10th Brazilian Congress on Computational Intelligence (CBIC'2011), November 8 to 11, 2011, Fortaleza, Ceará Brazil

MODELAGEM DE COMPORTAMENTO DE JOGADORES POR MEIO DE AGRUPAMENTO ONLINE E DETECÇÃO DE NOVIDADES

Rosane M. M. Vallim, Ivan C. de Andrade, André C. P. L. F. de Carvalho

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
rvallim@icmc.usp.br
icaramello@grad.icmc.usp.br
andre@icmc.usp.br

Abstract – A aplicação de técnicas de Inteligência Artificial em jogos tem recebido crescente interesse por parte da comunidade científica. Isso se dá pois os jogos computacionais modernos provêem ambientes altamente dinâmicos e competitivos. Um dos desafios da Inteligência Artificial aplicada em jogos é a modelagem de comportamento de jogadores por meio de técnicas de Aprendizado de Máquina. Essa tarefa visa criar modelos que sejam representativos de jogadores humanos com o objetivo de melhor compreender o comportamento do jogador e assim ajustar parâmetros do jogo de modo a melhor atender suas necessidades. Este trabalho investiga o uso de técnicas de agrupamento para fluxos de dados para descrever um modelo de jogador durante sua interação com um jogo do gênero *First Person Shooter*. Técnicas de detecção de novidades são também utilizadas para o reconhecimento de alterações comportamentais de um jogador.

Keywords – Jogos computacionais, agrupamento de dados, detecção de novidades.

1 Introdução

Nos últimos anos, um crescente interesse da Inteligência Artificial (IA) pelos jogos computacionais tem sido observado. Pesquisas recentes tem investigado a implantação de técnicas de Aprendizado de Máquina (AM) nos componentes de IA dos jogos, de modo a aumentar a qualidade do comportamento das personagens [1]. A maior parte desses trabalhos aborda o problema de aprendizado de modo *offline*, ou seja, utilizando dados obtidos a partir de partidas de jogos computacionais. Esse modo de aprendizado é aplicado durante a fase de desenvolvimento de um jogo, com o objetivo de expor possíveis falhas de projeto e assim aumentar a qualidade da IA. Embora raros, existem alguns trabalhos que tentam aplicar aprendizado de modo *online*, ou seja, enquanto o jogador interage com o jogo. Esse modo de aprendizado, também denominado de IA adaptativa, visa aprender e atualizar modelos de decisão das personagens, adaptando seus comportamentos de modo a melhor se ajustar ao estilo do jogador.

Para que a IA se ajuste a um jogador, é necessário construir um modelo que represente como esse jogador interage com o jogo. Para construir esses modelos de modo *online*, algoritmos tradicionais de AM não são adequados, pois necessitam, em geral, de múltiplos passos de aprendizado, resultando em um tempo de processamento elevado. Para essas aplicações, são necessários algoritmos capazes de minerar fluxos de dados com baixo custo computacional, atualizando o modelo de decisão sempre que novos dados estejam disponíveis. Gama e Rodrigues [2] definem as características desejáveis de sistemas de aprendizado com essa capacidade: 1) Ser incremental; 2) Aprender de modo *online*; 3) Utilizar tempo constante para processar cada exemplo, usando quantidade fixa de memória; 4) Varrer o conjunto de treinamento uma única vez; 5) Levar em consideração mudanças de conceitos nos dados.

Este trabalho investiga o uso de técnicas para aprendizado em fluxos de dados em um jogo computacional do gênero FPS (do inglês, *First-Person Shooter*) para a construção de um modelo de jogador durante sua interação com o jogo, representando seu comportamento. A abordagem investigada procura identificar alterações no comportamento observado durante o jogo. Para isso, algoritmos para agrupamento de dados e detecção de novidades são utilizados. São estudados dois algoritmos para agrupamento *online* de dados em que o número de grupos *k* é desconhecido, em conjunto com uma técnica para detecção de novidades baseada em medidas de nível de informação.

