UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Trabalho de Graduação

Estudo sobre balanceamento dinâmico de jogos baseado em sistemas classificadores

por

Denys Lins de Farias

Recife, 13 de dezembro de 2011

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

Denys Lins de Farias

Estudo sobre balanceamento dinâmico de jogos baseado em sistemas classificadores

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Engenheiro da Computação.

Orientador: Geber Lisboa Ramalho

Recife, 13 de dezembro de 2011

Orientador

_____

Geber Lisboa Ramalho

Aluno

_____

Denys Lins de Farias

Recife, 13 de dezembro de 2011

Agradecimentos

À minha família, em especial à minha mãe, pela compreensão, carinho e cuidado com minha formação pessoal.

Aos amigos que já passaram e aos que ficaram pela experiência de vida que me proporcionaram.

À turma de Engenharia da Computação de 2011.2 pela caminhada acadêmica e conhecimento compartilhado.

Aos professores do Centro de Informática que contribuíram para minha formação acadêmica.

Ao meu orientador Geber pela inspiração e oportunidade de realizar este trabalho.

A Gustavo Danzi e a Carlos Maciel pelo apoio técnico.

Somewhere, something incredible is waiting to be known.

Carl Sagan

Resumo

O entretenimento do jogo está diretamente associado à capacidade de criar o engajamento dos jogadores, principalmente através de uma inteligência artificial equilibrada, mas desafiadora. Muitas abordagens propõem a contrução de mecanismos com o fim de gerenciar a dificuldade do jogador a partir de sua evolução. Este trabalho tem por objetivo estudar a aplicação de sistemas classificadores no contexto de balanceamento dinâmico de dificuldade em jogos.

Serão analisadas as principais características do *Zeroth-Level Classifier System* (ZCS) que favorecem a solução do problema, para então construir um agente adaptativo com essa estratégia e comparar seu desempenho com outras abordagens existentes em Inteligência Artificial através de um jogo de luta.

Palavras-chave: balanceamento de jogos, sistemas classificadores, Inteligência Artificial.

Abstract

The entertainment of the game is directly associated with the ability to create the engagement of players, mainly through a balanced, but challenging artificial intelligence. Many approaches propose the construction of mechanisms in order to manage the difficulty of the player from its evolution. This work aims to study the application of classifier systems in the context of dynamic game difficulty balance.

It will be analyzed the main features of the Zeroth-Level Classifier System (ZCS) that favor the solution of the problem, then built an agent with this adaptive strategy and compared its performance with other existing approaches in Artificial Intelligence through a fighting game.

Keywords: game balancing, classifier systems, artificial intelligence.

Sumário

Lista de figuras

Lista de tabelas

1. Introdução

Jogos são sistemas com os quais os jogadores interagem em um conflito artificial, definidos por regras que limitam o comportamento dos jogadores e estabelecem objetivos (SALEN e ZIMMERMAN, 2004). Assim sendo, a jogabilidade de um jogo está diretamente associada à interação e aos desafios proporcionados segundo regras.

Como uma atividade interativa, (JUUL, 2003) explica que um jogo apresenta respostas variáveis e quantificáveis às ações dos jogadores (i.e. pontuação obtida ao coletar itens, pontos de vida tirados do oponente com algum golpe), onde algumas ações são melhores que outras, fazendo com que o jogador empreenda esforço para obter as melhores respostas. Esse esforço representa a ligação entre o jogador e as repostas do jogo, onde em geral o jogador fica “feliz” ao ganhar em uma resposta positiva e fica “triste” ao perder em uma resposta negativa, caracterizando o engajamento do jogador (JUUL, 2003).

Assim, o engajamento está diretamente relacionado aos desafios apresentados ao jogador. Nesse contexto, a inteligência artificial (IA) de um jogo é responsável pela parte da jogabilidade associada ao gerenciamento dos desafios - quer seja no processo de tomada de decisão de oponentes controlados pelo computador, quer seja na configuração do ambiente com o qual o jogador deve interagir (PONSEN, 2004) – e, portanto, pelo nível de dificuldade percebido pelo jogador.

De acordo com (ANDRADE, G., 2006), a dificuldade do jogo deve ser equilibrada a fim de evitar os extremos de ter o jogador entediado com tarefas triviais ou frustrado com tarefas intransponíveis, mas tornando o jogo desafiador o suficiente para que o jogador sinta-se estimulado a explorá-lo continuamente. Para atender tais condições, faz-se necessário o balanceamento adequado do jogo, isto é, criar mecanismos que desafiem adequadamente os jogadores.

A forma tradicional de balancear um jogo consiste em estabelecer níveis fixos de dificuldade com desafios pré-definidos durante a fase de *game design* do jogo, conhecida como balanceamento estático. A complexidade do jogo e a grande diversidade de jogadores - cada qual com habilidade e conhecimento específicos, com formas e taxas de aprendizagem diferentes (ANDRADE, G., 2006) - podem

tornar o mapeamento dos níveis de dificuldade inviável durante a fase de desenvolvimento. Além disso, o estabelecimento prévio de estados e respostas às ações tende à previsibilidade da IA (PONSEN, 2004), o que possibilita que os jogadores explorem algumas características do jogo.

Uma alternativa é a utilização de mecanismos que gerenciem desafios dinamicamente a partir da evolução do jogador, através do ajuste de parâmetros, cenários e comportamentos, conhecido por balanceamento dinâmico (ANDRADE, G., 2006) ou por ajuste dinâmico de dificuldade (HUNICKE e CHAPMAN, 2004). Nessa abordagem, a inteligência artificial do jogo avalia a dificuldade percebida pelo jogador com base na experiência acumulada com o jogador para então executar a ação mais compatível com o seu nível.

Para a solução do problema de balanceamento dinâmico, diferentes técnicas de aprendizagem de máquina são estudadas atualmente, descritas brevemente no capítulo 3. Uma técnica pouco explorada no contexto de balanceamento de dificuldade em jogos é a de sistemas classificadores (BOOKER, GOLDBERG e HOLLAND, J.H., 1989). Ela se baseia em um sistema de produção com regras de pesos ajustáveis associadas a um motor de operações genéticas (seleção, mutação, reprodução e cruzamento). Este trabalho resultou no estudo e na avaliação da aplicação desta técnica no jogo Knock'em (ANDRADE, F. et al., 2003), comparando com agentes de outras abordagens e revelando suas qualidades e limitações.

1.1. Objetivos

Este trabalho tem por objetivo geral realizar um estudo sobre a aplicação de sistemas classificadores no contexto de balanceamento dinâmico de dificuldade em jogos. Serão analisadas as principais características da técnica que favorecem a solução do problema, comparando seu desempenho com outras abordagens existentes em Inteligência Artificial.

Para satisfazer o objetivo geral, foram necessários os seguintes objetivos específicos:

- Estudar os principais conceitos envolvidos no contexto do problema de balanceamento dinâmico de dificuldade;
- Revisar as técnicas mais utilizadas atualmente;

- Estudar fundamentos de sistemas classificadores, compreendendo a arquitetura e o comportamento do sistema;
- Implementar um agente adaptativo baseado em sistemas classificadores;
- Validar o desempenho da abordagem proposta comparado às demais abordagens através dos resultados obtidos no estudo de caso;
- Analisar os resultados para identificar vantagens e possíveis aperfeiçoamentos;

1.2. Estrutura do trabalho

Este trabalho está dividido em seis capítulos. O capítulo 2 apresenta uma descrição detalhada do problema de balanceamento, explicando mais sobre a abordagem tradicional, a abordagem dinâmica e sobre os requisitos associados à aplicação da solução. O capítulo 3 contém uma análise das principais técnicas aplicadas atualmente no contexto deste trabalho. O capítulo 4 contém uma explicação sobre a abordagem proposta, o ambiente do estudo de caso e a implementação realizada. O capítulo 5 concentra a metodologia dos experimentos e a análise dos resultados obtidos. O capítulo 6 expõe as conclusões encontradas e as indicações de melhorias para trabalhos futuros.

2. Balanceamento de jogos

(SPRONCK, PONSEN e POSTMA, 2006) define o balanceamento da dificuldade em jogos como a adaptação automática do jogo em ajustar o desafio que o jogo apresenta ao jogador humano. De acordo com ele, o balanceamento de dificuldade por parte da IA de um jogo tem por objetivo conseguir uma “partida acirrada”, onde a habilidade do computador no jogo se compara à habilidade do jogador.

O mecanismo de balanceamento deve garantir então desafios à altura do jogador, evitando que o jogador perca interesse no jogo devido a desafios fáceis ou difíceis demais. Por consequência, o balanceamento de dificuldade deve fazer com que os jogadores se baseiem principalmente em sua habilidade, devendo também atuar contra a exploração do comportamento da IA por parte dos jogadores, uma vez que eles podem tomar decisões que não correspondem à sua verdadeira habilidade no jogo.

Um exemplo é o de que em jogo *multiplayer* de tiro em primeira pessoa, o mecanismo de balanceamento não deveria permitir que um jogador iniciante pudesse ganhar vantagem sobre outros jogadores avançados apenas por possuir uma arma muito poderosa. O jogo deveria ser equilibrado então quanto aos atributos e possibilidades de ação de cada jogador.

Mais uma atribuição do mecanismo de balanceamento de dificuldade é o de observar a evolução do jogador, de forma a apresentar uma experiência mais adequada e menos previsível, conforme sugere (ANDRADE, G., 2006). Ele ainda afirma que o mecanismo de balanceamento deve tanto acompanhar a evolução quanto a regressão do jogador, que pode ocorrer, por exemplo, após um longo período de tempo sem interação entre o jogo e o jogador.

O responsável pelo processo de balanceamento durante o desenvolvimento de um jogo é o *game designer* (ANDRADE, G., 2006). Ele define parâmetros de itens e personagens, comportamento de agentes com os quais o jogador pode interagir, e cenários que imponham desafios e aumento de habilidade.

Todo o processo de balanceamento demanda por parte do *game designer* um estudo dos perfis para os possíveis jogadores e muitas rodadas de teste para alcançar o ajuste adequado da construção e balanceamento do jogo. Quão mais complexo o projeto do jogo, mais ele é propenso a erros e mais complexo o balanceamento.

O começo do jogo em particular tende a ser uma parte crítica no balanceamento, pois é quando há maior variação das habilidades dos jogadores. Além disso, é o momento do jogo com o qual um novo jogador tem sua primeira interação, definindo sua primeira impressão sobre o jogo e portanto seu interesse em continuar jogando ou não.

Nas duas seções seguintes serão descritas as duas principais abordagens para o problema de balanceamento de dificuldade em jogos.

2.1. Balanceamento estático

O balanceamento estático é a abordagem mais utilizada pela indústria (ANDRADE, G., 2006), onde níveis de dificuldades pré-definidos (i.e. fácil, médio e difícil) são apresentados ao jogador para que ele decida em qual dos níveis ele se encontra. Para cada nível, as técnicas desta abordagem definem os parâmetros relacionados aos desafios, como a força de um golpe em um jogo de luta ou o quanto de mana (magia) um golpe especial utiliza. A Figura 1 mostra uma tela com as opções de níveis de dificuldade possíveis oferecidas pelo jogo.

Figura 1 - Tela de balanceamento estático (fonte: Starcraft 2)

No balanceamento estático, o *game designer* tem controle sobre todo o processo de balanceamento, da construção dos elementos do jogo (i.e. um cenário, ou personagem) ao ajuste dos parâmetros e à forma como os desafios são apresentados. Geralmente os desafios são agrupados por fases ou estágios de dificuldade crescente, de forma a estabelecer etapas na evolução de habilidade do jogador. Isso impõe que os jogadores têm acesso a novas fases à medida que aumentam de habilidade pela nivelação imposta pelas fases anteriores (ANDRADE, G., 2006).

