

Knock'em: Um Estudo de Caso de Processamento Gráfico e Inteligência Artificial para Jogos de Luta

FERNANDO DA CUNHA ANDRADE NETO
GUSTAVO DANZI DE ANDRADE
ANDRÉ ROBERTO GOUVEIA DO AMARAL LEITÃO
ANDRÉ WILSON BROTTTO FURTADO
GEBER LISBOA RAMALHO

Universidade Federal de Pernambuco (UFPE) – Centro de Informática (CIn)
Av. Prof. Luís Freire, s/n, Cidade Universitária, CEP 50740-540 – Recife/PE/Brazil
{fcan,gda,argal,awbf,glr}@cin.ufpe.br

Resumo

Apesar da evidente popularidade que permeia a história dos jogos de luta, é notável a quase inexistência de informação técnica e formal sobre o tema. Visando contribuir para a mudança desse cenário, este artigo apresenta um estudo de caso de desenvolvimento de jogos de luta, com enfoque na inteligência artificial e no processamento gráfico, principais desafios na elaboração de jogos desse gênero.

Palavras-chave: *jogos de luta, inteligência artificial, processamento gráfico*

1 Introdução

Há cerca de 20 anos, a fabricante Data East® solidificava no mundo dos jogos eletrônicos um gênero que ainda iria conquistar milhares de fãs e muita popularidade: “*one-on-one beat-‘em-up games*” ou, mais simplesmente, jogos de luta no estilo “um-contra-um”, em que o objetivo é levar o adversário a nocaute antes que o tempo termine. O jogo da Data East® em questão, intitulado Karate Champ [1], é considerado o precursor do gênero. Resistindo heroicamente ao *crash* de 1984 da indústria de videogames [2], o jogo introduziu conceitos para jogos de luta que persistem até hoje, oferecendo à comunidade de jogadores e programadores de jogos uma alternativa aos tradicionais jogos no estilo “bate-rebate” ou “combate alienígena”.

Seguindo o Karate Champ, novas gerações de jogos de luta foram evoluindo em ritmo bastante acelerado. A série Street Fighter [3] revelou a importância da identidade dos personagens em jogos de luta. Mortal Kombat [4] fez sucesso pela variedade de golpes, efeitos e segredos especiais (*fatality, babality, etc.*). Jogos de luta 3D, como os da série Virtua

Fighter [5], literalmente situaram o gênero em uma nova dimensão, viabilizando um grau de realismo e complexidade antes inconcebíveis.

A Figura 1 ilustra parte dessa evolução dos jogos de luta, apresentando jogos desse gênero de diferentes épocas. Em geral, observa-se que jogos de luta evoluíram em relação a vários aspectos, como a qualidade gráfica, a jogabilidade, a qualidade sonora e a inteligência artificial, por exemplo, contribuindo para o aumento da diversão e do prazer da comunidade de jogadores e programadores de jogos de luta.

Vinte anos, entretanto, não foram suficientes para que esta mesma comunidade se organizasse formalmente no objetivo de documentar e registrar de maneira produtiva as técnicas e armadilhas no projeto e implementação de jogos de luta. Apesar da evidente popularidade e acelerado ritmo de evolução do gênero, é notável a quase inexistência de artigos ou informações que facilitem o trabalho de programadores, *designers* e *publishers* de jogos de luta.

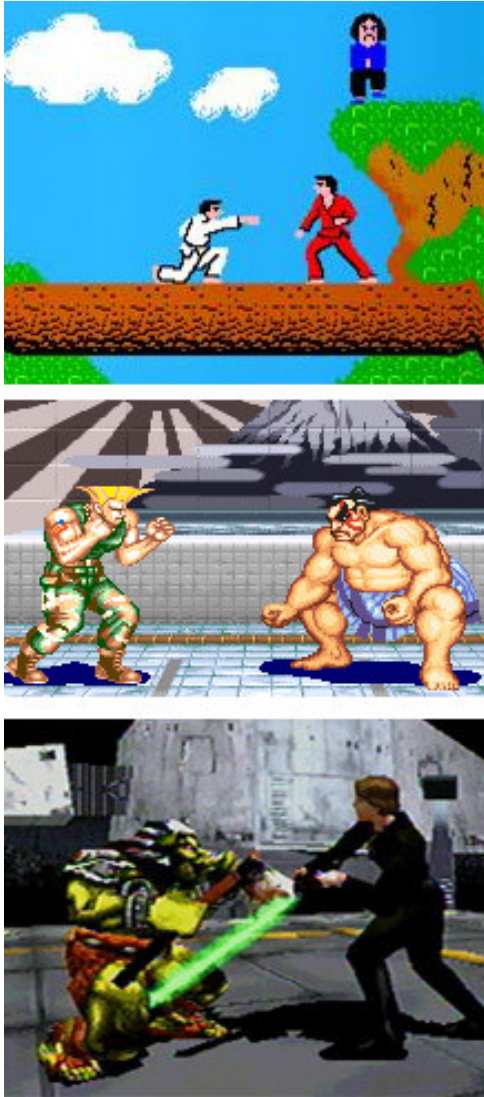


Figura 1. Três momentos distintos da evolução de jogos de luta (de cima para baixo): Karate Champ, Street Fighter 2 e Masters of Tera Kasi.