Este trabalho está organizado como segue. A Seção 2 apresenta uma breve revisão de trabalhos que utilizam AM em jogos. A Seção 3 descreve conceitos relacionados a agrupamento *online* de dados, assim como os algoritmos utilizados neste trabalho. A técnica para detecção de novidades investigada é apresentada na Seção 4. A Seção 5 descreve como agrupamento e detecção de novidades foram utilizados para modelar o comportamento de jogadores no jogo Unreal Tournament 2004 (UT2004) [27]. Os experimentos realizados e os resultados obtidos são apresentados, respectivamente, nas Seções 6 e 7. Finalmente, na Seção 8, são apresentadas as principais conclusões deste trabalho.

2 AM aplicado a Jogos Computacionais

Uma quantidade crescente de pesquisas relacionadas ao uso de AM em plataformas de jogos computacionais vem sendo realizada. A maior parte dessas pesquisas utiliza algoritmos de AM para construção automática de modelos de decisão de personagens a partir de dados obtidos da interação entre os jogadores e o jogo. Em geral, utilizam técnicas de aprendizado

supervisionado em métricas de jogos para a obtenção de modelos de decisão, sendo necessário para isso a criação de conjuntos de dados contendo informações rotuladas daquilo que se deseja aprender [3,4]. Existem ainda trabalhos que utilizam abordagens evolutivas como mecanismo de aprendizado aplicado aos algoritmos de AM [5,6,7]. Abordagens não supervisionadas também têm sido exploradas para a definição de grupos que representam estilos de jogo [8,9]. Em geral, o que se deseja nesse caso é a obtenção de grupos que diferenciem jogadores segundo a forma como eles exploram o ambiente de jogo e segundo suas habilidades próprias.

Uma quantidade menor de trabalhos procura aplicar técnicas de AM durante a execução do jogo. O objetivo, nesse caso, é a produção de personagens capazes de se adaptar durante o jogo, aprendendo sobre o ambiente e sobre o jogador. O trabalho de Spronck [10] apresenta a técnica de *Scripting* Dinâmico, em que bases de regras evoluem por meio de atualização de pesos associados às regras. A utilização de aprendizado por reforço é explorada nos trabalhos de [11] e [12], com o objetivo de evoluir estratégias utilizadas por personagens de modo *online*. Finalmente, [13, 14] utilizam algoritmos evolutivos para obter personagens melhor adaptadas às suas tarefas a medida que essas interagem com o jogador.

3 Agrupamento em Fluxos de Dados

A maior parte dos algoritmos de agrupamento em fluxos de dados seguem as duas principais abordagens de agrupamento de dados: particional e hierárquica. Na abordagem particional os dados disponíveis são agrupados em partições de acordo com alguma medida de similaridade. Alguns exemplos de algoritmos particionais são o *Single-Pass K-Means* [15], *STREAM* [16] e *Leader-Follower Clustering* [17]. Já na abordagem hierárquica, são construídas hierarquias de partições, permitindo encontrar estruturas de grupos diferentes em cada nível da hierarquia. Dos algoritmos hierárquicos, alguns exemplos são o *BIRCH* [18] e *CluStream* [19]. Embora menos abordados, existem ainda algoritmos baseados em *grids* [20] e em densidade, como o *DenStream* [21]. Nesse trabalho são abordados os algoritmos *Leader-Follower Clustering* e *DenStream*, visto que esses algoritmos não necessitam que o usuário forneça o número esperado de grupos.

O algoritmo *Leader-Follower* é um dos mais simples para agrupar dados quando o número de grupos é desconhecido. O primeiro passo desse algoritmo é o cálculo da distância Euclidiana entre um novo exemplo e os centróides dos grupos já criados, de modo a encontrar o centróide com menor distância ao exemplo de entrada. Essa distância é comparada com um valor de *threshold* pré-definido. Se a distância for menor que o *threshold*, o novo exemplo é agrupado pelo respectivo centróide. Caso contrário, um novo grupo é criado tendo como centróide o exemplo.