A figura 2 mostra como ocorre essa evolução da dificuldade por fases nesse tipo de abordagem:

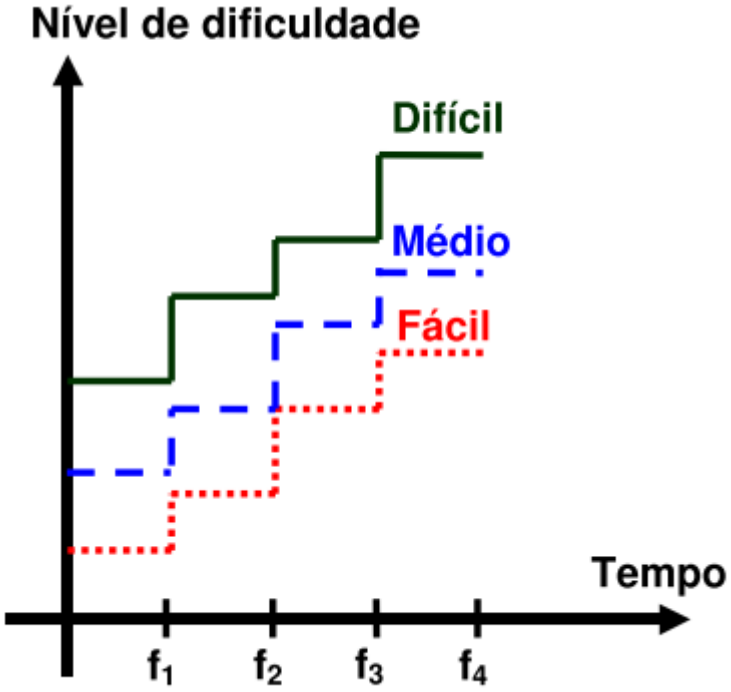

Figura 2 - Evolução do nível de dificuldade em jogos (Andrade, 2006)

As fases atuam então como barreiras de nivelamento, que servem para garantir que todos os jogadores que passarem por ela tenham um nível mínimo de habilidade (ANDRADE, G., 2006) e se sintam motivados a evoluir desenvolver sua habilidade. Assim o *game designer* sabe quando é adequado apresentar algum tipo de desafio dado as barreiras que o jogador conseguiu superar.

Como cada fase normalmente se baseia em um nível fixo de habilidade, torna-se inviável ao *game designer* mapear todas os níveis de habilidade dos jogadores. Assim, o jogo não considera as diferentes habilidades específicas e formas de aprender dos jogadores (ANDRADE, G., 2006). Há ainda para cada nível de habilidade a dificuldade por parte da equipe de desenvolvimento em realizar repetidos testes a fim de se ajustar adequadamente cada parâmetro do jogo. Devido a essa complexidade, é mais provável que a IA não consiga lidar com táticas não previstas e que seja explorada pelos jogadores para derrotar oponentes fortes (PONSEN, 2004).

(SPRONCK, PONSEN e POSTMA, 2006) ainda considera três problemas na seleção de níveis. Primeiro, os jogos apresentam um número limitado de opções de níveis de dificuldade (em geral três ou quatro). Segundo, os jogadores não

conseguem definir o nível de dificuldade mais apropriado para suas habilidades. Terceiro, a configuração da dificuldade de forma geral afeta apenas os atributos dos personagens controlados por computador, mas não sua tática - de modo que em um modo difícil de jogo, apesar de terem mais “força”, por exemplo, os oponentes exibem comportamento similar ao modo fácil. Ele sugere que esses três problemas podem ser resolvidos utilizando uma abordagem dinâmica.

2.2. Balanceamento dinâmico

A abordagem de balanceamento dinâmico consiste em utilizar mecanismos que automaticamente identifiquem as características específicas e o comportamento do jogador, e com base em sua experiência adapte os desafios ao seu nível de habilidade. Esses mecanismos são responsáveis por tornar a partida equilibrada em dificuldade, justa para todos os jogadores, e flexível em relação às novas e diferentes táticas que os jogadores podem adotar e às formas que podem aprender.

Além disso, como afirma (ANDRADE, G., 2006), esses mecanismos não devem apenas oferecer uma experiência mediana de dificuldade, mas também adicionar alguma imprevisibilidade, alternando entre desafios difíceis (os momentos de tensão) e os desafios fáceis (os momentos de relaxamento). Isso tem por objetivo estimular o jogador a evoluir e manter o contínuo interesse pelo jogo. A figura 3 ilustra bem esses conceitos, onde as regiões de jogo muito fácil ou muito difícil devem ser evitadas, a linha tracejada indica o balanceamento mediano previsível e a linha azul é um exemplo de comportamento balanceado imprevisível (ideal).

Figura 3 - Balanceamento não previsível (Andrade, 2006)

A adaptabilidade dos mecanismos de balanceamento é possível através da avaliação do nível de dificuldade percebido pelo jogador. Diferente da abordagem estática, esses mecanismos não identificam o nível de dificuldade de forma discreta através das fases e da seleção de dificuldade, mas a partir de uma função que traduza um estado do jogo em um valor indicativo do nível de dificuldade percebido pelo jogador. Essa função, conhecida como função desafio (ANDRADE, G., 2006) ou função de facilidade (DEMASI, PEDRO e CRUZ, A. J. DE O., 2003), opera sobre um conjunto discreto (i.e. fácil, médio, difícil) ou contínuo de valores, o qual for mais relevante ao problema. Fica a cargo do *game designer* o trabalho de definir a função desafio mais adequada para o jogo.

Segundo (DEMASI, PEDRO e CRUZ, A. J. DE O., 2003), é necessário que a função desafio seja confiável e rápida o suficiente de modo a não gastar muitos recursos computacionais. Como essa função é utilizada com bastante frequência e idealmente ela seria instantânea, a eficiência é mais importante que a precisão.

A função desafio geralmente se baseia em funções heurísticas simples. Como exemplo, no jogo de luta do estudo de caso, a função desafio utilizada se baseou na diferença de pontos de vida dos oponentes.

Além da definição da função desafio, é preciso definir a estratégia a ser adotada pelo mecanismo de balanceamento. Também fica a cargo do *game designer* selecionar a estratégia mais apropriada para o domínio do jogo. São diversas as técnicas possíveis de escolher nessa etapa, mas dois autores identificam alguns requisitos básicos que todas devem obedecer para que o balanceamento seja satisfatório.

(ANDRADE, G., 2006) identifica dois tipos de requisitos: os de jogabilidade e os de implementação. Os requisitos de jogabilidade são os que influenciam diretamente na percepção dos usuários em relação aos jogos. Os requisitos de implementação são influenciam no processo de desenvolvimento e nas características técnicas do jogo. Os requisitos de jogabilidade identificados por ele são:

- **Rápida adaptação inicial**: o jogo deve se adaptar rapidamente ao nível do jogador desde o início.
- **Rápida adaptação contínua**: o jogo deve acompanhar rapidamente as variações de níveis do jogador.
- **Inércia**: não deve haver alterações bruscas entre os desafios apresentados ao jogador.
- **Racionalidade**: o jogo não deve trapacear ou ter um comportamento estranho.

Os requisitos de implementação identificados pelo autor são:

- **Desenvolvimento**: o esforço do desenvolvimento da técnica de balanceamento deve ser simples.
- **Execução**: os algoritmos de balanceamento utilizados devem ter pouco consumo de processamento e memória.

(SPRONCK, PONSEN e POSTMA, 2006) considera uma lista de quatro requisitos computacionais obrigatórios e quatro requisitos funcionais desejáveis para que uma técnica de balanceamento seja aplicável na prática. Os requisitos computacionais identificados por ele são:

- **Velocidade**: o algoritmo de balanceamento deve ser computacionalmente rápido, uma vez que ocorre durante a execução do jogo.
- **Eficácia**: o algoritmo de balanceamento dinâmico deve apresentar um comportamento pelo menos tão bom quanto o balanceamento estático.
- **Robustez**: o algoritmo de balanceamento deve ser robusto quanto à aleatoriedade de jogos existentes.
- **Eficiência**: o algoritmo de balanceamento deve ser eficiente quanto às oportunidades de aprendizagem que aparecem.

Os requisitos funcionais identificados pelo autor são:

- **Clareza**: o algoritmo de balanceamento deve produzir resultados fáceis de interpretar.
- **Variedade**: o algoritmo de balanceamento deve produzir vários comportamentos distintos.
- **Consistência**: o sucesso do algoritmo de balanceamento deve ser independente do comportamento do jogador e de flutuações aleatórias do processo de aprendizagem.
- **Escalabilidade**: o algoritmo de balanceamento deve ser capaz de ajustar o nível de habilidade do jogador mantendo a taxa de sucesso de seus resultados.

Tais requisitos servem como uma forma de avaliar parcialmente estratégias de balanceamento, não sendo todas decisivas devido à subjetividade nas definições. Observou-se durante a implementação do agente adaptativo proposto neste trabalho o relaxamento de alguns requisitos em detrimento de outros, de forma a tornar prática a aplicação do agente no estudo de caso, enquanto que outros requisitos não puderam ser validados.

Vale ressaltar que abordagem de balanceamento dinâmico ao diminuir a complexidade do gerenciamento de desafios diminui o controle do *game designer* sobre o comportamento do sistema, o que pode ser considerado não desejável. Essa característica juntamente aos requisitos de eficiência dos recursos computacionais ainda impedem a maior utilização dessa abordagem na indústria.

Outro problema citado por (ANDRADE, G., 2006) é o da acomodação de jogadores, uma vez que as barreiras de níveis de habilidade são ajustadas por jogador. Isso dificulta bastante o desenvolvimento e os testes do balanceamento por parte do *game designer*, por não saberem o nível de habilidade ao projetar um novo ambiente. O autor aponta que é mais promissora a abordagem que combina métodos tradicionais de balanceamento com métodos adaptativos, transferindo maior controle do processo de adaptação ao *game designer* (ANDRADE, G., 2006).

No próximo capítulo serão descritos algumas abordagens dinâmicas existentes na literatura

3. Abordagens dinâmicas

Nas seções a seguir serão descritas algumas técnicas usadas para balanceamento dinâmico de jogos encontradas na literatura.

3.1. Manipulação de parâmetros

A abordagem realizada por (HUNICKE e CHAPMAN, 2004) trata da manipulação automática de parâmetros do jogo para o ajuste de desafios, baseando-se em uma visão econômica, dividindo tais parâmetros entre os de oferta e os de demanda. Os parâmetros de oferta estão relacionados à intervenção do jogador e correspondem aos seus atributos (i.e. pontos de vida, pontos de defesa, velocidade) e itens (i.e. arma, munição, kit médico). Os parâmetros de parâmetros de demanda estão relacionados aos desafios impostos e correspondem aos atributos e quantidades dos inimigos e itens colecionáveis. O balanceamento é resultado do equilíbrio entre as ofertas e demandas.

Junto ao uso de um inventário (como os de itens que um jogador pode possuir durante uma partida), os autores aplicaram um modelo probabilístico, onde na predição de um estado indesejado, o sistema ajusta os parâmetros de oferta ou demanda de forma a equilibrá-los. Como exemplo, em um jogo de tiro em primeira pessoa o jogador iniciante pode ter aumentada sua taxa de acerto nos disparos (ajuste na oferta) enquanto que um jogador avançado por ter de enfrentar um maior número de oponentes (ajuste na demanda).

Como é uma técnica que se baseia na manipulação de parâmetros apenas, ela pode ser integrada a outras técnicas que ajustam o comportamento dos oponentes. É importante observar que a técnica pode levar a situações não críveis (i.e. um disparo que elimina todos os oponentes de um ambiente) e que é necessário cuidado extra por parte do *game designer* em perceber quais modificações de parâmetros podem levar a tais situações.

3.2. Aprendizagem por reforço

A abordagem realizada por (ANDRADE, G., 2006) é caracterizada por um problema de aprendizagem por reforço. Essa classe de problemas consiste em um agente

aprender a política de ação que maximize a satisfação de seus objetivos, isto é, em que o agente procura aprender o conjunto de ações para os estados do problema que retornam as melhores recompensas (SUTTON e BARTO, 1998). Uma das principais características dessa abordagem é que não há supervisão para o treinamento do agente, de forma que ele aprende por tentativa e erro quais as melhores ações sem conhecimento prévio das regras que relacionam as ações aos estados.

Existem muitas técnicas para solucionar o problema de aprendizagem por reforço. O trabalho do autor utilizou a técnica Q-Learning (WATKINS e DAYAN, 1992) devido principalmente a duas propriedades interessantes para o contexto de jogos. A primeira é a garantia de convergência para a melhor solução desde que todas as combinações de estado-ação sejam visitadas o suficiente. A segunda é a relativa simplicidade comparada a outras técnicas de aprendizagem por reforço. Como estudo de caso foi utilizado o Knock'em (ANDRADE, F. et al., 2003), o mesmo que serviu de estudo de caso para este trabalho.

Para o contexto de balanceamento dinâmico, o autor dividiu o problema em competência e desempenho, seguindo os estudos realizados por (CHOMSKY, 1965). A competência é definida como o conhecimento que o agente possui sobre os estados e as ações, enquanto que o desempenho está relacionado ao nível de uso da competência. Dessa forma, o autor pretende separar o conhecimento de um agente e a execução da decisão tomada.

A competência é adquirida à medida que o agente descobre a melhor política de ação através da técnica de aprendizagem por reforço. A técnica Q-Learning utiliza uma matriz bidimensional para armazenar pesos a cada par estado-ação que determinam a qualidade de se tomar uma ação a partir de um estado. Nesse caso a competência é sempre adquirida, pois os valores da matriz são atualizados tanto por treinamento *offline* como durante a execução do jogo.

O desempenho ocorre através da avaliação do nível do jogador baseada na função desafio e executa a ação mais próxima do nível de habilidade do jogador. Assim, para um jogador iniciante o sistema vai selecionando ações sub-ótimas até compatibilizar o nível do agente com o do jogador, enquanto que um para um jogador avançado o sistema seleciona gradativamente seleções melhores, possivelmente escolhendo a solução ótima.

A separação do problema em competência e desempenho permite ao *game designer* utilizar diversas técnicas na parte de competência enquanto que permite o ajuste de parâmetros relativos à avaliação do nível do jogador à parte. Por outro lado, é necessária uma inicialização adequada dos parâmetros e da política de ação a fim de evitar alterações bruscas na no comportamento do agente.

3.3. Scripts dinâmicos

A abordagem de (SPRONCK, PONSEN e POSTMA, 2006) é um aprimoramento do uso de scripts, um método bastante conhecido na indústria para definir o comportamento de agentes. Scripts são conjuntos de regras de comportamento sobre um domínio definida por um par condição-ação. A condição corresponde à representação de um possível estado no ambiente percebido por um agente inteligente e a ação corresponde à interferência do agente no estado atual, gerando algum tipo de retorno ou recompensa.

Nesse contexto, o autor agregou um mecanismo de aprendizagem que associa um peso a cada regra, o qual determina a qualidade da regra dada a sua condição e a probabilidade da regra ser escolhida para ter sua ação executada. É necessário então que para cada situação haja alternativa de escolha.

A aprendizagem se dá através da resposta que o ambiente devolve ao agente após a aplicação da ação de uma regra, a qual tem seu peso aumentado para uma resposta positiva e diminuído para uma resposta negativa. Os pesos são atualizados por treinamento *offline* ou durante a execução do jogo. Com o passar do tempo, as regras mais eficientes tendem a possuir maiores pesos, corrigindo possíveis ruídos introduzidos inicialmente no peso das regras, e revelando as regras mais adequadas para cada jogador enfrentado.

Existem três estratégias na literatura para adaptar a técnica de scripts dinâmicos ao contexto de balanceamento dinâmico: adaptação de regras (*high-fitness penalising*), limitação dos pesos (*weight clipping*), e eliminação de regras fortes (*top culling*).

A estratégia de adaptação de regras ajusta os pesos de acordo em relação ao nível do jogador, de modo que as regras mais adaptadas ao nível do jogador tenham

um ajuste maior no peso e não as regras mais eficientes. Como exemplo, as regras menos eficientes aumentam mais de peso contra um jogador iniciante.

Já a estratégia de limitação dos pesos estabelece um valor máximo de peso para as regras, de forma que as regras que venham a atingir esse limite deixem de ter seus pesos aumentados. O balanceamento provém do ajuste do limite de acordo com o nível do jogador: o limite aumenta quando o nível do jogador está alto e o limite diminui caso contrário. Quando muitas regras estão limitadas, pode ocorrer um comportamento similar à escolha aleatória de ações.

Por sua vez, a estratégia de eliminação de regras define um valor máximo de peso na seleção de regras. Da mesma forma que a segunda estratégia, o balanceamento se dá através da modificação desse limite superior, porém não desconsidera o reajuste que ocorre nas melhores regras, aproveitando todas as oportunidades de aprendizagem. Essa estratégia se mostrou a mais eficiente dentre as três (SPRONCK, PONSEN e POSTMA, 2006), sendo também utilizada na construção do agente proposto neste trabalho.

Vários fatores atraem a indústria para utilização dessa abordagem: a similaridade com o uso clássico de script, a simplicidade do algoritmo, a inserção de conhecimento explícito e a capacidade de adaptação com previsibilidade. A abordagem de scripts dinâmicos mostrou um bom desempenho contra táticas fixas ou alternantes, mas devido à natureza estática da abordagem, a abordagem teve um bom desempenho contra táticas imprevistas.

Entretanto, uma base criada manualmente pode se tornar muito complexa, levar os agentes a apresentarem falhas de comportamento e fazer com que os jogadores explorem essas falhas.

A base de regra usada com esse método clássico é construída manualmente, o que tende a aumentar complexidade do comportamento do agente. Essa complexidade pode resultar na vulnerabilidade da IA, permitindo que os jogadores explorem alguma particularidade do comportamento dos oponentes. Além disso, a natureza estática das regras não prepara o comportamento dos agentes contra táticas imprevistas do jogador (PONSEN, 2004).

3.4. Algoritmos genéticos

A abordagem estudada por (DEMASI, PEDRO e CRUZ, A. J. DE O., 2003), ao contrário da anterior, não se baseia em regras, mas em algoritmos genéticos que codificam características de comportamento como genes dos agentes aprendizes. Eles são avaliados por uma função heurística dependente do domínio do problema chamada de função facilidade, cujo valor de aptidão indica o quão cada agente está adaptado ao nível do jogador. Essa função é capaz de realizar essa avaliação assim que ela conhece o desempenho de toda a população.

As operações básicas para quaisquer algoritmos genéticos são a seleção, a recombinação e a mutação. A seleção é o processo de escolha dos pais para a recombinação, sendo os mais comuns a seleção por classificação e a seleção por roleta. A recombinação é o processo em que os indivíduos recebem parte do código genético de cada um dos pais. A mutação altera o código genético aleatoriamente com uma probabilidade muito baixa, a fim de evitar a estagnação do indivíduo.

Uma característica da abordagem dinâmica é o fato da evolução ser enviesada por um grupo de agentes modelos, que possuem comportamentos alvos. Em geral, novos agentes são criados a partir de operações genéticas tais como a recombinação dos agentes considerados mais aptos e a mutação individual dos agentes. Os mais adaptados ao nível do jogador tendem a ser preservados e utilizados como pais na criação de indivíduos para uma nova geração.

Quando os agentes são destruídos ou ficam estagnados por um período pré-definido de tempo, a função de facilidade decide se a então geração de agentes evolui ou é mantida. Existem duas formas de evoluir: uma se dá através da comparação da quantidade de genes diferentes entre dois agentes e da modificação gradativa dos genes a fim de diminuir sua diferença genética; outra é a evolução através do cruzamento entre os agentes mais aptos e os modelos. Para o problema do balanceamento, os autores apresentam quatro variantes de produção dos modelos.

A primeira variante consiste na definição manual dos agentes modelos, quando deve existir um modelo para cada nível de dificuldade desejado. A evolução se dá através do cruzamento entre os agentes mais aptos e os modelos.

A segunda variante difere no que se refere à criação dos modelos, os quais passam a ser gerados através de treinamento *offline*. O objetivo é fazer com que tenham um bom comportamento inicial sem uso de conhecimento explícito do domínio. O treinamento pode ser realizado testando agentes aprendizes contra agentes randômicos, humanos ou outros agentes aprendizes (*self-learning*).

A terceira variante não utiliza modelos para guiar o processo evolutivo. A evolução acontece apenas com os dados presentes durante a execução (*online*) e as operações tradicionais de recombinação e mutação. Nessa variante, os agentes mais aptos de uma população são reunidos em um conjunto. Novos agentes são criados a partir da recombinação entre agentes desse conjunto. Esse conjunto é modificado toda vez que um agente for mais apto que o pior do conjunto, de modo que a priorizar os mais adaptados.

A quarta variante corresponde à utilização híbrida de 2 das variantes anteriores, obtendo-se modelos manualmente ou por treinamento *offline* numa primeira etapa (respectivamente variantes 1 e 2) para então passar a operar em modo *online* conforme a variante 3. Assim, é possível acelerar a adaptação dos agentes a partir de conhecimento explícito do domínio, mas permitindo a livre evolução dos agentes, sem a referência de modelos.

Essa abordagem de balanceamento dinâmico baseado em algoritmos genéticos permite a busca por estratégias novas, adaptadas ao jogador, sem a necessidade de codificação manual de regras, mas não excluindo também essa possibilidade. Ainda assim, não é possível controlar totalmente o processo evolutivo, de forma que a abordagem possa explorar agentes cujos comportamentos são indesejados, não havendo garantia de se encontrar uma boa solução.

Na próxima seção, será avaliada analisada a técnica que compõe a abordagem proposta, baseada em algumas ideias de scripts dinâmicos e algoritmos genéticos.

4. Estudo de caso

Neste capítulo serão analisadas duas técnicas de aprendizagem de máquina importantes para a proposta deste trabalho. A primeira é a *Learning Classifier System*, desenvolvida por (HOLLAND, J.H., 1986), cujos conceitos são necessários para que possamos entender a segunda técnica, a *Zeroth-Level Classifier System*, desenvolvida por (WILSON, 1994). Essa última técnica é a base do agente adaptativo proposto neste trabalho. Vale ressaltar que o sistema classificador da abordagem proposta não segue o paradigma *Michigan-style*, onde se utiliza mais de um conjunto de classificadores, mas sim o paradigma *Pittsburgh-style*, onde se utiliza apenas um conjunto de regras, se assemelhando ao conjunto de regras vistos em scripts dinâmicos (SPRONCK, PONSEN e POSTMA, 2006).

4.1. Visão básica de um sistema classificador

Sobre um dos primeiros sistemas classificadores (*Learning Classifier System, ou LCS*), (HOLLAND, JOHN H., HOLYOAK e NISBETT, 1989) o descreveu como um sistema baseado em regras com mecanismos gerais para o processamento paralelo das regras, para geração adaptativa de novas regras, e para testar a eficácia das regras já existentes. Estes mecanismos tornam o desempenho e a aprendizagem possíveis sem as características frágeis de muitos sistemas especialistas em IA. A partir da definição, podemos identificar que o sistema é dividido em três partes: um componente de mensagens, um componente de reforço (ou atribuição de crédito) e um componente de descoberta de regras, como apresentado na figura X.

O componente de mensagens é caracterizado como um sistema de produção, isto é, um sistema com entradas e saídas que converte uma mensagem recebida do ambiente em mensagens de saída a cada ciclo através de regras. As regras, chamadas também de “classificadores”, em geral são da forma *condição-ação*, o que significa que a ação possa ser tomada se tem a condição satisfeita. Essa regras utilizam um alfabeto ternário {0,1,#}, onde o símbolo # indica generalização, isto é, ele pode significar tanto 0 quanto 1. De início, o conjunto de regras desse sistema classificador é gerado aleatoriamente, e é associado um peso (ou predição) a cada

regra, que representa a qualidade da regra em receber recompensas positivas ao ser aplicada ao ambiente (ISAAC, 2005).

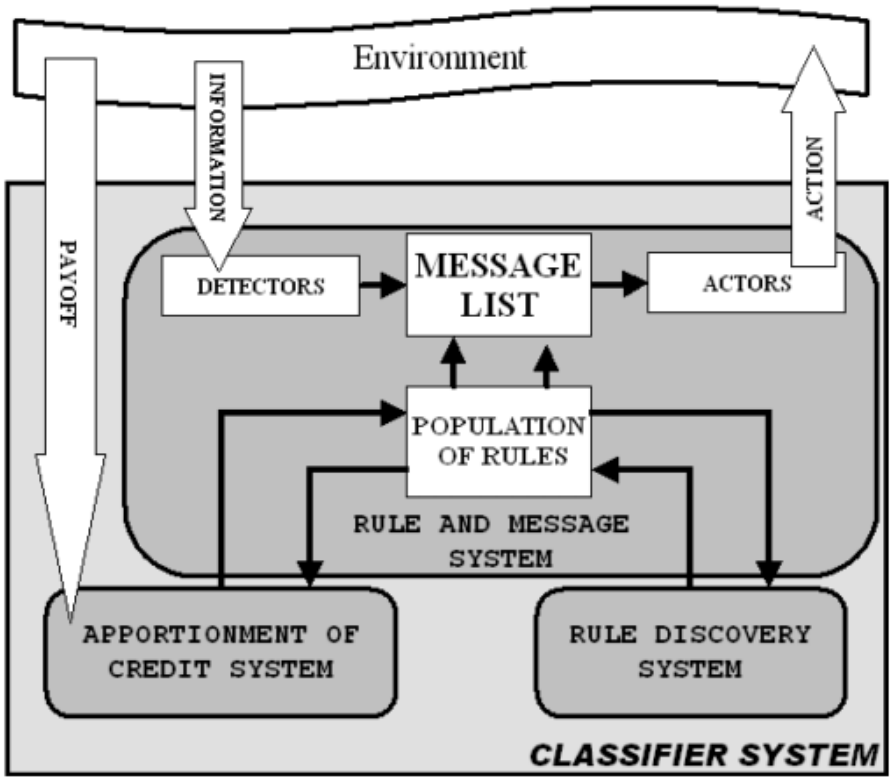

Figura 4 - Arquitetura de um sistema classificador padrão (Isaac, 2005)

A cada início de ciclo, os detectores do sistema mapeiam estímulos de entradas em novas mensagens. Regras que tenham condições satisfeitas com alguma das mensagens do ambiente são ativadas, compondo o conjunto de regras compatíveis [M]. É então realizada sobre esse conjunto alguma operação de sorteio (i.e. *Roulette Wheel*) baseada nos valores das predições (BULL e KOVACS, 2005). As regras selecionadas podem enviar suas ações como mensagens para a lista de mensagens do sistema. Vale ressaltar que na definição tradicional de LCS, a lista de mensagens pode conter tanto mensagens de entrada quanto mensagens de saída, permitindo o que se chama de cadeias de regras, onde uma mensagem disparada por uma regra ativada pode satisfazer outra regra no próximo ciclo sucessivamente (ISAAC, 2005).

Ao fim do ciclo, os atuadores executam as ações disparadas possíveis de serem realizadas. Caso haja um valor de recompensa retornando do ambiente, ele é então utilizado para atualizar as predições das regras através do componente de

reforço. Porém como uma ação pode ser resultante de uma cadeia de regras, é importante atualizar a predição de todas as regras que levaram à execução da ação. Esse problema de atribuição de crédito pode ser resolvido por várias técnicas, sendo a utilizada originalmente a *bucket brigade* (BOOKER, GOLDBERG e HOLLAND, J.H., 1989). Nessa técnica de atribuição de crédito, cada regra realiza uma aposta proporcional ao seu peso. Quando uma regra é sorteada e executada, ela tem seu peso aumentado em pela recompensa recebida do ambiente, enquanto que as regras que advogaram pela ação executada recebem parte do total acumulado das apostas. Por último, é cobrada uma taxa de desconto sobre as regras que não foram satisfeitas e também sobre os que participam das apostas (RICHARDS, 1995).

Já o componente de descoberta de regras garante a criação de regras, cuja aptidão reside no valor da predição. Na maioria das vezes, esse sistema é composto por algoritmos genéticos, os quais combinam as regras com melhores predições (mantendo o tamanho da população) e modificam regras para produzir novas gerações, as quais se espera que resultem em melhores predições (vide 3.4). Segundo (BULL e KOVACS, 2005), o papel do motor genético no sistema classificador é o de criar um conjunto de regras cooperativo, que juntos resolvem o problema. Diferente do cenário de otimização tradicional, a busca não é para a melhor regra, mas um número de regras de diferentes tipos que juntas correspondem a um comportamento apropriado.

Dessa forma, o sistema é capaz tanto de explorar o espaço de busca através do uso de algoritmos genéticos, quanto de explorar as predições, testando os classificadores e atualizando suas predições.

4.2. Agente adaptativo proposto

Este trabalho se baseia na adaptação proposta por (WILSON, 1994) do modelo de sistemas classificadores *Learning Classifier System* desenvolvido por (HOLLAND, J.H., 1986). Os estudos do autor levaram-no à criação do *Zeroth-Level Classifier System* (ZCS), um modelo simplificado do LCS para facilitar o entendimento dos processos e melhorar o desempenho da técnica, mantendo a essência da ideia original.

A primeira grande mudança foi a de abandonar o componente de mensagens, sendo substituído apenas por uma população de classificadores e seus pesos. A codificação dos classificadores também mudou, proibindo o uso de ações genéricas (através do símbolo #), possível com o componente de mensagens.

No início do ciclo, o detector repassa à população uma codificação do estado do ambiente, havendo a comparação direta do estado com as condições de cada classificador, resultando em um subconjunto de classificadores compatíveis [M]. Em seguida é realizado um sorteio entre todos os classificadores da população baseado em seus pesos, e na seleção do vencedor, é criado o subconjunto dos classificadores compatíveis que possuem a mesma ação, que é enviada para os atuadores executar.

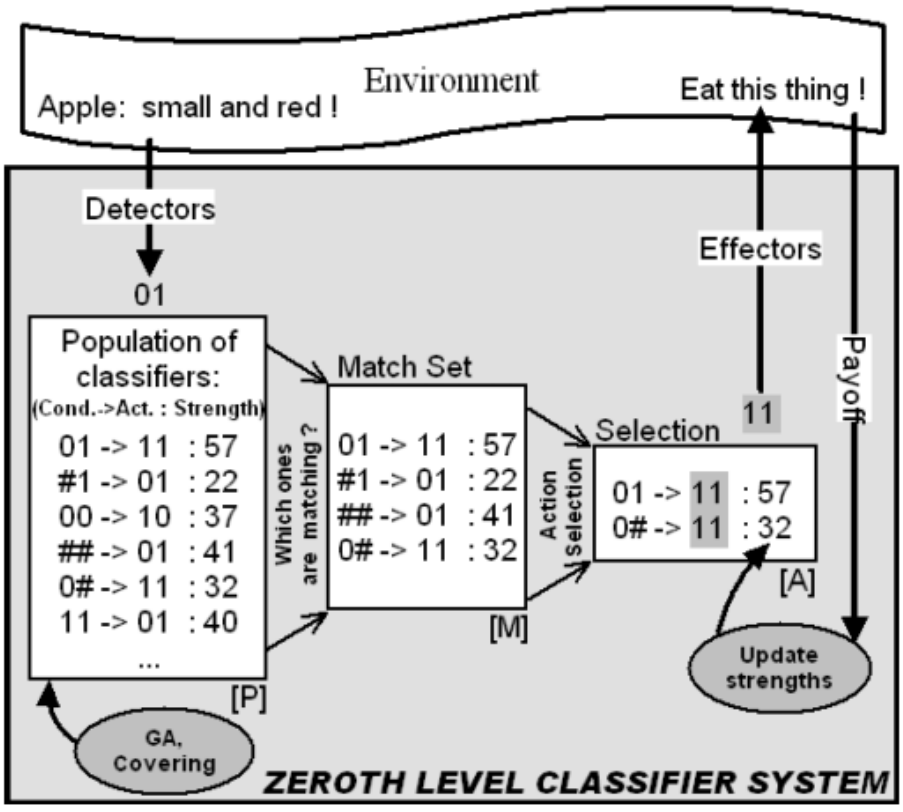

Figura 5 - *Zeroth-Level Classifier System* (Isaac, 2005)

O componente de reforço nesta técnica não realiza mais o processo de aposta. Os classificadores passam a ter seus pesos atualizados através de uma processo de *bucket brigade* implícito que ocorre entre os classificadores de [A] (ciclo atual) e de $[A]_{-1}$ (ciclo anterior). Primeiro cada classificador em [A] cede uma fração β

(0 < β < 1) de seu peso para um acumulador B. Depois disso, caso haja recompensa do ambiente r_{imm}, ela é dividida entre os classificadores de [A], cada qual com seu peso aumentado em $\beta r_{imm}/|A|$, onde |A| é o número de classificadores em [A]. Em seguida, se $[A]_{-1}$ não for vazio, os classificadores desse conjunto tem seus pesos aumentados de $\gamma B/|A_{-1}|$, onde γ (0 < γ < 1) é um fator de desconto, B é o total do acumulador obtido do passo 1 e $|A_{-1}|$ é o número de classificadores em $[A_{-1}]$. Então, [A] substitui $[A_{-1}]$ e o acumulador B é esvaziado. Por último, cada classificador compatível de [M] que não possui a mesma ação que os classificadores de [A], ou os classificadores que se encontram no conjunto [M] – [A], tem seus pesos reduzidos por um $\tau \cong \beta$.

O componente de descoberta de regras é composto por um algoritmo genético de estado estacionário e por um mecanismo de *covering*. O algoritmo genético funciona da mesma maneira que no LCS quanto às operações genética, com a diferença de que é executado em um período regular de passos, de forma que todos os classificadores tenham executado sua ação alguma vez. O mecanismo de *covering* opera quando ao se criar o conjunto de classificadores compatíveis [M] ele é vazio ou a média dos pesos desses classificadores for menor que uma fração ϕ da média dos pesos da população. Nesse caso, ele cria um novo classificador com as condições correspondentes à codificação do estado, mas com uma probabilidade por gene uma generalização, além de selecionar uma ação aleatória. Esse novo classificador é então adicionado à população, e serve como um teste de hipótese. Esse modo de operar evita testes de hipóteses com proposições muito diferentes da realidade (MOIOLI, VARGAS e ZUBEN, 2008).

Tendo visto o referencial teórico, a proposta deste trabalho é a construção de um agente adaptativo baseado em *Zeroth-Level Classifier System* (ZCS) integrado à técnica *Top Culling* desenvolvida por (SPRONCK, PONSEN e POSTMA, 2006), a fim de comparar o seu desempenho aos demais agentes adaptativos desenvolvidos por (ANDRADE, F. et al., 2003; ANDRADE, G., 2006). A técnica *Top Culling* apenas prevê um limite máximo ajustável para a participação do classificador da escolha de ação. Nas próximas seções serão descritos o ambiente utilizado para implementação e validação dos agentes.

4.3. Ambiente de testes

O jogo Knock'em (ANDRADE, F. et al., 2003) é o ambiente escolhido para o desenvolvimento e a realização de testes com os agentes adaptativos propostos, por ser um jogo de luta de tempo real que apresenta agentes inteligentes de várias abordagens implementados por (ANDRADE, G., 2006). Nesse jogo, os lutadores controlados pelos agentes são definidos pelos atributos: pontos de vida, pontos de energia espiritual (mana), força, agilidade, resistência e pontos de energia. A premissa básica é comum dos jogos de luta: dois lutadores possuem uma quantidade inicial e pontos de vida e de pontos de mana, de modo que em uma partida eles se enfrentam até que um deles perca todos os pontos de vida ou até que o tempo acabe (90 segundos). O vencedor é quem terminar a partida com mais pontos de vida para ambas as condições de parada. A Figura 6 ilustra uma partida do jogo, apresentando na região superior os pontos de vida e de mana bem como o tempo restante.

Figura 6 - Knock'em (F. Andrade, 2003)

Como vemos na Figura 6, o jogo possui uma perspectiva 2D, permitindo ao jogadores se movimentar livremente na horizontal e através de pulos na vertical. Cada lutador pode executar socos forte ou rápido, chutes forte ou rápido, defender-se, pular, agachar, se movimentar horizontalmente e lançar bola de fogo. Ao utilizar esse último ataque, uma quantidade de mana é consumida, de forma que o lutador deve esperar recarregar mana suficiente para então lançar bola de fogo novamente.

Quanto à arquitetura, o jogo é composto por 3 sistemas: *FullGame*, *Playtest* e *Simulator*. Todos se utilizam das funcionalidades básicas providas pelo sistema *CoreEngine*. A Figura 7 mostra o diagrama de pacotes do ambiente de testes.

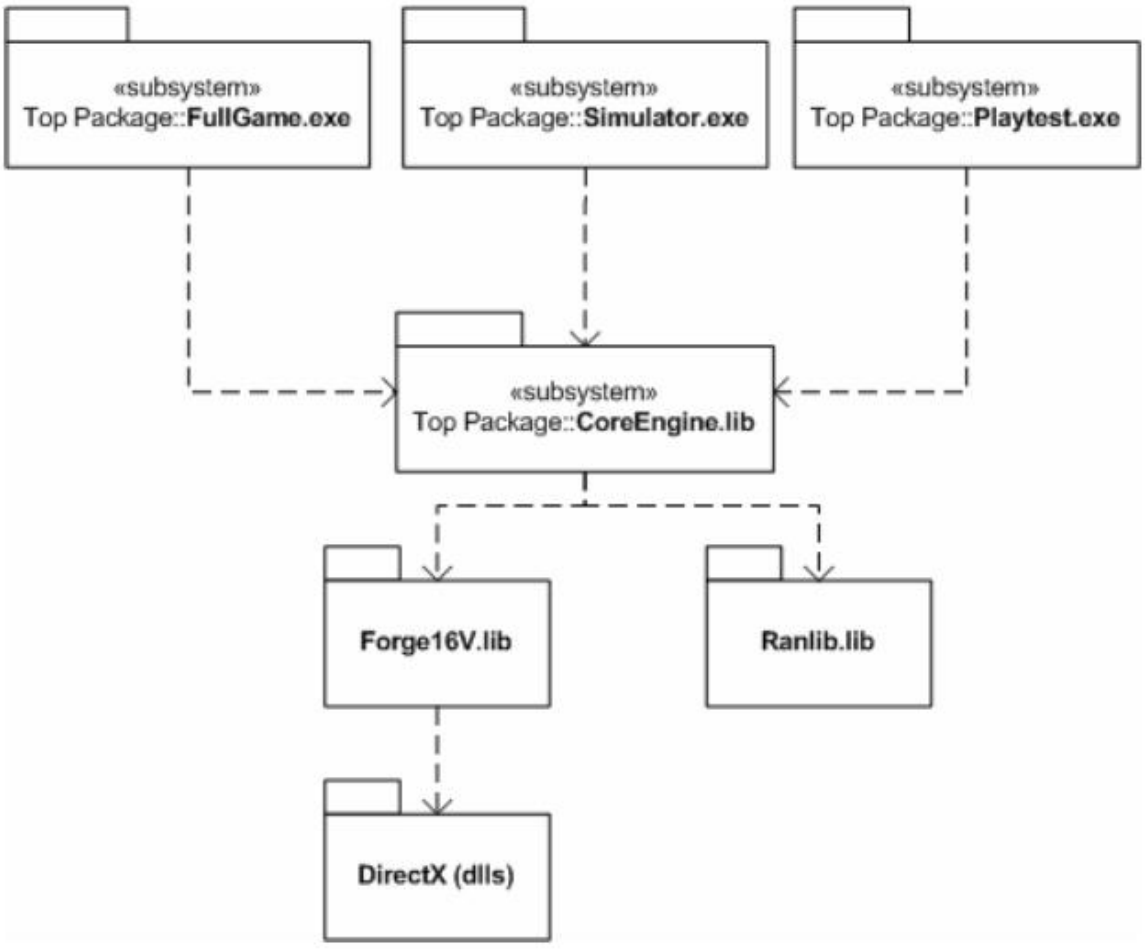

Figura 7 - Sistemas e módulos do Knock'em (F. Andrade et al., 2003)

O módulo *FullGame* consiste na versão completa do jogo, onde o jogador joga uma sequência de lutas em estágios diferentes para ser o vencedor do torneio. O módulo *PlayTeste* consiste na versão reduzida do jogo, utilizada para a realização de testes dos diversos agentes com jogadores humano. O módulo *Simulator* executa

uma série de lutas entre agentes no modo treinamento e validação, sendo útil no treinamento *offline* de agentes que implementem algum tipo de aprendizagem. Os módulos *Forge16V* e *Ranlib* são bibliotecas que tratam respectivamente das funções de comuns de jogos de luta e da geração de números aleatórios. O módulo principal *CoreEngine* concentra os elementos relativos à dinâmica do ambiente, como modelagem física, funções de dano e regras do jogo (ANDRADE, F. et al., 2003).

Andrade em seu trabalho (ANDRADE, G., 2006) implementou agentes inteligentes com diferentes estratégias: um agente randômico (RDPlayer), um agente baseado em máquina de estados (SMPlayer), um agente baseado em aprendizagem por reforço tradicional (TRLPlayer), um agente que utiliza uma abordagem baseada em desafio (*CBRLPlayer*) proposto em seu trabalho, um agente que implementar a proposta de (SPRONCK, PONSEN e POSTMA, 2006), e um agente baseado na abordagem proposta por (DEMASI, P. e CRUZ, A., 2002) de algoritmos genéticos (*GAPlayer*). Todos esses agentes herdam da classe *Fighter* características e métodos comuns aos lutadores no jogo. A definição de estratégias de atuação bem como o processo de seleção de ações e de aprendizagem de cada agente são implementados na função *ProcessGameState*.

Segundo (RUSSELL e NORVIG, 2004), a descrição do mundo é definida pela codificação dos conjuntos (ou espaços) de estados e ações. A descrição do mundo possui então um grande impacto no desempenho dos agentes, pois ela é quem especifica as informações que podem ser usadas na tomada de decisão e quais ações podem ser executadas (ANDRADE, G., 2006).

Esse impacto é maior em abordagens como a de aprendizagem por reforço, nas quais os agentes precisam visitar constantemente cada par estado-ação até que aprendam a utilidade de se executar cada ação a partir de cada estado. O tempo de convergência para uma competência satisfatória nesse tipo de abordagem cresce proporcional ao tamanho do conjunto estado-ação. Isso torna necessária a simplificação da representação de estados e ações no jogo para manipulação por parte do jogo. A representação do estado **S** no jogo foi definida então pela seguinte tupla:

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

S_{agente} significa o estado atual do agente (parado, pulando ou agachado). **$S_{oponente}$** significa o estado atual do oponente (parado, pulando, agachado, atacando, atacando e pulando simultaneamente, atacando e agachando simultaneamente ou bloqueando). **D** representa a distância do agente ao oponente (perto, médio ou longe). **M_{agente}** e **$M_{oponente}$** significam a disponibilidade de pontos de mana do agente e do oponente respectivamente (suficiente ou insuficiente para lançar bola-de-fogo). **F**, por último, significa a situação das bolas-de-fogo lançadas pelo oponente (perto, médio, longe ou não existente) (ANDRADE, G., 2006).

Basicamente, em posse da descrição de mundo (*WorldDescription* ou *WorldDescription2*) e do significado da representação dos estados e ações, é possível a construção de um agente pela implementação da função *ProcessGameState*. A explicação da implementação dos agentes presentes no jogo pode ser encontrada no trabalho de (ANDRADE, G., 2006). Na próxima seção, será descrito o processo de implementação do agente adaptativo aqui proposto.

4.4. Implementação

O agente adaptativo implementado neste trabalho é baseado em *Zeroth-Level Classifier System* com *Top Culling* (ZCSTCPlayer). A primeira classe necessária à implementação desse agente foi a dos classificadores, que foi adaptada de uma classe de regras utilizada pelo agente de scripts dinâmicos presente no jogo.

Em seguida, as demais partes do agente foram implementadas de acordo com as fases de execução no método *ProcessGameState*. A primeira fase é a de captura do novo estado, onde também ocorrem as inicializações de algumas variáveis para uma nova rodada.

A segunda fase é a de atualização dos pesos, onde teve de ser construído o mecanismo de *bucket brigade* implícito (*UpdateWeights*). Nesse método, as ações executadas por uma geração e pela anterior têm seus pesos ajustados com base na recompensa, normalizando-se em seguida conforme a explicação da seção 4.2.

A terceira fase é a de evolução da população, que ocorre a uma taxa fixa de ciclos. Para esta fase, o agente proposto se utilizou de uma função responsável pelas operações genéticas (*EvolvePopulation*), de forma que parte da população fosse selecionada por elitismo, outra parte (maioria) fosse recombinada a partir dos

melhores selecionados por elitismo e a terceira parte fosse substituída por classificadores aleatórios. Para manter o mesmo tamanho da população, foi necessário implementar também o mecanismo de sorteio por roleta (*RouletteWheel*) em sua versão inversa: os classificadores com maiores predições são menos prováveis de serem substituídos. A função ao final de cada operação realiza também o teste para mutação (*DoMutation*).

A quarta e última fase é a de seleção e execução da ação (*GetAction*). Nela, os classificadores precisaram de uma função que comparasse suas condições com um dado estado (*Match*) e de um mecanismo de seleção, que por simplicidade foi o de sorteio por roleta baseado em predição: os classificadores com maiores predições têm maiores chances de serem sorteados. Esse mecanismo de sorteio apesar de simples garante a possibilidade de execução para todos os classificadores. Ainda foi preciso desenvolver o mecanismo de *covering* (*Cover*) para a possibilidade de teste de hipóteses.

Além disso, devido ao *Top Culling*, antes de todas as fases há a avaliação do nível de dificuldade do jogo por uma função heurística. Essa função heurística é a função desafio implementada para os agentes baseados em aprendizagem por reforço no trabalho de (ANDRADE, G., 2006):

$$f = \begin{cases} fácil, caso\ \Delta_{vida} < 0 \\ médio, caso\ 0 \leq \Delta_{vida} < 10 \\ difícil, caso\ contrário \end{cases}$$

, onde f é o nível de dificuldade estimado e Δ_{vida} corresponde à diferença de pontos de vida entre o agente e o oponente. Esse nível de dificuldade ajusta o limite *weightLimit* para mais (caso o nível esteja fácil) ou para menos (caso o nível esteja difícil). O limite é aplicado então no momento da seleção de ação, impedindo a escolha de ações que não estejam dentro deste limite.

Devido à lentidão do motor de operações genéticas em encontrar uma solução aceitável de desempenho do agente, buscou-se uma forma de acelerar tal busca. Uma alternativa é o uso do mecanismo de *covering*, que permite testes de hipóteses mais prováveis de acontecerem. Testes preliminares indicaram que a inicialização do algoritmo poderia acontecer baseada em um conhecimento inicial mínimo (conjunto de classificadores definidos manualmente) que servisse de

referencial pelo menos para as primeiras gerações. Essa ideia surgiu com base na abordagem de agentes-objetivo aplicada com sucesso por (DEMASI, PEDRO e CRUZ, A. J. DE O., 2003).

No próximo capítulo serão apresentados a metodologia dos experimentos e os resultados encontrados para esta abordagem.

5. Experimentos e resultados

No decorrer da seção 4.4, pudemos notar que enquanto algumas partes da implementação foram planejadas, outras tiveram de ser construídas a partir de resultados de experimentos preliminares. Nas seções a seguir, a metodologia da abordagem proposta será detalhada, bem como os resultados obtidos.

5.1. Metodologia experimental

Os experimentos consistiram em uma série de lutas entre os agentes do grupo de avaliação contra os agentes do grupo de controle. A fim de avaliar o desempenho do agente proposto, o seguinte grupo de avaliação foi formado:

- Agente baseado em máquina de estados (SMPlayer);
- Agente baseado em aprendizagem por reforço tradicional (TRLPlayer);
- Agente adaptativo de abordagem baseada em desafio (CBRLPlayer), desenvolvido no trabalho de (ANDRADE, G., 2006);
- Agente adaptativo genético adaptativo (GAPlayer), baseado na abordagem de balanceamento com agentes-objetivo de (DEMASI, PEDRO e CRUZ, A. J. DE O., 2003);
- Agente adaptativo baseado em scripts dinâmicos com *Top Culling* (DSTCPlayer), desenvolvido no trabalho de (SPRONCK, PONSEN e POSTMA, 2006);
- Agente adaptativo proposto baseado em *Zeroth-Level Classifier System* (WILSON, 1994) com *Top Culling* (ZCSTCPlayer);

O seguinte grupo de controle foi formado para servir de referência aos grupo de avaliação:

- Agente randômico (RDPlayer), que realiza ações aleatórias sem qualquer conhecimento explícito ou algoritmo de aprendizagem;
- Agente baseado em máquina de estados (SMPlayer);
- Agente baseado em aprendizagem por reforço tradicional (TRLPlayer) treinado por *self-learning*;

O agente TRLPlayer treinado por *self-learning* no grupo de controle foi posto para treinar contra outro agente TRLPlayer por 100 lutas, ambos com base inicial de

conhecimento aleatória. Uma descrição detalhada das estratégias adotadas por cada um dos demais agentes está disponível no trabalho desenvolvido por (ANDRADE, G., 2006).

Antes das lutas de avaliação, foi realizado o treinamento offline dos agentes que aprendem (TRLPlayer, CBRLPlayer e ZCSTCPlayer) para que pudessem adquirir uma base inicial de conhecimento. Esse treinamento consistiu em uma série de 1000 lutas contra um agente randômico. Essa decisão se baseia nos argumentos construídos nos trabalhos de (ANDRADE, G., 2006) e (MACIEL JUNIOR, 2010) que investigaram o treinamento com outros tipos de agentes e concluíram que o agente randômico é o que apresenta comportamento mais abrangente e resulta em maior conhecimento residual por parte do agente aprendiz.

O desempenho dos agentes foi comparado de forma que cada agente do grupo de avaliação lutasse 50 partidas com cada um dos agentes do grupo de controle. As variáveis observadas para avaliação do desempenho foram a média e o desvio padrão das diferenças dos pontos de vida entre os agentes ao final de cada partida.

Buscou-se replicar a maior parte da configuração dos experimentos, como por exemplo o número de partidas de treinamento, o número de partidas de avaliação e o uso de personagens com mesmo atributo. Outro ponto considerado foi o desempenho da aplicação, pois experimentando parâmetros diferentes, observou-se que com o aumento do número máximo de classificadores na população, o agente aparentava apresentar um desempenho inicialmente melhor: quando o nível do agente era considerado fácil ou médio pela função desafio, o seu comportamento tornava-se mais agressivo e diversificado – ora o agente ZCSTCPlayer se aproximava para aplicar socos e chutes, ora ele de distanciava para lançar bolas-de-fogo. Entretanto, à medida que a população de classificadores crescia, o desempenho da aplicação se degradava, de forma que não foi possível estabelecer com certeza a relação direta entre essas duas variáveis (tamanho da população e desempenho) (PEROURNALNAIK e ÉNÉE, 2008). Gradativamente estabilizou-se em um limite máximo com o qual o desempenho da execução fosse aceitável ao enfrentar todos os agentes.

Apesar dos agentes que empregam algum tipo de aprendizagem continuar aprendendo durante as partidas de avaliação, tomou-se o devido cuidado para que

esses agentes iniciassem o período de avaliação apenas com a experiência acumulada durante o treinamento.

Dada a característica da abordagem aqui proposta de conhecimento representado por classificadores, que são nada mais que regras com pesos, poderíamos utilizar o conjunto inicial de regras do agente de scripts dinâmicos, esperando apenas que novas regras fossem descobertas. Porém, decidiu-se avaliar o agente proposto sob as mesmas condições dos demais agentes que aprendem: adquirindo conhecimento inicial unicamente do treinamento offline descrito anteriormente.

5.2. Resultados obtidos

A figura 8 e a figura 9 apresentam o desempenho do grupo de avaliação contra o agente randômico RDPlayer, de comportamento imprevisível e sem estratégia de seleção de ações.

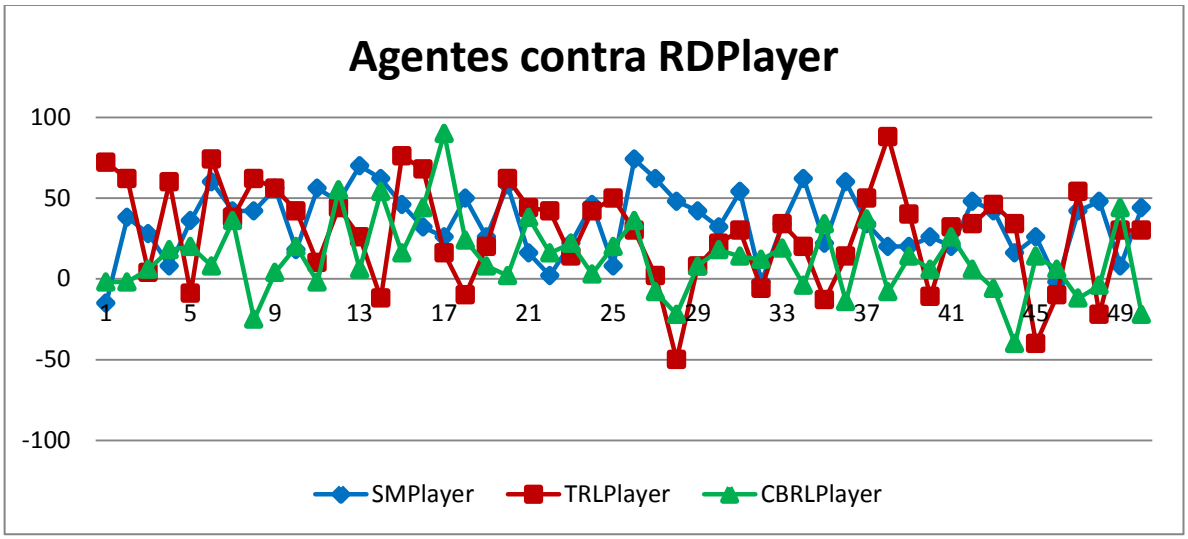

Figura 8 - Desempenho dos agentes SMPlayer, TRLPlayer e CBRLPlayer contra o agente RDPlayer

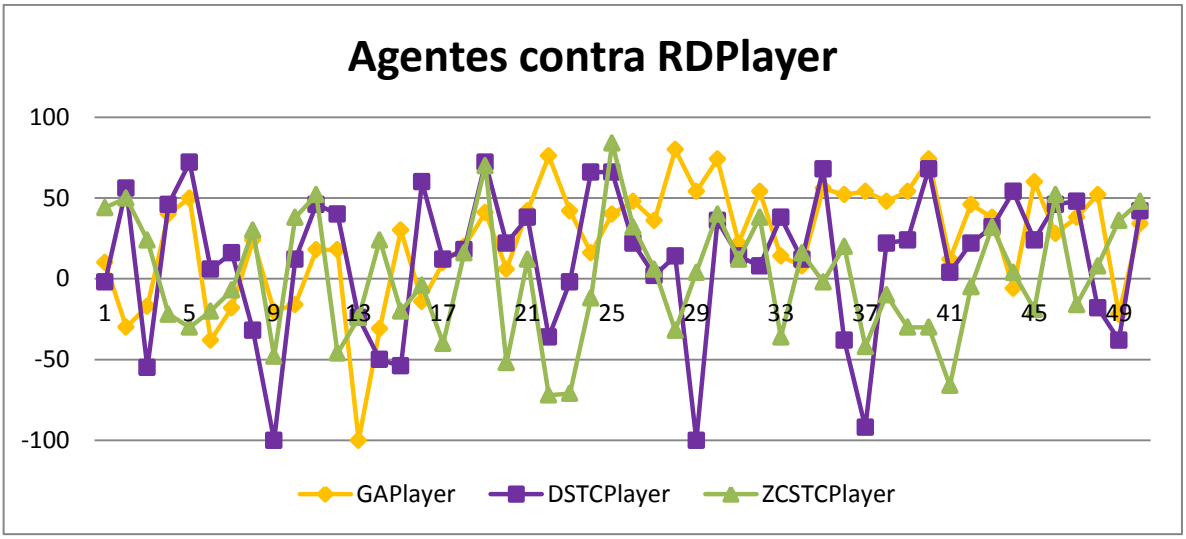

Figura 9 - Desempenho dos agentes GAPlayer, DSTCPlayer e ZCSTCPlayer contra o agente RDPlayer

Os gráficos mostram a diferença de pontos de vida entre o agente avaliado e o agente oponente (neste caso, o agente RDPlayer) ao longo da série de 50 lutas. A vitória do agente avaliado acontece quando essa diferença é positiva, enquanto que sua derrota ocorre se negativa. O empate é dado pela diferença nula de pontos de vida ao final de uma partida.

A tabela 1 resume para cada agente do grupo de avaliação a média da diferença dos pontos de vidas ao término das partidas (vantagem positiva para o agente, negativa para o oponente) e o grau de dispersão dos resultados.

Tabela 1 - Análise de desempenho contra o agente RDPlayer

Oponente	Média	Desvio padrão
SMPlayer	35,22	20,39
TRLPlayer	27,98	30,77
CBRLPlayer	12,66	22,94
GAPlayer	24,14	35,21
DSTCPlayer	12,14	44,14
ZCSTCPlayer	0,72	37,31

A figura 10 e a figura 11 apresentam o desempenho do grupo de avaliação contra o agente baseado em máquina de estados SMPlayer, de estratégia fixa definida manualmente por um game designer.

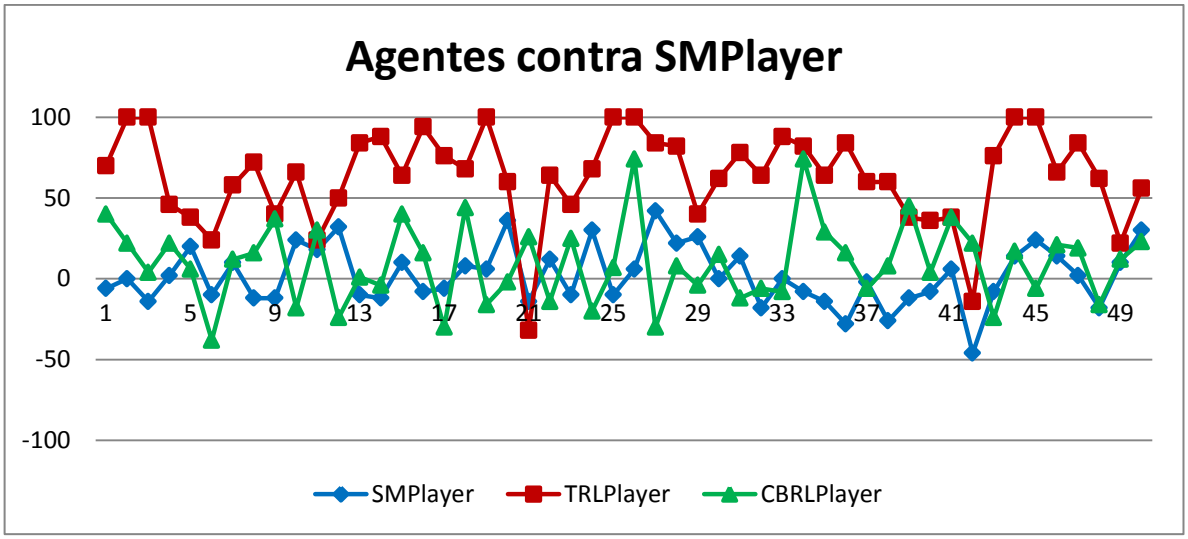

Figura 10 - Desempenho dos agentes SMPlayer, TRLPlayer e CBRLPlayer contra o agente SMPlayer

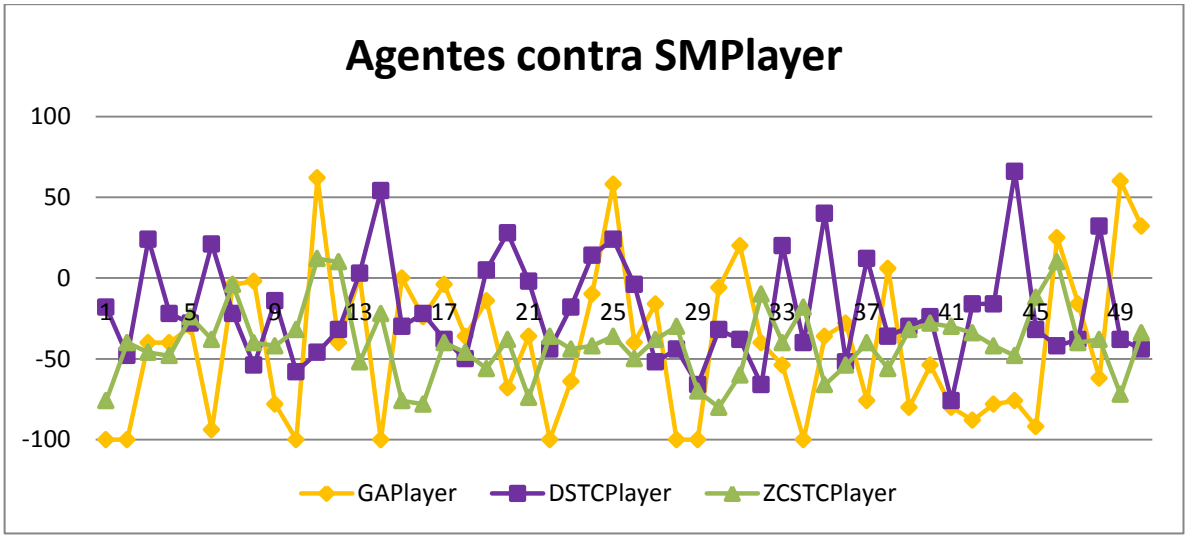

Figura 11 - Desempenho dos agentes GAPlayer, DSTCPlayer e ZCSTCPlayer contra o agente SMPlayer

A tabela 2 resume para cada agente do grupo de avaliação a média da diferença dos pontos de vidas ao término das partidas (vantagem positiva para o agente, negativa para o oponente) e o grau de dispersão dos resultados.

Tabela 2 - Análise de desempenho contra o agente SMPlayer

Oponente	Média	Desvio padrão
SMPlayer	2,12	18,09
TRLPlayer	63,60	28,29
CBRLPlayer	9,90	24,68
GAPlayer	-40,84	46,18
DSTCPlayer	-19,78	32,54
ZCSTCPlayer	-40,40	21,81

A figura 12 e a figura 13 apresentam o desempenho do grupo de avaliação contra o agente baseado em aprendizagem por reforço TRLPlayer treinado por *self-learning*, capaz de aprender melhores estratégias.

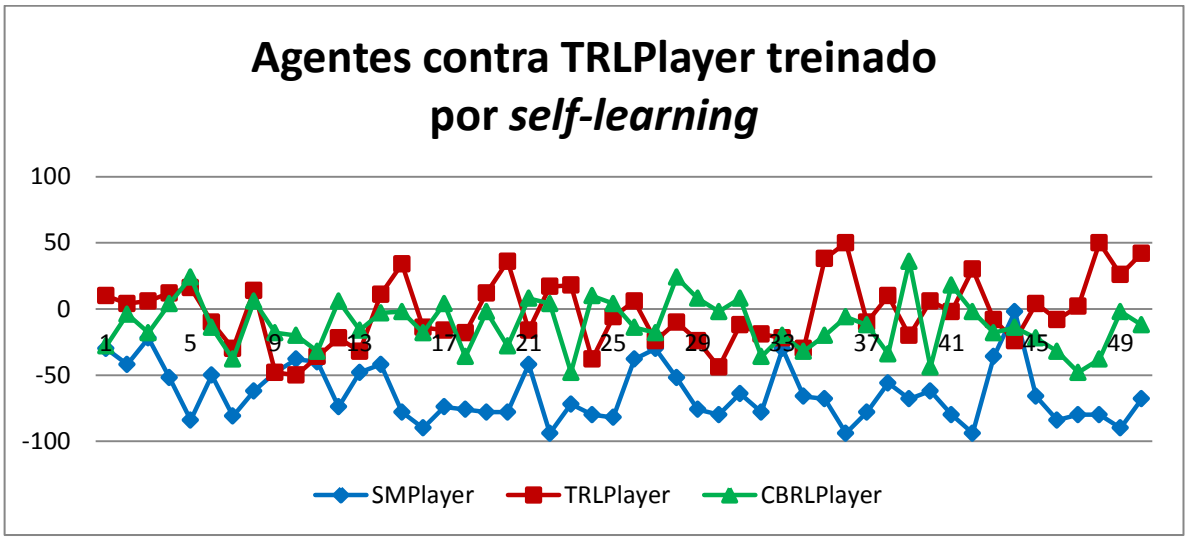

Figura 12 - Desempenho dos agentes SMPlayer, TRLPlayer e CBRLPlayer contra o agente TRLPlayer treinado por *self-learning*

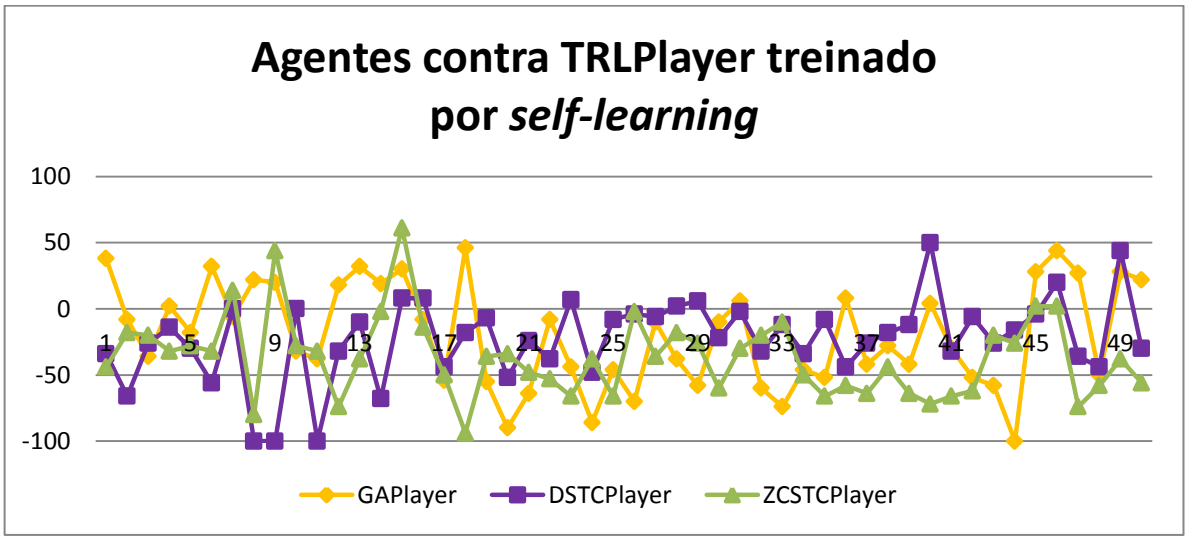

Figura 13 - Desempenho dos agentes GAPlayer, DSTCPlayer e ZCSTCPlayer contra o agente TRLPlayer treinado por *self-learning*

A tabela 3 resume para cada agente do grupo de avaliação a média da diferença dos pontos de vidas ao término das partidas (vantagem positiva para o agente, negativa para o oponente) e o grau de dispersão dos resultados.

Tabela 3 - Análise de desempenho contra o agente TRLPlayer treinado por *self-learning*

Oponente	Média	Desvio padrão
SMPlayer	-63,54	21,49
TRLPlayer	-2,78	25,18
CBRLPlayer	-11,74	19,51
GAPlayer	-19,74	39,10
DSTCPlayer	-22,88	30,84
ZCSTCPlayer	-36,48	30,21

5.3. Discussão

Contra o oponente sem estratégia formada RDPlayer, os agentes SMPlayer e TRLPlayer conseguem vencer na maioria das vezes com uma diferença razoável de pontos de vida. Já o agente GAPlayer perde algumas vezes inicialmente, mas depois passa a vencer partidas com mais frequência. Os agentes CBRLPlayer e DSTCPlayer conseguem vencer com uma diferença pequena de pontos de vida, apresentando o segundo agente maior variação dos resultados. Na média, o agente ZCSTCPlayer está em nível de empate com o agente randômico, apresentando relativa variação nos resultados. Observa-se que esses três últimos agentes conseguem atuar no nível do oponente randômico.

Já contra o oponente de estratégia fixa SMPlayer, contendo um comportamento razoavelmente eficiente estabelecido manualmente por um game designer, apenas o agente TRLPlayer vence a maioria das partidas, pois consegue aprender quais as falhas do oponente. O agente avaliado SMPlayer atua no mesmo nível do oponente, enquanto que o agente CBRLPlayer atua em um nível um pouco maior (vencendo a maioria das partidas) e o agente DSTCPlayer atua em um nível um pouco menor (perdendo mais vezes). Os agentes GAPlayer e ZCSTCPlayer não se adaptaram de forma satisfatória e perderam na maioria das vezes.

Por último, o oponente TRLPlayer treinado por *self-learning*, que possui a capacidade de aprender novas estratégias, teve apenas o agente avaliado TRLPlayer como páreo. Todos os outros agentes avaliados não conseguiram atuar no nível do oponente, com o agente CBRLPlayer perdendo em média com uma pequena diferença de pontos de vida, o agente SMPlayer perdendo com uma grande diferença de pontos de vida e os agentes restantes perdendo razoavelmente.

É possível observar dos resultados que comparado ao agente SMPlayer, o agente TRL foi capaz de modificar seu comportamento de forma a atuar da maneira mais efetiva contra o oponente, quaisquer que tenham sido as estratégias tomadas por ele. Porém, é possível observar que o agente alcança o objetivo de proporcionar uma partida balanceada apenas contra o oponente de estratégia mutável.

Percebe-se também que o agente CBRLPlayer foi o que mais se aproximou do objetivo de balanceamento em uma partida, pois em média foi o que mais atuou no nível do usuário. Porém, o agente CBRLPlayer não apresentou desafio a um oponente de estratégia mutável como o TRLPlayer neste trabalho, que comparado ao trabalho de (ANDRADE, G., 2006) se justifica por não ter utilizado conhecimento específico do jogo na escolha de ações a fim de acelerar o treinamento.

O desempenho do agente GAPlayer, por sua vez, indica que a evolução das gerações demora a acompanhar o comportamento do oponente, de forma que contra oponentes de estratégia fixa o agente demorou em atuar no nível do oponente, e que contra um oponente de estratégia mutável que responde mais rápido à mudança de comportamento como o TRLPlayer o agente falhou em atuar no mesmo nível de dificuldade.

Por possuir uma base de conhecimento fixa composta por regras ótimas o agente DSTCPlayer apresentou uma grande variação de comportamento frente aos três oponentes. Uma possível explicação para esse problema pode ser a necessidade por uma mudança mais significativa do nível de dificuldade e por uma maior variedade de regras conhecidas pelo agente. Como o nível de dificuldade é medido unicamente pela diferença dos pontos de vida, durante muitas situações estacionárias dessa diferença o agente mantém um conjunto de ações pouco eficazes. Além disso, conforme (ANDRADE, G., 2006) verificou, no contexto de jogos as ações codificadas são fundamentalmente ótimas de ataque (golpes) ou de defesa (bloqueios). Como consequência, pequenas variações que ocorram no nível de dificuldade podem resultar em uma grande variação de eficiência das ações, como foi possível verificar no decorrer dos experimentos. Outro problema desse agente é a falta de um mecanismo que possibilite a aprendizagem de novas regras.

Quanto ao objetivo deste trabalho, o agente ZCSTCPlayer não apresentou um desempenho satisfatoriamente balanceado ou desafiador para os oponentes. Uma

primeira justificativa poderia ser o tempo de treinamento insuficiente para que o agente aprendesse regras eficazes. De início as regras são todas formadas a partir de testes de hipóteses realizados pelo mecanismo de *covering*, sendo possível observar que a generalização é influenciada pela recombinação e mutação das regras, bem como pela técnica *Top Culling*, a qual impede algumas hipóteses de serem avaliadas. Possivelmente um mecanismo que oriente parcialmente a evolução das regras pode diminuir o tempo de treinamento requerido.

(WILSON, 1995) prevê que a técnica *Zeroth-Level Classifier System* não é capaz de evitar a generalização excessiva das regras e que o mecanismo de seleção de ação por roleta pode levar à convergência em soluções sub-ótimas. Ele sugere o uso de outro mecanismo que divida claramente o regime de busca (*exploration*) e do refinamento das regras existentes (*exploitation*) (SÁ et al., 2008).

Outra possibilidade é a de que a configuração dos parâmetros não foi adequada, uma vez que essa técnica é particularmente sensível a alguns dos parâmetros (BULL, 2004). A frequência de aplicação dos operadores genéticos pode estar elevada, por exemplo.

6. Conclusões

Este trabalho realizou um estudo de técnicas de sistemas classificadores a fim de avaliar a aplicação dessas no contexto de balanceamento dinâmico de jogos. Foram revisados os principais conceitos relacionados ao problema, bem como algumas das principais abordagens dinâmicas existentes na literatura.

Das diversas categorias de sistemas classificadores, uma foi escolhida como base para implementação utilizando a técnica *Top Culling* como mecanismo de balanceamento semelhante ao utilizado por scripts dinâmicos.

Os resultados experimentais indicaram a necessidade da realização de uma melhor configuração de parâmetros do agente implementado. A partir de pesquisas adicionais, considerou-se também a importância de explorar mais mecanismos utilizados no sistema, como o de seleção de ação, atribuição de crédito e função de aptidão das regras (ou a função desafio, utilizada neste trabalho).

Apesar dos resultados encontrados, os estudos na área de sistemas classificadores parecem promissores, dada a diversidade de aplicações existentes (BULL, 2004). Técnicas mais sofisticadas de sistemas classificadores como Fuzzy XCS (CASILLAS et al., 2004) podem vir a sanar alguns dos problemas aqui encontrados, servindo como extensão à abordagem aqui proposta.

Referências bibliográficas

ANDRADE, F. et al. Knock'em: Um Estudo de Caso de Processamento Gráfico e Inteligência Artificial para Jogos de Luta. In: II WORKSHOP BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL. **Anais**... [S.l: s.n.]. Disponível em: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Knock+'+em+:+Um+Estudo+de+Caso+de+Processamento+Gráfico+e+Inteligência+Artificial+para+Jogos+de+Luta#0>. Acesso em: 10 dez. 2011, 2003.

ANDRADE, G. **Balanceamento Dinâmico de Jogos: Uma Abordagem Baseada em Aprendizagem por Reforço**. Universidade Federal de Pernambuco - [S.l.]. 2006.

BOOKER, L. B.;; GOLDBERG, D. E. e HOLLAND, J.H. Classifier systems and genetic algorithms. **Artificial Intelligence**, v. 40, n. 1-3, p. 235-282, doi:10.1016/0004-3702(89)90050-7, 1989.

BULL, L. e KOVACS, T. Learning Classifier Systems. In: BULL, L.; KOVACS, T. (Eds.). **Foundations of Learning Classifier Systems**. [S.l.]: Springer, 2005. .

BULL, L. Learning Classifier Systems: A Brief Introduction. In: BULL, L. (Ed.). **Applications of Learning Classifier Systems**. [S.l.]: Springer, 2004. v. 6p. 3-14.

CASILLAS, J. et al. Fuzzy XCS : An Accuracy-Based Fuzzy Classifier System. 2004.

CHOMSKY, N. **Aspects of the Theory of Syntax**. Massachusetts: The MIT press, 1965. v. 119

DEMASI, Pedro e CRUZ, A. J. de O. Evolução de Agentes em Tempo Real para Jogos Eletrônicos de Ação. **II Workshop Brasileiro de Jogos e**, 2003.

DEMASI, P. e CRUZ, A. Online Coevolution for Action Games. **Proceedings of The 3rd International Conference on Intelligent Games And Simulation**, p. 113-20, 2002.

HOLLAND, J.H. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning An Artificial Intelligence Approach**. [S.l.]: Morgan Kaufmann, 1986. v. 2p. 593-623.

HOLLAND, John H.;; HOLYOAK, K. J. e NISBETT, R. E. **Induction: processes of inference, learning, and discovery**. [S.l: s.n.], 1989.

HUNICKE, R. e CHAPMAN, V. AI for dynamic difficulty adjustment in games. In: CHALLENGES IN GAME ARTIFICIAL INTELLIGENCE AAAI WORKSHOP. **Anais**... San Jose: [s.n.]. Disponível em: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AI+for+Dynamic+Difficulty+Adjustment+in+Games#0>. Acesso em: 11 dez. 2011, 2004.

ISAAC, B. **Learning Behaviours of Autonomous Agents**. Oxford University - [S.l.]. 2005.

JUUL, J. The Game, the Player, the World: Looking for a Heart of Gameness. (M. Copier & J. Raessens, Eds.)In: LEVEL UP: DIGITAL GAMES RESEARCH CONFERENCE PROCEEDINGS. **Anais**... Utrecht: Utrecht University. Disponível em: <http://www.jesperjuul.net/text/gameplayerworld/>. Acesso em: 9 dez. 2011, 2003.

MACIEL JUNIOR, C. A. V. **Imitation-based approach in game balancing**. Univerdade Federal de Pernambuco - [S.l.]. 2010.

MOIOLI, R. C.;; VARGAS, P. A. e ZUBEN, F. J. V. Analysing Learning Classifier Systems in Reactive and Non-Reactive Robotic Tasks. **Lecture Notes in Computer Science**, v. 4998/2008, p. 286-305, 2008.

PEROURNALNAIK, M. e ÉNÉE, G. Towards Self-Adjustment of Adapted Pittsburgh Classifier System Cognitive Capacity on Multi-Step Problems. **2008 Eighth International Conference on Hybrid Intelligent Systems**, p. 885-892, doi:10.1109/HIS.2008.105, 2008.

PONSEN, M. **Improving Adptive Game AI with Evolutionary Learning**. 2004.

RICHARDS, R. A. **Classifier Systems and Genetic Algorithms (chapter 3 of Zeroth-order Shape Optimization Utilizing a Learning Classifier System)**. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/gudwin/ea072/classifier.pdf>. Acesso em: 8 dez. 2011.

RUSSELL, S. e NORVIG, P. Inteligência artificial. **Inteligência Artificial**, v. 3, p. 1, 2004.

SALEN, K. e ZIMMERMAN, E. **Rules of play: game design fundamentals**. Disponível em: <http://books.google.com/books?id=UM-xyczrZuQC&pgis=1>. Acesso em: 6 dez. 2011.

SPRONCK, P.;; PONSEN, M. e POSTMA, I. S.-kuyper E. Adaptive game AI with dynamic scripting. **Machine Learning**, p. 217-248, doi:10.1007/s10994-006-6205-6, 2006.

SUTTON, R. S. e BARTO, A. G. **Reinforcement Learning : An Introduction**. Disponível em: <http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html>. Acesso em: 28 nov. 2011.

SÁ, Â. et al. Exploration vs . Exploitation in Differential Evolution. **AISB 2008 Convention**, n. i, p. 1-7, 2008.

WATKINS, C. J. C. H. e DAYAN, P. Q-learning. **Machine Learning**, v. 8, n. 3-4, p. 279-292, doi:10.1007/BF00992698, 1992.

WILSON, S. W. ZCS: A Zeroth Level Classifier System. **Evolutionary Computation**, v. 2, n. 1, p. 1-18, doi:10.1162/evco.1994.2.1.1, 1994.

WILSON, S. W. Classifier Fitness Based on Accuracy. **Evolutionary Computation**, v. 3, n. 2, p. 149-175, doi:10.1162/evco.1995.3.2.149, 1995.