Uma pesquisa em dois dos maiores *sites* de suporte ao desenvolvimento de jogos eletrônicos, o Gamasutra [6] e o GameDev.net [7], apresentou resultados pouco satisfatórios. Utilizando-se as ferramentas de busca presentes em cada *site*, os seguintes termos foram procurados: “fighting games”, “fighting game”, “fight games” e “fight game”. Apesar das pesquisas retornarem 99 *sites* distintos, em apenas **um** deles o autor realmente aborda jogos de luta como tema principal (Carless 2001 [8]).

Nos demais resultados, foi constatado que jogos de luta são citados apenas para exemplificar um outro assunto principal, como detecção de colisão, categorização de jogos, casamento de padrão para combinações de teclas, efeitos sonoros, performance e dispositivos móveis¹, entre outros. Em resumo, à exceção do artigo de Carless, subsídios para o projeto e implementação de jogos de luta estão limitados a informações não-objetivas do tipo:

- “A detecção de colisão determina se dois objetos do jogo entraram em contato. Em um jogo de luta, por exemplo, é importante saber se o soco de um personagem atingiu ou não o adversário.” [10]
- “Em um jogo do tipo *arcade*, o único momento em que o som deve ficar mais calmo é quando o jogador pára de fazer alguma coisa, como se afastar do adversário em um jogo de luta, por exemplo.” [11]

Como se pode observar, informações deste tipo deixam muito a desejar aos desenvolvedores de jogos de luta em geral, principalmente por se tratar da essência do resultado de uma pesquisa em dois dos maiores *sites* de suporte ao desenvolvimento de jogos.

Os autores deste trabalho, portanto, acreditam que há muito conteúdo não-divulgado que pode enriquecer e facilitar o trabalho de futuros desenvolvedores de jogos de luta. Este artigo pretende contribuir para o início de um registro mais formal sobre a experiência de equipes de desenvolvimento de software em projetos de criação de jogos do gênero. Para tanto, é apresentado o Knock'em, um estudo de caso de desenvolvimento de jogos de luta vivenciado pelos autores. Neste momento, foi preferido abordar os aspectos gráficos e de inteligência artificial utilizados no jogo, por serem dois dos mais complexos desafios inerentes a jogos desse tipo. Não é assumido aqui que a abordagem utilizada foi a mais correta. Pelo contrário, é objetivo deste trabalho

¹ Neste caso, inclusive, jogos de luta são citados como contra-exemplo. Em outras palavras, é defendido que a aplicabilidade de jogos de luta é restrita em dispositivos móveis. [9]

promover uma discussão sobre o tema e registrar a experiência vivenciada no projeto.

Este artigo está estruturado da seguinte maneira: a seção 2 apresenta uma visão geral do Knock'em, permitindo a inserção do leitor no contexto do jogo antes de serem abordados aspectos mais avançados. A seção 3, por sua vez, detalha as abordagens utilizadas para a modelagem gráfica do jogo, como renderização de *sprites* ou detecção de colisão, por exemplo. A seção 4 introduz as estratégias utilizadas para modelar a inteligência artificial do computador que, apesar de simples, mostrou-se satisfatória. A seção 5, por fim, realiza conclusões acerca do trabalho realizado.

2 Visão Geral do Knock'em

O Knock'em consiste em um jogo bidimensional de luta entre dois jogadores. A primeira versão do jogo² foi desenvolvida durante a disciplina *Projeto e Implementação de Jogos* do Centro de Informática (CIn) da UFPE, pioneira no gênero no Brasil. Tal fato implicou em uma escassez do tempo disponível para o projeto e a implementação do Knock'em: apenas três meses. Devido a este curto período de tempo, grande parte da arte gráfica e sonora do Knock'em foi reutilizada a partir de *sites* especializados em arte para jogos de luta [13].

Para a fase de concepção do Knock'em, que demandou grande criatividade da equipe do projeto, diversos jogos de luta foram pesquisados com o objetivo de identificar as principais características e peculiaridades inerentes a este gênero de jogo. As idéias iniciais foram formalizadas no documento de *game design* do Knock'em, registrando os mais variados aspectos do jogo como, por exemplo, sua trama, personagens, ambiente, objetivos do jogador e esquema de pontuação.

O foco principal do Knock'em consiste em disputar, com um personagem previamente escolhido, torneios ao redor do mundo objetivando vencê-los. Este modo de jogo é

² Uma segunda versão do jogo, envolvendo *aprendizagem por reforço* para melhorar sua IA, estava sendo desenvolvida quando da publicação deste artigo. Ela não será abordada nesta visão geral.

denominado *Tournament*. Cada torneio é localizado em diferentes regiões do globo terrestre, e seus respectivos combates possuem ambientes, adversários e recompensas variados. Esta última característica introduz um elemento de estratégia do jogo. Ao vencer lutas e torneios, o personagem adquire uma certa quantia em dinheiro. Esta quantia pode ser usada pelo personagem para pagar diferentes tipos de treinamentos. Cada treinamento melhora uma habilidade específica do personagem, como sua destreza, força ou resistência. Além de treinamentos físicos, existem alguns treinamentos espirituais que permitem ao jogador aprimorar certos golpes especiais, chamados *magias*. Quanto mais complexo for o torneio a ser disputado pelo jogador, mais bem-preparados serão seus adversários. Dessa forma, o jogador planeja a evolução de seu personagem através de treinamentos, com o objetivo de se equiparar com seus inimigos. Esta característica torna o Knock'em positivamente diferenciado dos tradicionais jogos de luta³.

A Figura 2 apresenta um *screenshot* do jogo durante uma luta entre dois personagens. A Figura 3, por sua vez, apresenta a tela de seleção de personagens. A Figura 4, por fim, mostra o "Templo de Treinamento de Okina", interface na qual o jogador compra seus treinamentos.



Figura 2. Screenshot do jogo durante uma luta.

³ Esta característica não é singular ao Knock'em. Alguns jogos de luta, como OMF (One Must Fall) [14] e Budokan [15] também a apresentam.



Figura 3. Tela de seleção de personagens.



Figura 4. Tela de escolha de treinamentos.

Além do modo *Tournament*, o Knock'em pode ser jogado em modo *Two Players*. Neste caso, dois jogadores, utilizando-se do mesmo teclado, lutam entre si, estando ausente o elemento de estratégia. Neste modo, como no anterior, os combates possuem apenas um único round, considerando-se vencedor o personagem que possuir menos danos ao término do tempo ou o que conseguir derrotar seu adversário.

Depois de definida e validada a concepção do jogo, teve início a fase de implementação. Como linguagem de programação, foi definida a utilização da linguagem orientada a objetos C++, que se adequou à arquitetura projetada para o Knock'em. Adicionalmente, foi utilizada a biblioteca *DirectX*, que facilita o

desenvolvimento e a otimização de jogos ao oferecer um conjunto de interfaces para o controle eficiente do hardware. Em especial, destaca-se o uso do módulo *DirectDraw*, que provê suporte a gráficos e manipulação direta na memória de vídeo, sendo bastante utilizada para jogos 2D no tratamento de imagens e texturas.

A escolha da linguagem C++ e da biblioteca *DirectX*, na verdade, foi um requisito para a utilização do motor para jogos *Forge16V* [17] que proporciona uma excelente abstração para operações de mais baixo-nível. A sua utilização viabilizou, assim, mais funcionalidades em menos tempo, ao disponibilizar vários módulos como o manipulador de som, o controlador de interface de entrada e o módulo gráfico. A performance do *Forge16V* se mostrou bastante satisfatória para o Knock'em, apesar de alguns poucos problemas de estabilidade. No geral, apesar do foco do *Forge16V* ser jogos isométricos, pode-se dizer que o mesmo foi validado com sucesso para jogos não-isométricos, como o Knock'em.

3 Processamento Gráfico do Knock'em

A interface gráfica sempre ocupou lugar de destaque no projeto e implementação de jogos. Em jogos de luta, esta característica é acentuada em virtude do apelo visual exigido, em que a variedade de personagens e de golpes é um forte atrativo para jogadores.

Uma simplificação do diagrama de classes do Knock'em é exibido na Figura 5. Nesse diagrama, não são mostrados os atributos e métodos relativos a aspectos de implementação, como variáveis de controle, e métodos get/set.

Uma classe básica é a classe *Sprite*, que possui as seqüências de imagens de golpes e ações dos personagens. A classe possui um *KnockEmTileset* e informações do *sprite* atual e do tempo de *delay* na animação dos *sprites*. A classe *KnockEmTileset* surgiu na necessidade de se inverter os pontos de âncora de um *sprite* na execução de um *Flip*. Esta classe utiliza os métodos de gerenciamento de *tiles* do motor *Forge16V*, e apenas sobrescreve seu método de *Flip*.

A classe *Fighter* representa os lutadores do jogo, e possui atributos como os *sprites* de cada

jogador, suas características (força, resistência, etc.) que podem ser alteradas com o decorrer do jogo, sua posição, seu estado atual e outros fatores. O comportamento de um lutador é implementado como uma máquina de estados, que limita as ações do lutador de acordo com seu estado atual. Os lutadores também estão associados à classe *Fireball*, que corresponde às magias que podem ser lançadas durante uma luta.

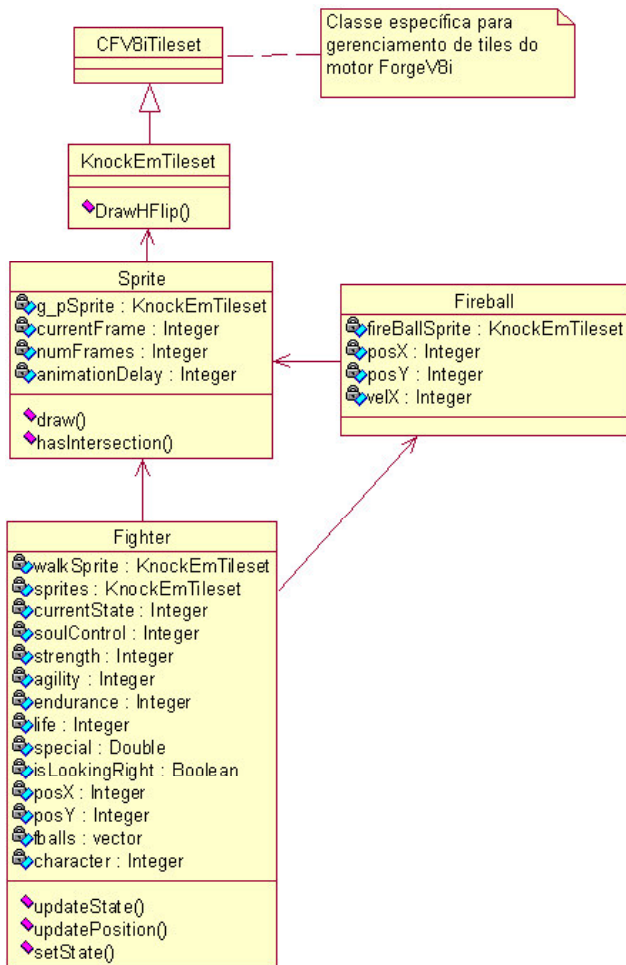


Figura 5. Classes do componente gráfico do Knock'em.

O processamento gráfico do Knock'em é baseado em *sprites*. Estas são imagens, de tamanho arbitrário, usadas por agentes que se movem pela tela. No Knock'em, os personagens possuem um *sprite animado* para cada ação que

executam: soco, chute, movimentação, pulo, etc. Esses *sprites animados* são compostos por *sprites simples*, ou *frames* (quadros), que representam cada etapa da animação.

Em cada *sprite*, é necessário definir algumas informações gráficas do jogo: cor de transparência, moldura dos *frames*, tamanho real de cada *frame* e pontos de âncora [12]. A cor de transparência é uma cor que não é desenhada na tela durante a exibição do *sprite*. A moldura é necessária para a delimitação de *frames* de um mesmo *sprite*. O tamanho real de um *frame* indica a área da imagem realmente utilizada pelo *frame*. Por fim, pontos de âncora são pontos que definem a correlação de um ponto de um espaço inicial em um espaço final. Por exemplo, com uma âncora de (x, y) cada ponto do espaço inicial precisa ser transladado de x e y posições para ser representado no espaço final.

O tratamento destas questões no Knock'em é realizado pelo motor *Forge16V*, que implementa o processamento de *sprites*, eficientemente, através do uso da biblioteca gráfica *DirectDraw*. Esta característica de implementação é essencial para jogos de luta, que exigem muita performance das bibliotecas gráficas. O motor *Forge16V* também se mostrou adequado na implementação de efeitos gráficos clássicos, como *fade-in*, *fade-out* e *blending*, por exemplo.

No Knock'em, a manipulação das imagens gráficas é encapsulada em um único componente, responsável pelo controle das animações dos personagens (ações de ataque, de ferimentos, golpes especiais, etc.). Como estas animações envolvem muitos *frames*, torna-se necessário resolver de forma eficiente o problema de codificação das informações gráficas. Abordagens comuns consistem em inseri-las no próprio código-fonte (abordagem *hard-coded*) ou em arquivos de configuração, como um XML ou arquivos de texto em um formato previamente definido. Entretanto, a quantidade de *sprites* utilizadas no Knock'em inviabiliza estas abordagens.

Para solucionar este problema, foi utilizado um método de inserção de informações nos próprios arquivos gráficos, definido inicialmente para gerenciamento de *tiles* [12]. Este método

atribui a um determinado conjunto de cores do próprio arquivo funções especiais de marcação, como delimitação de *frames*, pontos de âncora e cor de transparência. Com isso, o trabalho de design pôde ser desenvolvido independente dos padrões de implementação. A Figura 6 ilustra essa abordagem.

O componente de gerenciamento de imagens também é responsável pela detecção de colisão. Em jogos de luta, a checagem de colisão tem papel fundamental na jogabilidade, à medida que precisa traduzir realismo nos efeitos dos ataques. No Knock'em, foi utilizado a combinação de duas técnicas: bounded-box, e checagem pixel a pixel [16]. A técnica bounded-box representa uma *sprite* como o menor retângulo que o contém, realizando uma macro checagem que consome pouco processamento. Caso os retângulos de dois *sprites* se intersectem, a checagem de pixel é realizada dentro da área de interseção. Com isso, foi possível obter resultados precisos, com pouco custo de processamento.

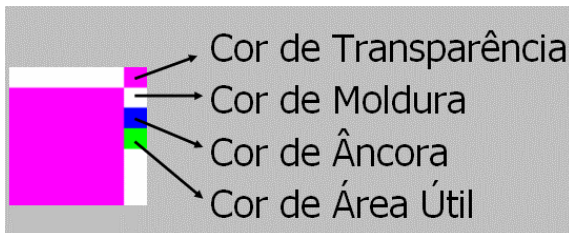


Figura 6. Codificação de informações gráficas.

A checagem de colisão ainda precisou lidar com outro problema: nem todos os *frames* de um *sprite animado* de ataque causam, efetivamente, dano ao oponente. Por exemplo, na Figura 7 apenas o *frame* central causa dano, embora os demais também sejam *frames* de ataque. Este mesmo problema se repete no lançamento de magias, em que uma “bola de fogo” não é necessariamente lançada no primeiro *frame* da animação. Para solucionar este problema, específico de jogos de luta, foi necessário implementar checagens individuais para cada golpe de cada personagem, visando descobrir se um dado *frame* da animação realmente causa dano ou se é o momento correto de lançamento da magia. Essa checagem é realizada por um procedimento que associa a

cada *sprite animado* os correspondentes *sprites* de dano ou lançamento de magia. No primeiro caso, pode haver mais de um *sprite* causador de dano, de forma que o procedimento define intervalos de *sprites* de impacto. Já no segundo caso, só pode haver uma “bola de fogo” lançada em cada magia. As associações são fixas, estabelecidas como constantes no código-fonte do jogo.



Figura 7. Exemplo de *sprite animado* para a ação “soco”.

Uma vez definido quando uma ação de ataque efetivamente causa dano, é interessante determinar como o oponente é danificado. Isso corresponde a diferenciar golpes no rosto de golpes na barriga, por exemplo, com os primeiros provocando maior dano. Existem abordagens sofisticadas para tratar este problema. Uma possibilidade é a checagem de vários *bounded-boxes*, um para cada parte relevante do lutador (cabeça, tronco e membros). Outra alternativa é a construção de máscaras para todos os *sprites*, associando cores, que representam diferentes resistências a danos, a regiões distintas do lutador; a cada golpe diferido, é checada a cor atingida na máscara do oponente. Entretanto, no Knock'em essas abordagens não foram implementadas, optando-se por considerar danos uniformes em todo o corpo do oponente.

Para a temporização das animações, isto é, o tempo entre exibição de *frames* de um mesmo *sprite*, é realizada a checagem do *timer* ainda no laço principal do jogo.

Em jogos de luta, uma dificuldade a mais é a necessidade de se manter sincronia nas animações de pulos. Em geral, essas animações possuem poucos *frames* e o tempo da animação precisa ser controlado de acordo com a duração do pulo. Para isso, cada *frame* é exibido várias vezes, de forma que o total de *frames* exibidos

na animação seja igual à duração do pulo. Além disso, um controle mais cauteloso deve ser realizado para que o *sprite* que representa o auge do pulo seja exibido no momento certo, passando uma sensação maior de realismo. No Knock'em, é definido que cada animação de pulo possui a mesma duração e a mesma quantidade de *sprites*, de modo que dividindo-se a duração total pela quantidade de *frames*, obtém-se a quantidade de vezes que cada *frame* é exibido na tela. Para que o auge do pulo seja exibido no momento em que o lutador atinge a maior posição vertical, é implementado um procedimento que divide a duração do pulo em várias unidades. A cada unidade são adicionados ou subtraídos valores que dependem de constantes e do atributo de agilidade, que é variável, de cada personagem. As constantes são determinadas de modo que no começo e final do pulo, por exemplo, o deslocamento seja maior do que no meio.

Para a exibição de textos, o Knock'em utilizou uma combinação de funções do motor *Forge16V* com imagens gráficas. As funções de desenho de texto do motor foram utilizadas para a exibição de textos longos e de menor importância, como a história dos jogadores. Para textos de impacto, como a indicação do vencedor (*player wins* ou *computer wins*), foram elaboradas imagens gráficas para cada texto, permitindo uma arte gráfica mais atrativa.

4 Inteligência Artificial no Knock'em

Assim como o processamento gráfico, a inteligência artificial é uma das características mais perceptíveis em jogos de luta. Ao determinar o comportamento dos adversários, influenciando diretamente no balanceamento da dificuldade, a IA mostra-se um fator decisivo para a aceitação e o sucesso de jogos do gênero.

No sentido de identificar aspectos que revelem o impacto da IA no *gameplay* do Knock'em, foram levantadas, em sua concepção, diversas questões a serem analisadas durante a evolução do jogo:

- O nível de desafio está balanceado? Os inimigos devem ser bons o suficiente para manter o interesse do jogador, mas a

experiência com o jogo não pode ser frustrante pelo excesso de dificuldade.

- A IA está justa? O jogo deve promover desafio sem passar a sensação de que o computador sabe mais do que deveria, isto é, o jogador não pode ter a sensação de “trapaça”.
- É possível haver um aumento gradual da dificuldade, acompanhando proporcionalmente a evolução do jogador?
- Como agregar mais valor à identidade dos personagens? Além da variação de golpes, deve existir um mecanismo que permita a identificação de personalidades e estilos de luta próprios para cada personagem.
- Como inserir no jogo os elementos de imprevisibilidade e criatividade, maximizando o *replay-value*?
- Como obter um resultado satisfatório no desenvolvimento da IA em apenas 3 meses, de modo a cumprir os *deadlines* do projeto?

Em resposta a esta última pergunta, decidiu-se adotar uma estratégia de desenvolvimento incremental. O primeiro passo foi fazer uma modelagem básica de comportamento para os adversários. Esta modelagem inicial consistiu na criação de um sistema baseado em regras, em que os lutadores controlados pelo computador foram concebidos como agentes reativos [20]. A partir de percepções sobre o próprio estado e o estado do adversário, agentes deste tipo tomam decisões associando tais percepções a uma ação de uma base de regras pré-definida. Alguns exemplos de regras para estes agentes reativos estão apresentados na Tabela 1.

Tabela 1. Exemplos de regras básicas para agentes do Knock'em

Percepção	Ação
Inimigo lançando magia	Defesa
Inimigo próximo e sem defesa	Ataque (socos ou chutes)
Inimigo distante	Lançar magia

Esta modelagem foi estimulada pela facilidade de implementação, mas resultou em diversos pontos negativos, como a rigidez do

comportamento e a previsibilidade dos personagens.

No contexto do jogo, os estados estão relacionados àquilo que um lutador está fazendo em um determinado momento (parado, saltando ou chutando, por exemplo). Existem diversas restrições nas transições destes estados a serem consideradas para a animação correta dos personagens e por questões de jogabilidade. Por exemplo, foi definido que um lutador não pode dar dois chutes no mesmo salto ou dar um golpe antes que o movimento de outro golpe anterior termine. O modelo de máquina de estados e a utilização de orientação a objeto possibilitaram o encapsulamento destas restrições, abstraídas no processo de programação da IA. Em outras palavras, o desenvolvimento da base de regras não se preocupou com essas restrições nas transições entre os estados.

Em termos de implementação, isto significa que o estado do lutador é um atributo privado e só pode ser modificado através de um método específico (“SetState”). Este método encapsula regras do tipo:

- Os estados que representam a vitória ou derrota do jogador podem interromper qualquer outro estado;
- Os estados “andando” (para esquerda ou direita), “parado” e “agachado” podem ser interrompidos por qualquer outro estado;
- Estados de pulo (“pulando para a esquerda”, “pulando para a direita” e “pulando sem direção”), só podem ser interrompidos por estados de chute ou soco no ar.

Desse modo, a IA do jogo deve se preocupar apenas em identificar qual será o próximo estado de um lutador e solicitar essa mudança de estado através do método SetState, que se encarregará das restrições e validações referentes à máquina de estados do Knock'em.

Posteriormente, algumas adaptações na base de regras permitiram que conjuntos distintos de regras fossem associados a grupos de personagens específicos, possibilitando uma variação de comportamento que diferencia lutadores com estilos próprios de luta. Foram modelados três estilos:

- Defensivo: pouco ousado, procura fugir e tem preferência por golpes de longa distância.
- Ofensivo: agressivo, ataca sempre e procura não dar distância para o adversário.
- Ágil: movimenta-se bastante, varia os golpes e alterna pulos com ataques.

Os personagens foram classificados nesses estilos de acordo com seu perfil físico e descrição conceitual, o que agregou bastante valor à variedade de lutadores ao dar-lhes mais identidade.

Embora mais adaptações pudessem ser feitas na base de regras, elevando a complexidade do comportamento adversário, percebeu-se que a abordagem de IA utilizada não apresenta mecanismos mais adequados para que os adversários não se revelem previsíveis após um certo tempo de jogo. Dessa forma, o desafio proposto ao jogador fica limitado à descoberta do padrão de comportamento inimigo.

É sabido que a adaptabilidade do adversário tem impacto direto no aumento do *replay-value* do jogo, ao sempre exigir do jogador mais treino e a criação de novas estratégias. Dessa forma, a equipe de desenvolvimento reconheceu que a IA até então utilizada no Knock'em estava comprometendo sua qualidade como um todo. Os esforços foram direcionados, portanto, para o lançamento de uma segunda versão do jogo que priorizasse esse aspecto. A abordagem utilizada foi a de *aprendizado por reforço* [19].

Aprendizagem por reforço é um problema em que um agente escolhe as ações que serão executadas com base apenas na sua interação com o ambiente. Neste modelo, um agente decide sua ação a partir de sua percepção do ambiente, recebendo um reforço (recompensa ou punição). Analisando os reforços de ações passadas, o agente pode aprender aquelas que produziram melhores resultados.

Para implementar um agente desse tipo, é necessário definir como os estados serão percebidos, as ações que podem ser tomadas e como a recompensa de cada ação será calculada. No Knock'em, os agentes aprendizes são os lutadores, e a forma como esses agentes atuam no ambiente é especificada a seguir.

A representação dos estados do ambiente foi definida como uma tupla:

$$S = (S_{\text{AGENTE}}, S_{\text{OPONENTE}}, D, M, F)$$

No caso, S_{AGENTE} e S_{OPONENTE} significam o estado do agente e do oponente no jogo, respectivamente, podendo assumir os seguintes valores: Atacando, Atacando agachado, Agachado, Defendendo, Parado, Lançando magia, Fugindo, Aproximando, Pulando, Pulando atacando, Pulando fugindo e Pulando aproximando. D significa a distância entre os lutadores, discretizada para os valores “perto”, “médio” e “longe”. M representa a quantidade de *mana* (“combustível da magia”) do agente, que pode ser “suficiente” ou “insuficiente”. F , por fim, significa a existência e distância de bolas de fogo inimigas (*fireballs*), cujos valores são “perto”, “médio”, “longe” ou “inexistente”.

A escolha desses atributos foi baseada no impacto que essas variáveis possuem no desempenho de um lutador. Por exemplo, a inclusão da variável D (distância) na representação do estado é relevante para que o agente diferencie os resultados de um soco executado quando o oponente está longe ou quando está no raio de alcance do golpe. Por outro lado, foi necessário discretizar a representação das variáveis contínuas D (distância) e M (mana) visando diminuir o número de estados possíveis a serem percebidos pelo agente. Com essa abordagem chegou-se a um total de 3.456 estados diferentes.

As ações executadas por um agente aprendiz são os golpes e movimentações do lutador: Socar forte, Socar fraco, Chutar forte, Chutar fraco, Pular, Fugir Pulando, Aproximar-se pulando, Lançar magia, Defender, Aproximar-se, Fugir, Agachar-se e Ficar parado. Estas ações representam o mesmo conjunto de comandos que podem ser executados por um lutador humano, de forma que ambos os lutadores briguem sob as mesmas condições.

O reforço de cada ação, por fim, foi modelado como a diferença de *life* dos lutadores no estado atual menos a diferença de *life* dos lutadores no último estado. Essa abordagem é simples de ser implementada e bastante representativa para o aprendizado do agente. Quando o agente efetua um golpe de sucesso, o

life do oponente é decrementado, e o valor desse decremento é transformado em recompensa. Quando o agente sofre um golpe, seu *life* é decrementado e esse decremento é transformado em punição.

Apesar desta segunda versão do Knock'em ainda não estar concluída quando da publicação deste artigo, observações iniciais já permitem concluir uma melhora significativa na IA, na diversão e no *replay value* do jogo de uma maneira geral. Adversários inicialmente derrotados com facilidade, uma vez treinados por um período de tempo razoável, mostraram-se bastante desafiadores. Espera-se que a implementação de “modos de simulação”, em que um lutador aprendiz é colocado para lutar por tempo indeterminado com outros lutadores também controlados pelo computador, permita liberdade e facilidade na determinação da dificuldade do jogo.

Em resumo, pôde-se constatar que a elaboração da inteligência artificial para jogos de luta é uma etapa bastante desafiadora e complexa do projeto, sobretudo pelos vários requisitos de IA identificados e pela grande possibilidade de abordagens.

5 Conclusões

No geral, apesar de algumas limitações, o Knock'em pode ser considerado um caso bem-sucedido: apesar do curto tempo disponível para a conclusão do jogo, ele obteve um *feedback* bastante positivo dos usuários, principalmente em relação à performance e jogabilidade, e superou as expectativas da disciplina de Projeto e Implementação de Jogos, segundo os avaliadores.

Este artigo teve o propósito de apresentar uma experiência particular dos autores na elaboração de jogos de luta, com foco no processamento gráfico e na inteligência artificial. Apesar de este ser apenas um subconjunto do universo de características e detalhes de jogos deste gênero, espera-se que este artigo contribua para que a comunidade de jogadores e programadores de jogos de luta possa se organizar no sentido de enriquecer a bibliografia sobre o assunto.

6 Referências

1. Karate Champ at Coinop.org Arcade Game Resource, http://coinop.org/g.aspx/100088/Karate_Champ.html (01/08/2003)
2. Dark Watcher's Console History: The Crash of 1984, <http://darkwatcher.psxfanatics.com/console/thecrash.htm> (01/08/2003).
3. Street Fighter Legends, <http://www.chikapu.com/streetfighterlegends/index.html> (23/07/2003)
4. Mortal Kombat Special Forces, <http://www.earthrealm.com/specialforces/walkthrough.shtml> (23/07/2003)
5. PlayStation.com, http://uk.playstation.com/previews/previewStory.jhtml?storyId=101962_en_GB_PREV&linktype=GOO&locale=en_GBg (25/07/2003)
6. Gamasutra: The Art & Science of Making Games, <http://www.gamasutra.com/> (15/07/2003)
7. GameDev.net, <http://www.gamedev.net/> (15/07/2003)
8. Punch Kick Punch: A History of One-On-One Beat-'Em-Ups, http://www.gamasutra.com/features/game_design/19980424/punchkickpunch_01.htm
9. Developing Action-based Mobile Games, http://www.gamasutra.com/resource_guide/20021125/matthews_03.htm (17/07/2003)
10. GameDev.net Dictionary: Collision Detection, <http://www.gamedev.net/dict/term.asp?TermID=901> (17/07/2003)
11. Tales from the Trenches of Coin-Op Audio, http://www.gamasutra.com/features/19991118/Granner_pfv.htm (25/07/2003)
12. GameDev.net - Extended Graphical Templates for Sprite Management <http://www.gamedev.net/reference/articles/article1122.asp> (30/05/2003)
13. Moah's KOF91 2D Fighting Game Engine, <http://www.kof91.com/> (30/07/2003)
14. One Must Fall Home, <http://www.omf.com/> (20/07/2003)
15. Free Game Downloads, http://free-game-downloads.mosw.com/abandonware/pc/arcade_action/games_bm_bz/budokan.html (19/07/2003)
16. Collision Detection , <http://www.gamedev.net/reference/articles/article735.asp> (18/07/2003)
17. Rocha, E.S., Menezes, T.R., Vieira, M.F., Ramalho, G.L. & Santos, A. Forge 16V: Um Framework para Jogos Isométricos, WJogos 2003.
18. Machine Learning textbooks slides, <http://www-2.cs.cmu.edu/~tom/mlbook-chapter-slides.html> (03/08/2003)
19. Sutton & Barto Book: Reinforcement Learning: An Introduction, <http://www-anw.cs.umass.edu/~rich/book/the-book.html> (03/08/2003)
20. Russel, S., & Norvig, P. Artificial Intelligence: a Modern Approach. Englewood Cliffs: Prentice-Hall, Inc., 1995.