O algoritmo *DenStream* baseia-se em duas etapas: a identificação e atualização dos chamados *microclusters*; e a formação de agrupamentos de *microclusters* utilizando o conceito de densidade. Os *microclusters* são protótipos de sumarização semelhantes ao Cluster Feature (CF) utilizado pelo algoritmo BIRCH. Os microclusters estendem o conceito de CF ao armazenar informação sobre o tempo. No *DenStream*, pontos suficientemente distintos dos *microclusters* existentes são caracterizados como *microclusters outliers*, bem como *microclusters* que não foram atualizados recentemente, e são eventualmente descartados caso não recebam novos exemplos, i.e., caso não cheguem novos dados semelhantes ao *microcluster* em questão. De maneira oposta, caso diversos exemplos possam ser sumarizados por um dado *microcluster outlier*, ele é transformado em um *microcluster* potencial, e utilizado na etapa seguinte de agrupamento. A etapa superior de agrupamento do *DenStream* utiliza como base os *microclusters* e aplica um agrupador *offline* baseado em densidade, permitindo que grupos de formato arbitrário sejam identificados.

4 Detecção de Novidades

Detecção de Novidades (DN) é um campo de pesquisa que busca identificar conceitos emergentes, ou seja, identificar novos conceitos à medida que dados adicionais são recebidos [23]. As pesquisas em DN eram inicialmente focadas na identificação de informações raras ou desconhecidas. Nesses casos, os dados são temporalmente independentes, ou seja, não há relação de causa entre exemplos distintos. A identificação de novidades nesse cenário é conhecida como DN não temporal e, em geral, é realizada por meio de abordagens estatísticas ou abordagens que utilizam métodos de agrupamento e classificação. Outras aplicações, por outro lado, consideram relações de dependência entre sequências de dados. Nesse caso, existe uma relação de causa entre diferentes exemplos. A identificação de novidades nesse cenário é conhecida como DN temporal e é normalmente realizada a partir de séries temporais. Neste trabalho será abordada a DN temporal.

Abordagens recentes para DN temporal consideram modificações no modelo de decisão após o recebimento de um novo exemplo. O trabalho de Itti e Baldi [22] aplica entropia e divergência de Kullback-Leibler, para mensurar diferenças entre modelos probabilísticos durante o tempo. Albertini e Mello [23] propõem uma técnica denominada SONDE (*Self-Organizing Novelty Detection Neural Network*). Essa técnica utiliza *Self Organizing Maps* (SOM) para modelar o comportamento dos dados. A partir do agrupamento realizado pela SOM, a técnica estima uma cadeia de Markov que representa as relações entre grupos, modelando assim o comportamento do processo. Novidades são detectadas a partir da alteração do modelo de decisão induzido, por meio da análise da entropia da cadeia gerada. Na mesma linha da SONDE, o trabalho de Pereira [24] também utiliza cadeias de Markov para estimar as relações entre estados de um processo e níveis de entropia para detecção de novidades. Ao contrário da SONDE, no entanto, um método de agrupamento baseado no algoritmo *Leader-Follower* é utilizado.

5 Abordagem Utilizada

O objetivo deste trabalho é construir um modelo representativo de um jogador a partir de sua interação com um jogo, de modo que seja possível detectar quando o comportamento do jogador se altera. A abordagem utilizada neste trabalho para detecção de novidades a partir de agrupamento de dados é baseada nos trabalhos de [23, 24].

Métricas de jogo são armazenadas durante a interação jogador-jogo. Em intervalos de tempo fixo, essas métricas são transformadas em exemplos não rotulados adequados para processamento por um algoritmo de agrupamento de dados. Cada comportamento de jogador possui um conjunto próprio de características que o descrevem, como a frequência na qual atacam ou o número de mortes por período. Essas características, por sua vez, podem ser observadas a partir dos agrupamentos obtidos ao aplicar agrupadores nos exemplos extraídos do jogo. Nesse trabalho, optou-se por fixar o intervalo de tempo em 15 segundos. Um agrupador *online* processa os exemplos a medida que estes estejam disponíveis, criando grupos que representam o comportamento do processo que, nesse caso, equivale ao comportamento do jogador. Neste trabalho foram investigados dois métodos de agrupamento *online*: *DenStream* e *Leader-Follower Clustering*. A versão do algoritmo *DenStream* utilizada é a mesma apresentada na seção 3. A versão do *Leader-Follower Clustering* é a utilizada por [24] e ligeiramente diferente da apresentada na seção 3. Nessa versão, as distâncias Euclidianas do novo exemplo aos centróides dos grupos já criados não são comparadas diretamente com o *threshold*. As distâncias do exemplo a cada centróide são utilizadas como entrada para funções de base radial (RBFs, do inglês *Radial Basis Functions*), que calculam a ativação de cada centróide ao exemplo de entrada. O maior valor de ativação é selecionado e comparado ao *threshold*. A função utilizada foi a Gaussiana, dada pela Equação 1:

$$h(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad \textbf{(1)}$$

sendo que σ é constante e representa a dispersão da função. Além disso, o algoritmo realiza adaptação de centróides, ou seja, a cada novo exemplo agrupado o centróide do grupo escolhido é atualizado segundo a influência desse novo exemplo no grupo, utilizando para isso uma média móvel.

Os grupos criados equivalem a estados representativos do comportamento de um processo. A partir desses estados é possível estimar uma cadeia de Markov que expressa as relações entre os estados do processo. As relações entre estados, expressas pelas probabilidades de se transitar de um estado para outro, representam o comportamento normal de um processo. Para obter as probabilidades de transição entre estados foi utilizada a abordagem proposta por [23]. Essas probabilidades são atualizadas de acordo com uma média móvel exponencial ponderada, com um fator de ponderação β. A Equação 3 apresenta a abordagem para atualização da cadeia.

$$P_{i,j} = (1 - \beta)P_{i,j} + \beta \quad \textbf{(2)}$$

Quando as probabilidades estimadas para a cadeia de Markov de um processo se alteram significativamente, considera-se que algo novo ocorreu no comportamento do processo. Para medir os índices de novidade no processo, utiliza-se uma medida de entropia como proposta por Shannon [25]:

$$H(M) = -\sum_{i=0}^{N}\sum_{j=0}^{N} P_{i,j}\log_2(P_{i,j}) \quad \textbf{(3)}$$

em que M é uma cadeia de Markov, $P_{i,j}$ é a probabilidade de se transitar do estado i para o estado j da cadeia e N é o número total de estados. As probabilidades da cadeia são utilizadas para medir a entropia a cada novo exemplo processado pelo sistema, de modo que uma variação da entropia acima de um limiar indica a divergência do processo de seu padrão esperado. Quando isso ocorre, considera-se que uma novidade foi detectada. Nas abordagens originais propostas por [23, 24] o nível de entropia atual da cadeia ao receber um novo exemplo é comparado com o nível de entropia anterior e, caso esteja acima de um limiar fixo, é declarada a ocorrência de uma novidade. Estabelecer esse limiar *a priori* não é uma tarefa trivial. Neste trabalho foi utilizada uma alternativa que estima o limiar dinamicamente, calculando uma média móvel das entropias, assim como um desvio padrão móvel, dados respectivamente pelas Equações 4 e 5.

$$MME = (1 - \gamma) * MME + \ \gamma * EA \quad (4)$$

$$SDE = (1 - \delta) * SDE + \delta * |EA - MME| \quad (5)$$

onde MME é a média móvel de entropias, EA é o valor da entropia atual da cadeia e SDE é o desvio padrão móvel das entropias. