado.

## 4. Bases da Abordagem Proposta

Este capítulo apresenta o referencial teórico utilizado na criação da abordagem proposta neste trabalho. Primeiramente, são apresentados os conceitos de competência e desempenho, e como eles podem ser utilizados em um princípio geral de balanceamento de jogos. Em seguida, são detalhados os conceitos de aprendizagem por reforço, uma das técnicas possíveis de implementação da abordagem proposta.

### 4.1. Competência Vs. Desempenho

A distinção entre competência e desempenho (*competence vs. performance*) começou a ser usada nos estudos de lingüística, a partir dos trabalhos de Ferdinand de Saussure e Noam Chomsky. As teorias resultantes afetaram os estudos em campos como teoria da informação, cibernética e inteligência artificial.

#### 4.1.1. As Idéias de Saussure

Ferdinand de Saussure (1857 – 1913) adotou uma abordagem **estruturalista** ao estudo da lingüística [Saussure 1986]<sup>1</sup>. Como estruturalista, Saussure estava interessado nos sistemas e estruturas das linguagens, e não em suas unidades de comunicação. Seus estudos buscavam entender as *inter-relações* entre as diversas unidades de uma língua e as *regras* que determinam como essas unidades podem ser combinadas. Com isso, suas idéias podem ser aplicadas em qualquer tipo de linguagem: inglês, francês, ou até linguagens de programação.

Para delimitar esse campo de estudo, Saussure separou o estudo da linguagem em **Langue** (a *língua*, que representa o sistema formal da linguagem que governa os eventos da fala) e **Parole** (a *fala* ou *palavra* propriamente dita, que representa os eventos de uma fala). Enquanto os sistemas de linguagem (*Langue*) são influenciados por dimensões sociais e coletivas, as unidades que a compõe (*Parole*) são influenciadas pelas características individuais do falante (ou escritor).

Uma língua (seus sistemas e estruturas) surge a partir de convenções sociais. Segundo as idéias de Saussure, a língua representa o conhecimento comunitário sobre as possibilidades de comunicação, enquanto a fala (ou escrita) é a aplicação desse conhecimento em uma situação específica. Por exemplo, uma criança que lê um livro de Língua Portuguesa está aprendendo o sistema formal de sua língua (*Langue*), mas quando ouve um discurso de

---

<sup>1</sup> Tradução do livro “Cours de Linguistique Générale (published by C. Bally and A. Sechehaye in collaboration with A. Riedlinger)”, Lausanne and Paris: Payot, 1916.

seu pai está apenas observando um evento informal de utilização da língua (*Parole*). Distinguindo a dimensão coletiva (genérica) da dimensão individual (específica), Saussure pôde focar seus estudos nos aspectos universais da lingüística, legando contribuições duradouras para diversas áreas das ciências humanas.

#### 4.1.2. A Abordagem de Chomsky

Alguns anos após os estudos de Saussure, Noam Chomsky [Chomsky 1965] voltou ao problema de delimitação das áreas de estudo da lingüística. Chomsky identificou que as teorias lingüísticas tradicionais (até então) se referiam a situações com um conjunto *falante-ouvinte* ideal. Nessas situações, a comunicação é feita por uma comunidade lingüística completamente homogênea, que conhece sua língua perfeitamente, e não é afetada por condições como distrações, mudanças de interesses, limitações de memória, e erros de expressão. Nos casos ideais, a atuação de um conjunto falante-ouvinte é uma consequência direta de seus conhecimentos da língua.

Entretanto, em situações reais, diversos elementos causam impacto na comunicação, como os erros gramaticais e os falsos começos de expressão. Isso significa que a aplicação prática do conhecimento pode ser influenciada por fatores como stress, emoções, e limitações de tempo e memória. Em função disso, Chomsky criou uma distinção fundamental entre **competência** (o conhecimento da língua) e **desempenho** (o efetivo uso da língua em situações concretas). A primeira se refere a uma propriedade abstrata, enquanto a segunda se refere a um evento:

*"(...) We thus make a fundamental distinction between competence (the speaker-hearer's knowledge of his language) and performance (the actual use of language in concrete situations)." (Noam Chomsky, Aspects of the Theory of Syntax, 1965, pp. 4)*

Por exemplo, durante uma apresentação de mestrado, o aluno pode não conseguir expressar claramente seu conhecimento, uma vez que seu desempenho é afetado pelo stress da avaliação, pela ansiedade do resultado, e pela limitação do tempo de apresentação. Esses fatores podem fazer com que o aluno gagueje, omita algum detalhe do trabalho, ou use termos ambíguos. O desempenho, portanto, não expressa diretamente a sua competência.

Como apenas o desempenho é claramente percebido pelas pessoas, Chomsky voltou suas análises para os discursos **aceitáveis** (*acceptables*), em que as expressões dos participantes são naturais e compreensíveis, independentemente de suas características gramaticais. O conceito de *aceitável* está relacionado ao estudo do desempenho, enquanto a *gramaticalidade* se relaciona com o estudo da competência. Essa separação se mostrou

importante porque não necessariamente uma expressão aceitável está gramaticalmente correta, assim como não necessariamente uma expressão gramaticalmente correta é aceitável.

Na prática, o conceito de aceitabilidade está relacionado a gradações, e envolve diferentes dimensões, umas das quais é a gramaticalidade. Desta forma, é possível identificar expressões mais ou menos aceitáveis. Expressões gramaticalmente corretas têm mais chance de serem aceitáveis, mas outros fatores podem acabar por torná-las inaceitáveis.

Embora Chomsky tenha sido o primeiro a utilizar os termos Competência e Desempenho, suas idéias encontram um paralelo nas definições de *Langue* e *Parole* de Saussure, como o próprio autor reconhece [Chomsky 1965]. Ambos os casos fazem uma distinção entre o **conhecimento** (Competência ou *Langue*) e a **aplicação** desse conhecimento (Desempenho ou *Parole*).

### 4.1.3. Competência e Desempenho em Jogos

As idéias expostas anteriormente podem ser aplicadas ao balanceamento de jogos combinando-as com o conceito de função de desafio apresentado anteriormente. Nossa proposta é criar agentes capazes de adquirir conhecimentos genéricos (possivelmente sobre mais de um jogador), e portanto com elevada competência, mas que adaptem seu desempenho a cada oponente enfrentado.

Como o desempenho não é uma expressão direta da competência, não precisamos restringir esta última. A conveniência dessa distinção é que a competência pode ser continuamente evoluída, independentemente das habilidades do usuário, e apenas o desempenho ser limitado. Mais especificamente, utilizamos agentes que aprendem com jogadores humanos ou virtuais (outros agentes) diferentes estratégias de comportamento. A cada oponente enfrentado, a função de desafio é usada para definir que estratégia utilizar: caso o jogo esteja muito fácil, utiliza-se a melhor estratégia aprendida; caso esteja muito difícil, utiliza-se alguma estratégia sub-ótima. O termo “sub-ótima” aqui utilizado se refere à qualidade de uma estratégia (ou de uma ação) do ponto de vista da aprendizagem: são aquelas estratégias que não utilizam todo o potencial do agente.

A competência do agente pode ser continuamente evoluída por meio de algum mecanismo de aprendizagem, que descubra, durante a interação com cada usuário, novas estratégias eficientes. Essas estratégias podem nunca ser utilizadas, se o desempenho com uma alternativa sub-ótima for suficiente, mas mesmo assim a competência sempre é evoluída.

A abordagem proposta pode ser implementada de diferentes maneiras. Em todas elas, é preciso definir: (1) o mecanismo de aquisição e desenvolvimento da competência e (2) o mecanismo de adaptação do desempenho.

Uma possibilidade de implementação é utilizar regras de comportamento. Nesse caso, a competência do agente é explicitamente definida durante seu desenvolvimento. Associando a cada regra um peso que indique sua qualidade, podemos adaptar o desempenho escolhendo as regras mais adequadas a cada usuário: regras de pesos altos são executadas quando o oponente for habilidoso, e regras de pesos menores são executadas caso contrário.

De fato, a melhor proposta de balanceamento (*top culling*) de Spronck [Spronck *et al.* 2004b] utiliza a mesma abordagem aqui apresentada, embora a distinção entre competência e desempenho não seja explicitada no seu trabalho e os dois projetos tenham sido desenvolvidos em paralelo<sup>2</sup>. Nessa abordagem, a competência inicial do agente é sua base de regras (especificada manualmente). Para cada oponente enfrentado, os pesos das regras são modificados, de modo a evoluir a competência (scripts dinâmicos). A adaptação do desempenho é realizada através do mecanismo de eliminação de regras fortes (*top culling*), que não considera regras muito boas para usuários com poucas habilidades. Esse mecanismo difere das outras possibilidades de adaptação (*high-fitness penalising* e *weight clipping*) porque não limita a competência de acordo com cada usuário: apenas o desempenho pode ser restringido. Em suma, não é porque o agente enfrenta um adversário mais fraco que ele deixa necessariamente de aprender; a aprendizagem ocorre sempre, apenas o desempenho é limitado. Essa distinção é fundamental para entender porque o primeiro mecanismo (*top culling*) foi mais eficiente do que os demais nos experimentos realizados [Spronck *et al.* 2004b].

Um dos problemas com essa implementação baseada em regras é a necessidade de inserção explícita de conhecimento (através da codificação das regras). Embora em alguns problemas seja possível identificar facilmente regras de comportamento eficientes, à medida que a complexidade dos sistemas cresce esse processo se torna trabalhoso e sujeito a erros. Na realidade, é preciso codificar não apenas regras eficientes, mas também regras sub-ótimas, que serão executadas pelos usuários menos habilidosos. Além disso, uma vez definidas as regras, o desempenho se limita às melhores regras pré-definidas.

Uma outra possibilidade de implementação da abordagem baseada em competência e desempenho é utilizar algum mecanismo de aprendizagem de máquina que não dependa da codificação explícita de regras. A utilização de aprendizagem em jogos, sistemas tipicamente complexos, possui desafios importantes a serem superados. O uso de algum método de aprendizagem supervisionada exige a presença de um professor, ou seja, uma pessoa ou sistema capaz de informar, para cada situação possível, a melhor ação a ser executada. Infelizmente, em muitos jogos não existe esse professor, o que dificulta a aplicação desse

---

<sup>2</sup> Em encontro pessoal com Spronck no IJCAI'05 (2005 *International Joint Conference on Artificial Intelligence*), ele nos disse: “Só depois que li o artigo de vocês é que entendi porque o *top culling* funciona melhor”.

paradigma de aprendizagem. Os métodos de aprendizagem não-supervisionada, por sua vez, embora não exijam conhecimento prévio do domínio, podem requerer muito tempo até a descoberta de estratégias de atuação satisfatórias. Uma alternativa intermediária é a utilização de aprendizagem por reforço, que exige apenas um crítico para guiar a aprendizagem. Esse crítico não precisa informar os resultados esperados, mas apenas indicar se as ações executadas são boas ou ruins. Aprendizagem por reforço já foi amplamente estudada na literatura, é aplicada com sucesso em várias situações práticas (inclusive jogos), mas ainda não tinha sido utilizada no problema de balanceamento de jogos.

Por isso, no presente trabalho, utilizamos aprendizagem por reforço como mecanismo de aquisição e evolução da competência. Os mesmos elementos da aprendizagem também são usados para a adaptação do desempenho, através de uma modificação na estratégia de atuação tradicional. A próxima seção apresenta conceitos de aprendizagem por reforço que serão utilizados na construção de nossa proposta.

## 4.2. Aprendizagem por Reforço (AR)

Aprendizagem por reforço (AR) não é caracterizada como uma técnica de aprendizagem, mas sim com um problema de aprendizagem. O problema da aprendizagem por reforço é definido como “*aprender o que fazer (mapear situações em ações), de modo a maximizar um sinal numérico de recompensa*” [Sutton e Barto 1998]. Existem diferentes técnicas na literatura que resolvem esse problema. No nosso trabalho, é usada como base a técnica *Q-Learning* [Watkins e Dayan 1992], que possui algumas características úteis para a aplicação em jogos.

### 4.2.1. O Problema de Aprendizagem por Reforço

No problema de aprendizagem por reforço, tem-se um agente que seqüencialmente toma decisões em um ambiente. A cada instante, o agente percebe o estado atual  $s$ , de um conjunto  $S$ , e executa uma ação  $a$ , de um conjunto  $A$ , sendo levado a um novo estado  $s' \in S$  com uma probabilidade  $P(s,a,s')$ . A probabilidade de transição  $P$  representa a dinâmica do ambiente, e indica a probabilidade do agente atingir o estado  $s'$  após executar a ação  $a$  no estado  $s$ . Para cada tupla  $\langle s,a,s' \rangle$ , o agente recebe um sinal de reforço  $R(s,a,s') \in \mathfrak{R}$ . O reforço  $R(s,a,s')$  indica quão bom (ou ruim) foi executar a ação  $a$  no estado  $s$ , tendo como consequência o novo estado  $s'$ . Implicitamente, o sinal de reforço  $R$  indica o objetivo do agente, uma vez que é o único indicador do comportamento desejado.

Esse *framework* do problema de aprendizagem por reforço é tipicamente definido em termos de um *Processo de Decisão de Markov* (PDM). Um PDM é definido por uma tupla  $\langle S, A, P, R \rangle$  dos elementos especificados anteriormente. O conjunto  $S$  especifica os possíveis

estados percebidos pelo agente, o conjunto  $\mathbf{A}$  especifica as ações que o agente pode executar no ambiente, a distribuição de probabilidade  $\mathbf{P}$  (ou função de transição de estados) indica a dinâmica do ambiente, e a função  $\mathbf{R}$  indica o reforço recebido pelo agente em cada situação.

As funções  $\mathbf{P}$  e  $\mathbf{R}$  devem satisfazer a *propriedade de Markov*: o valor da função aplicada a uma determinada tupla  $\langle s, a, s' \rangle$  depende apenas dos valores da tupla, e não da sequência de estados ou ações precedentes do agente. Em outras palavras, cada elemento do conjunto de estados  $\mathbf{S}$  deve incorporar percepções atuais e passadas, de tal maneira que toda informação relevante para a tomada de decisão esteja contida nele.

Neste formalismo, o agente atua de acordo com uma política de ações  $\pi(s, a) \rightarrow \mathfrak{R}$ , que representa a probabilidade dele escolher a ação  $a \in \mathbf{A}$  quando está no estado  $s \in \mathbf{S}$ . O objetivo é maximizar um critério de desempenho a longo prazo, chamado de *retorno*, que representa o valor esperado das recompensas futuras. O agente tenta então aprender a política de ações ótima  $\pi^*(s, a)$  que maximiza o retorno esperado.

Outro conceito nesse formalismo é a função de ação-valor,  $Q^\pi(s, a)$ , definida como o retorno esperado quando o agente, a partir do estado  $s$ , executa a ação  $a$ , e segue a política  $\pi$  daí em diante. Se o agente aprender a função ação-valor ótima,  $Q^*(s, a)$ , a política ótima pode ser derivada facilmente: para cada estado  $s$ , a melhor ação  $a$  é aquela que maximiza a função  $Q^*(s, a)$ .

### 4.2.2. *Q-Learning*

*Q-Learning* [Watkins e Dayan 1992] é um algoritmo bastante utilizado para a resolução do problema de aprendizagem por reforço. O algoritmo consiste em, iterativamente, computar os valores da função ação-valor usando a regra de atualização:

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha [r(s, a, s') + \gamma \cdot \max_{a'} Q(s', a')]$$

Essa regra depende apenas dos valores estimados da função ação-valor  $Q$  para o estado atual e para o próximo estado, e do sinal de reforço  $r$ . Os demais termos,  $\alpha$  e  $\gamma$ , são parâmetros que representam, respectivamente, a taxa de aprendizagem ( $0 < \alpha \leq 1$ ) e o fator de desconto ( $0 \leq \gamma \leq 1$ ). O fator de desconto  $\gamma$  indica a importância relativa das recompensas das ações futuras sobre recompensas imediatas: valores pequenos indicam preferência por recompensas pequenas e imediatas, enquanto valores altos favorecem recompensas maiores após muitas iterações.

Esse algoritmo é garantido de convergir para a função ação-valor ótima, supondo que cada par estado-ação seja visitado infinitamente. Vale ressaltar que não é utilizado nenhum conhecimento da dinâmica do ambiente, isto é, não é necessário conhecer a função de

transição de estados  $\mathbf{P}$ . Como a regra de atualização é executada repetidamente, a probabilidade de ser executada com uma tupla  $(s, a, s')$  é exatamente o valor de  $\mathbf{P}(s, a, s')$ , o que dispensa o conhecimento da função  $\mathbf{P}$  mantendo a garantia de convergência [Watkins e Dayan 1992]. O algoritmo pode ser facilmente implementado usando uma matriz bidimensional representando a função ação-valor. Em uma dimensão são colocados os estados e em outra as ações possíveis, de modo que cada célula represente um par estado-ação. Os valores da matriz são iniciados com zero e atualizados de acordo com a regra anterior.

A regra de atualização especificada anteriormente é suficiente para ser aplicada a um Processo de Decisão de Markov (PDM). Entretanto, PDMs convencionais não são capazes de lidar com ações com extensões temporais, isto é, ações cuja execução requer intervalos de tempo variáveis. Dessa forma, um PDM não consegue diferenciar se é melhor executar uma ação que demora 10 segundos ou 10 ações que duram 1 segundo cada. Como a duração não afeta o retorno obtido por uma ação, supõe-se que todas elas têm duração igual. Essa limitação pode fazer com que um agente aprendiz prefira executar uma ação de 10 segundos que produza um retorno de 10 do que executar 10 ações de 1 segundo que produzam um retorno de 2 cada (totalizando um retorno de 20 no mesmo intervalo de tempo).

Para aplicações cujas ações possuem intervalos de tempos diferentes (como jogos, em que cada ação pode possuir uma duração distinta), deve-se utilizar um Processo de Decisão Semi-Markoviano (PDSM). PDSMs são uma versão mais geral do formalismo de PDM, no qual as ações podem ter durações variáveis. Por isso, todos os fundamentos dos PDMs, especificados anteriormente, continuam válidos nos PDSMs, que apenas insere a informação de tempo nas suas propriedades. Para o algoritmo *Q-Learning*, a regra de atualização mais genérica [Watkins e Dayan 1992] é mostrada a seguir:

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha [r(s, a, s') + \gamma^t \cdot \max_{a'} Q(s', a')]$$

Essa regra considera a variável  $t$  (o tempo de duração da ação  $a$ ) no fator de desconto  $\gamma$ . Como o fator de desconto é sempre menor (ou igual) a 1, quanto maior for o valor de  $t$ , maior será o valor descontado do retorno esperado das ações futuras, punindo as ações mais demoradas. Para o caso em que todas as ações tenham a mesma duração unitária, essa regra equivale à regra para PDMs. Por fim, vale ressaltar que as provas de convergência do algoritmo *Q-Learning* também são aplicáveis à sua versão genérica para PDSMs.

Uma vez definida a função ação-valor  $\mathbf{Q}(s, a)$ , a política de ações do agente consiste em, dado um estado  $s$ , executar a ação que possui o maior valor naquele estado, de acordo com a função ação-valor. Essa política é ótima para o caso de o agente já ter aprendido a função ótima  $\mathbf{Q}^*(s, a)$ . Entretanto, caso o agente ainda esteja aprendendo, é preciso garantir



que todos os pares estado-ação sejam visitados. Para garantir isso, é necessário explorar situações desconhecidas, se arriscando a receber punições mesmo quando já se conhece uma estratégia boa. Esse fator de exploração evita que o agente fique preso em uma estratégia sub-ótima, forçando-o a conhecer sempre novas estratégias. A questão de balancear a exploração de novos caminhos com o usufruto do conhecimento já aprendido é denominada de **dilema da exploração versus usufruto** (*exploration versus exploitation dilemma*) [Sutton e Barto 1998].

Diversas soluções já foram propostas para esse dilema, sendo a mais simples delas o método  **$\epsilon$ -greedy**. O  $\epsilon$ -greedy consiste em utilizar um parâmetro  $\epsilon$  que indica a frequência de ações de exploração a serem executadas pelo agente. Nesse método, a cada escolha de ação o agente executa uma ação aleatória com probabilidade  $\epsilon$  e executa a melhor ação (definida pela função ação valor) com probabilidade  $(1 - \epsilon)$ . Utilizando-se valores altos de  $\epsilon$  obtém-se agentes que aprendem mais rapidamente (uma vez que exploram mais), porém no longo prazo apresentam desempenho limitado (porque sempre estão executando ações aleatórias). Com valores pequenos, o tempo de aprendizagem é maior, mas o desempenho no longo prazo é melhor.

Embora a abordagem  $\epsilon$ -greedy seja bastante popular, uma de suas grandes limitações é escolher indistintamente uma ação nos passos de exploração. Isso significa que o agente tanto pode escolher uma ação próxima da melhor possível, quanto escolher a pior de todas. Em ambientes nos quais a pior ação é muito ruim (por exemplo, por causar a morte do agente), esse método não é satisfatório. A abordagem alternativa é criar probabilidades de escolha com base na função ação-valor. Esse mecanismo de escolha de ações é denominado de **softmax**.

No método softmax, o agente sempre executa uma ação  $a$  no estado  $s$  com uma probabilidade  $P$  que depende diretamente do valor de  $Q(s,a)$ . Como a melhor ação possui o maior valor na função, sua probabilidade de escolha é a maior. Entretanto, todas as outras ações ainda podem ser escolhidas, embora com uma probabilidade menor. Para as piores ações possíveis, essa probabilidade é muito baixa, o que dificulta sua ocorrência como ação de exploração. Um método softmax bastante comum utiliza a distribuição de probabilidade de Boltzmann [Sutton e Barto 1998], em que uma ação  $a$  é escolhida com a probabilidade:

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^n e^{Q(b)/\tau}}$$

Nessa função,  $Q$  é a função ação-valor,  $n$  é o número de ações possíveis no estado atual e  $\tau$  é um parâmetro positivo denominado *temperatura*. Altas temperaturas resultam em probabilidades próximas, enquanto baixas temperaturas aumentam a distância de ações com

valores diferentes. No limite em que  $\tau \rightarrow 0$ , o softmax equivale ao método guloso, em que apenas a melhor ação é escolhida, com probabilidade 1.

O algoritmo *Q-Learning* é um método de aprendizagem *off-policy* [Sutton e Barto 1998]. Isso significa que a aprendizagem da função ação-valor ótima independe da política de ações seguida. A política de ações pode influenciar na velocidade da aprendizagem, mas a única condição que precisa ser satisfeita para garantir a convergência do algoritmo é que todos os pares estado-ação sejam continuamente atualizados. Essa condição pode ser facilmente satisfeita utilizando-se algum método de exploração (como  $\epsilon$ -greedy ou softmax) combinado com qualquer outro método de usufruto. Essa propriedade se verifica porque a regra de atualização do *Q-Learning* considera a execução de uma política ótima (representada pelo termo  $\max_a Q(s', a')$ , ou seja, escolher sempre a melhor ação). Por isso, mesmo que o agente utilize uma política diferente, a aprendizagem da função ação-valor será feita de acordo com a política ótima.

### 4.2.3. Aprendizagem por Reforço em Jogos

Algumas características da aprendizagem por reforço a tornam bastante atraente para ser utilizada em aplicações complexas como jogos. Primeiramente, ao contrário de outras formas de aprendizagem de máquina, AR não requer exemplos prévios de comportamentos desejados, sendo capaz de encontrar estratégias ótimas (ou pelo menos próximo de ótimas) apenas através de tentativa e erro. Essa característica reduz o esforço de desenvolvimento necessário para a construção da inteligência artificial de jogos, uma vez que dispensa a codificação de longos scripts de comportamento para os agentes. Além disso, pode-se usar uma etapa off-line, como um pré-processamento durante o desenvolvimento, para acumular um conhecimento mínimo na versão inicial do jogo, juntamente com uma etapa on-line para continuamente melhorar o aprendizado após o lançamento [Manslow 2003].

As grandes limitações da aplicação de AR em jogos são a velocidade da aprendizagem e a dificuldade em garantir que não sejam executadas ações incoerentes. Dependendo da codificação dos estados, ações e sinal de reforço, o tempo necessário para aprender uma estratégia boa pode ser inviável para ser utilizada contra um jogador humano. O tempo de aprendizagem pode ser diminuído pela realização de um treinamento off-line. Entretanto, os melhores treinadores são os próprios jogadores humanos, o que torna a aprendizagem off-line bastante custosa (demandando muito tempo). Além disso, como todo o processo de tomada de decisão é realizado automaticamente, de acordo com cada usuário, torna-se bastante difícil prever os resultados da aprendizagem através dos testes

convencionais. Dessa forma, na versão final do jogo ainda podem ser executadas ações inconsistentes, que prejudiquem a sua qualidade.

Para atenuar esses problemas, é necessário realizar uma cuidadosa codificação dos espaços de estado e ações, do sinal de reforço e dos parâmetros da aprendizagem. Por exemplo, eliminando estados irrelevantes dispensa-se o agente de tentar aprender algo em situações que não serão utilizadas. Outra possibilidade é a inserção explícita de conhecimento que limite as ações disponíveis, eliminando aquelas claramente ineficientes do processo de aprendizagem. Em geral, quanto mais estados e ações codificadas, mais pares estado-ação o agente precisa visitar para aprender a se comportar no ambiente.

Mesmo com essas limitações, aprendizagem por reforço tem sido aplicada com sucesso em jogos de tabuleiro, como Damas [Samuel 1967], Go [Abramson e Wechsler 2001], e Gamão [Tesauro 1994]. Neste último, uma das primeiras aplicações práticas de sucesso de AR, o agente desenvolvido conseguiu atingir o nível dos melhores jogadores mundiais humanos, além de descobrir novas estratégias ótimas não percebidas pelos campeões mundiais. AR também tem sido aplicada com sucesso em outros domínios, como a RoboCup (competição de futebol de robôs) [Merke e Riedmiller 2001], sendo utilizada pelos principais campeões do torneio.

Em relação a jogos de ação, a Microsoft Research UK possui um trabalho de aplicação de AR a um jogo comercial (o *Tao Feng: First of the Lótus*, do console Xbox) [Graepel *et al.* 2004]. Neste trabalho, os autores utilizam o algoritmo SARSA [Sutton e Barto 1998], semelhante ao *Q-Learning*, para criar agentes aprendizes em um jogo de luta 3D. A utilização do SARSA foi necessária porque a arquitetura do jogo limitava a percepção de estados e ações para o agente, exigindo um método de aprendizagem *on-policy*, que estima o valor da mesma política utilizada pelo agente (ao contrário dos métodos *off-policy*, nos quais a política aprendida não depende daquela efetivamente utilizada). O trabalho mostra que agentes que implementam aprendizagem por reforço efetivamente conseguem aprender a lutar, partindo de nenhum conhecimento prévio do ambiente. Além disso, é mostrado que AR é capaz de criar diferentes personalidades de agentes apenas através da modificação da função de reforço. Por exemplo, uma função que dá maior peso aos pontos de vida retirados do oponente pode resultar em agentes mais agressivos, enquanto uma função que pune tanto os golpes deferidos quanto os sofridos pode resultar em agentes mais “pacifistas”. Essa característica é positiva porque dispensa o esforço de codificação explícita do comportamento de cada agente que se precise implementar em um jogo.

O trabalho aqui apresentado, entretanto, diferencia-se dessas outras aplicações em um aspecto básico. Enquanto os demais trabalhos estão interessados basicamente em criar

agentes computacionais que derrotem o mais eficientemente possível quaisquer oponentes (comportamento ótimo), nosso objetivo é criar agentes capazes de desafiar seus oponentes adequadamente, sejam eles habilidosos ou não [Andrade *et al.* 2004].

### 4.3. Conclusões

Esse capítulo apresentou as bases de uma proposta original de balanceamento dinâmico de jogos. Nossa abordagem consiste em, a partir da distinção entre competência e desempenho, criar agentes que melhoram continuamente sua competência, mas que utilizam o conhecimento aprendido da forma mais adequada a cada usuário. O próximo capítulo apresenta como os conceitos de aprendizagem por reforço podem ser utilizados para implementar essa proposta [Andrade *et al.* 2005a].

## 5. Balanceamento Baseado em Aprendizagem por Reforço

Este capítulo descreve uma alternativa original para o problema de balanceamento dinâmico de jogos utilizando aprendizagem por reforço. Primeiramente, analisamos como AR pode ser aplicada diretamente ao problema, através da escolha cuidadosa do sinal de reforço. A idéia é direcionar a aprendizagem para o balanceamento, e não para a otimização do desempenho. Entretanto, diante das limitações dessa solução, desenvolvemos uma política de atuação original para obter um método eficiente de balanceamento dinâmico baseado em aprendizagem por reforço [Andrade *et al.* 2005d].

A abordagem desenvolvida consiste em dividir o problema em duas dimensões: competência (criar agentes NPCs com potencial para derrotar qualquer oponente) e desempenho (fazer os agentes agirem no mesmo nível de cada oponente). Para evoluir a competência, a inteligência dos agentes é implementada como uma tarefa de aprendizagem por reforço, e são realizados treinamentos prévios (off-line) para a aquisição de conhecimento inicial. Para adaptar o desempenho, a política de escolha de ações tradicional é modificada de forma que os agentes escolham, a cada momento, a ação mais adequada a cada perfil de usuário, e não necessariamente a melhor ação aprendida. Essa distinção é uma forma inovadora de lidar com uma das principais dificuldades da aplicação de AR em jogos: o tempo de aprendizagem, que é tipicamente mais lento do que a evolução de desempenho exigida dos agentes aprendizes.

### 5.1. Balanceamento com AR Tradicional

O balanceamento dinâmico pode ser abordado diretamente com aprendizagem por reforço, através da escolha correta do sinal de reforço [Andrade *et al.* 2005b]. A idéia é recompensar o agente aprendiz quando ele atuar no mesmo nível do usuário (reforço positivo) e puni-lo quando estiver muito pior ou muito melhor do que o mesmo (reforço negativo). Essa abordagem tem a vantagem de usar, para o balanceamento, o mesmo *framework* da aprendizagem, sem a necessidade de modificação nos algoritmos tradicionais de AR.

Usando o conceito de função de desafio apresentado anteriormente, essa abordagem pode ser resumida como a seguir. É definido um valor para o sinal de reforço (inicialmente, zero), e a cada interação do agente com o ambiente, esse sinal  $I_t$  é informado. Periodicamente, o agente usa a função de desafio (expressa como um mapeamento do estado

atual no nível de dificuldade do jogo) para inferir o nível de dificuldade percebido pelo usuário. Caso esteja fácil ou difícil, o valor do sinal de reforço é atualizado para um valor negativo (por exemplo, -1). Caso contrário, o sinal é atualizado para um valor positivo (+1). Esse valor passa ser informado ao agente a cada ação realizada, sendo continuamente atualizado de acordo com os períodos de avaliação. Dessa forma, o agente é recompensado por atuar no mesmo nível do usuário e punido caso contrário.

A grande vantagem dessa abordagem é que o modelo de adaptação ao usuário do agente é o mesmo modelo da aprendizagem. Com isso, há um grande indicativo de que o agente atuará no mesmo nível do oponente após um tempo suficiente de treinamento, supondo que cada estado seja visitado freqüentemente. Essa última suposição pode ser facilmente satisfeita com a utilização de algum método de exploração, como  $\epsilon$ -greedy ou softmax.

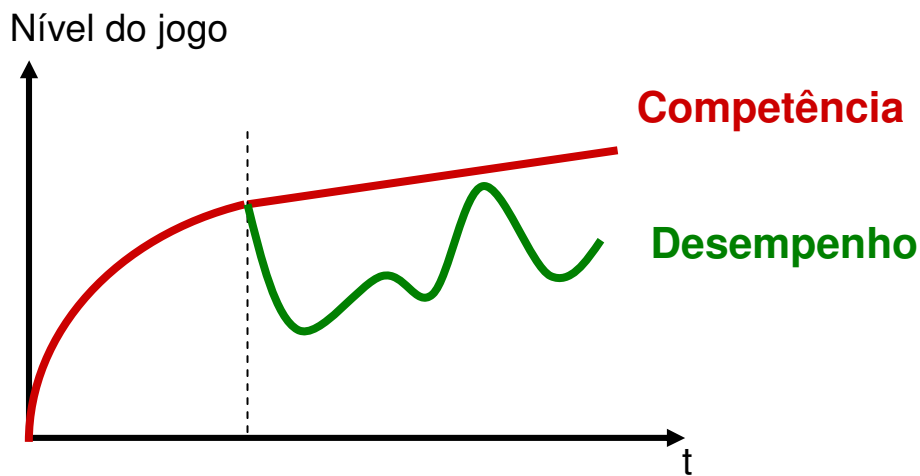
Entretanto, a aplicação prática dessa solução possui algumas limitações críticas. Em primeiro lugar, o agente não é capaz de atuar no mesmo nível do usuário logo no início do jogo, uma vez que precisa aprender a atuar contra cada jogador. Essa limitação fica evidente contra usuários experientes, que precisam esperar muito tempo até que o agente aprenda estratégias boas de atuação. Em segundo lugar, é muito difícil acompanhar o desenvolvimento do usuário, uma vez que o processo de aprendizagem de máquina é tipicamente mais lento do que a aprendizagem humana. Por fim, essa abordagem pode resultar em agentes que executam ações inconsistentes e irracionais, uma vez que a aprendizagem não armazena conhecimento sobre a qualidade das ações. Em um jogo de luta, por exemplo, o agente pode ser induzido a atacar duramente o oponente e depois permanecer parado e permitir o contra-ataque, de modo que o desempenho geral mantenha-se balanceado.

Em resumo, o balanceamento com AR tradicional dificilmente consegue satisfazer os requisitos de jogabilidade especificados anteriormente. A próxima seção apresenta uma alternativa eficiente para aplicar aprendizagem por reforço ao balanceamento, utilizando a abordagem de distinção entre competência e desempenho apresentada no Capítulo 4.

## 5.2. Balanceamento com AR Baseado em Desafio

Para implementar a abordagem de distinção entre competência e desempenho, é preciso definir os métodos utilizados para adquirir e evoluir a competência (construir agentes capazes de aprender estratégias ótimas) e adaptar o desempenho (escolher ações, possivelmente sub-ótimas, que garantam o balanceamento do jogo). A idéia é que o nível do jogo tenha as características ilustradas na Figura 7. Após o desenvolvimento do jogo (depois da linha tracejada) a competência deve continuar crescendo (devido à aprendizagem on-line), mas o

desempenho pode variar de acordo com cada usuário. A proposta aqui apresentada utiliza aprendizagem por reforço em ambas tarefas.



**Figura 7: Evolução da competência e adaptação do desempenho.**

### 5.2.1. Adquirindo e Aperfeiçoando a Competência

Na nossa proposta, a aquisição e evolução da competência são realizadas através de aprendizagem por reforço. O comportamento dos agentes é implementado como uma tarefa de AR, utilizando o método *Q-Learning* descrito anteriormente. O sinal de reforço usado representa o objetivo do jogo, isto é, derrotar o oponente da melhor maneira possível. Por exemplo, uma alternativa simples é fornecer um valor positivo (+1) quando o agente derrotar o oponente e um valor negativo quando for derrotado ou empatar o jogo (-1). Dessa forma, o processo de codificação dos estados, das ações e do sinal de reforço do agente segue as idéias tradicionais de AR [Sutton e Barto 1998]. O Algoritmo 1 apresenta o processo de aquisição e evolução de conhecimento de nossa proposta.

Esse algoritmo recebe como entrada o conjunto de estados  $S$ , o conjunto de ações  $A$ , a função de reforço  $R(s, a, s')$ , a função de desafio  $f(.)$  e uma política de atuação  $\pi$ . O conjunto  $S$  deve satisfazer a propriedade de Markov, como apresentado anteriormente. O Algoritmo 1 utiliza uma implementação tabular da função ação-valor: cria-se uma matriz bidimensional de valores reais com  $\#S$  linhas e  $\#A$  colunas (no qual  $\#$  significa o tamanho do conjunto referenciado). Embora o algoritmo apresentado utilize uma representação matricial da função ação-valor, outras implementações são possíveis, como redes neurais. A cada interação do jogo, utiliza-se uma política de escolha de ações  $\pi$  para escolher uma ação  $a$  com um nível de dificuldade  $d$ . A ação escolhida é executada, é observado o novo estado  $s'$  e a recompensa

$r(s,a,s')$  é calculada. O valor de  $Q[s][a]$  é atualizado na matriz bidimensional utilizando a regra de atualização do *Q-Learning* para um Processo de Decisão Semi-Markoviano. Na linha 5.e, o elemento  $\alpha$  é a taxa de aprendizagem,  $\gamma$  é o fator de desconto e  $t$  é a duração da ação  $a$ . A cada ciclo de avaliação  $\Delta t$  é calculada a função de desafio  $f(.)$ : se o resultado for menor do que o limite  $L_{fácil}$ , o valor de  $d$  é diminuído (o que equivale a aumentar o nível de dificuldade); se for maior do que o limite  $L_{difícil}$  o valor de  $d$  é aumentado. A descrição da função  $f(.)$  depende de cada jogo específico. Por exemplo, essa função pode receber o estado atual do agente e do oponente e informar um valor no conjunto {fácil, adequado, difícil} indicando o nível de dificuldade do jogo.

### Algoritmo 1: Aquisição e aperfeiçoamento do conhecimento

```

Entrada:
Conjunto de estados  $S$ 
Conjunto de ações  $A$ 
Função de reforço  $R(s,a,s')$ 
Função de desafio  $f(.)$ 
Política de atuação  $\pi(s,d)$ 

Parâmetros:
Taxa de aprendizagem  $\alpha$ 
Fator de desconto  $\gamma$ 
Limites  $L_{fácil}$  e  $L_{difícil}$ 

1. Crie uma matriz de valores reais  $Q$  ( $\#S$  linhas e  $\#A$  colunas)
2. Inicialize a matriz com valores zero
3.  $s \leftarrow$  estado inicial
4.  $d \leftarrow 1$ 
5. Para cada iteração do jogo, faça:
    a. Utilize uma política  $\pi(s,d)$  para escolher uma ação  $a$ 
    b. Execute  $a$ 
    c. Observe o novo estado  $s'$ 
    d. Calcule a recompensa recebida  $r$ 
    e.  $Q[s][a] \leftarrow (1-\alpha)Q[s][a] + \alpha(r + \gamma^t \max_{a'} Q[s'][a'])$ 
    f. A cada intervalo  $\Delta t$ :
        f.1. Calcule  $f(.)$ 
        f.2. Se  $f(.) < L_{fácil}$  e  $d > 1$ 
             $d \leftarrow d - 1$ 
        f.3. Se  $f(.) > L_{difícil}$  e  $d < \#A$ 
             $d \leftarrow d + 1$ 
    g.  $s \leftarrow s'$ ;

```

Esse algoritmo utiliza uma política de escolha de ações  $\pi$ . A determinação da política  $\pi$  faz parte do problema de adaptação do desempenho e será abordado posteriormente. A aquisição e evolução do conhecimento, que equivale às atualizações da matriz bidimensional



$Q[s][a]$ , independem da política utilizada. Supondo que a política de atuação possua um componente de exploração ( $\epsilon$ -greedy ou softmax), mantemos a garantia de convergência do algoritmo *Q-Learning*.

Uma vez implementado o agente, é realizado um processo de treinamento prévio (aprendizagem off-line) para promover a sua competência. Devido ao requisito de ser imediatamente capaz de atuar no nível do usuário (incluindo jogadores habilidosos) logo no início do jogo, os agentes precisam ser previamente treinados para garantir uma competência inicial. O treinamento off-line pode ser realizado deixando um agente lutar consigo mesmo (*self-learning*) [Kaelbling *et al.* 1996], contra outros agentes pré-programados [Madeira *et al.* 2004], ou contra um conjunto de jogadores humanos. Outra possibilidade de criação da competência inicial é através da definição, manual, dos valores iniciais da função ação-valor (diferentes de zero). Isso equivale a inserir explicitamente conhecimento no agente, o que pode requerer um grande esforço de desenvolvimento.

A finalidade do treinamento off-line é fazer com que o agente adquira e armazene, automaticamente, a maior quantidade possível de conhecimento sobre o ambiente. No nosso caso, busca-se acumular conhecimentos generalizados, que sejam úteis para oponentes de diferentes perfis. Essa situação é diferente do problema de Modelagem do Oponente, na qual o objetivo é acumular conhecimento necessário para derrotar um oponente específico.

Como não há um perfil bem definido de oponente, o treinamento realizado deve cobrir o maior número possível de estratégias. A questão de encontrar um treinador que simule a variação de situações encontradas em problemas reais (e não apenas a estratégia ótima) já foi anteriormente abordada. Epstein [1994] mostrou que o treinamento baseado em um oponente perfeito é insuficiente para preparar um agente para as variações existentes nos problemas reais. Por outro lado, agentes aleatórios podem fornecer uma grande diversidade de situações no treinamento, mas falham em não enfatizar as estratégias mais eficientes. Uma estratégia de treinamento promissora é complementar o oponente perfeito com algumas ações aleatórias que façam o agente conhecer caminhos sub-ótimos.

Escolhendo os parâmetros de treinamento corretamente, o que inclui a escolha do treinador, é possível, em princípio, desenvolver agentes satisfatoriamente competentes. Entretanto, para garantir a constante evolução do aprendizado, é importante que os agentes continuem adquirindo conhecimentos que melhorem sua competência contra cada oponente específico (aprendizagem on-line). Dessa forma, quando enfrenta jogadores humanos, os agentes continuam aprendendo para que o conhecimento prévio seja adaptado, descobrindo particularidades de cada oponente que possam maximizar a sua competência.

A próxima seção descreve como os agentes competentes desenvolvidos podem ser utilizados para atuar de forma balanceada contra seus oponentes.

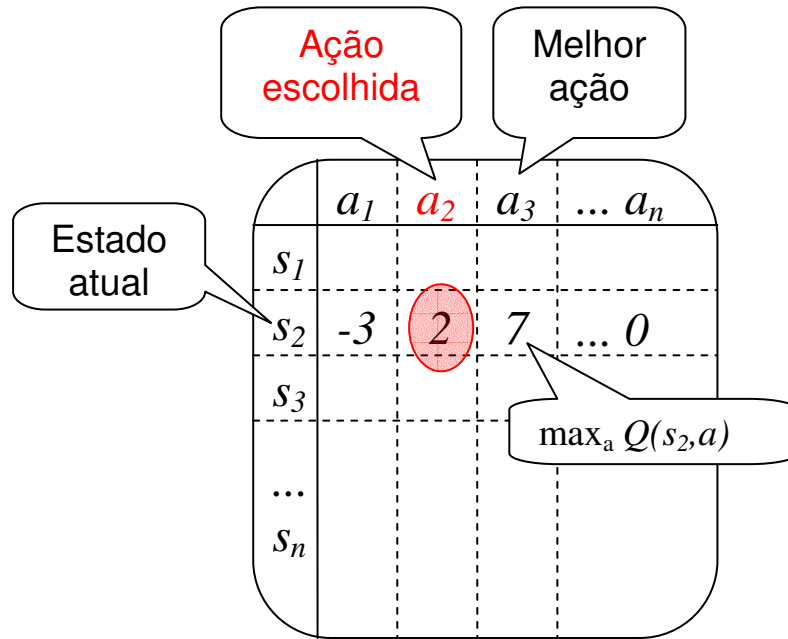
### 5.2.2. Adaptando o Desempenho

Para adaptar o desempenho dos agentes, foi desenvolvida uma política de escolha de ações baseada em desafio que escolhe ações que mantenham o agente e o oponente (jogador humano) no mesmo nível de desempenho.

Considerando que o agente tenha aprendido uma estratégia de atuação ótima (ou próxima de ótima), uma implementação simples dessa política consiste em periodicamente escolher ações aleatórias para serem executadas, evitando a frustração do oponente com longas seqüências de ações difíceis. A freqüência de execução de ações aleatórias dependeria da função de desafio: quando o nível está muito difícil, mais ações aleatórias são executadas; quando o nível está muito fácil, apenas as melhores ações são escolhidas. Entretanto, essa abordagem possui duas limitações básicas. Primeiramente, dificilmente os requisitos de inércia [R/JOG 2.A] e racionalidade [R/JOG 2.B] são satisfeitos, uma vez que as ações escolhidas randomicamente podem gerar um comportamento inconsistente. Por exemplo, um agente lutador pode ser levado a ficar parado para sofrer dano do usuário após executar uma seqüência de golpes difíceis. Além disso, como não se tem controle sobre a qualidade das ações aleatórias, a variação do desempenho pode não acompanhar o nível de habilidade do usuário, o que não satisfaz os requisitos de rápida adaptação [R/JOG 1.A] e [R/JOG 1.B]. A seguir, apresentamos como podemos implementar uma política que supera essas limitações.

Em nossa abordagem, propomos que, de acordo com o nível de dificuldade que o usuário percebe, sejam escolhidas e realizadas ações com maior ou menor retorno esperado. Para um dado estado, se o nível do jogo está muito difícil, o agente não executa a melhor ação (aquela com maior resultado na função ação-valor), mas escolhe progressivamente ações sub-ótimas até que o seu nível de atuação seja equivalente ao do jogador. Isso implica em escolher a segunda melhor ação, a terceira melhor, e assim por diante, até que os desempenhos do agente e do oponente estejam balanceados. Similarmente, se o jogo se torna fácil, as ações com maiores resultados na função ação-valor são progressivamente escolhidas, possivelmente até atingir a melhor ação aprendida. A Figura 8 ilustra uma configuração de um agente atuando no seu segundo melhor nível de desempenho, usando uma representação matricial do conhecimento aprendido. Nessa figura, embora o agente tenha o conhecimento de que a melhor ação a ser executada é  $a_3$ , uma ação sub-ótima ( $a_2$ ) é escolhida para que o desempenho seja adequado ao oponente.

Nossa abordagem utiliza a relação de ordem naturalmente definida em um dado estado pela função ação-valor, que é automaticamente construída durante o processo de aprendizagem. Como esses valores são uma estimativa da qualidade individual de cada ação, temos um controle rápido e eficiente do comportamento do agente, e, conseqüentemente, do seu nível de desempenho.



**Figura 8: Um agente atuando no seu segundo melhor nível de desempenho**

A Figura 8 utiliza uma matriz bidimensional (denominada *Q-Table*) para a implementação da função ação-valor porque esta opção nos pareceu a mais evidente. Entretanto, outras funções podem ser utilizadas como substitutas da tabela, como por exemplo, redes neurais.

Uma possível fonte de instabilidade nesse mecanismo de escolha de ações são as recompensas atrasadas, isto é, recompensas associadas a uma ação que não são recebidas imediatamente após sua execução. Como os resultados da função ação-valor expressam a recompensa esperada obtida pela ação atual somada a todas as suas subseqüentes (descontadas pelo parâmetro  $\gamma$ ), é necessário que o agente persista na política escolhida (ou seja, no nível definido) para receber o retorno estimado pela função ação-valor. Por exemplo, uma estimativa de retorno de +10 pode estar associada às ações: (1) aproximar-se do oponente, (2) realizar um pulo, e (3) deferir um chute no ar. Se o agente realizar apenas a ação (1), o retorno obtido não será aquele estimado pela função ação-valor, uma vez que é preciso realizar todas as 3 ações. Isso significa que precisamos definir um ciclo de avaliação, que

determina a periodicidade em que o agente analisa e, possivelmente, modifica seu nível de atuação. Entre cada avaliação, o nível de atuação não é modificado, de forma que o agente persista por algum tempo em uma mesma política. Esse pode ser o mesmo ciclo usado pela função de desafio para medir a dificuldade que o usuário está enfrentando.

Na política de atuação proposta, o agente periodicamente avalia se está atuando no mesmo nível do usuário, através da função de desafio, e de acordo com o resultado, mantém ou modifica o seu nível de atuação. O agente não modifica seu nível até o próximo ciclo. O tamanho do ciclo de avaliação é profundamente dependente do ambiente em que o agente atua, sendo influenciado, em particular, pelo tempo de recebimento de recompensas atrasadas. Se o ciclo for muito curto, o agente pode exibir um comportamento aleatório; se for muito longo, o agente não conseguirá acompanhar a evolução (ou regressão) do usuário suficientemente rápido. Assim como a função de desafio, o ciclo de avaliação precisa ser determinado heurísticamente.

É importante destacar que a técnica apresentada modifica apenas o procedimento de escolha de ações, enquanto o agente mantém o mecanismo de aprendizagem por reforço tradicional durante todo o jogo, detalhado no Algoritmo 1. Essa separação só é possível porque o método *Q-Learning* é um método de controle *off-policy*, no qual a aprendizagem não depende da política de escolha de ações utilizada. A única restrição existente para garantir a convergência da aprendizagem é que a política utilizada realize um método de exploração, como  $\epsilon$ -greedy ou softmax, de modo que todo par estado-ação seja visitado. O Algoritmo 2 mostra como o método  $\epsilon$ -greedy pode ser usado em uma política de escolha de ações baseada em desafio.

O Algoritmo 2 possui como entrada o conjunto de ações  $A$ , a função ação-valor  $Q(s,a)$ , o estado atual  $s$  e o nível de atuação atual  $d$ . A cada execução, o algoritmo escolhe randomicamente uma ação de  $A$  com uma probabilidade  $\epsilon$  e escolhe uma ação balanceada com probabilidade  $(1-\epsilon)$ . Para escolher uma ação balanceada, as ações do conjunto  $A$  são ordenadas de acordo com a função ação-valor. Para um nível de desempenho  $d$ , é escolhida aquela na posição  $d$  da lista ordenada. A forma de ordenação (crescente ou decrescente) do conjunto de ações depende da codificação do valor de  $d$ : caso os valores próximos de 1 se refiram a desempenho elevado, o conjunto deve ser ordenado decrescentemente; caso contrário (valores próximos de 1 representam desempenho sub-ótimo), a ordenação deve ser crescente.

O parâmetro  $\epsilon$  especifica a taxa de exploração do algoritmo. Quanto maior seu valor, maior a exploração. Entretanto, como a exploração é realizada através de ações aleatórias, valores muito altos de  $\epsilon$  podem prejudicar o desempenho do agente. Para evitar isso, é

necessário utilizar um valor que permita a exploração de novos pares estado-ação, mas que não degrade o desempenho geral. Uma outra possibilidade é variar a taxa de exploração ao longo do tempo: nas execuções iniciais, quando o agente ainda não acumula muito conhecimento, é usada uma taxa alta; ao longo do tempo essa taxa é diminuída de forma que, à medida que a competência aumenta, menos ações aleatórias são utilizadas.

### Algoritmo 2: Política $\epsilon$ -greedy de escolha de ações baseada em desafio

Entrada:  
 Conjunto de ações  $A$   
 Matriz  $Q[\#S][\#A]$   
 Estado atual  $s$   
 Nível de desempenho  $d$

Saída:  
 Uma ação a ser executada

Parâmetros:  
 Taxa de exploração  $\epsilon$

1. Com uma probabilidade  $\epsilon$ 
  - a. Escolha randomicamente uma ação de  $A$
2. Com uma probabilidade  $(1-\epsilon)$ 
  - a. Ordene os elementos de  $A$  com base nos valores de  $Q[s][.]$
  - b. Escolha o  $d$ -ésimo elemento da lista ordenada
3. Retorne a ação escolhida

Outra possibilidade de implementação da política de escolha de ações baseada em desafio é a utilização do método de exploração softmax. Para aplicar o softmax à política definida nessa seção, é necessário realizar uma pequena alteração quando se associa a cada ação uma probabilidade de ser escolhida. Essa probabilidade deve ser baseada na distância em relação à ação desejada, de modo que as ações próximas do nível de atuação definido tenham mais chance de serem escolhidas do que as ações mais distantes. Dessa forma, considerando o softmax com a função de Boltzmann, temos a função de probabilidade especificada a seguir:

$$P(a) = \frac{e^{D(a)/\tau}}{\sum_{b=1}^n e^{D(b)/\tau}}, \quad \text{onde} \quad D(a) = Q(a_{\text{sub-ótima}}) - |Q(a) - Q(a_{\text{sub-ótima}})|$$

Nessa função,  $\tau$  é o parâmetro temperatura,  $n$  é o número de ações possíveis de serem executadas no estado atual,  $Q(a)$  é o resultado da ação  $a$  na função ação-valor e  $Q(a_{\text{sub-ótima}})$  é o resultado da ação sub-ótima desejada (definida pelo nível de atuação). A função  $D$  é calculada com base na distância (isto é, no valor absoluto da diferença) entre a ação avaliada  $a$

e a ação desejada  $a_{sub-ótima}$ . Com essa função, garantimos que a ação desejada terá maior probabilidade de escolha e as ações distantes dela (muito melhores ou muito piores) terão probabilidades baixas. O Algoritmo 3 apresenta a implementação desses conceitos.

### Algoritmo 3: Política softmax de escolha de ações baseada em desafio

```

Entrada:
Conjunto de ações  $A$ 
Matriz  $Q[\#S][\#A]$ 
Estado atua  $s$ 
Nível de desempenho  $d$ 

Saída:
Uma ação a ser executada

Parâmetros:
Distribuição de probabilidade  $P$ 

1. Ordene os elementos de  $A$  com base nos valores de  $Q[s][.]$ 
2. Escolha o  $d$ -ésimo elemento  $a_d$  da lista ordenada
3.  $V_d \leftarrow Q[s][a_d]$ 
4. Para cada ação  $a$  em  $A$ 
    a.  $V_a \leftarrow V_d - \text{abs}(Q[s][a] - V_d)$ 
    b. Escolha  $a$  com uma probabilidade  $P(V_a)$ 
5. Retorne a ação escolhida

```

O Algoritmo 3 apresenta as mesmas variáveis de entrada do Algoritmo 2. Entretanto, ao invés de possuir uma taxa de exploração, tem como parâmetro uma distribuição de probabilidade  $P$ , como a função de Boltzmann citada anteriormente. A cada execução, o conjunto de ações  $A$  é ordenado com base na função ação-valor. A ação na posição  $d$  da lista ordenada é escolhida como ação desejada (os passos 1 e 2 são equivalentes aos 2.a e 2.b do Algoritmo 2), e seu resultado na função ação-valor é guardado em  $V_d$ . Para todas as ações possíveis, suas distâncias em relação à ação  $a_d$  são calculadas utilizando a regra 4.a. Essa regra garante que todos os valores serão menor ou igual ao valor da ação desejada, de modo que essa tenha maior probabilidade de ser escolhida. No entanto, todas as demais ações ainda podem ser executadas, de acordo com a distribuição de probabilidade usada. Para o caso de uma função de Boltzmann o valor do parâmetro temperatura pode ser regulado para priorizar mais exploração ou mais usufruto do conhecimento aprendido. Assim como no método  $\epsilon$ -greedy, esse parâmetro também pode variar no decorrer do tempo para priorizar explorações nas execuções iniciais e desempenho após o agente estiver consolidado.

A utilização do softmax no balanceamento de jogos serve para evitar a previsibilidade do comportamento dos agentes, sem degradar seu desempenho com ações puramente

aleatórias. Através desse método, podemos alcançar o efeito ilustrado na Figura 5, na qual um agente não acompanha precisamente a evolução do usuário, mas sim intercala momentos de relaxamento e tensão, garantindo um componente de imprevisibilidade ao seu comportamento.

A política de escolha de ações apresentada nesta subseção é apenas uma das modificações possíveis no *framework* de *Q-Learning*. A idéia de repensar e sofisticar a escolha de ações é bem geral e pode ser explorada em outras direções, criando-se variações na política de adaptação. Por exemplo, poderíamos implementar uma política que se mantém sempre 3 níveis acima do usuário nas fases difíceis e se conserva 1 nível abaixo nas fases fáceis. Outra possibilidade é estabelecer um limite mínimo de desempenho, nunca executando, por exemplo, menos do que a 5ª melhor ação.

### 5.3. Conclusões

Esse capítulo apresentou uma abordagem original para o problema de balanceamento dinâmico de jogos digitais. A abordagem proposta é baseada na distinção das dimensões de competência e desempenho. A aquisição e evolução da competência são obtidas através de aprendizagem por reforço, complementada por treinamentos off-line e a constante atualização on-line do conhecimento adquirido. A adaptação do desempenho é conseguida com a utilização de uma política de escolha de ações baseada em desafio que possibilita a um agente atuar no mesmo nível de seu oponente.

O mecanismo de escolha de ações proposto, combinado com uma fase de aprendizagem off-line eficiente, permite, em princípio, que os agentes sejam capazes de rapidamente atuar no mesmo nível de cada usuário já no começo de um jogo, não importando o seu nível de experiência e habilidade. Além disso, como a adaptação do desempenho não depende da aquisição de conhecimento, as evoluções e regressões do usuário podem ser eficientemente acompanhadas. Essas características indicam que a proposta aqui descrita pode potencialmente satisfazer os requisitos de rápida adaptação [R/JOG 1.A] e [R/JOG 1.B] apresentados no Capítulo 2. Do ponto de vista da credibilidade (*believability*), uma parametrização correta dos ciclos de avaliação pode evitar alterações bruscas no comportamento do agente, de modo a satisfazer, pelo menos parcialmente, o requisito de inércia [R/JOG 2.A]. Podemos diminuir o impacto das ações aleatórias de exploração utilizando uma política softmax, o que dificulta a ocorrência de comportamentos inconsistentes, e satisfaz parcialmente o requisito de racionalidade [R/JOG 2.B]. Por fim, em termos de implementação, a abordagem aqui proposta também satisfaz parcialmente o requisito de desenvolvimento [R/IMP 3], pois embora torne desnecessária a codificação de extensas bases de regras, requer a correta parametrização dos algoritmos de aprendizagem e adaptação. O requisito de execução

[R/IMP 4] provavelmente é satisfeito pois os algoritmos aqui apresentados não exigem extensivos recursos computacionais.

No próximo capítulo, expomos o ambiente utilizado como estudo de caso para validar e analisar nossa proposta, bem como os detalhes de implementação do nosso método e de outras abordagens de balanceamento encontradas na literatura.



## 6. Implementação

As idéias apresentadas neste trabalho foram implementadas no domínio de Jogos de Luta. Jogos de Luta são aqueles que simulam alguma forma de arena na qual lutadores se enfrentam com o objetivo de derrotar o adversário. Alguns jogos comerciais de sucesso desse gênero são *Street Fighter* (Capcom) e *Mortal Combat* (Midway).

### 6.1. Knock'em: Descrição do Ambiente

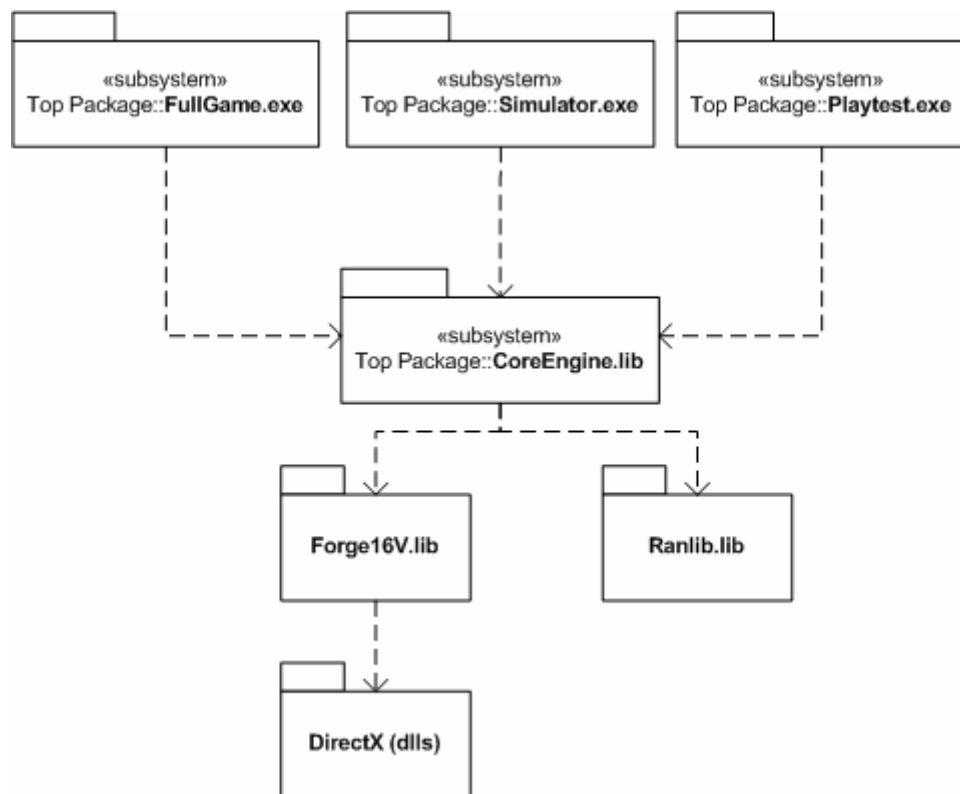
O ambiente utilizado nas implementações é baseado em um jogo previamente desenvolvido no Centro de Informática da UFPE, o Knock'em [Andrade *et al.* 2003]. O Knock'em consiste em um jogo de luta em tempo real no qual dois lutadores se enfrentam dentro de uma arena. O objetivo principal é derrotar o oponente. Uma luta termina quando os pontos de vida de um dos lutadores (inicialmente cada um possui 100 pontos) chega a zero, ou após 1min30secs de luta, a situação que ocorrer primeiro. O vencedor é aquele que possuir a maior quantidade de pontos de vida no final. O ambiente é uma arena bidimensional na qual movimentos horizontais são livres e movimentos verticais são possíveis através de pulos. As ações de ataque que cada lutador pode executar são *soco* (forte ou rápido), *chute* (forte ou rápido), e *lançamento de bola-de-fogo*. Socos e chutes também podem ser deferidos no ar, durante um *pulo*. As ações de defesa são *bloqueio* ou *agachamento*. Enquanto agachado, um lutador também pode deferir socos ou chutes. Se o lutador possuir uma quantidade suficiente de energia espiritual (denominada de *mana*), bolas-de-fogo podem ser lançadas. O *mana* é reduzido após cada lançamento e é continuamente recuperado no decorrer do tempo, a uma taxa fixa. Algumas telas do Jogo são mostradas na Figura 9.



Figura 9: Telas do ambiente de testes Knock'em

## 6.2. Arquitetura do Ambiente

O ambiente implementado possui 3 sistemas: Jogo (FullGame.exe), Simulador (Simulator.exe), e Playtest (Playtest.exe). Esses sistemas utilizam as funções básicas de um módulo principal (CoreEngine.lib), na qual estão as regras básicas do jogo e diferentes implementações de agentes inteligentes. A Figura 10 apresenta o diagrama de pacotes do ambiente.



**Figura 10: Sistemas e módulos do ambiente de testes Knock'em**

Todos os sistemas e módulos são implementados em C++ e utilizam a biblioteca de jogos Forge16V [Rocha 2003], também desenvolvida no Centro de Informática da UFPE. Essa biblioteca fornece uma abstração das funções típicas de jogos (como gerenciamento gráfico, de som, e de entrada de dados), a partir da API multimídia DirectX [Microsoft.com 2006], produzida pela Microsoft e amplamente utilizada em jogos comerciais. Além do Forge16V, também é utilizada a biblioteca Ranlib [Ranlib 2006], que fornece funções matemáticas e de geração de números randômicos.

O módulo Jogo consiste na versão completa de um jogo de luta, bastante semelhante aos jogos clássicos do gênero. O módulo Simulação executa longas seqüências de lutas, e serve para realizar o treinamento off-line de agentes que implementam algum mecanismo de

aprendizagem. Por fim, o módulo Playtest consiste em uma versão reduzida do jogo voltada para a realização de testes com jogadores humanos, integrando a coleta de dados ao enredo.

O módulo principal (CoreEngine.lib) possui todos os elementos que determinam a dinâmica do ambiente, como modelagem física, funções de dano, e regras do jogo. Todos esses itens são utilizados pela classe principal FightGameState, que implementa o estado correspondente a uma luta. Essa classe possui atributos como a imagem de fundo, o placar (*score*), o tempo de luta, e os lutadores. A Figura 11 apresenta o diagrama de classes resumido desse estado.

Uma luta possui 2 lutadores, que são representados pela classe Fighter. Essa é uma classe abstrata que implementa as funções comuns a todos os lutadores, como exibição na tela e atualização de estados e posições. Alguns de seus atributos são os parâmetros de cada personagem (força, resistência, agilidade, energia espiritual, e pontos de vida) e as variáveis relacionadas a um lutador, como posição, *sprites* (as imagens que formam as animações dos golpes e movimentos), e estado atual.

A classe Fighter é abstrata porque não implementa o comportamento do lutador. O comportamento é determinado pelas classes que a estendem, através da implementação do método ProcessGameState(). Esse método é executado a cada ciclo de jogo, o que permite que cada lutador perceba o ambiente e escolha a ação a ser executada em cada interação. Alguns exemplos de implementação são as classes HMPPlayer, RDPlayer, e SMPlayer. Cada uma implementa uma opção diferente de lutador. A classe HMPPlayer (*Human Player*) representa um jogador humano, e determina as ações a serem executadas a partir das teclas digitadas. As demais classes correspondem a agentes inteligentes que implementam diferentes estratégias de atuação.

Essa arquitetura é interessante por abstrair o comportamento dos lutadores das funcionalidades de controle. As classes FightGameState e Fighter são fixas e definidas pelo projeto do jogo (game design). Para implementar um agente lutador, é preciso apenas criar uma classe que herda de Fighter e codificar seu comportamento no método ProcessGameState(), sem necessitar se preocupar com a integração do lutador ao jogo. A próxima seção apresenta os agentes implementados neste trabalho com base na arquitetura aqui definida.

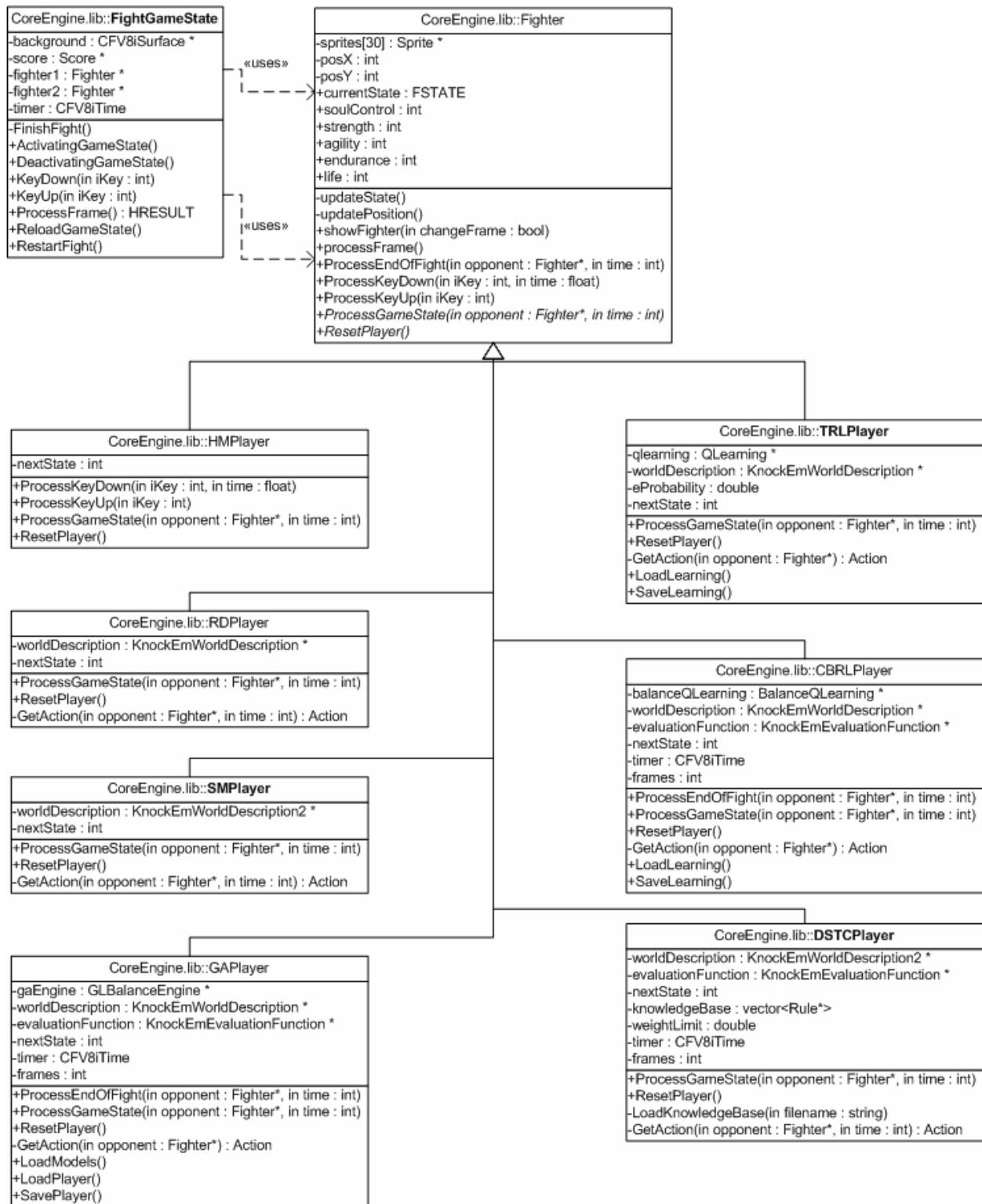


Figura 11: Diagrama de classes resumido do estado de luta

## 6.3. Criação dos Agentes

Todos os personagens NPCs do Knock'em foram implementados como agentes, e sua inteligência artificial baseou-se em diferentes estratégias. Primeiramente, foram desenvolvidos dois agentes básicos: um agente Randômico (RDPlayer) e um agente *State-Machine* (SMPlayer). Em seguida, foram desenvolvidos dois agentes baseados em aprendizagem por reforço: um utilizando a abordagem tradicional (TRLPlayer) e um utilizando a abordagem baseada em desafio proposta nesse trabalho (CBRLPlayer). Por fim, para realizar uma análise comparativa de diferentes métodos de balanceamento dinâmico, foram desenvolvidos um agente que implementa a proposta de Spronck *et al.* [2004b], baseada em scripts dinâmicos (DSTCPlayer) e um agente que implementa a proposta de Demasi e Cruz [2002], baseada em algoritmos genéticos (GAPlayer). Todos esses agentes, de alguma maneira, percebem e atuam no ambiente. A próxima seção especifica como esse ambiente (também denominado de mundo) é codificado para todos os agentes NPCs.

### 6.3.1. Codificação do Ambiente

Agentes inteligentes são sistemas que, a cada momento, percebem o estado do mundo, através de um conjunto de sensores, e executam ações, através de um conjunto de atuadores. Definir os sensores de um agente consiste em especificar o conjunto de estados que serão percebidos, e definir os atuadores consiste em especificar o conjunto de ações que podem ser realizadas.

A codificação dos conjuntos (ou espaços) de estados e ações é denominada de descrição do mundo [Russel e Norvig 2004]. A descrição do mundo possui um grande impacto no desempenho dos agentes porque é ela que especifica as informações que podem ser usadas na tomada de decisão e as ações que podem ser executadas.

A velocidade de aprendizagem é diretamente relacionada ao tamanho dos espaços de estados e ações. O agente precisa visitar cada par estado-ação várias vezes até aprender quão bom é executar uma determinada ação em um determinado estado. Com isso, quanto maior os conjuntos de estados e ações, mais tempo o agente precisará para atingir uma competência satisfatória.

Para que a velocidade de aprendizagem seja coerente com a duração de um jogo de luta, algumas técnicas foram utilizadas para reduzir a representação de estados e ações e, conseqüentemente, reduzir os espaços  $S$  e  $A$ . A redução do espaço de estados foi obtida com a discretização de variáveis contínuas. Essa estratégia consiste em não apenas transformar valores reais nos inteiros mais próximos, mas sim definir intervalos que sejam efetivamente relevantes para a percepção dos agentes. Por exemplo, em um jogo FPS (*First Person*

*Shooter*), um agente não precisa conhecer a distância real de seu oponente para atirar, mas apenas se ele está ou não dentro do raio de alcance de sua arma. A redução do espaço de ações, por sua vez, foi obtida através da codificação de seqüências de ações (*Moves*, na denominação de Merke e Riedmiller [2001]). Esses movimentos são seqüências de ações atômicas com um objetivo comum. Ao invés de codificar cada ação e esperar o agente aprender a seqüência correta, é codificada uma ação abstrata que corresponde a toda a seqüência.

A representação de estados (percepções) é implementada como a seguinte tupla:

$$S = (S_{agente}, S_{oponente}, D, M_{agente}, M_{oponente}, F)$$

$S_{agente}$  significa o estado atual do agente (parado, pulando, ou agachado).  $S_{oponente}$  significa o estado do oponente (parado, pulando, agachado, atacando, atacando durante um pulo, atacando enquanto agachado, e bloqueando).  $D$  representa a distância do agente para o oponente (perto, médio, ou longe).  $M_{agente}$  e  $M_{oponente}$  significam a disponibilidade de mana do agente e oponente, respectivamente (suficiente ou insuficiente para lançar uma bola-de-fogo). Por fim,  $F$  significa a situação das bolas-de-fogo lançadas pelo oponente (perto, médio, longe, ou não existente).

Esses atributos foram escolhidos heurísticamente (após alguns testes com outras codificações). Os valores de  $S_{agente}$  significam aquelas situações em que o agente pode efetivamente tomar uma decisão (isto é, mudar seu estado). Por exemplo, o agente não pode mudar seu estado durante a execução de um soco, apenas após o golpe ter sido deferido. Já durante um pulo ou enquanto agachado, o agente pode decidir, por exemplo, deferir um chute no ar ou um chute rasteiro, respectivamente. Por isso, não é preciso identificar quando o agente está deferindo um soco, mas é relevante identificar quando ele está pulando ou agachado. Já os valores de  $S_{oponente}$  representam as ações do oponente que o agente precisa perceber para se defender ou contra-atacar. Para reduzir o espaço de estados, todas as ações de ataque (soco e chute forte ou rápido) foram representadas com um único estaco “atacando”. Os valores de  $D$  indicam se o oponente está ou não no raio de alcance de um ataque, os valores de  $M$  indicam se cada lutador pode ou não lançar uma bola-de-fogo, e os valores de  $F$  representam os valores relevantes para que o agente aprenda a se defender de bolas-de-fogo inimigas, pulando-as ou bloqueando-as.

As ações possíveis de cada agente são as mesmas de todos lutadores: socar e chutar (forte ou rápido), se aproximar, fugir, pular, pular se aproximando, pular fugindo, lançar bola-de-fogo, bloquear, se agachar ou permanecer parado. Apenas as ações de movimentação (aproximação e fuga) foram codificadas como seqüências de ações atômicas. Dessa forma,

cada uma dessas ações corresponde a 100 repetições da movimentação (o agente se aproxima ou se afasta por 100 frames de jogo).

No Knock'em, a descrição do mundo é determinada pela classe abstrata `KnockEmWorldDescription` (Figura 11). Essa classe é abstrata porque permite que diferentes codificações sejam utilizadas e testadas, sem a necessidade de modificação da implementação dos agentes. Com pode ser observado na Figura 11, cada agente inteligente (todos os lutadores, com exceção do `HMPlayer`) possui um atributo `KnockEmWorldDescription`. Essa classe possui duas implementações: a primeira, `KnockEmWorldDescription1`, foi realizada no desenvolvimento da versão inicial do jogo [Andrade *et al.* 2003], e a segunda, `KnockEmWorldDescription2`, foi realizada durante a realização deste trabalho a partir da correção e evolução da primeira. A codificação apresentada anteriormente corresponde à classe `KnockEmWorldDescription2`.

Para os agentes que constroem sua competência a partir de algum mecanismo de aprendizagem de máquina, sem a codificação de regras explícitas, (`TRLPlayer`, `CBRLPlayer` e `GAPlayer`), a forma como a descrição do mundo é codificada não interfere na sua implementação, uma vez que as estratégias de atuação são automaticamente aprendidas. Por isso, se precisarmos adicionar um novo elemento na tupla de representação de estados (por exemplo, a força do oponente), não precisaremos modificar os agentes aprendizes, mas apenas fazê-los instanciar uma nova classe `KnockEmWorldDescription3`. A partir do treinamento, os agentes aprenderão automaticamente como utilizar essa nova informação.

Por outro lado, os agentes que necessitam de inserção explícita de conhecimento (`SMPlayer` e `DSTCPlayer`), precisam ter acesso às implementações concretas da descrição do mundo, pois é através dela que serão criadas (em tempo de desenvolvimento) as regras de atuação do agente. Por exemplo, para criar uma regra do tipo “*Se estiver perto, deferir um soco*” é preciso conhecer como o atributo “perto” e a ação “soco” são codificadas. Dessa forma, embora os demais agentes tenham como atributo a classe abstrata `KnockEmWorldDescription`, esse grupo acessa diretamente a classe concreta `KnockEmWorldDescription2`. A cada mudança na descrição do mundo, a implementação desses agentes também precisa ser atualizada.

Além de especificar os estados e ações acessíveis aos agentes, a descrição do mundo possui um método, `isPossibleAction(State s, Action a)`, que informa se é possível realizar a ação *a* no estado *s*. Dessa forma, em cada tomada de decisão, os agentes analisam apenas as ações possíveis de serem executadas no estado atual.

Esse método é útil para limitar as possibilidades de atuação do agente, proibindo a execução de ações claramente ineficientes. Como esse método é implementado durante o

desenvolvimento, essa é uma forma de inserir explicitamente conhecimento nos agentes aprendizes. Por exemplo, na classe `KnockEmWorldDescription2`, são codificadas regras que impedem a execução da ação pular durante um pulo (conforme limitação do jogo), ou impedem que seja deferido um golpe quando o oponente se encontra muito longe (ação claramente ineficiente). Embora a implementação desse método não seja obrigatória, sua utilização é uma maneira eficiente de diminuir o tempo de aprendizagem, na medida que reduz as possibilidades que precisam ser exploradas pelo agente.

Uma vez definida a codificação do ambiente, é preciso especificar como cada agente utiliza as percepções de estados e ações na sua tomada de decisão. Os detalhes de implementação de cada um são apresentados a seguir.

### 6.3.2. Agentes Básicos

Foram implementados dois agentes básicos para servirem de referencial para os demais: um agente *Random* e um *State-Machine*. O agente *Random* (`RDPlayer`) escolhe aleatoriamente as ações a serem executadas. De modo a evitar ações exóticas e mantê-lo coerente com as regras do jogo, esse agente utiliza o conhecimento básico codificado na descrição do mundo (através das regras do método `isPossibleAction` da classe `KnockEmWorldDescription`). No entanto, essas regras apenas limitam a execução de algumas ações, não fornecendo nenhuma informação sobre a estratégia a adotar (como se aproximar do oponente, como atacá-lo, como se defender, etc.). Essas decisões táticas são realizadas aleatoriamente.

O agente *State-Machine* (`SMPlayer`) utiliza uma máquina de estados para definir suas ações. Para cada estado percebido, um conjunto de 25 regras indica a ação a ser executada. Essas regras de comportamento são estáticas e definidas durante o desenvolvimento do jogo. Alguns exemplos de regras utilizadas são exibidos na Figura 12. Como cada estado é mapeado em uma única ação, o comportamento do agente não muda no decorrer do tempo.

```
SE (distancia = Perto) E (estado-do-oponente = Parado)
ENTÃO ação = Soco-Rápido

SE (distancia = Perto) E (estado-do-oponente = Atacando)
ENTÃO ação = Bloquear

SE (distancia = Longe) E (bolas-de-fogo = Perto)
ENTÃO ação = Bloquear
```

**Figura 12: Exemplos de regras do agente `SMPlayer` escritas em pseudo-linguagem**



### 6.3.3. TRL – *Traditional Reinforcement Learning*

Uma vez definidos os agentes básicos SMPlayer e RDPlayer, foi desenvolvido um agente *Traditional Reinforcement Learning* (TRLPlayer) que implementa seu comportamento como uma tarefa de aprendizagem por reforço, usando o método *Q-Learning* com representação linear da função ação-valor. Para isso, foi utilizado e estendido o motor de aprendizagem por reforço RLEngine [Santana 2005], desenvolvido no Centro de Informática da UFPE. Nesse agente, temos uma matriz bidimensional de estados e ações, na qual os conjuntos S (espaço de estados) e A (espaço de ações) precisam ser finitos. Esses conjuntos são definidos pela descrição do mundo especificada anteriormente.

O sinal de reforço do agente é baseado na diferença de pontos de vida entre os lutadores causada por cada ação: pontos tirados do oponente, menos pontos perdidos.

$$r = \Delta_{oponente} - \Delta_{agente}$$

Como inicialmente cada lutador tem 100 pontos de vida, o sinal de reforço está sempre no intervalo  $[-100; 100]$ . Reforços negativos indicam um mau desempenho, uma vez que o agente perdeu mais pontos do que ganhou, enquanto reforços positivos são o objetivo desejado. Essa medida representa satisfatoriamente o objetivo da aprendizagem porque o resultado de uma luta é determinado pelos pontos de vida ao final dela: se o agente tiver tirado mais pontos do que perdido, será vencedor; caso contrário, será perdedor.

O sinal de reforço pode ser informado ao agente apenas ao final de cada luta, quando seu objetivo (vencer) está completamente alcançado. Entretanto, trabalhos anteriores [Mataric 1994] já mostraram que a aprendizagem pode ser acelerada pela utilização de funções de reforço que recompensem o agente quando sub-objetivos são alcançados. No Knock'em, isso equivale a informar o sinal de reforço a cada ação realizada pelo agente, permitindo que as conseqüências de cada ação sejam mais rapidamente percebidas.

A política de escolha de ações do agente TRLPlayer é uma política gulosa (*greedy*): a cada estado percebido, é executada a ação que possui maior valor na função ação-valor. Como essa função é implementada como uma matriz bidimensional, em cada estado  $s$  basta analisar a linha da matriz correspondente a esse estado e escolher a ação que possui o maior valor. Essa política equivale a um agente que atua da melhor forma possível, de acordo com o conhecimento adquirido.

Uma vez implementado o agente TRLPlayer, foram realizados treinamentos off-line, de modo a lhe garantir uma inteligência inicial básica. Embora esses agentes sejam capazes de aprender e modificar seu comportamento on-line, durante uma luta, a etapa de treinamento off-

line é importante para impulsionar seu desempenho nas lutas iniciais, quando ainda não há muita informação sobre o oponente desafiado.

Como atingir o limite de convergência da aprendizagem pode ser inviável na prática, uma vez que requer um grande número de iterações, foram definidos parâmetros da etapa de treinamento para que um bom balanceamento entre qualidade da aprendizagem e viabilidade computacional fosse obtido. Inicialmente, foram realizados treinamentos com 50 lutas, durante as quais o agente executa uma política  $\epsilon$ -greedy com 10% de ações randômicas. A taxa de aprendizagem utilizada é 0,5 e o fator de desconto, 0,9.

#### 6.3.4. CBRL – *Challenge-Based Reinforcement Learning*

O mecanismo de balanceamento baseado em desafio proposto neste trabalho é implementado em um agente *Challenge-Based Reinforcement Learning* (CBRLPlayer). O CBRLPlayer também implementa seu comportamento como uma tarefa de aprendizagem por reforço, através do mesmo algoritmo *Q-Learning* utilizado pelo TRLPlayer. A codificação do sinal de reforço de ambos lutadores é a mesma, assim como os parâmetros de treinamento e de aprendizagem.

A diferença entre os agentes está na política de escolha de ações. Enquanto o agente TRLPlayer executa as melhores ações possíveis indicadas pelo aprendizado, o CBRLPlayer escolhe aquelas mais adaptadas ao nível de habilidade do oponente. Para isso, é utilizada uma função de desafio que infere, para cada estado, o nível de dificuldade sentido pelo oponente.

A função de desafio utilizada é baseada na diferença de pontos de vida entre os lutadores. Com o objetivo de estimular a evolução do jogador, a função foi implementada de forma que o agente atue um pouco acima do nível do oponente. Após a realização de alguns testes, foi fixada, heurísticamente, a seguinte função:

$$f = \begin{cases} \text{fácil}, & \text{caso } (L_{\text{agente}} - L_{\text{oponente}}) < 0 \\ \text{médio}, & \text{caso } (L_{\text{agente}} - L_{\text{oponente}}) < 10 \\ \text{difícil}, & \text{caso contrário} \end{cases}$$

Essa função indica que, caso os pontos de vida do agente ( $L_{\text{agente}}$ ) seja menor do que os do oponente ( $L_{\text{oponente}}$ ), o nível de dificuldade está fácil e o agente deve melhorar o desempenho. Caso a diferença seja menor do que 10 (o equivalente a 10% do total possível de pontos de vida), a dificuldade está razoável e o agente continua atuando no mesmo nível. Caso contrário, o nível está difícil e o agente deve diminuir seu desempenho, executando ações sub-ótimas.

A Figura 13 apresenta o diagrama de classes resumido do CBRLPlayer. Além dos atributos comuns aos demais agentes, como a descrição do mundo, essa classe possui uma

função de desafio (KnockEmEvaluationFunction) e uma implementação da nossa proposta, baseada em *Q-Learning* (BalanceQLearning).

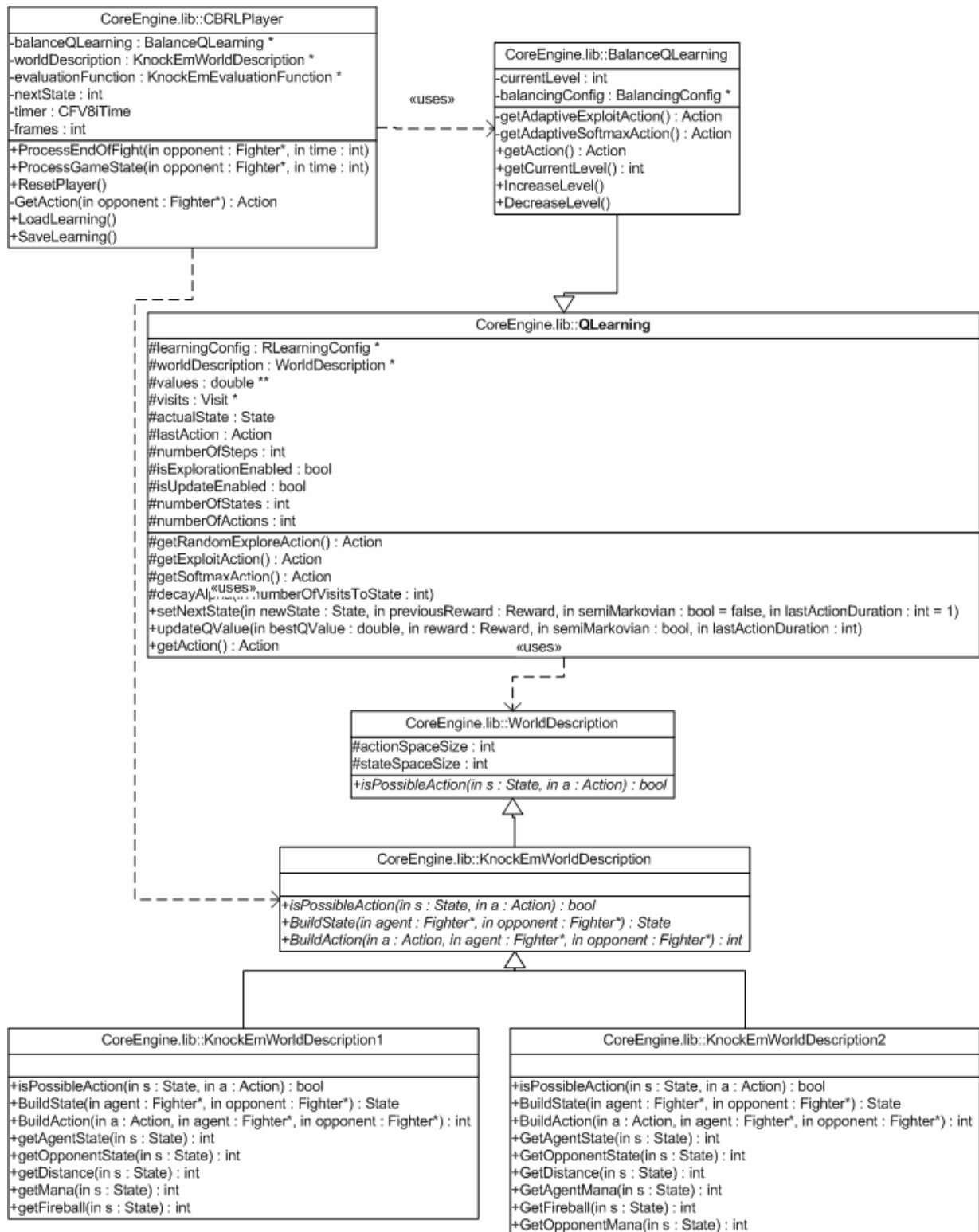


Figura 13: Diagrama de classes resumido do agente CBRLPlayer

A cada ciclo de avaliação, o agente utiliza a função de desafio e, de acordo com seu resultado, aumenta ou diminui o nível de dificuldade implementado pela classe `BalanceQLearning`. O `BalanceQLearning` estende a classe `QLearning`, também usada pelo agente `TRLPlayer`, e utiliza seus métodos de representação e evolução do conhecimento (respectivamente, a matriz bidimensional e o método `UpdateQValue()`). Apenas o método de escolha de ações (`GetAction()`) é sobrescrito por implementações próprias (`GetAdaptiveExploitAction()` e `GetAdaptiveSoftmaxAction()`).

Como pode se observar na Figura 13, a classe `QLearning` utiliza uma descrição do mundo que independe da aplicação (`WorldDescription`). Essa classe é definida pelo motor genérico de aprendizagem por reforço `RLEngine`. Dessa forma, assim como a classe `QLearning`, a implementação de nossa proposta, `BalanceQLearning`, também pode ser aplicada em diferentes aplicações, sejam jogos ou não.

As classes que implementam lutadores (`TRLPlayer` e `CBRLPlayer`), por sua vez, acessam uma descrição específica do ambiente do Knock'em (`KnockEmWorldDescription`) porque as codificações de estados e ações usadas pelos agentes não correspondem diretamente àqueles implementados pelo motor do jogo. Por exemplo, a ação “andar para frente” no jogo corresponde a 1 frame de movimentação, enquanto na codificação dos agentes, essa ação significa uma movimentação de 100 frames. Dessa forma, é necessária uma classe intermediária que realize as conversões entre codificação do agente e do jogo.

A função de desafio é utilizada periodicamente, de acordo com o ciclo de avaliação do agente `CBRL`. O ciclo de avaliação utilizado foi definido, também heurísticamente, como 100 *frames* do jogo (ou, aproximadamente, a cada 3 segundos). Isso significa que, a cada 100 *loops*, o agente infere o nível de dificuldade através da função de desafio e adapta seu comportamento de acordo com o resultado. Esse ciclo de avaliação foi definido porque é suficientemente longo para que o agente acumule dados do oponente antes de avaliá-lo, e suficientemente curto para que o agente consiga adaptar-se rapidamente às mudanças de comportamento do oponente.

A implementação do mecanismo de escolha de ações baseado em desafio proposto neste trabalho pode ser assim resumida. O agente inicia o jogo atuando no seu melhor desempenho. Como o espaço de ações possui 13 elementos (ações possíveis), o agente `CBRL` pode utilizar 13 níveis de desempenho, e inicia o jogo no nível 1 (o melhor, de acordo com o aprendizado). Durante o jogo, e enquanto estiver no nível 1, o agente escolhe as ações que tenham o maior valor na função ação-valor. A cada ciclo de avaliação, o nível de dificuldade é estimado de acordo com a função de desafio. Caso o jogo esteja muito difícil, o

agente diminui sua atuação para o nível 2 e passa a executar as ações que tenham o segundo maior valor em casa estado. O nível pode ser diminuído até que o jogo se torne balanceado ou até atingir o menor nível possível (no caso, 13). Se em algum momento a função de desafio indicar que o jogo está fácil, o nível do agente é progressivamente aumentado até o jogo voltar a ser balanceado ou atingir o nível 1. Quando a função de desafio indica que o jogo está no nível adequado, o agente não altera seu desempenho.

Como o agente CBRL também implementa a tarefa de aprendizagem por reforço, é importante ressaltar que seu conhecimento é sempre atualizado. A aprendizagem on-line permite ao agente adquirir conhecimentos específicos de cada oponente enfrentado, evoluindo sua competência de acordo com cada um. Dessa forma, caso o agente já esteja atuando no melhor nível e o jogo ainda esteja fácil (por exemplo, se o oponente for um jogador experiente), seu desempenho ainda pode ser melhorado de acordo com a evolução no aprendizado.

### 6.3.5. DSTC – *Dynamic Scripting Top Culling*

A abordagem de balanceamento proposta por Spronck *et al.* [2004b] é implementada pelo agente *Dynamic Scripting Top Culling* (DSTCPlayer). Esse agente possui, além da codificação do mundo, da função de desafio e do ciclo de avaliação (os mesmos utilizados pelo CBRLPlayer), uma base de conhecimento e um limite de atuação. A base de conhecimento é um conjunto de regras, do tipo condição-ação, no qual cada uma possui um peso associado. Diferentemente do agente SMPlayer, essas regras não são disjuntas: para um dado estado, mais de uma regra pode ter sua condição satisfeita.

Dessa forma, enquanto a implementação do agente SMPlayer pode ser baseada em uma grande sequência de condicionais, para o DSTCPlayer é necessário implementar, pelo menos, um motor de inferência simples [Christian 2002]. O motor desenvolvido carrega as regras na base de conhecimento através de um arquivo texto. As regras são codificadas como exemplificado na Figura 6, e todas são do tipo condição-ação, de modo que não é necessário processar o encadeamento de regras. Durante a execução, as condições de cada uma são analisadas com respeito ao estado atual. As regras disparadas (isto é, aquelas que têm sua condição satisfeita) são agrupadas em um conjunto temporário, e associadas a uma probabilidade de escolha a partir do seu peso. Dessa forma, regras com pesos maiores têm mais chance de serem executadas do que aquelas com pesos menores. Por fim, é escolhida uma regra a partir de uma distribuição uniforme de probabilidade.

A ação correspondente à regra escolhida é executada. Na próxima tomada de decisão, é analisado o retorno obtido, através do mesmo sinal de reforço utilizado pelos agentes TRLPlayer e CBRLPlayer: a diferença de pontos de vida entre agente e oponente.

Esse retorno é adicionado ao peso da regra, de modo que ações boas (que tenham tirado mais pontos do que perdido) tenham seu peso aumentado e ações ruins tenham o peso diminuído. Para evitar que uma única regra que tenha seu peso continuamente aumentado impossibilite que as demais sejam tentadas, os pesos de todas as regras da base de conhecimento são sempre normalizados para o intervalo  $[0; 1]$ .

O balanceamento dinâmico é obtido através da definição de um limite superior para as regras que podem ser executadas. Regras com peso maior do que o limite são descartadas da tomada de decisão, mas permanecem na base de conhecimento. A cada ciclo de avaliação, a função de desafio é aplicada e, de acordo com o resultado, o limite é aumentado (quando o jogo está fácil) ou diminuído (quando está difícil) por um fator fixo. Como os pesos das regras estão sempre no intervalo  $[0; 1]$ , esse fator de incremento foi definido, heurísticamente, como 0,05. Desprezando esse limite, temos um agente que implementa o mecanismo de scripts dinâmicos clássico (*dynamic scripting*).

As regras do agente DSTCPlayer foram codificadas a partir das regras do SMPlayer, com a adição de regras sub-ótimas. Esse agente não realiza treinamento off-line. Em vez disso, foram realizados testes preliminares com os agentes SMPlayer e RDPlayer para verificar a qualidade das regras codificadas. Com base nesses testes, foi definido um conjunto de aproximadamente 35 regras e os seus respectivos pesos iniciais. Nesse conjunto estão regras aparentemente boas como “se o oponente estiver perto, deferir chute, com peso 1.0”, e regras não ótimas como “se a distância for mediana, fugir, com peso 0.3”. Em particular, há uma regra “ficar parado, com peso 0.2”, que pode ser executada em qualquer estado. Embora essa regra não faça muito sentido para um agente que tenta agir da melhor maneira possível, ela é muito importante para um agente que precisa atuar de maneira não ótima em alguns momentos.

### 6.3.6. GA – *Genetic Adaptive*

Para analisar a abordagem de balanceamento dinâmico proposta por Demasi e Cruz [2002], foi implementado um agente *Genetic Adaptive* (GAPlayer). Assim como o CBRLPlayer e o DSTCPlayer, esse agente possui uma descrição do mundo, uma função de desafio e um ciclo de avaliação.

Além desses atributos, o agente genético implementa um motor de algoritmos genéticos [Eiben e Smith 2003]. Esse motor possui um conjunto de indivíduos (equivalente à população) e funções de evolução (cruzamento, mutação e elitismo). Os cromossomos dos indivíduos são codificados como conjuntos de pares estado-ação. Para todos os estados possíveis de serem percebidos (de acordo com a codificação do ambiente), existe um gene no

cromossomo que indica a ação a ser realizada naquele estado. Dessa forma, cada indivíduo possui um cromossomo e o seu número de genes é igual ao tamanho do espaço de estados.

O comportamento do agente GAPlayer é determinado por um dos indivíduos da população, que é periodicamente mudado ou evoluído. A cada tomada de decisão, o agente consulta o mapeamento estado-ação desse indivíduo e executa a ação correspondente ao estado atual.

O mapeamento estado-ação é determinado pela evolução genética. Inicialmente, os indivíduos são iniciados com genes aleatórios (para cada estado, é atribuída randomicamente uma ação). A cada momento, um dos indivíduos da população é utilizado como base do comportamento. Periodicamente, esse indivíduo é analisado, através de uma função de aptidão (*fitness*) e substituído por outro indivíduo da mesma população. Cada população possui 10 indivíduos. A função de aptidão utilizada é a mesma que serve como sinal de reforço nos agente TRLPayer e CBRLPlayer e como retorno do agente DSTCPlayer: a diferença de pontos de vida entre o agente e o oponente.

Após todos os indivíduos de uma população serem testados, ocorre uma evolução. A evolução consiste em gerar uma nova população com o mesmo número de indivíduos. Os novos indivíduos são gerados por cruzamento (80%), por elitismo (10%) e aleatoriamente (10%). O cruzamento corresponde a utilizar dois indivíduos da população atual como pais, escolher aleatoriamente um ponto de divisão no cromossomo, e gerar um filho com parte dos genes de um pai (antes do ponto de divisão) e parte do outro (depois do ponto de divisão). A escolha dos pais é baseada na aptidão de cada um: aqueles com maiores aptidões têm mais chance de serem escolhidos. Embora em um dado cruzamento os pais utilizados sejam sempre diferentes, um mesmo indivíduo poder ser usado como pai para gerar mais de um filho.

O elitismo consiste em replicar os melhores indivíduos da população atual, o que é definido pelas aptidões de cada um. Por fim, uma parte da nova população é composta por indivíduos gerados aleatoriamente. Para todos os filhos gerados, pode-se realizar uma mutação nos seus genes. Uma mutação ocorre em um par estado-ação com probabilidade 0.001; nesse caso, a nova ação correspondente a esse estado é escolhida aleatoriamente.

Embora os pares estado-ação sejam iniciados aleatoriamente, após algum tempo de treinamento os indivíduos conseguem aprender as melhores ações a serem executadas em cada estado e propagar esse conhecimento nas gerações futuras.

Para implementar o mecanismo de balanceamento dinâmico nesses agentes, foi preciso definir um conjunto de indivíduos modelos, que guia a evolução diante de cada oponente. Os modelos foram criados através de treinamentos off-line contra o agente RDPlayer (randômico). Após 50 lutas de treinamento, foram escolhidos 5 indivíduos com diferentes

aptidões. O agente GAPIayer utiliza esses modelos para guiar sua evolução diante de cada oponente. Dessa forma, a cada evolução, ao invés de se realizar o cruzamento entre dois indivíduos da mesma população, são utilizados como pais um dos modelos e o indivíduo atual. O modelo utilizado no cruzamento depende da função de desafio: caso o jogo esteja muito fácil, é utilizado um modelo com baixa aptidão; caso esteja difícil, é utilizado um com alta aptidão.

É importante ressaltar que o agente GAPIayer implementa a proposta de balanceamento de Demasi e Cruz [2002], que *não* utiliza a distinção entre competência e desempenho apresentada neste trabalho. Essa abordagem genérica pode ser implementada por diferentes técnicas, como aprendizagem por reforço (o que é feito pelo agente CBRLPlayer) e scripts dinâmicos (DSTCPlayer). Uma terceira opção seria a utilização de algoritmos genéticos, o que não é realizado por nenhum dos agentes deste trabalho.

## 6.4. Conclusões

Este capítulo apresentou os detalhes de implementação da nossa abordagem de balanceamento dinâmico e de outras propostas da literatura em um jogo de luta. Foram apresentados a arquitetura do ambiente e os principais aspectos da implementação de 6 agentes lutadores, que se baseiam em diferentes estratégias de comportamento: randômica, regras estáticas, aprendizagem por reforço tradicional, aprendizagem por reforço com desafio, scripts dinâmicos, e algoritmos genéticos.

O ambiente aqui apresentado é utilizado como estudo de caso para a realização de experimentos com os agentes desenvolvidos. Esses experimentos visam validar os agentes com relação aos requisitos especificados no Capítulo 2. O próximo capítulo apresenta a metodologia de avaliação utilizada nos experimentos e os resultados obtidos.



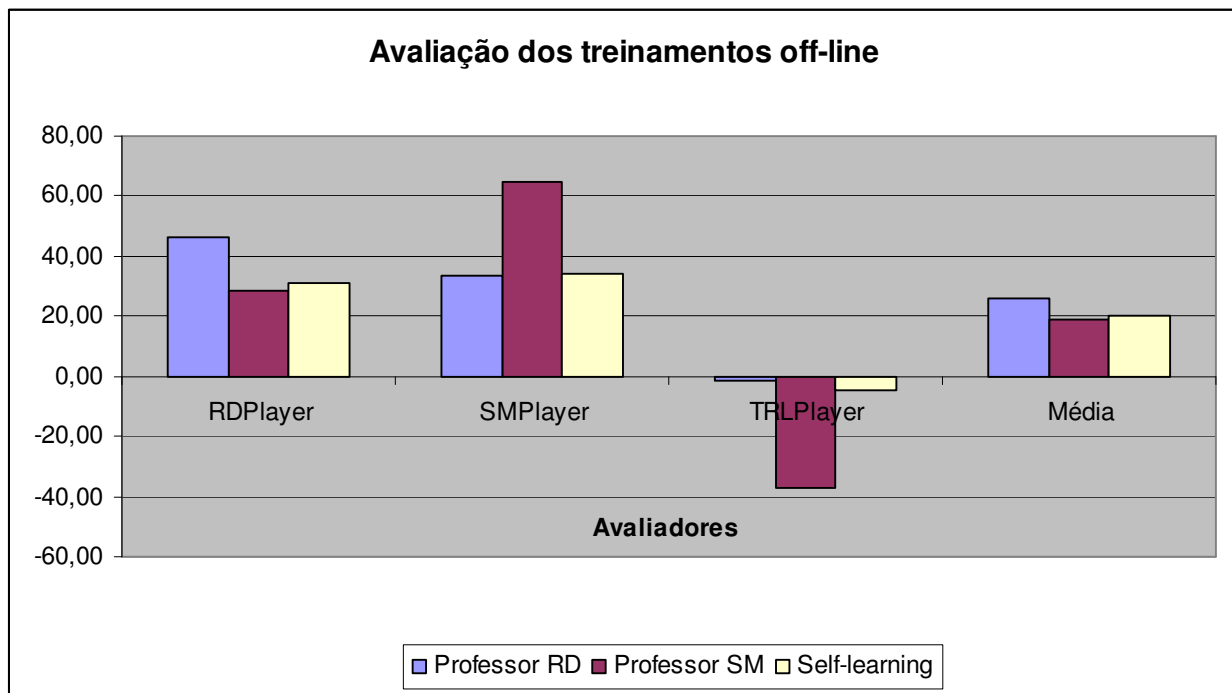
## 7. Experimentos: Agentes x Agentes

Este capítulo apresenta os experimentos realizados no jogo de luta Knock'em para analisar os agentes implementados no capítulo anterior. Primeiramente, são especificados os treinamentos off-line realizados para a aquisição da competência inicial nos métodos de aprendizagem. Em seguida, são apresentados os experimentos conduzidos para a avaliação do balanceamento. Por fim, é detalhado um conjunto de experimentos executados para inferir a capacidade de aprendizagem dos agentes que implementam o método proposto neste trabalho.

### 7.1. Treinamentos Off-line

O objetivo da realização de treinamentos off-line é garantir uma competência inicial para os agentes que implementam algum mecanismo de aprendizagem de máquina. Nesta seção, é analisado o treinamento de um agente TRL (*Traditional Reinforcement Learning*). O principal fator que influencia no treinamento é o adversário utilizado como professor. Como buscamos criar agentes capazes de enfrentar diferentes perfis de jogadores, o melhor professor deveria ser, em princípio, aquele que proporciona a maior *aprendizagem residual*, equivalente à retenção de conhecimentos genéricos, independentes de adversários.

Foram consideradas 3 opções de professores: o agente Randômico (RD), o agente baseado em regras estáticas (SM) e o próprio agente aprendiz (*self-learning*, TRL). Um agente TRLPlayer, sem conhecimento inicial, foi treinado, por 50 lutas, com cada um desses 3 professores. Esse agente utiliza uma política de exploração softmax, utilizando a função de Boltzmann com temperatura 1.0, e possui taxa de aprendizagem de 0,50 e fator de desconto de 0,90. Após a realização do treinamento, foram realizadas 5 lutas de avaliação contra cada um dos seguintes agentes: RDPlayer, SMPlayer, e TRLPlayer, este último com o conhecimento inicial obtido no treinamento *self-learning*. Mesmo durante as lutas de avaliação, os agentes aprendizes (TRLPlayer) continuam aprendendo. A idéia dessas avaliações é checar se o conhecimento aprendido com um professor específico é útil diante de outros adversários. A análise de desempenho é baseada na diferença de pontos de vida ao final de cada luta. Valores positivos representam vitórias do agente treinado, enquanto valores negativos representam derrotas. Os resultados são exibidos na Figura 14. Os 4 grupos nos eixos das categorias (eixo X) apresentam os resultados das avaliações contra 3 oponentes distintos e a média dessas avaliações. Para cada grupo, são exibidos os resultados considerando o agente TRLPlayer treinado contra um oponente RD, contra um oponente SM, e contra si mesmo (*self-learning*).



**Figura 14: Avaliação do treinamento off-line com diferentes professores**

Como pode ser observado, o agente treinado com um professor RD (primeira barra de cada grupo) possui seu melhor desempenho diante do avaliador RDPlayer (primeiro grupo). Entretanto, sua perda de desempenho diante do SMPlayer (segundo grupo) é bastante pequena, e, diante do TRLPlayer (terceiro grupo), sua atuação é similar à do avaliador (desempenho equilibrado). Por outro lado, o agente treinado com um professor SM (segunda barra de cada grupo) possui um desempenho excepcional diante do avaliador SMPlayer, mas não reflete tanta competência diante dos demais avaliadores. Por fim, o agente treinado contra o TRL (*self-learning*, terceira barra de cada grupo) possui um desempenho similar ao daquele treinado contra o RD. Na média (quarto grupo), entretanto, o melhor professor se mostrou o agente randômico (RD), que conseguiu ter bons desempenhos mesmo diante de avaliadores que utilizam estratégias diferentes das aprendidas na fase de treinamento off-line.

Os resultados obtidos são coerentes quando lembramos que o objetivo é maximizar a aprendizagem residual. Nesse caso, o treinamento contra um agente randômico se mostrou efetivo porque, como o professor não segue uma estratégia fixa, o conhecimento aprendido não é limitado a um único comportamento. Por outro lado, o treinamento contra um agente estático faz com que o aprendizado seja limitado à única estratégia utilizada pelo professor, o que pode não ser útil para enfrentar oponentes diferentes.

O problema de maximizar a aprendizagem residual está relacionado com a generalização do conhecimento aprendido. Os testes aqui realizados são apenas uma maneira simples e preliminar de lidar com esse problema. Outras possibilidades são a realização de treinamentos com diversos professores (por exemplo, realizar metade do treinamento contra um oponente randômico e a outra metade contra si mesmo) ou a utilização de algum mecanismo de aprendizagem capaz de generalizar o conhecimento aprendido (como redes neurais) [Haykin 1998]. Essas abordagens estão fora do nosso escopo, pois requerem uma análise aprofundada de diversos componentes que influenciam no treinamento, o que vem sendo feito por outros trabalhos no próprio Centro de Informática da UFPE [Dantas 2005].

Nos próximos experimentos, todos os agentes aprendizes utilizados são previamente treinados contra um oponente randômico por 50 lutas, a menos que explicitamente especificadas outras configurações. Esse professor é usado tanto para os agentes baseados em aprendizagem por reforço quanto para o agente baseado em algoritmos genéticos.

## **7.2. Avaliação do Balanceamento**

Esta seção apresenta os experimentos realizados para avaliar diferentes propostas de balanceamento dinâmico de jogos encontradas na literatura, incluindo a abordagem proposta neste trabalho. Primeiramente, é especificada a metodologia de avaliação utilizada. Em seguida, são apresentados e analisados os resultados obtidos.

### **7.2.1. Metodologia de Avaliação**

Esta seção apresenta a metodologia utilizada nos experimentos para avaliação do balanceamento. Esses experimentos possuem como objetivo básico validar os métodos de balanceamento dinâmico, analisando se agentes que implementam esse mecanismo efetivamente conseguem atuar no mesmo nível de jogadores com diferentes perfis. Nesses experimentos, partimos da pressuposto de que jogos que oferecem um desafio adequado são mais divertidos do que aqueles que não distinguem seus usuários. O processo desenvolvido consiste em realizar experimentos utilizando outros agentes inteligentes que simulam diferentes estratégias de atuação, isto é, diferentes perfis de jogadores humanos.

A grande questão em realizar testes com jogadores virtuais (implementados por agentes) consiste em definir os oponentes que simularão diferentes estratégias de atuação. Para isso, podem ser utilizadas diferentes implementações de agentes. Por exemplo, um agente simples aleatório poderia ser suficiente para simular um oponente iniciante. Um agente baseado em regras estáticas poderia simular satisfatoriamente um oponente mediano, mesmo que as regras criadas não sejam extensivamente testadas e melhoradas. A dificuldade maior é

simular um oponente inteligente e com capacidade de aprendizagem, que apresente evolução do comportamento no decorrer do tempo.

Considerando que dispomos de um agente que utiliza uma abordagem de balanceamento baseada na distinção entre competência e desempenho, esse último problema poderia ser a princípio resolvido por uma implementação de agente que não adapte seu desempenho, mas atue sempre no melhor nível de competência. Por exemplo, considerando que dispomos de um agente baseado em scripts dinâmicos, basta desprezarmos o limite de atuação (definido pelo *top culling*) para dispormos de um agente que atua de forma inteligente e que melhora seu comportamento no decorrer do tempo.

Nos experimentos de avaliação, foram analisados 5 agentes diferentes:

- **SM:** *State-Machine*, baseado em regras estáticas;
- **TRL:** *Traditional Reinforcement Learning*, atuando no melhor nível possível, de acordo com a aprendizagem;
- **CBRL:** *Challenge-Based Reinforcement Learning*, implementando a nossa proposta de balanceamento dinâmico com AR baseado em desafio;
- **DSTC:** *Dynamic Scripting Top Culling*, implementando a proposta de Spronck *et al.* [2004b], e;
- **GA:** *Genetic Adaptive*, implementando a proposta de Demasi e Cruz [2002].

Os agentes que implementam AR (TRL e CBRL) foram previamente treinados contra um oponente randômico. O agente genético utiliza 5 modelos de evolução (com diferentes níveis de aptidão) também construídos a partir de treinamentos off-line com oponentes randômicos. O agente DSTC, embora implemente um mecanismo de aprendizagem, não é previamente treinado, pois suas regras e os pesos de cada uma são manualmente ajustados. Todos os agentes que implementam algum mecanismo de aprendizagem de máquina continuam aprendendo durante as avaliações.

O objetivo dos experimentos é checar se os agentes que implementam algum mecanismo de balanceamento dinâmico efetivamente conseguem atuar no mesmo nível de diferentes oponentes (com estratégias de atuação distintas). Os resultados esperados são que o desempenho dos agentes CBRL, DSTC, e GA sejam sempre próximos ao desempenho dos oponentes, independentemente de suas características.

O cenário de avaliação consiste em uma série de lutas contra 3 oponentes, que simulam diferentes estratégias: um agente *State-Machine* (comportamento estático), um agente *Random* (comportamento imprevisível), e um agente *Traditional RL* previamente treinado e que se mantém aprendendo (inteligente e com capacidade de aprimoramento ao longo do tempo). Cada agente avaliado realiza 30 lutas contra cada oponente. A análise de desempenho é

baseada na diferença de pontos de vida ao final de cada luta. Valores positivos representam vitórias do agente avaliado, enquanto valores negativos representam derrotas. O objetivo, portanto, é que a diferença final seja próxima de zero, o que indica que os lutadores atuaram no mesmo nível.

Em cada avaliação, foram realizados Testes de Hipóteses (*p-value significance tests*) [Meyer 2000] para checar se a média dos pontos de vida de cada série é diferente de zero, considerando um nível de significância de 5%. A hipótese nula é que ambos os lutadores atuaram no mesmo nível, e portanto a diferença de desempenho *não* é significativa. Quanto mais baixo for o valor-p (*p-value*), maior é a evidência contra a hipótese nula [Lane 1999]. Em outras palavras, quando o valor-p (que varia entre 0 e 1) for menor do que 0,05 (o nível de significância), a diferença de desempenho é significativa. Nesse caso, um dos lutadores é melhor do que o outro (médias positivas indicam que o agente avaliado é melhor, enquanto médias negativas indicam que o oponente é melhor). Caso contrário, temos um indicativo de que ambos os lutadores atuam no mesmo nível.

### 7.2.2. Resultados e Interpretação

A Figura 15 apresenta o desempenho do agente *State-Machine* (SM) contra cada um dos 3 oponentes.

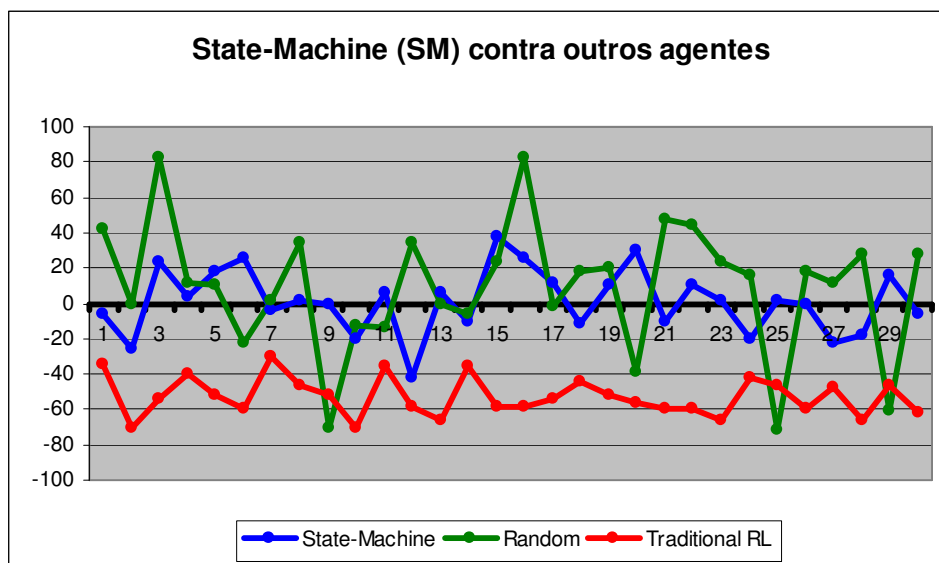


Figura 15: Desempenho do *State-Machine* (SM) contra outros agentes

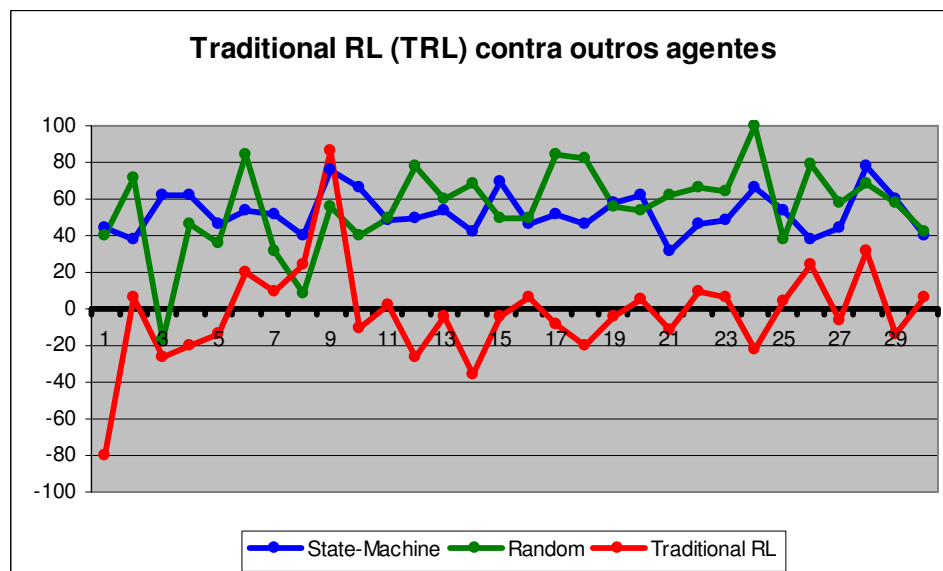
Os valores positivos contra o oponente *Random* indicam que o agente SM consegue derrotá-lo na maioria das lutas, mas eventualmente ainda é derrotado, como demonstram os valores negativos. Dois agentes SM lutando entre si apresentam um desempenho bastante semelhante. Por fim, os pontos negativos contra o oponente *Traditional RL* indicam que o

agente SM é derrotado por ele em todas as lutas. A Tabela 4 resume essa avaliação, e mostra que não há diferença significativa entre o desempenho do agente e dos oponentes *State-Machine* e *Random*. Por outro lado, o agente SM é significativamente pior do que o oponente *Traditional RL*, como mostra a média negativa dessa série de avaliações.

**Tabela 4: Análise de desempenho do agente *State-Machine* (SM)**

Oponente	Média	Desvio padrão	<i>p-value</i>	Diferença é significativa?
<i>State-Machine</i>	1,20	18,32	0,72	Não
<i>Random</i>	9,23	37,13	0,18	Não
<i>Traditional RL</i>	-52,73	10,92	0,00	Sim

A Figura 16 apresenta os dados da avaliação do agente *Traditional RL* (TRL). Esse agente derrota com uma certa facilidade os oponentes *Random* e *State-Machine*, o que indica que o treinamento prévio foi suficiente para enfrentar essas classes de lutadores. Como na análise anterior do *State-Machine*, o desempenho foi equivalente quando dois agentes idênticos (TRL) se enfrentaram. Esses dados são confirmados pela Tabela 5, que mostra que o agente TRL é significativamente melhor do que os oponentes *Random* e *State-Machine*.



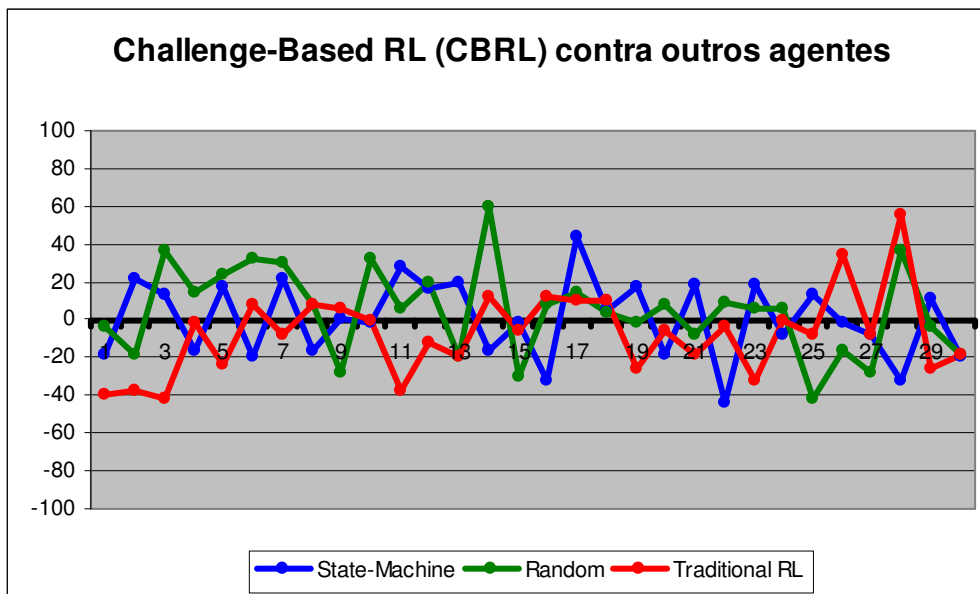
**Figura 16: Desempenho do *Traditional RL* (TRL) contra outros agentes**

A Figura 17 apresenta o desempenho do agente *Challenge-Based RL* (CBRL). Embora esse agente tenha a mesma competência do TRL, uma vez que os algoritmos de aprendizagem e o conhecimento inicial são os mesmos, a política de escolha de ações

baseada no desafio induz o agente a atuar no mesmo nível do oponente. Como pode ser visto na Figura 17, a maior parte das lutas desse agente termina com uma pequena diferença nos pontos de vida dos lutadores (valores próximos de zero), o que indica que ambos atuaram no mesmo nível. A Tabela 6 confirma esses resultados, mostrando que o agente CBRL não é significativamente diferente de seus oponentes.

**Tabela 5: Análise de desempenho do agente *Traditional RL* (TRL)**

Oponente	Média	Desvio padrão	<i>p</i> -value	Diferença é significativa?
<i>State-Machine</i>	52,47	11,54	0,00	Sim
<i>Random</i>	55,47	23,35	0,00	Sim
<i>Traditional RL</i>	-2,17	27,12	0,66	Não

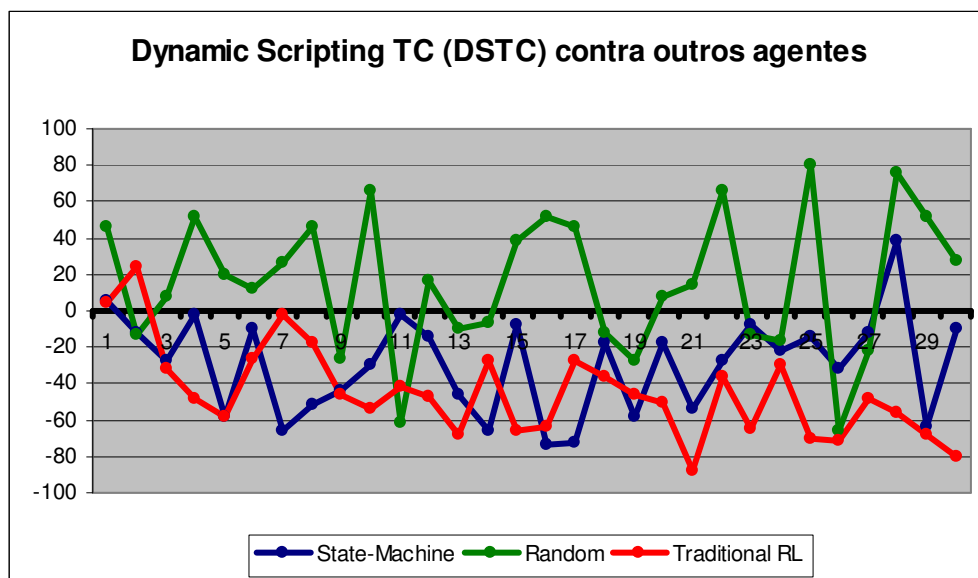


**Figura 17: Desempenho do *Challenge-Based RL* (CBRL) contra outros agentes**

**Tabela 6: Análise de desempenho do agente *Challenge-Based RL* (CBRL)**

Oponente	Média	Desvio padrão	<i>p</i> -value	Diferença é significativa?
<i>State-Machine</i>	0,37	20,67	0,92	Não
<i>Random</i>	4,47	23,47	0,30	Não
<i>Traditional RL</i>	-7,33	21,98	0,07	Não

A Figura 18 apresenta os resultados dos experimentos com o agente *Dynamic Scripting Top Culling* (DSTC). Os valores positivos e negativos contra o oponente *Random* mostram que o agente tenta se adaptar a ele, mas muitas vezes não consegue atuar no seu nível. Diante dos demais oponentes, o agente DSTC não consegue atuar satisfatoriamente, sendo constantemente derrotado. Diante do oponente *State-Machine*, é possível identificar uma leve evolução do agente no decorrer do tempo (conforme o aumento do desempenho nas últimas lutas). Já no caso do oponente *Traditional RL*, o perfil decrescente da linha de desempenho mostra que a competência do inimigo cresce ao longo do tempo, como resultado do processo de aprendizagem por reforço, o que acaba degradando o desempenho do agente avaliado. A Tabela 7 consolida esses dados, mostrando que o agente DSTC não consegue atuar no mesmo nível de nenhum oponente: é melhor do que o randômico e pior do que os demais. A alto desvio padrão nos experimentos contra o oponente randômico indica que o agente DSTC não conseguiu manter a consistência entre as lutas, intercalando desempenhos muito bons e muito ruins.



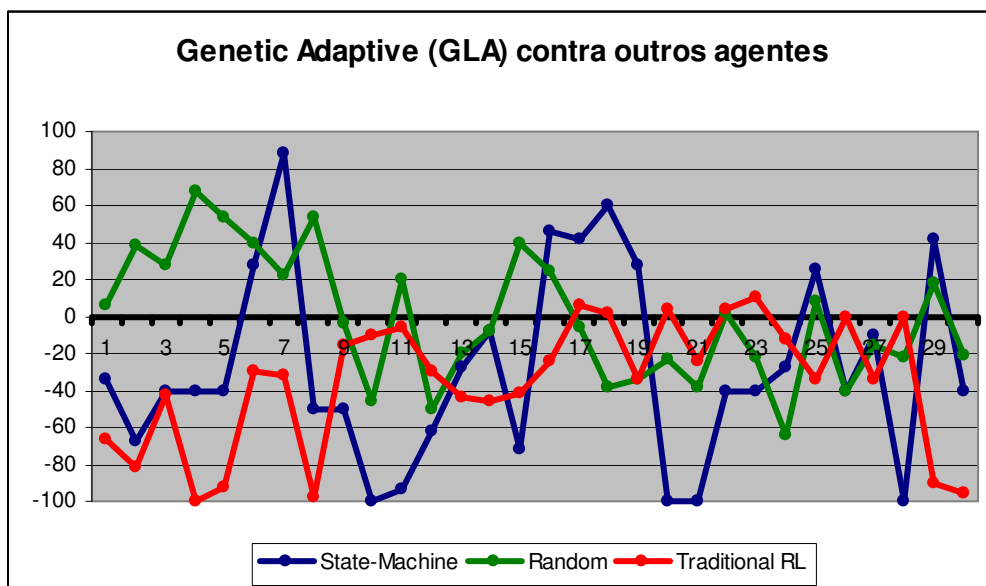
**Figura 18: Desempenho do *Dynamic Scripting TC* (DSTC) contra outros agentes**

**Tabela 7: Análise de desempenho do agente *Dynamic Scripts Top Culling* (DSTC)**

Oponente	Média	Desvio padrão	<i>p-value</i>	Diferença é significativa?
<i>State-Machine</i>	-29,27	27,10	0,00	Sim
<i>Random</i>	15,87	38,42	0,03	Sim
<i>Traditional RL</i>	-44,77	25,12	0,00	Sim



A Figura 19 apresenta os resultados do agente *Genetic Adaptive* (GA). Contra o oponente *State-Machine*, há uma grande variação no seu comportamento, com desempenhos muito bons e muito ruins (próximos dos limites 100 e -100, respectivamente). Diante do oponente *Random*, essa variação é menor e o agente consegue atuar, aproximadamente, no seu mesmo nível. Por fim, o perfil crescente da linha vermelha mostra que o agente genético efetivamente consegue evoluir no decorrer do tempo, melhorando seu desempenho mesmo diante de um oponente com capacidade de aprendizagem. A Tabela 8 resume esses resultados indicando que o agente GA não consegue atuar no mesmo nível dos oponentes *State-Machine* e *Traditional RL*, mas se aproxima do desempenho do *Random*. Os altos valores de desvio padrão em todos os experimentos indicam que o agente apresentou grande variação no desempenho entre lutas diante de um mesmo oponente.



**Figura 19: Desempenho do *Genetic Adaptive* (GLA) contra outros agentes**

**Tabela 8: Análise de desempenho do agente *Genetic Adaptive* (GLA)**

Oponente	Média	Desvio padrão	<i>p-value</i>	Diferença é significativa?
<i>State-Machine</i>	-27,50	51,88	0,00	Sim
<i>Random</i>	-1,00	34,64	0,87	Não
<i>Traditional RL</i>	-35,30	34,69	0,00	Sim

### 7.2.3. Análise dos Resultados

Os resultados apresentados anteriormente mostram que o agente CBRL consegue atuar, na média, no mesmo nível de seus oponentes, independentemente da competência de cada um. Por outro lado, o desempenho dos agentes DSTC e GA varia muito bruscamente conforme o oponente enfrentado. Dessa forma, a hipótese inicial de que toda implementação de balanceamento dinâmico atua no mesmo nível do oponente foi apenas parcialmente verificada.

Uma das limitações do agente DSTC que pode ter influenciado nos seus resultados é a dificuldade de codificação de suas regras de comportamento. Para obter um agente mais eficiente, seria necessário aprofundar a observação dos lutadores para identificar mais regras e estender a base de conhecimento utilizada. Além disso, para o estudo de caso utilizado, há uma certa dificuldade em codificar regras não ótimas. Em um jogo de luta, a maioria das ações se refere a golpes (ações ótimas de ataque) ou bloqueios (ações ótimas de defesa). É difícil codificar regras que façam o agente atuar de forma sub-ótima. As principais ações disponíveis para isso são ações de movimentação (se afastar ou se aproximar). Entretanto, apenas a criação de regras para cada uma dessas ações não garante uma adaptação satisfatória do desempenho. Para isso, é necessário escolher cuidadosamente os parâmetros de atualização dos pesos das regras, sob pena de se criar um agente inconsistente. Essa dificuldade causa a grande variação no desempenho entre lutas com um mesmo oponente apresentado pelo agente DSTC: ou o agente executa regras muito boas, ou executa regras muito ruins.

A dificuldade na codificação dos parâmetros para o agente DSTC é evidenciada por seu desempenho diante do agente SM. Embora as regras do SM sejam um subconjunto daquelas do DSTC, este possui regras não ótimas (inexistentes no outro) que podem estar sendo executadas por causa de uma inicialização imperfeita de seus pesos. Com o passar do tempo (cerca de 15 lutas), esse pesos são corrigidos pelo mecanismo de scripts dinâmicos (*dynamic scripting*). Entretanto, o fraco desempenho do DSTC nas lutas iniciais acaba por prejudicar sua avaliação como um todo, de forma que, no geral, seu desempenho ainda fica abaixo daquele do SM.

Por fim, os resultados evidenciam que embora esse agente seja capaz de atualizar o peso de suas regras e aprender a melhor forma de atuação contra cada oponente, essa aprendizagem é restrita às melhores regras existentes na base de conhecimento. Diante de um oponente que explora estratégias não previstas, como o agente TRL, o desempenho do DSTC pode sofrer degradação no decorrer do tempo, como pode ser observado pelo desempenho decrescente diante desse agente na Figura 18.

Analisando os resultados do agente *Genetic Adaptive* (GA), podemos identificar como sua principal limitação a grande variação de desempenho entre lutas contra um mesmo

oponente. Essa limitação decorre da dificuldade em controlar a evolução genética (que se baseia no cruzamento entre indivíduos). Mesmo utilizando modelos pré-definidos como um dos pais do cruzamento, não temos garantia de que os genes que serão replicados nos filhos são as melhores características. Como a função de aptidão está relacionada ao indivíduo como um todo, não sabemos a qualidade específica de cada um dos genes. Dessa forma, não sabemos previamente se estamos transmitindo características muito boas ou muito ruins aos filhos. Essa dificuldade faz com que o agente apresente grande variação de comportamento, mesmo quando, na média, consegue atuar no mesmo nível do inimigo (como no caso do oponente randômico).

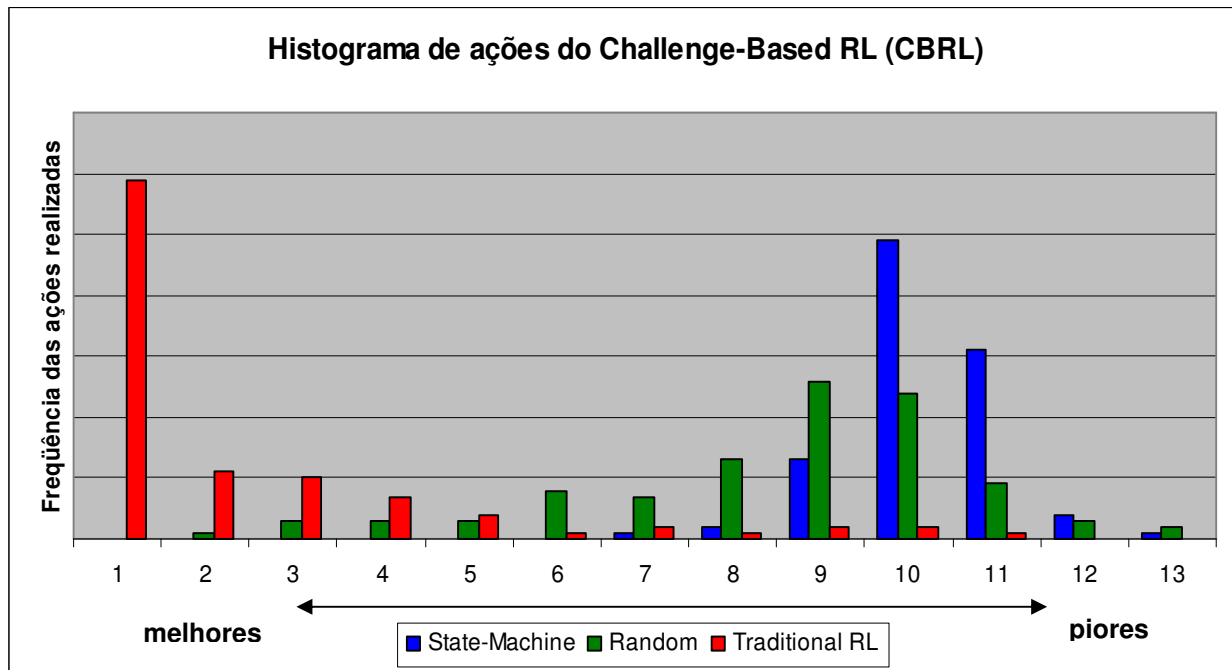
Essa variação de comportamento fica evidente no desempenho diante do oponente SM, quando o agente apresentou um desvio padrão bem acima dos demais. Nesse caso, são intercaladas atuações agressivas (com resultado próximo de 100) com atuações pífias (com resultado próximo de -100). A imprevisibilidade de comportamento do agente GA acaba sendo uma vantagem diante do oponente TRL, que aprende no decorrer do tempo. Como o agente GA executa uma estratégia bastante variável, a aprendizagem do TRL não consegue acompanhar rapidamente essas mudanças de comportamento, o que resulta na degradação de seu desempenho ao longo do tempo. Mesmo assim, o agente GA não consegue, no curto período de tempo das avaliações (30 lutas), evoluir suficientemente para atuar no mesmo nível dos oponentes SM e TRL.

O único agente adaptativo que conseguiu atuar adequadamente diante dos diferentes perfis de oponentes foi o agente CBRL. A adaptação do desempenho do CBRL é possível porque o agente intercala ações fáceis e difíceis, de acordo com cada situação. A Figura 20 apresenta um histograma que representa a frequência de ações executadas pelo agente CBRL em cada série de avaliação. Os 13 valores de cada série correspondem a todos os níveis em que o agente pode atuar. Os valores mais à esquerda são os níveis em que o agente executa as ações com maior resultado na função ação-valor, enquanto os mais à direita são as ações associadas aos menores valores.

A alta frequência de ações no nível 1, quando o agente enfrenta um oponente *Traditional RL*, indica que, na maior parte dessas lutas, o agente CBRL atuou no seu melhor nível. Por outro lado, as altas frequências de ações entre os níveis 9 e 11 quando são enfrentados os oponentes *Random* e *State-Machine* indicam que, nessas lutas, o agente CBRL atuou em um nível baixo.

Esse histograma mostra como a separação dos conceitos de competência e desempenho é eficiente para resolver o problema de balanceamento dinâmico de jogos. A atuação do agente CBRL é uma expressão direta de sua competência apenas quando se está

no nível 1. Nos demais casos, o desempenho do agente está abaixo de sua competência, de modo que sua atuação esteja no mesmo nível do oponente.



**Figura 20: Histograma de ações do agente *Challenge-Based RL* (CBRL)**

O agente CBRL consegue diminuir seu desempenho porque é capaz de identificar seqüências de ações que tenham resultado sub-ótimo. Por isso, não é necessário codificar explicitamente um conjunto de regras de qualidade mediana (ao contrário do agente DSTC), uma vez que o próprio agente descobre essas regras (caso existam) ou seqüências de regras boas e ruins que sejam mais adequadas a cada oponente. Essa capacidade de avaliar seqüências de ações é proveniente da aprendizagem por reforço, que considera não apenas o resultado da próxima ação, mas também todo o retorno obtido daí em diante.

### 7.3. Aprendizagem com Atuação Ótima Vs Sub-ótima

A execução de ações sub-ótimas pelo agente CBRL pode causar impacto na aprendizagem. Utilizando o algoritmo *Q-Learning* com um componente de exploração na política de escolha de ação ( $\epsilon$ -greedy ou softmax), temos um indicativo formal de que, com uma quantidade suficiente de iterações, a função ação-valor aprendida converge para a função ótima. Entretanto, na prática, o fato de executar ações não ótimas pode fazer com que o tempo necessário para atingir esse limite aumente. Esta seção apresenta alguns experimentos realizados para quantificar o impacto da escolha de ações sub-ótimas na aprendizagem por reforço. Em outras palavras, os experimentos aqui apresentados comparam a aquisição de conhecimento dos

agentes TRL (atuação ótima) e CBRL (atuação sub-ótima) através de suas curvas de aprendizagem.

Após a realização de algumas simulações iniciais, foi verificado que a maior evolução do aprendizado ocorre nas primeiras lutas de treinamento. Dessa forma, os parâmetros dos experimentos foram determinados como a seguir. Cada agente aprendiz executa 50 ciclos de simulação contra um dado oponente. Em cada ciclo, é realizada 1 luta de treinamento e 4 de avaliação. A aprendizagem (atualização da função ação-valor) só ocorre na luta de treinamento. As lutas de avaliação servem para medir o conhecimento acumulado adquirido com as lutas de treinamento. São realizadas 4 lutas para cada uma de treinamento para reduzir o impacto de ações randômicas (do agente e do oponente) na avaliação. Durante a avaliação, o agente CBRL sempre atua otimamente (equivalentemente ao TRL), uma vez que se procura mensurar a competência acumulada. Apenas durante as lutas de treinamento o CBRL pode realizar ações sub-ótimas.

O desempenho na avaliação é medido pela média da diferença de pontos de vida ao final de cada uma das 4 lutas. Para cada experimento, o conjunto de 50 ciclos é repetido por 10 séries. Em cada série, o agente não possui conhecimento inicial. Os resultados a seguir mostram a média de desempenho nas 10 séries contra os oponentes *State-Machine* (SM) e *Random* (RD). Como o objetivo dos experimentos é analisar o impacto de ações sub-ótimas na aprendizagem, foram escolhidos oponentes contra os quais o agente CBRL realiza ações não ótimas, como pode ser observado no histograma da Figura 20.

A Figura 21 apresenta a curva de aprendizagem dos agentes TRL e CBRL contra o oponente SM. Como pode ser observado, o agente CBRL apresenta um desempenho ligeiramente abaixo do TRL. Entretanto, a proximidade das curvas indica que a degradação da aprendizagem ocorrida com a escolha de ações sub-ótimas é bastante sutil. A curva de aprendizagem contra o oponente RD apresentada na Figura 22 mostra que, nesse caso, o desempenho é ainda menos prejudicado, o que é expresso pela sobreposição das curvas.

Esses resultados indicam que, na prática, a utilização de uma política de escolha de ações que possibilite a execução de ações não ótimas não apresenta grande degradação na aprendizagem do agente. Isso ocorre porque, embora as ações executadas não sejam as melhores, o reforço obtido expressa corretamente as suas qualidades, punindo ações ruins e recompensando ações boas. Dessa forma, além do indicativo teórico (pelo fato de usarmos um método de aprendizagem *off-policy*), temos um indicativo empírico de que a utilização de uma política de escolha de ações baseada em desafio não prejudica a aprendizagem por reforço.

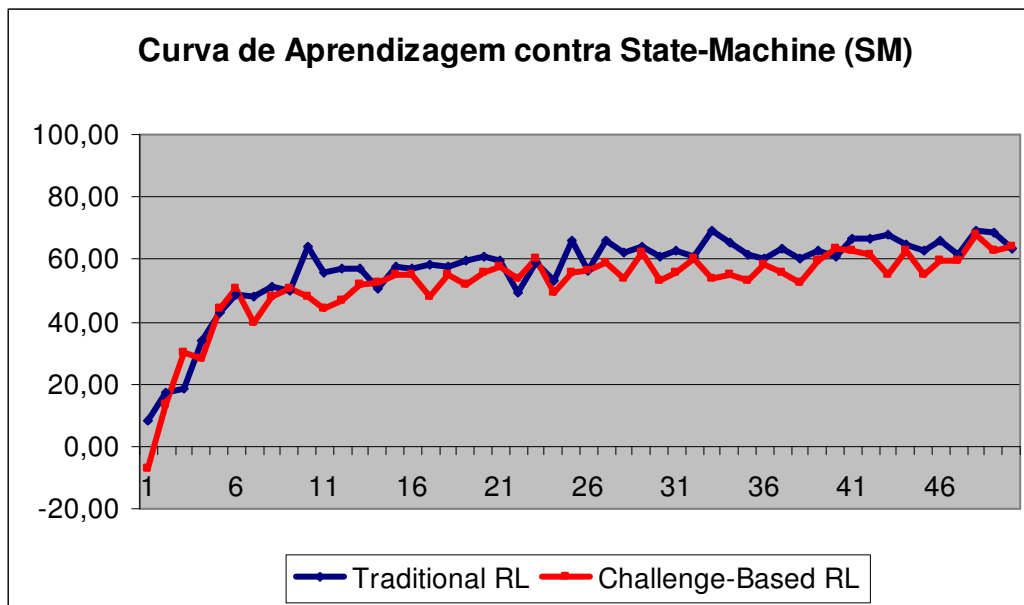


Figura 21: Curva de aprendizagem dos agentes TRL e CBRL contra oponente SM

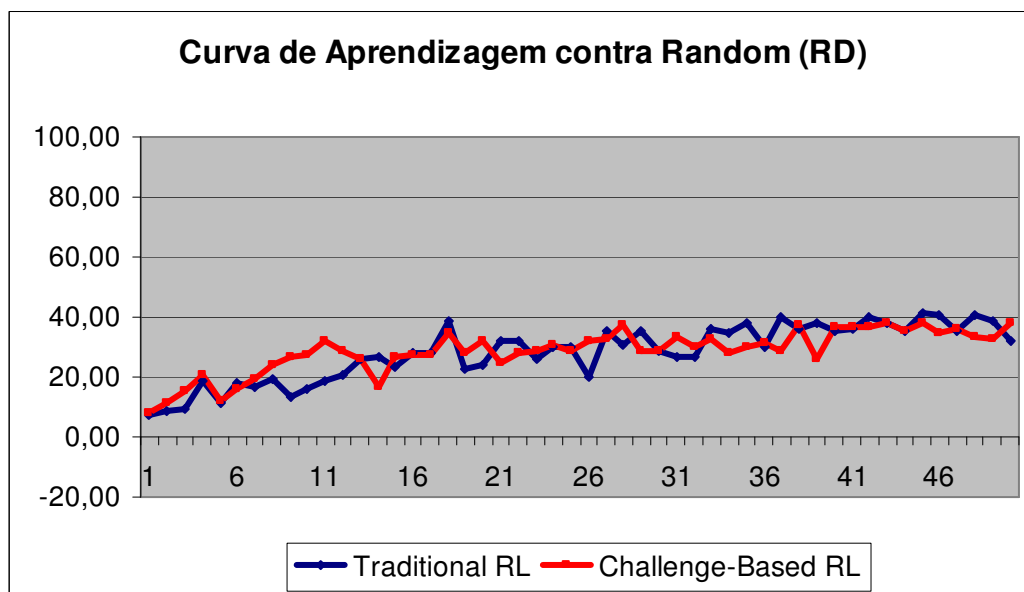


Figura 22: Curva de aprendizagem dos agentes TRL e CBRL contra oponente RD

## 7.4. Extensão a Outros Gêneros

Embora os experimentos tenham sido realizados em um jogo de luta, vale ressaltar que o método de balanceamento proposto neste trabalho é aplicável a outros gêneros. Em jogos onde existem NPCs, como os de Ação (*FPS* ou *Aventura*), Esportes, Estratégia e *Role-Playing*, os próprios agentes podem ser modelados de acordo com o método exposto: utilizando aprendizagem por reforço para desenvolver a competência e uma política de escolha de ações

baseada em desafio para adaptar o desempenho. A principal mudança em relação ao exemplo apresentado consiste na modificação das codificações de estados, ações e sinal de reforço dos agentes.

Em gêneros que utilizam ambientes complexos, como no caso dos jogos de Estratégia, a limitação da abordagem proposta reside na dificuldade, própria da aprendizagem por reforço, de lidar com grandes espaços de estados e ações. Nesse caso, pode-se optar por utilizar implementações que tenham capacidade de generalizar o conhecimento aprendido, como a substituição da matriz bidimensional do método *Q-Learning* por redes neurais. Outra opção é definir uma hierarquia de tomada de decisão (como estratégia, tática e operações) e utilizar a aprendizagem por reforço separadamente em cada uma. Em um jogo de guerra pode-se, por exemplo, realizar o balanceamento apenas no nível estratégico (quando atacar, quem atacar, com quem se aliar, etc.), e implementar as operações (que unidade utilizar em um ataque, que caminho adotar, etc.) por meio de outro método.

Alternativamente à mudança de comportamento dos NPCs, pode-se utilizar o método proposto para modificar as características do ambiente em um jogo. Nesse caso, a codificação de estados corresponde aos diferentes parâmetros que definem o ambiente e as ações equivalem a uma mudança desses parâmetros. Por exemplo, em um jogo como *Tetris*, do tipo puzzle, pode-se realizar o balanceamento pela modificação da velocidade de queda das peças ou pelos formatos apresentados em sequência. Assim, o estados corresponde aos valores desses parâmetros e as ações às modificações nesses valores.

Em resumo, abstraindo a forma de codificação e implementação do método proposto, podemos aplicá-lo a diferentes gêneros de jogos, com pouca modificação da idéias apresentadas neste trabalho.

## 7.5. Conclusões

Esse capítulo apresentou os experimentos realizados com agentes que implementam diferentes abordagens de balanceamento de jogos. Os testes, realizados com outros agentes que simulam diferentes comportamentos humanos, mostraram a capacidade de adaptação de cada um a diferentes perfis de atuação e revelaram a efetividade do método desenvolvido neste trabalho. Embora os experimentos tenham sido realizados em um jogo de luta, as idéias aqui propostas podem ser aplicadas com poucas modificações a outros gêneros de jogos.

## 8. Experimentos: Agentes X Usuários

Os experimentos do capítulo anterior validam a capacidade do método de balanceamento proposto neste trabalho em atuar no mesmo nível de oponentes com diferentes habilidades. Um passo complementar a essa validação é a avaliação do método diante de jogadores reais, com perfis diferentes daqueles utilizados nos experimentos anteriores.

Este capítulo apresenta os resultados obtidos com testes preliminares realizados por jogadores humanos, bem como a metodologia de avaliação desenvolvida para avaliar o balanceamento de jogos por meio de testes com usuários [Andrade *et al.* 2006]. Os experimentos realizados possuem como objetivo básico validar os métodos de balanceamento dinâmico, analisando se agentes que implementam esse mecanismo efetivamente conseguem atuar no mesmo nível de jogadores reais com diferentes perfis. O processo desenvolvido consiste na realização de testes com usuários em um ambiente controlado, no qual são coletados dados quantitativos e qualitativos.

### 8.1. Metodologia de Avaliação

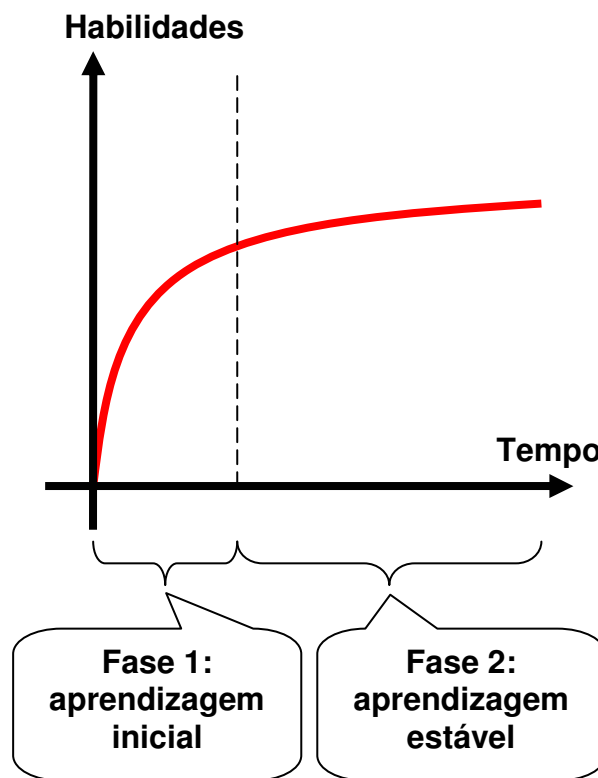
Para realizar testes com usuários, é necessário definir um plano de testes, criar um ambiente propício, e recrutar usuários representativos. O plano de testes consiste na especificação detalhada do teste, como as tarefas a serem executadas, os dados registrados e os métodos utilizados para coletá-los. Uma vez definido o plano, é necessário criar um ambiente que reduza o impacto de variáveis externas nos experimentos. Essas variáveis podem estar relacionadas ao ambiente físico do teste (como barulho e interrupções) ou ao ambiente virtual (por exemplo, a satisfação produzida por um balanceamento adequado pode ser mascarada pela frustração com uma interface deficiente). Por fim, é necessário recrutar usuários representativos do sistema analisado, e, possivelmente, acompanhá-los durante a realização dos testes.

O plano de testes define 5 elementos básicos: o objetivo dos testes, o perfil dos usuários (isto é, os testadores), as tarefas a serem executadas, as variáveis usadas para medir o resultado, e os métodos utilizados para coletá-las [Nielsen 1993].

O objetivo principal dos nossos testes é analisar o balanceamento e comparar diferentes abordagens existentes na literatura. Para isso, são usados como testadores jogadores reais, que expressem diversidade de habilidades. As únicas restrições aplicadas são aquelas dependentes do jogo utilizado no teste, e geralmente se relacionam a requisitos de idade (por exemplo, jogos com violência não são recomendados para crianças).



As tarefas executadas pelos testadores são pré-definidas e divididas em duas fases, de acordo com a Figura 23, que mostra a curva de aprendizagem dos usuários. Essa divisão é necessária porque no começo da interação com um sistema, as habilidades tendem a aumentar rapidamente, e apenas após algum tempo a evolução é estabilizada.



**Figura 23: Divisão das tarefas executadas nos testes com usuários**

Como uma das partes de nosso objetivo é a comparação de diferentes métodos de balanceamento, é importante realizar a análise comparativa apenas quando a aprendizagem estiver estabilizada. Caso contrário, o desempenho entre análises subsequentes pode ser profundamente influenciado pela evolução do próprio usuário.

A primeira fase dos testes é utilizada para checar o quão fácil é começar a interagir com o jogo (atributo *learnability*). Essa é uma questão crítica porque os usuários podem desistir de continuar jogando caso notem que seu começo é muito fácil ou muito difícil. Nessa fase, não são realizadas análises comparativas por um mesmo testador. A segunda fase é utilizada para realizar as principais medições do teste, como o nível de balanceamento, a comparação entre abordagens, e uma análise simplificada de satisfação.

Uma questão crítica nesse processo é a definição do ponto a partir do qual a aprendizagem do usuário se torna estável. Como cada um possui habilidades e experiências iniciais distintas, esse ponto é variável. Enquanto usuários iniciantes devem demorar mais

tempo para atingi-lo, os experientes tendem a ser mais rápido. O objetivo desse ponto não é garantir que, após ele, os usuários tenham as mesmas habilidades (o que pode ser muito difícil de se realizar), mas apenas que a evolução de cada um tenha se estabilizado. Uma abordagem simples que elaboramos para esse problema é a utilização de pontos de controle (*checkpoints*). Exemplos de *checkpoints* são: conquistar uma porcentagem das peças do oponente em um jogo de tabuleiro, derrotar um chefe intermediário em um jogo de ação, ou explorar o mapa completo em um jogo de estratégia. Após atingir o *checkpoint*, o usuário avança para a segunda fase.

Enquanto o usuário executa tarefas pré-definidas, diferentes dados que influenciam o balanceamento são coletados, como *scores*, tempo para atingir uma meta, eficiência, percepções e opiniões sobre o jogo. Para registrar todos esses aspectos, são utilizados testes controlados, questionários de satisfação, e entrevistas pós-experiência [Maguire 2001].

Testes controlados consistem em especificar versões do sistema e solicitar que usuários (representativos do universo real) realizem um conjunto de tarefas. O objetivo é coletar informações sobre o desempenho dos usuários e seus comentários durante a execução do teste. Os dados registrados podem ser genéricos (estarem relacionados com o jogo como um todo) ou específicos (estarem focados em uma questão definida, como o balanceamento). Para coletá-los, são utilizados *logs* de informações do sistema, gerados automaticamente durante o uso, e observações realizadas por um especialista. O especialista assiste passivamente o usuário, sem interferir em sua atuação, e registra dados como expressões, comentários ou reações ocorridas durante o teste. Outras possibilidades são a gravação em vídeo do teste ou a captura de todas as interações pelo próprio computador.

Os questionários de satisfação são utilizados para coletar informações subjetivas de forma estruturada. Consistem em questionários pré-definidos que os testadores devem responder antes, durante ou depois do teste. Através deles, é possível registrar informações sobre dificuldade percebida, aprendizagem do usuário, e nível de satisfação. Os questionários podem ser baseados em listas de opções e escalas *Likert* [Chin *et al.* 1988], o que permite análises estatísticas sobre os dados obtidos.

Para registrar dados não cobertos pelos métodos acima, são utilizadas entrevistas pós-experiências. Nessas entrevistas, o usuário tem a oportunidade de expressar livremente suas opiniões, sugestões, e sentimentos sobre o jogo. O objetivo é deixá-lo confortável para expor todas suas idéias. Portanto, embora uma lista de perguntas pré-definidas seja útil para guiar o entrevistador, pode-se inserir ou remover questões de acordo com o andamento da conversa.

A metodologia proposta nessa seção procura contemplar a coleta de dados concretos, semi-estruturados, e abertos, permitindo a análise de todas as dimensões que podem influenciar o balanceamento. A seguir, é apresentado como esta metodologia é aplicada no jogo de luta Knock'em [Andrade *et al.* 2005c].

## 8.2. Plano de Testes

Os próximos experimentos utilizam os mesmos agentes avaliados no capítulo anterior: SM, TRL, CBRL, DSTC, e GA. Alguns desses agentes tiveram seus parâmetros modificados para contemplar possibilidades de melhorias identificadas durante os experimentos anteriores.

Para o agente SM, essas modificações consistiram na adição de regras de comportamento que o tornasse menos previsível. Em situações onde há mais de uma ação eficiente que pode ser realizada, o agente escolhe uma delas randomicamente. Por exemplo, quando o oponente está próximo e parado, o agente pode deferir soco forte, soco rápido, chute forte ou chute rápido. Como todas são opções boas, a escolha entre essas 4 possibilidades é feita aleatoriamente. Entretanto, mesmo com essas novas regras, o agente SM ainda apresenta um comportamento estático em muitas situações.

As mesmas regras adicionadas ao agente SM foram adicionadas ao DSTC. Além disso, esse último ainda teve os pesos de algumas regras atualizados para corrigir distorções encontradas nos experimentos. Para os agentes TRL e CBRL, foram modificadas as políticas de exploração. Para os testes com usuários, esses agentes utilizam uma política softmax com função de Boltzmann e temperatura 1.0.

As tarefas executadas pelos usuários nos testes com o Knock'em são divididas em duas fases, conforme a Figura 23. Na primeira fase, quando a aprendizagem do usuário apresenta grande crescimento, cada usuário enfrenta apenas dois tipos de agentes: um treinador e um avaliador. O treinador é uma das 3 implementações de balanceamento dinâmico (CBRL, DSTC, ou GA) e é escolhido aleatoriamente pelo sistema. O avaliador é um agente fixo para todos os usuários, sendo usado como *checkpoint* para avançar para a próxima fase. Esse agente é um TRL, treinado previamente contra um oponente randômico.

A primeira fase tem duração mínima de 3 lutas contra o treinador, de modo que seja possível analisar a evolução do usuário logo no início da interação com o jogo. O critério de avanço para a segunda fase é derrotar, seguidamente, o treinador e o avaliador. Cada vez que o treinador é derrotado (após as 3 lutas iniciais), o usuário ganha o direito de enfrentar o avaliador. Caso o derrote, avança para a segunda fase. Caso seja derrotado, volta a enfrentar seu treinador até derrotá-lo e ganhar uma nova chance de enfrentar o avaliador. O objetivo desta fase é permitir que os usuários adquiram conhecimento do jogo (desenvolvam sua

competência por meio da prática) e estabilizem a sua aprendizagem. Após ela, esperamos que a variação nas habilidades dos jogadores seja pequena, possibilitando a realização de análises comparativas.

Durante a segunda fase, todos os usuários enfrentam 5 agentes: SM, TRL, CBRL, DSTC, e GA (em uma ordem aleatoriamente definida para cada usuário). Cada agente é enfrentado, seqüencialmente, por 5 lutas. Logo, embora a primeira fase tenha duração variável, a segunda tem a duração fixa de 25 lutas.

Todos os agentes avaliados são implementados em um personagem do jogo com as mesmas características do personagem controlado pelo usuário: força, resistência, etc. Para manter as condições de luta equivalentes, são realizadas apenas alterações de suas cores, como mostra a Figura 24. Apenas o agente avaliador (da fase de treinamento), usado como *checkpoint*, é implementado como um personagem diferente.



**Figura 24: Visual dos personagens utilizados nos testes com usuários**

Durante cada luta, a tarefa básica do usuário é vencer o oponente. As ações disponíveis são aquelas de controle do lutador, que variam de deferir um soco a lançar magia. O usuário pode executar, com o teclado, as mesmas ações dos agentes NPCs. Todas as ações são executadas através de uma única tecla, com exceção do lançamento de magia. Essa última é executada através de uma combinação seqüencial de 3 teclas, no estilo dos jogos clássicos de luta. O esforço adicional necessário para o lançamento de magia é recompensado pela possibilidade de fazer dano ao oponente sem precisar se arriscar ao se aproximar dele. Os controles do jogo (as teclas válidas e suas ações associadas) são informados ao usuário durante a fase de treinamento.

Em todos os testes, o usuário é acompanhado por um especialista, que é responsável por fazer a introdução, observá-lo enquanto executa as tarefas pré-definidas, e entrevistá-lo ao final. Na introdução, todos os usuários são informados de que estarão sendo monitorados, mas que o objetivo dos experimentos é analisar o sistema, e não os usuários, de forma que eles devem agir da maneira mais natural possível [Nielsen 1993]. Eles também são informados de

que todos os dados coletados são usados de forma que a identidade de cada jogador não seja revelada.

O teste em si é automaticamente conduzido pelo sistema, com as tarefas a serem realizadas integradas ao enredo do jogo. Enquanto o jogador executa as tarefas, o especialista observa passivamente suas ações, reações e expressões, sem oferecer nenhum tipo de ajuda. Todos os dados observados são registrados em uma folha de observação.

Após os testes, os usuários são entrevistados pelo mesmo especialista. Embora tenha sido definido um roteiro básico de perguntas (por exemplo, “Como avalia o jogo?”, “O que surpreendeu / decepcionou?”, “Quais as características de um bom jogo?”), o entrevistador tem liberdade para conduzir a entrevista inserindo ou removendo perguntas.

Além dos dados manualmente coletados, por meio das observações e entrevistas, algumas métricas (*logs* e respostas de questionários de satisfação) são automaticamente registradas pelo sistema. As variáveis registradas por meio dos *logs* são tempo decorrido, pontos de vida e eficiência (do agente e do jogador) ao final de cada luta. Para o agente CBRL, também é registrado o nível de desempenho utilizado. Os questionários de satisfação são aplicados após cada uma das duas fases. Além disso, é aplicado um questionário no início do teste para coletar o perfil de cada jogador. Todos os questionários, assim como o roteiro utilizado nas entrevistas e o modelo de folha de observação, encontram-se em anexo. A próxima seção apresenta os resultados obtidos com os testes.

### 8.3. Resultados e Interpretação

Os testes foram realizados com um conjunto de usuários de diferentes perfis. Excluindo os testes que não puderam ser aproveitados (devido a falhas do sistema ou do equipamento) sobraram os dados de 12 jogadores. Com base no questionário de auto-avaliação e na observação de seus comportamentos e dos *logs*, os usuários foram divididos em dois grupos: iniciantes (*casual players*) e avançados (*gamers*). Cada grupo possui 6 representantes.

O grupo iniciante é composto por usuários que não costumam jogar ou consideram-se com poucas habilidades em jogos de ação (luta, estratégia, FPS). Dessa forma, um membro desse grupo pode ser experiente em outros gêneros (RPG, estratégia, simuladores). O grupo experiente possui jogadores que se consideram habilidosos em jogos de ação ou que demonstram um bom desempenho durante o teste. A observação do comportamento também influi na definição do grupo à que um usuário pertence porque, em alguns casos, os jogadores respondem conservadoramente ao questionário de auto-avaliação. Assim, alguns jogadores que se avaliam como regulares, apresentaram desempenho superior (como pôde ser confirmado pela análise dos *logs*) a outros que se consideram avançados. Em geral, a análise

dos *logs* só foi necessária para classificar os usuário que se colocavam no nível intermediário do questionário de auto-avaliação: aqueles que se consideravam fracos foram classificados como iniciantes, e aqueles que se consideravam bons foram classificados como avançados.

A diferença de desempenho entre os dois grupos fica evidente nas primeiras interações do jogo. A duração da fase de treinamento, que depende de cada usuário, apresentou uma grande variação entre os grupos: enquanto os iniciantes realizaram, em média, 16,67 lutas nessa fase, os usuários avançados precisaram apenas de 7,5 lutas (em média) para avançar à fase de avaliação. O desempenho dos usuários, analisado por meio da diferença de pontos de vida ao final de cada luta, também apresentou variação entre os grupos. Durante as três lutas iniciais, quando cada usuário enfrenta um único tipo de agente (CBRL, DSTC ou GA), o desempenho médio do grupo iniciante é de -3,39 pontos, enquanto o do grupo avançado é de 28,11 pontos. Os valores negativos indicam que os iniciantes, em média, perdem do oponente treinador durante as 3 lutas iniciais, mas por uma diferença de pontos pequena. Por outro lado, os usuários avançados, já conseguem derrotar o treinador mesmo nas lutas iniciais.

Como cada usuário enfrenta apenas um tipo de agente (escolhido aleatoriamente) na fase de treinamento, a análise do desempenho individual de cada um não indicou resultados conclusivos. A escolha aleatória fez com que o agente CBRL enfrentasse 3 oponentes (todos avançados), o GA 4 (todos iniciantes) e o DSTC 3 avançados e 2 iniciantes. Dessa forma, embora o agente GA tenha tido melhor desempenho (média de -21,08 pontos) do que o agente CBRL (média de 22,67 pontos), esses resultados podem ter sido profundamente influenciados pelos perfis dos usuários enfrentados por cada um. A seguir, são apresentados os dados gerados durante a fase de avaliação, quando a aprendizagem de cada jogador está potencialmente estabilizada.

A Tabela 9 apresenta a média da diferença de pontos de vida (ao final de cada luta) entre os usuários e os agentes, durante a fase de avaliação. Os valores entre parênteses representam o desvio padrão.

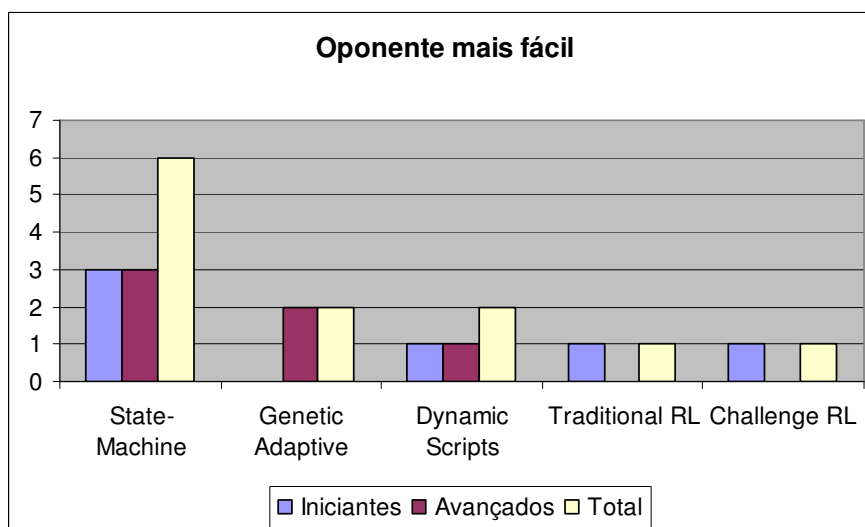
Valores positivos representam vitórias dos usuários (e valores negativos representariam derrotas). Quanto maior o valor, mais facilmente o usuário conseguiu derrotar o oponente. Como pode ser observado na Tabela 9, nenhum agente conseguiu, na média, derrotar os usuários (mesmo no grupo iniciante) durante a fase de avaliação. O agente que ofereceu maior dificuldade foi o TRL, que, na média, foi derrotado por 33,63 pontos. O agente mais fácil foi o SM, com média de 77,68 pontos. Esses agentes apresentaram uma grande variação entre os jogadores iniciantes e avançados, como pode ser observado pelas médias de pontos por grupo. Por outro lado, os agentes CBRL e DSTC mantiveram aproximadamente o

mesmo nível de desempenho, o que é expresso pelos baixos valores do desvio padrão das médias por grupo. Vale ressaltar ainda a alta variação do agente GA, que apresentou o mais alto desvio padrão dentre todos os agentes avaliados. Em resumo, os dados da Tabela 9 revelam que apenas os agentes CBRL e DSTC se adaptaram eficientemente ao oponente enfrentado, uma vez que seu desempenho não é significativamente alterado entre os diferentes grupos.

**Tabela 9: Análise das diferenças de pontos de vida durante a fase de avaliação**

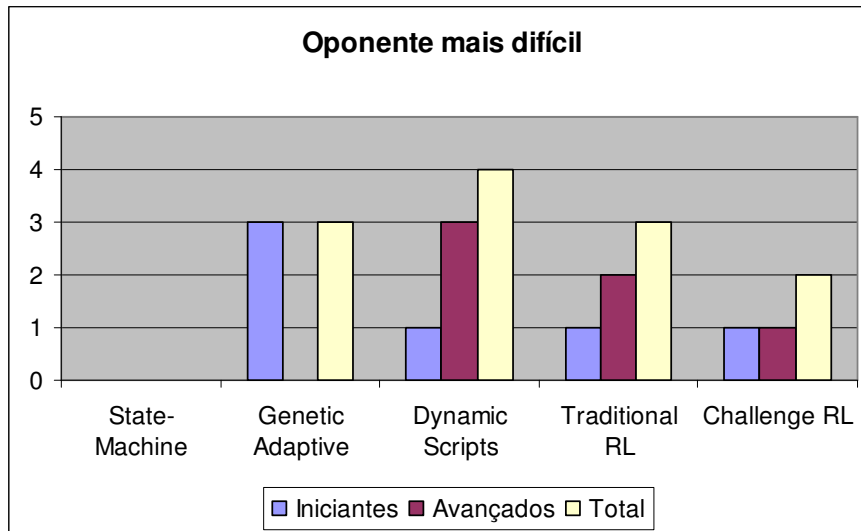
	Média (desvio padrão) por grupo		
	Iniciantes	Avançados	Total
State-Machine	68,10	87,27	<b>77,68 (13,56)</b>
Traditional RL	25,23	42,03	<b>33,63 (11,88)</b>
Challenge RL	42,57	41,27	<b>41,92 (0,92)</b>
Dynamic Scripts	44,10	49,30	<b>46,70 (3,68)</b>
Genetic Adaptive	36,13	67,27	<b>51,70 (22,02)</b>

As Figuras 25 e 26 apresentam alguns resultados do questionário de satisfação aplicado ao final dos testes, após a fase de avaliação. Na primeira, são apresentadas as respostas à pergunta “Qual o oponente mais fácil?”. Os resultados indicam que o agente SM é muitas vezes identificado como trivial, tanto por iniciantes quanto por usuários avançados. A Figura 26 apresenta as respostas à pergunta “Qual o oponente mais difícil?” e os resultados revelam que os agentes DSTC, GA, e TRL são mais associados a um nível difícil.



**Figura 25: Questionário de satisfação: “Qual o oponente mais fácil?”**

As respostas desses questionários mostram que o agente CBRL não foi referenciado, em geral, nem como o mais fácil nem como o mais difícil, sendo sempre o menos votado em ambos critérios. Esses resultados indicam que o agente é percebido pelos jogadores como o mais equilibrado dentre todos os analisados, atuando no mesmo nível de desempenho independentemente do perfil dos oponentes.



**Figura 26: Questionário de satisfação: “Qual o oponente mais difícil?”**

Além de escolher o oponente mais divertido, cada usuário também indicou as características desse oponente em termos de previsibilidade, inteligência e desafio. Para cada um desses fatores, foram respondidas questões (conforme questionários em anexo) através de uma escala Likert: (1) Concordo plenamente, (2) Concordo, (3) Indiferente, (4) Discordo, e (5) Discordo Plenamente. Os resultados mostram que o fator que mais influenciou na escolha do oponente mais divertido varia conforme o grupo. Para os iniciantes, a característica que mais se sobressaiu foi o *desafio* (isto é, o oponente mais divertido é desafiador), enquanto que para os usuários avançados, a característica mais importante foi a *imprevisibilidade* (isto é, o oponente mais divertido é imprevisível).

## 8.4. Análise dos Resultados

A seção anterior apresenta os resultados dos testes realizados a partir de uma divisão entre usuários iniciantes e avançados. Com base nas observações realizadas, é possível caracterizar esses grupos não apenas pela experiência prévia com jogos, mas também pelo comportamento demonstrado durante os testes.

Os usuários iniciantes aparentam ter como objetivo básico vencer o jogo o mais rapidamente possível. Durante os testes, esses usuários experimentavam algumas estratégias de atuação (opções de golpes) e assim que identificavam um padrão eficiente, passavam a



usá-lo durante todo o resto do jogo. Dessa forma, é possível identificar um pequeno conjunto recorrente de estratégias usadas pelos iniciantes: (1) aproximar-se, deferir um chute (rápido ou forte) e afastar-se; (2) manter distância e lançar magias; (3) encurralar o oponente e deferir rasteira. Todas essas estratégias são eficientes (em diferentes graus) e cada usuário iniciante utilizou uma delas (a primeira a ser descoberta), repetidamente, até o final do jogo.

Os usuários classificados como avançados, por outro lado, possuíam um comportamento mais explorador. Embora eles também identificassem essas estratégias eficientes (e, geralmente, mais rapidamente do que os iniciantes), foi observado que muitos preferiam testar outras possibilidades ao invés de repetir um mesmo padrão por todo o jogo. Mesmo que essa exploração resultasse em uma pequena degradação de desempenho, esses usuários preferiam tentar novos caminhos a repetir uma mesma estratégia por todo o teste.

Essas diferenças de comportamento ajudam a explicar os fatores que determinam as percepções dos usuários em relação aos oponentes. Dado que nenhum agente conseguiu desafiar suficientemente os usuários avançados na fase de avaliação, o fator que mais lhes surpreendeu foi a imprevisibilidade. Esses jogadores avançados gostavam quando, mesmo após explorar várias estratégias, não conseguiam identificar padrões estáticos no comportamento nos oponentes. Por outro lado, os iniciantes ainda foram mais influenciados pelo nível de desafio, mesmo não sendo suficientemente provocados pelos oponentes.

Um outro efeito dessas diferenças de comportamento é a aproximação do desempenho dos dois grupos. Por um lado, os iniciantes conseguem um bom desempenho porque, mesmo não conhecendo profundamente o ambiente do jogo, identificam algumas poucas estratégias eficientes e as re-utilizam por todo o teste. Por outro lado, embora os usuários avançados fossem capazes de (algumas vezes) atuar mais eficientemente, seus desempenhos são um pouco degradados pela utilização de estratégias não-ótimas para explorar o ambiente. Essas diferenças são um dos fatores que explicam a proximidade de desempenho dos dois grupos, expressa na Tabela 9.

A proximidade do desempenho também é influenciada pela eficiência da fase de treinamento. Como o treinamento é suficientemente longo para que cada jogador adquira bastante conhecimento sobre o jogo, após a primeira fase os usuários já possuem um nível de habilidade alto. Por isso, os agentes são incapazes de derrotar, durante a fase de avaliação, mesmo os usuários iniciantes. Essa análise é confirmada pelo exame do nível de desempenho do agente CBRL, isto é, a ordem das ações escolhidas em cada estado. Durante a fase de avaliação, o nível médio de atuação desse agente foi de 1,91, o que indica que o agente atuou quase otimamente (nível 1) contra todos os usuários.

O alto nível de atuação do agente CBRL também mostra que o treinamento off-line realizado com não foi suficiente para enfrentar jogadores humanos. A utilização de um método de treinamento mais eficiente, que enfatize diferentes estratégias de comportamento, poderia aumentar a competência inicial dos agentes aprendizes (CBRL, TRL, GA, e DSTC).

Em relação à fase de treinamento, muitos usuários comentaram que ela foi bastante útil para que fossem aprendidas estratégias boas de atuação. Em particular, alguns jogadores ressaltaram a qualidade do agente utilizado como avaliador (*Traditional RL*), que fornecia um bom desafio durante o treinamento. Esses jogadores evoluíram suas estratégias de luta após observar as ações mais eficientes adotadas pelo agente. Por exemplo, alguns jogadores só aprenderam a deferir chutes rasteiros após sofrerem muitos danos com esse golpe deferido pelo agente avaliador.

Uma vez analisados os comportamentos dos usuários, a seguir são analisados os padrões de comportamento dos agentes avaliados, com base nos resultados apresentados anteriormente, nas observações realizadas, e nos comentários dos usuários durante os testes e as entrevistas. Para cada agente, são apresentados os principais comentários (positivos e negativos) dos usuários e algumas possibilidades de melhoria elaboradas com base no *feedback* informado nas entrevistas.

### ***State-Machine (SM)***

O agente SM possui regras de comportamento eficientes, mas tem a grande limitação de nunca modificar sua atuação. A sua principal virtude é a coerência de comportamento. Como suas ações são determinadas por regras previamente desenvolvidas e testadas, temos a garantia de que sua atuação será consistente. Além disso, como alguns usuários iniciantes não conseguem perceber um padrão de jogo, as regras estáticas podem ser suficientes para que o agente seja percebido como um oponente divertido.

Entretanto, quando é descoberta uma falha em suas regras, o comportamento estático se torna desestimulante. Por exemplo, uma das regras codificadas expressa “se o oponente estiver a uma distância mediana, lançar magia”. O lançamento de magia demora, seguindo a lógica dos jogos de luta, alguns segundos (15 frames) antes que a bola-de-fogo seja lançada, e durante essa animação o lutador fica vulnerável a ataques. Dessa forma, é possível que o oponente se aproxime e defira um golpe enquanto o agente está vulnerável.

Os usuários avançados (e alguns iniciantes) percebiam, com o tempo, esse padrão e passavam a explorá-lo repetidamente. Como não há mudança de comportamento, alguns usuários descobriram que é possível derrotar o agente SM apenas usufruindo dessa situação. Esses usuários mantinham uma distância mediana, e quando o oponente começava a lançar

magia, se aproximavam, deferiam um golpe e voltavam a se afastar. Repetindo esse comportamento continuamente, muitos usuários venceram lutas sem perder nenhum ponto de vida.

Embora a demora do lançamento de magia seja uma característica do personagem (de modo que não pode ser evitada), os demais agentes conseguiam identificar quando essa ação se tornava ineficiente e adaptavam sua estratégia durante a luta. Por exemplo, embora o agente DSTC possua a mesma regra, após executá-la sem sucesso algumas vezes, diminui seu peso de modo que outra regra seja utilizada em seu lugar. Desse modo, esse agente (assim como os TRL, CBRL, e GA) não podia ser derrotado com a simples exploração de uma falha de comportamento.

A incapacidade de adaptação do agente SM é determinante na sua capacidade de entretenimento. Enquanto seus padrões não são identificados pelo usuário, o agente se mostra bastante divertido; porém, quando a repetição de padrões é percebida, se torna bastante desestimulante e previsível. Essa constatação foi comentada por alguns usuários durante as lutas de avaliação, os quais se dirigiram ao observador e ressaltaram: “Esse aqui é muito fácil. Vou derrotar ele só dando esse chute.”

### ***Traditional RL (TRL)***

O agente TRL foi, em geral, um oponente desafiador. O principal ponto positivo citado pelos usuários sobre este agente é a sua perceptível capacidade de adaptação.

A capacidade de adaptação é muito importante porque evita que esse agente caia em “armadilhas” do jogo (como no caso do lançamento de magias). Nesses casos, mesmo quando ele tenta executar uma ação ineficiente, após algumas repetições sua percepção sobre a qualidade dessa ação é atualizada e uma outra estratégia é adotada. Com relação à velocidade com que essa adaptação ocorre, muitos usuários comentaram que perceberam a mudança de comportamento do agente TRL durante os testes (e algumas vezes até durante uma mesma luta). Essas observações indicam que a aprendizagem por reforço é suficientemente eficiente mesmo diante de jogadores humanos.

A capacidade de adaptação do agente foi especialmente notada pelos iniciantes que repetiam uma mesma estratégia de atuação contra todos os oponentes. Nesses casos, os usuários demonstravam surpresa quando, após usar contra alguns oponentes uma única estratégia boa, percebiam que essa mesma estratégia não funcionava tão facilmente contra o agente TRL, que mudava seu comportamento para evitá-la.

O ponto negativo citado pelos usuários é que muitas vezes o agente TRL era pouco agressivo (atuava na defensiva). De fato, as observações revelaram que, em muitos casos, ele

mantinha uma grande distância do jogador e se limitava a lançar magias. Caso o oponente se aproximasse, ele reagia bem aos ataques e apresentava um bom desempenho no combate corpo-a-corpo. Mas caso contrário, o agente TRL não tomava a iniciativa e podia passar toda a luta sem se aproximar do inimigo. Essa atitude defensiva foi destacada como negativa porque transmite a idéia de que o lutador é fraco, embora em uma análise objetiva esta estratégia seja muito boa. Alguns usuários o destacaram como um lutador fácil por causa dessa passividade durante a luta. Esse resultado está de acordo com trabalhos anteriores [Yannakakis e Hallam 2005], que ressaltam a agressividade como um dos componentes que influenciam na satisfação dos usuários.

Essa limitação do comportamento é decorrente da codificação do sinal de reforço. Como o sinal de reforço não depende do tempo da luta, para o agente não há diferença entre vencer a luta em 10 segundos ou 1 minuto. A utilização de um sinal de reforço decrescente com o tempo pode induzir o agente a atuar mais agressivamente. Além disso, a realização de treinamentos off-line por um professor que induza o ataque pode ajudar a aumentar a agressividade do agente.

### ***Challenge-based RL (CBRL)***

O agente CBRL possui as mesmas vantagens e desvantagens do TRL. No caso da atitude defensiva, essa característica foi menos perceptível, uma vez que o mecanismo de adaptação de desempenho aumenta as variações de seu comportamento.

Do ponto de vista do balanceamento, esse agente foi algumas vezes criticado (durante as lutas contra usuários iniciantes) por realizar ações ineficientes (como ficar parado por algum tempo). Analisando os registros dos seus níveis de atuação, foi observado que o agente variava intensamente o desempenho (em uma mesma luta, podia variar do nível ótimo para o nível mais baixo possível). Embora, em média, seu desempenho fosse alto, essas variações talvez tenham prejudicado o entretenimento. Além disso, como jogos de luta são muito sensíveis a pequenas alterações de comportamento (alguns poucos segundos de vulnerabilidade pode ser suficiente para um lutador derrotar o inimigo), essa limitação também prejudicava o desempenho médio, como pode ser observado pela incapacidade em derrotar mesmo oponentes iniciantes. Por isso, o agente CBRL não consegue ser equivalente ao TRL diante do grupo iniciante (Tabela 9), embora estivesse atuando quase no seu melhor nível.

Essa instabilidade de comportamento pode ser amenizada por algumas alterações nos parâmetros do agente, de forma a provocar uma maior inércia na adaptação. Por exemplo, o aumento do ciclo de avaliação evita mudanças bruscas no desempenho e o estabelecimento

de um limite mínimo para o nível de atuação (como não atuar em um nível mais baixo do que 7, mesmo que o jogo esteja difícil) evita que ações incoerentes sejam executadas.

### ***Dynamic Scripting Top Culling (DSTC)***

O agente DSTC apresentou com principais vantagens a coerência de comportamento e a capacidade de adaptação. Como todas as suas ações são determinadas por regras pré-definidas (como no agente SM), não há o risco de que suas ações sejam bizarras. Além disso, como a atualização dos pesos consegue identificar quando uma regra se torna ruim, o agente também não cai em “armadilhas” do jogo.

A modificação dos pesos das regras ocorre suficientemente rápida, tendo sido percebida por vários usuários. Observou-se que, similarmente aos agentes TRL e CBRL, o DSTC também mudava de comportamento quando os jogadores insistiam em uma única estratégia eficiente (que fosse prejudicial para o agente). Essa adaptação foi apontada nas entrevistas como muito positiva para o entretenimento.

A limitação desse agente é a dificuldade de codificar suas regras de comportamento. Para melhorar o seu desempenho diante de usuários avançados e aperfeiçoar a sua capacidade de adaptação a iniciantes, é necessário criar mais regras (não-ótimas para estes e eficientes para aqueles). Entretanto, esse processo requer um estudo profundo das estratégias de atuação dos usuários e das melhores regras para lidar com essas situações, o que demanda um grande esforço de desenvolvimento.

### ***Genetic Adaptive (GA)***

O agente GA apresenta como principal característica a grande variação de comportamento, expresso pelo alto desvio padrão de seu desempenho (Tabela 9). Foi observado que esse agente intercala atuações bastante agressivas com outras bastante ineficientes. Por isso, alguns usuários o percebem como muito difícil e outros como muito fácil (Figuras 25 e 26). Além disso, a aprendizagem desse agente ocorre de forma mais demorada do que os demais, de modo que alguns usuários não chegavam nem a perceber a sua evolução nas 5 lutas de avaliação.

Para atenuar essa limitação, é necessário realizar uma mudança da codificação dos cromossomos dos indivíduos, diminuindo o espaço de busca da aprendizagem genética através da eliminação de estados não-essenciais. Embora com isso o agente possa perder a capacidade de se aproveitar de detalhes do oponente, a sua velocidade de adaptação tende a ser consideravelmente aumentada.

## 8.5. Conclusões

Os usuários demonstraram, em geral, contentamento com os testes realizados. Embora tenham sido observadas falhas nos comportamentos dos agentes, outros pontos surpreenderam as expectativas dos jogadores, como a adaptação do comportamento e a imprevisibilidade de atuação. Em especial, a maioria dos usuários percebeu diferenças de atuação entre agentes diferentes, o que confirma que cada técnica de implementação realmente produz um comportamento distinto.

Quando questionados, durante as entrevistas, sobre as características de um bom jogo, os usuários citaram diferentes fatores: nível de desafio oferecido, imprevisibilidade, eficiência dos controles, design gráfico, efeitos sonoros. Essas observações ressaltam a multidisciplinaridade de jogos e a necessidade de se estudar todos os seus elementos. O trabalho aqui apresentado proporciona uma contribuição relevante para um desses fatores (nível de desafio), mas ainda precisa ser estendido para aprofundar a análise das demais dimensões que influenciam na satisfação dos usuários, objetivo que está fora do nosso escopo.

Esse capítulo apresentou os experimentos realizados com agentes que implementam diferentes abordagens de balanceamento de jogos. Os testes com jogadores humanos (avançados e iniciantes) revelaram resultados interessantes sobre as vantagens e limitações das abordagens analisadas. Em especial, verificou-se que um agente baseado em regras estáticas pode ser bastante satisfatório para usuários iniciantes. Já diante de jogadores avançados, a capacidade de adaptação presente nos agentes baseados em aprendizagem por reforço ou scripts dinâmicos revelou-se uma característica essencial dos NPCs. Por fim, o agente baseado em algoritmos genéticos apresentou grande variação no comportamento, o que prejudicou sua avaliação pelos usuários.

Os testes com usuários servem não apenas para examinar a qualidade de um sistema, mas principalmente para coletar possibilidades de melhorias. No nosso caso, foi possível extrair novas idéias, consolidadas no próximo capítulo, que podem efetivamente aumentar o nível de entretenimento de jogos digitais. O próximo capítulo conclui todo o trabalho realizado e expõe algumas direções de trabalhos futuros.

## 9. Conclusões

Este trabalho abordou o problema de balanceamento de jogos digitais e, de forma colateral, sua relação com a satisfação sentida pelo usuário. Partimos de duas premissas: (1) o balanceamento está diretamente relacionado à satisfação do usuário; e (2) uma abordagem dinâmica é mais eficiente do que o balanceamento estático. A partir da definição do problema de balanceamento, seus requisitos e as limitações das atuais abordagens, fomos levados à necessidade de desenvolvimento de uma proposta alternativa de solução. Para isso, foi desenvolvida uma abordagem de balanceamento baseada na distinção entre competência e desempenho, que avalia o nível de dificuldade percebido por cada jogador e modifica o comportamento de agentes NPCs para ajustá-lo. Essa proposta permite várias implementações e tem o poder de explicar alguns sucessos encontrados na literatura. No nosso caso, a abordagem foi implementada através de aprendizagem por reforço, resultando em agentes capazes de aprender, automaticamente, as melhores estratégias de atuação em cada jogo, e adaptar, dinamicamente, o seu desempenho a cada usuário enfrentado. A distinção entre aprendizagem e atuação permite a concretização de uma idéia geral: repensar e sofisticar a política de escolha de ações dos agentes e explorá-la em diversas direções. Neste trabalho, foi implementada uma política que procura manter o desempenho do agente próximo ao nível do usuário, mas outras estratégias de atuação são possíveis de serem utilizadas.

A avaliação desse método foi realizada através de simulações com outros agentes. De forma a compará-lo com outras abordagens encontradas na literatura, também foram implementados e analisados outras propostas de balanceamento. Os resultados obtidos mostram que o balanceamento dinâmico baseado em aprendizagem por reforço consegue com sucesso atuar no mesmo nível de oponentes com diferentes perfis. Complementarmente, foram realizados testes preliminares com usuários que revelaram as vantagens e desvantagens de cada proposta, indicando possibilidades de melhoria para cada um dos métodos analisados. Esses resultados estão sendo consolidados em um único texto, voltado para sua publicação em periódicos internacionais, como o *ACM Computers in Entertainment* [ACM.org 2006]. A seguir, são detalhadas as principais contribuições deste trabalho realizadas durante esse processo, e, em seguida, são apresentadas algumas direções de trabalhos futuros para a extensão das idéias aqui apresentadas.

## 9.1. Principais Contribuições

A principal contribuição deste trabalho é apresentar uma abordagem original de balanceamento dinâmico de jogos. Essa tarefa abrangeu vários desafios que precisaram ser superados, resultando em contribuições relevantes para o estudo de jogos digitais:

- Definição de uma abordagem original de balanceamento dinâmico baseado na distinção entre competência e desempenho;
- Definição de uma implementação automática dessa abordagem através de aprendizagem por reforço;
- Criação e implementação de uma metodologia de testes para a avaliação do balanceamento e comparação das principais abordagens existentes na literatura.

Os resultados obtidos indicam o potencial de utilização de técnicas de aprendizagem em jogos digitais, que possibilitam a criação de agentes inteligentes sem a necessidade de codificação manual de seu comportamento.

O trabalho aqui apresentado pode ser aplicado em domínios diferentes de jogos. Por exemplo, o desenvolvimento de mecanismos de adaptação de desafios ao usuário é uma necessidade também em sistemas de tutores inteligentes (instrução assistida por computador) [Beck 1997]. Em ambos casos, a habilidade do usuário deve ser constantemente inferida para guiar a escolha de desafios adequados a serem propostos, sejam eles missões em um jogo ou questões e exercícios em um sistema de ensino.

## 9.2. Trabalhos Futuros

O trabalho aqui apresentado pode ser estendido por diferentes caminhos. Alguns apontam para melhorias nas abordagens aqui propostas, enquanto outros são opções de investigação identificadas com base nos resultados obtidos nos experimentos.

A primeira possibilidade de melhoria é o desenvolvimento de novas técnicas de treinamento off-line que aumentem a competência inicial dos agentes. Os testes com usuários revelaram que os treinamentos realizados com agentes randômicos não foram suficientes para lidar com jogadores humanos. Dessa forma, é necessário não apenas aumentar a duração das simulações, mas principalmente melhorar o oponente utilizado na aprendizagem off-line. Uma alternativa nessa direção é realizar treinamentos off-line com jogadores experientes, que ensinem os agentes a lidar com estratégias de atuação avançadas. O desafio dessa questão é definir como o conhecimento adquirido com jogadores diferentes pode ser combinado e consolidado em uma única representação que defina a competência básica do agente.

Outra possibilidade de melhoria é a implementação de algum mecanismo de antecipação das ações do oponente, que aumente a eficiência do processo de aprendizagem



por reforço. Embora o método *Q-Learning* seja capaz de criar um modelo implícito do usuário, alguns trabalhos já demonstraram que uma modelagem explícita é mais eficiente para a aprendizagem. Littman [1994] estende a função ação-valor do algoritmo *Q-Learning* para incluir as ações realizadas pelo oponente, criando uma função  $Q(s, a_{\text{agente}}, a_{\text{oponente}})$ , que estima o retorno esperado da ação  $a_{\text{agente}}$  no estado  $s$  considerando que o oponente execute a ação  $a_{\text{oponente}}$ . Essa função é utilizada em um algoritmo *Minimax-Q* que tenta maximizar o retorno do agente e minimizar o do oponente. Essa estratégia é útil em ambientes competitivos (como jogos) porque considera o impacto das ações do inimigo nos resultados das ações. Uther e Veloso [1997] combinam o *Minimax-Q* com um modelo estatístico do oponente (construído pela medição da frequência de ações realizadas) obtendo uma previsão ainda mais acurada do seu comportamento. Essa previsão é utilizada para, a partir da ação mais provável de ser executada pelo inimigo, escolher uma outra que resulte em um maior retorno para o agente. Embora ambos métodos possam ser utilizados em jogos, a abordagem de Uther e Veloso é mais promissora, diante dos custosos requisitos computacionais da proposta de Littman.

Outra opção de investigação é a integração de AR com outras técnicas de aprendizagem, como aprendizagem por observação (*learning by watching*). Nesse modelo, o agente observa as estratégias eficientes seguidas pelo seu oponente e as re-utiliza em seu comportamento. Em aprendizagem por reforço, isso equivale a atualizar o conhecimento do agente (a função ação-valor) considerando o reforço recebido pelo oponente. A aparente vantagem dessa técnica é que podem ser aprendidas mais rapidamente estratégias boas usadas pelo adversário (como fizeram vários jogadores durante os testes com usuários). Entretanto, algumas limitações também podem ser previstas, como a sua restrição de aplicação a jogos simétricos (como o Knock'em), em que o agente consiga identificar o reforço obtido pelo oponente.

A aprendizagem por observação pode ser implementada de diferentes maneiras. Uma opção simples é a criação de uma política de exploração (como  $\epsilon$ -greedy ou softmax) que favoreça a escolha das ações executadas pelo oponente [Dahlstrom e Wiewiora 2002], de modo que o agente periodicamente imite as estratégias dos adversários e analise sua qualidade. Uma outra opção consiste em utilizar diretamente o reforço obtido pelo oponente (sem necessariamente executar a sua ação) simultaneamente ao reforço obtido pelo agente na construção da função ação-valor. Nesse caso, é necessário adaptar o *framework* de AR para considerar as informações do adversário [Price e Boutilier 2003] e checar o seu impacto nas provas de convergência do algoritmo *Q-Learning*, uma vez que este se baseia na hipótese de que o reforço é consequência das ações do agente (e não do oponente).

Outra direção de trabalho futuro é refinar a metodologia de avaliação de balanceamento para realizar uma análise de satisfação detalhada. Outros autores têm trabalhado em métodos alternativos para analisar conceitos subjetivos como divertimento (*fun*) e desejo (*desirability*). Além de escalas *Likert*, pode se usar métodos como cartões de reações e questionário de imagens para inferir a satisfação sentida pelo usuário durante a utilização de um sistema [Benedek e Miner 2002]. O primeiro método consiste em usar cartões que expressam diferentes atributos (eficiente, estressante, flexível, etc.) e solicitar que cada usuário escolha aqueles que melhor representam seus sentimentos. O segundo consiste em substituir as perguntas dos questionários por imagens que expressem o atributo que se deseja medir (divertimento, desafio, frustração, etc.). Agregando esses métodos aos testes com usuários, pode se obter uma maior diversidade de informações sobre a satisfação proporcionada por cada técnica de balanceamento avaliada.

Por fim, é importante analisar como o método de balanceamento dinâmico baseado em aprendizagem por reforço desenvolvido neste trabalho se adapta a outros tipos de jogos, em especial naqueles onde os espaços de estados, de ações e a função de recompensa são mais complexas. Tais ambientes podem exigir implementações mais avançadas (como redes neurais para a função ação-valor) e mais esforço na realização dos treinamentos off-line.

As possibilidades de melhoria apresentadas anteriormente mostram que ainda podem ser estudadas muitas questões envolvendo o balanceamento de jogos. Este trabalho representa um apenas passo na evolução dos atuais métodos de solução do problema, proporcionando uma maior adaptação dos jogos aos seus usuários. O objetivo de todos esses esforços é construir sistemas de entretenimento cada vez mais inteligentes, usáveis, e, principalmente, divertidos.

# Referências

- [Abramson e Wechsler 2001] Abramson, M., e Wechsler, H. Competitive Reinforcement Learning for Combinatorial Problems. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 2333-2338, Washington, DC, 2001.
- [ACM.org 2006] ACM: Computers in Entertainment. <http://www.acm.org/pubs/cie.html>. (14/07/2006)
- [Andrade et al. 2006] Andrade, G., Ramalho, G. Gomes, A., e Corruble, V. "Dynamic Game Balancing: an Evaluation of User Satisfaction". In *Proceedings of the 2<sup>nd</sup> Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE'06)*, Marina del Rey, USA, AAAI Press, 2006.
- [Andrade et al. 2005a] Andrade, G., Ramalho, G., Santana, H., e Corruble, V. "Challenge-Sensitive Action Selection: an Application to Game Balancing". In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, pp. 194-200, Compiègne, France. IEEE Computer Society, 2005.
- [Andrade et al. 2005b] Andrade, G., Ramalho, G., Santana, H., Corruble, V. "Extending Reinforcement Learning to Provide Dynamic Game Balancing". In *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games* (eds. David W. Aha, Héctor Muñoz-Avila, and Michael van Lent), Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, pp. 07-12, Edinburgh, Scotland, 2005.
- [Andrade et al. 2005c] Andrade, G., Ramalho, G., Gomes, A., e Corruble, V. "Challenge-Sensitive Game Balancing: an Evaluation of User Satisfaction". In *Proceedings of the 4<sup>rd</sup> Brazilian Workshop on Computer Games and Digital Entertainment (WJogos'05) part of SBGAMES 2005*, pp. 66-76, São Paulo, 2005.
- [Andrade et al. 2005d] Andrade, G., Ramalho, G., Santana, H., Corruble, V. "Automatic Computer Game Balancing: A Reinforcement Learning Approach". In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pp. 1111-1112, Utrecht, The Netherlands, ACM Press, 2005.
- [Andrade et al. 2004] Andrade, G., Santana, H., Furtado, A., Leitão, A., and Ramalho, G. "Online Adaptation of Computer Games Agents: A Reinforcement Learning Approach". *Scientia*, 15(2), UNISINOS, Special Journal edition based on the best papers from "WJogos 2004", pp. 140-147, São Leopoldo, Brazil, 2004.

- [Andrade *et al.* 2003] Andrade, F., Andrade, G., Leitão, A., Furtado, A., e Ramalho, G. “Knock'em: Um Estudo de Caso de Processamento Gráfico e Inteligência Artificial para Jogos de Luta”. *II Workshop Brasileiro de Jogos e Entretenimento Digital*, Salvador, BA, 2003.
- [Barnes e Hutchens 2002] Barnes, J., e Hutchens, J. “Testing Undefined Behavior as a Result of Learning”. In *Steve Rabin, editor, AI Game Programming Wisdom*. Charles River Media. Hingham, MA, 2002.
- [Beck 1997] Beck, J., Stern M., e Woolf B. “Using the Student Model to Control Problem Difficulty”. In *Proceedings of Sixth International Conference on User Modeling*, pp. 277-288, Vienna, 1997.
- [Benedek e Miner 2002] Benedek, J., e Miner, T. “Measuring Desirability: New methods for evaluating desirability in a usability lab setting”. In *Proceedings of Usability Professionals Association Conference*, Orlando, 2002.
- [Chin *et al.* 1988] Chin, J., Diehl, V., e Norman, K. “Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface”. In: *Soloway, Elliot, Frye, Douglas, Sheppard, Sylvia B. (ed.): Proceedings of the ACM CHI 88 Human Factors in Computing Systems Conference*, pp. 213-218, Washington, DC, 1988.
- [Chomsky 1965] Chomsky, N. *Aspects of the Theory of Syntax*, The MIT Press, Massachusetts, 1965.
- [Christian 2002] Christian, M. “A Simple Inference Engine for a Rule-Based Architecture”. In *Steve Rabin, editor, AI Game Programming Wisdom*. Charles River Media. Hingham, MA, 2002.
- [Crawford 1982] Crawford, C. *The Art of Computer Game Design*, Osborne/McGraw-Hill, Berkeley, CA, 1982.
- [Crosby 1979] Crosby, P. *Quality is Free: the art of making quality certain*. McGraw-Hill, New York, 1979.
- [Csikszentmihalyi 1990] Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York. 1990.
- [Dahlstrom e Wiewiora 2002] Dahlstrom, D., e Wiewiora, E. “Imitative Policies for Reinforcement Learning”. Technical Report, Department of Computer Science and Engineering, University of California. Unpublished, 2002.
- [Dantas 2005] Dantas, V. *Integrando Aprendizagem por Reforço e Aprendizagem Supervisionada*. Recife: UFPE/CIn, Trabalho de Graduação, 2005.

- [Demasi e Cruz 2002] Demasi, P., e Cruz, A. "Online Coevolution for Action Games". In *Proceedings of The 3rd International Conference on Intelligent Games And Simulation*, pp. 113-120, London, 2002.
- [Demasi 2003] Demasi, P. *Estratégias Adaptativas e Evolutivas em Tempo Real para Jogos Eletrônicos*. Rio de Janeiro: UFRJ/IM/NCE, Dissertação de Mestrado, 2003.
- [Eiben e Smith 2003] Eiben, A., e Smith, J. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003.
- [Epstein 1994] Epstein, S., "Toward an Ideal Trainer", *Machine Learning*, 15(3), pp. 251-277, 1994.
- [Falstein 2004] Falstein, N., "The Flow Channel". *Game Developer Magazine*, Vol. 11, Number 05, 2004.
- [Fullerton et al. 2004] Fullerton, T., Swain, C., and Hoffman, S. *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. CMP Books, Gilroy, CA, 2004.
- [GameAI.com 2006] The Game AI Page: Building Artificial Intelligence into Games. <http://www.gameai.com/> (22/05/2006).
- [Graepel et al. 2004] Graepel, T., Herbrich, R., e Gold, J. "Learning to Fight". In *Proceedings of The International Conference on Computer Games: Artificial Intelligence, Design and Education*. Reading, UK, 2004.
- [Hastings 2005] Hastings, B. "Up Your Arsenal: On and Offline in Ratchet & Clank". *Game Developer Magazine*, Vol. 12, Number 2, 2005.
- [Haykin 1998] Haykin, S. *Neural Networks: A Comprehensive Foundation (2<sup>nd</sup> edition)*. Prentice Hall, New Jersey, 1998.
- [Hunicke 2005] Hunicke, R. "The Case for Dynamic Difficult Adjustment in Games". In *Proceedings of the ACM SIGCHI Conference*, Valencia, 2005.
- [Hunicke e Chapman 2004] Hunicke, R., e Chapman, V. "AI for Dynamic Difficulty Adjustment in Games". *Challenges in Game Artificial Intelligence AAAI Workshop*, pp. 91-96, San Jose, 2004.
- [Kaelbling et al. 1996] Kaelbling, L., Littman, M., e Moore, A. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*, pp. 4:237-285, 1996.
- [Koster 2004] Koster, R. *Theory of Fun for Game Design*, Paraglyph Press, Phoenix, 2004.
- [Kovacs 2002] Kovacs, T. "Learning Classifier Systems Resources". *Journal of Soft Computing*, No.6(3-4), pp. 240-243, 2002.

- [Lane 1999] Lane, D. *Hyperstat*, Atomic Doug Publishing, Mason, Ohio, 1999.
- [Langley 1997] Langley, P., "Machine Learning for Adaptive User Interfaces". *Kunstliche Intelligenz*, pp. 53-62, 1997.
- [Littman 1994] Littman, M. "Markov games as a framework for multi-agent reinforcement learning". *Machine Learning* 11, pp. 157–163, 1994.
- [Madeira et al. 2004] Madeira, C., Corruble, V., Ramalho, G., e Ratitch, B. "Bootstrapping the Learning Process for the Semi-automated Design of a Challenging Game AI". *Challenges in Game Artificial Intelligence AAAI Workshop*, pp. 72-76, San Jose, 2004.
- [Maguire 2001] Maguire, M. "Methods to support human-centred design". *International Journal on Human-Computer Studies* 55, pp. 587-634, 2001.
- [Manslow 2003] Manslow, J. "Using Reinforcement Learning to Solve AI Control Problems". In *Steve Rabin, editor, AI Game Programming Wisdom 2*. Charles River Media. Hingham, MA, 2003.
- [Mataric 1994] Mataric, M. "Reward Functions for Accelerated Learning". In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 181-189, New Jersey, 1994.
- [Merke e Riedmiller 2001] Merke A., e Riedmiller, M. "Karlsruhe Brainstormers - A Reinforcement Learning Approach to Robotic Soccer". *RoboCup 2001. RoboCup-2000: Robot Soccer World Cup IV*, pp. 435-440, London, 2001.
- [Meyer 2000] Meyer, P. "Probabilidade: Aplicações à Estatística". LTC Editora. Rio de Janeiro, RJ, 2000.
- [MGSUserResearch.com 2006] Games User Research at Microsoft Game Studios. <http://www.mgsuserresearch.com/> (27/04/2006)
- [Microsoft.com 2006] Microsoft DirectX: Home Page. [www.microsoft.com/directx/](http://www.microsoft.com/directx/) (10/05/2006)
- [Nielsen 1993] Nielsen, J. *Usability Engineering*. Morgan Kaufmann. San Francisco, CA, 1993.
- [Pagulayan et al. 2003] Pagulayan, R., Keeker, K., Wixon, D., Romero, R., e Fuller, T. "User-centered design in games". In *Human-Computer Interaction Handbook, J. Jacko and A. Sears*, Lawrence Erlbaum, pp. 883-905, Mahwah, NJ, 2003.
- [Price e Boutilier 2003] Price, B., e Boutilier, C. "Accelerating Reinforcement Learning through Implicit Imitation". *Journal of Artificial Intelligence Research* (19):569-629, 2003
- [Rabin 2003] Rabin, S. "Promising Game AI Techniques". In *Steve Rabin, editor, AI Game Programming Wisdom 2*. Charles River Media. Hingham, MA, 2003.

- [Ranlib 2006] Ranlib: Library of Routines for Random Number Generation Readme by Barry Brown and James Lovato. <http://lib.stat.cmu.edu/general/Utexas/ranlib.readme> (11/05/2006)
- [Robert e Guillot 2005] Robert, G., e Guillot, A. "A motivational architecture of action selection for non-player characters in dynamic environments". *International Journal of Intelligent Games & Simulation*, 4(1):1-12, 2005.
- [Robertson 2001] Robertson, S. "Requirements trawling: techniques for discovering requirements". *International Journal on Human-Computer Studies* 55, 405-421. 2001.
- [Rocha 2003] Rocha, E. *Forge 16V: Um Framework para Desenvolvimento de Jogos Isométricos*. Recife: UFPE/CIn, Dissertação de Mestrado, 2003.
- [Rollings e Morris 2003] Rollings, A., e Morris, D. *Game Architecture and Design: A New Edition*. Peachpit Press, Berkeley, CA, 2003.
- [Russel e Norvig 2004] Russel, S., e Norvig, P. *Inteligência Artificial*. Elsevier. Rio de Janeiro, 2004.
- [Santana 2005] Santana, H. *Patrulha Multi-Agente com Aprendizagem por Reforço*. Recife: UFPE/CIn, Dissertação de Mestrado, 2005.
- [Samuel 1967] Samuel, A. "Some studies in machine learning using the game of checkers II-Recent progress". *IBM Journal on Research and Development*, 11:601-617, 1967.
- [Saussure 1986] Saussure, F. *Course in General Linguistics*. Open Court, Illinois, 1986.
- [Scott 2002] Scott, B. "The Illusion of Intelligence". In *Steve Rabin, editor, AI Game Programming Wisdom*. Charles River Media. Hingham, MA, 2002.
- [Spronck et al. 2004a] Spronck, P., Sprinkhuizen-Kuyper, I., e Postma, E. "Online Adaptation of Game Opponent AI with Dynamic Scripting". *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 3, No. 1. University of Wolverhampton and EUROSIS, pp. 45–53, 2004.
- [Spronck et al. 2004b] Spronck, P., Sprinkhuizen-Kuyper, I., e Postma, E. "Difficulty Scaling of Game AI". In *Proceedings of the 5th International Conference on Intelligent Games and Simulation*, pp. 33-37, Belgium, 2004.
- [Sutton e Barto 1998] Sutton, R., e Barto, A. *Reinforcement Learning: An Introduction*. The MIT Press, Massachusetts, 1998.
- [Sweetser e Wyeth 2005] Sweetser, P., e Wyeth, P. "GameFlow: a Model for Evaluating Player Enjoyment in Games". *ACM Computers in Entertainment*, Vol. 3, No. 3, July, 2005.

- [Tesauro 1994] Tesauro, G. "TD-Gammon, a self-teaching backgammon program, achieves master-level play". *Neural Computation*, 6(2): 215-219, The MIT Press, Massachusetts, 1994.
- [Tozour 2002] Tozour, P. "The Perils of AI Scripting". In *Steve Rabin, editor, AI Game Programming Wisdom*. Charles River Media. Hingham, MA, 2002.
- [Uther e Veloso 1997] Uther, W., Veloso, M. "Adversarial Reinforcement Learning". Technical Report, Computer Science Department, Carnegie Mellon University. Unpublished, 1997.
- [Watkins e Dayan 1992] Watkins, C., e Dayan, P. "Q-learning". *Machine Learning*, 8(3):279–292, 1992.
- [Wiegand et al. 2002] Wiegand, R., Liles, W., e Jong, G. "Analyzing Cooperative Coevolution with Evolutionary Game Theory". In *Proceedings of the 2002 Congress on Evolutionary Computation*, IEEE Press, pp. 1600-1605, Honolulu, 2002.
- [Yannakakis e Hallam 2005] Yannakakis, G., e Hallam, J. "A Generic Approach for Generating Interesting Interactive Pac-Man Opponents". In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 94-101, Essex University, 2005.



# Anexos

## Anexo 1: Questionários e documentos usados nos testes

### Questionário 1

Idade (em anos)

---

- ☐ Menos de 15      ☐ 16 a 25      ☐ 26 a 35      ☐ 36 a 45      ☐ Acima de 46

Sexo

---

- ☐ Masculino      ☐ Feminino

Você gasta, em média, quantas horas por dia usando um computador?

---

- ☐ Menos de 2      ☐ 3 a 5      ☐ 6 a 10      ☐ Mais de 10      ☐ Não usa

Como você avalia sua experiência quanto ao uso do computador?

---

- ☐ Muito Ruim      ☐ Ruim      ☐ Regular      ☐ Bom      ☐ Muito Bom

Você costuma jogar jogos eletrônicos (em PC, consoles, etc.)?

---

- ☐ Sim      ☐ Não (*pode concluir o questionário*)

Há quantos anos você joga?

---

- ☐ Menos de 1      ☐ 2 a 5      ☐ 5 a 10      ☐ Acima de 10

Quantas horas por semana você joga?

---

- ☐ Menos de 2      ☐ 2 a 7      ☐ 8 a 14      ☐ 15 a 21      ☐ Acima de 21

Que gêneros você costuma jogar?

---

- |                                     |  |  |
|-------------------------------------|--|--|
| <input type="checkbox"/> Estratégia | <input type="checkbox"/> Aventura/Jogo de Plataforma         | <input type="checkbox"/> Simulação                         |
| <input type="checkbox"/> Luta       | <input type="checkbox"/> FPS ( <i>First Person Shooter</i> ) | <input type="checkbox"/> Esportes                          |
| <input type="checkbox"/> Educativo  | <input type="checkbox"/> RPG ( <i>Role Playing Game</i> )    | <input type="checkbox"/> Quebra-cabeças ( <i>Puzzles</i> ) |

Como você avalia sua habilidade em jogos eletrônicos em geral?

---

- ☐ Muito Fraca      ☐ Fraca      ☐ Regular      ☐ Boa      ☐ Muito Boa

Como você avalia sua habilidade em jogos de ação (Aventura/Luta/FPS/Esportes)?

---

- ☐ Muito Fraca      ☐ Fraca      ☐ Regular      ☐ Boa      ☐ Muito Boa

Quantas horas por semana você joga jogos de ação?

---

- ☐ Menos de 2      ☐ 2 a 7      ☐ 8 a 14      ☐ 15 a 21      ☐ Acima de 21

### **Questionário 2**

Como foi aprender a jogar o Knock'em?

---

- ☐ Muito Difícil      ☐ Difícil      ☐ Regular      ☐ Fácil      ☐ Muito Fácil

Com relação à afirmação: "A fase de treinamento foi demorada."

---

- ☐ Concordo plenamente  
☐ Concordo  
☐ Indiferente  
☐ Discordo  
☐ Discordo Plenamente

Com relação à afirmação: "A fase de treinamento foi divertida."

---

- ☐ Concordo plenamente  
☐ Concordo  
☐ Indiferente  
☐ Discordo  
☐ Discordo Plenamente

Como você avalia sua habilidade com o Knock'em?

---

- ☐ Muito Fraca      ☐ Fraca      ☐ Regular      ☐ Boa      ☐ Muito Boa

### **Questionário 3**

Qual dos lutadores foi o oponente mais fácil de derrotar?

---

- ☐ Koffman Greenleaf (verde)  
☐ Kandir Seawest (azul escuro)  
☐ Kay Reedneck (vermelho)  
☐ Koni Gray (cinza)  
☐ Kyro Ielou (amarelo)

Qual dos lutadores foi o oponente mais difícil de derrotar?

---

- ☐ Koffman Greenleaf (verde)  
☐ Kandir Seawest (azul escuro)  
☐ Kay Reedneck (vermelho)  
☐ Koni Gray (cinza)  
☐ Kyro Ielou (amarelo)

Qual dos lutadores foi o oponente mais divertido?

---

- ☐ Koffman Greenleaf (verde)
- ☐ Kandir Seawest (azul escuro)
- ☐ Kay Reedneck (vermelho)
- ☐ Koni Gray (cinza)
- ☐ Kyro Ielou (amarelo)

Com relação à afirmação: “O oponente mais divertido é previsível.”

---

- ☐ Concordo plenamente
- ☐ Concordo
- ☐ Indiferente
- ☐ Discordo
- ☐ Discordo Plenamente

Com relação à afirmação: “O oponente mais divertido parece inteligente.” (não é claramente ineficiente)

---

- ☐ Concordo plenamente
- ☐ Concordo
- ☐ Indiferente
- ☐ Discordo
- ☐ Discordo Plenamente

Com relação à afirmação: “O oponente mais divertido é desafiador.”

---

- ☐ Concordo plenamente
- ☐ Concordo
- ☐ Indiferente
- ☐ Discordo
- ☐ Discordo Plenamente

Como foi a experiência de jogar o Knock'em?

---

- ☐ Muito Boa
- ☐ Boa
- ☐ Regular
- ☐ Ruim
- ☐ Frustrante

### **Roteiro de Entrevista**

Como você avalia o jogo?

O que mais surpreendeu você no jogo?

O que mais lhe decepcionou no jogo?

Você identificou alguma forma de trapaça para vencer o jogo?

Em que o jogo é diferente de outros jogos que você costuma usar?

Descreva características típicas de um bom jogo para você.

Há mais alguma coisa que você gostaria de acrescentar?

### **Folha de Observação**

Ao início dos testes, explicar que o objetivo é analisar o *Jogo* e não o usuário, com o intuito de conhecer um pouco os fatores que influenciam a jogabilidade. Afirmar que ele(a) deve se sentir à vontade se não desejar responder alguma questão da entrevista ou não quiser que seja gravada. Explicar que todos os dados serão codificados de modo que no relatório, não será possível identificar os participantes pelo nome.

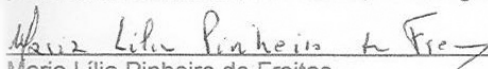
Data:	Hora de Início:
Local:	Hora de Término:




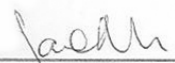
SERVIÇO PÚBLICO FEDERAL  
UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CENTRO DE INFORMÁTICA  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

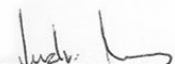
Ata de Defesa de Dissertação de Mestrado do  
Centro de Informática da Universidade Federal de Pernambuco, 21 de agosto de 2006.


Ao vigésimo primeiro dia do mês de agosto do ano dois mil e seis, às nove horas e trinta minutos, no Centro de Informática da Universidade Federal de Pernambuco, teve início a **quingentésima quadragésima sexta** defesa de dissertação de Mestrado em Ciência da Computação intitulada "**Balanceamento Dinâmico de Jogos: Uma Abordagem Baseada em Aprendizagem por Reforço**", do candidato **Gustavo Danzi de Andrade**, o qual já havia preenchido anteriormente as demais condições exigidas para a obtenção do grau de mestre. A Banca Examinadora, composta pelos professores Aluizio Fausto Ribeiro Araújo, pertencente ao Centro de Informática desta Universidade, Jair Cavalcanti Leite, pertencente ao Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, André Menezes Marques das Neves, pertencente ao Departamento de Design desta Universidade e Geber Lisboa Ramalho, pertencente ao Centro de Informática desta Universidade, sendo o primeiro presidente da Banca Examinadora e o último orientador do trabalho de dissertação, resolveu: **Aprovar por unanimidade e dar o prazo de trinta dias para a entrega da versão final do trabalho**. E para constar lavrei a presente ata que vai por mim assinada e pela Banca Examinadora. Recife, 21 de agosto de 2006.

  
\_\_\_\_\_  
Maria Lilia Pinheiro de Freitas  
(secretária)

  
\_\_\_\_\_  
Prof. Aluizio Fausto Ribeiro Araujo  
(primeiro examinador)

  
\_\_\_\_\_  
Prof. Jair Cavalcanti Leite  
(segundo examinador)

  
\_\_\_\_\_  
Prof. André Menezes Marques das Neves  
(terceiro examinador)

  
\_\_\_\_\_  
Prof. Geber Lisboa Ramalho  
(quarto examinador)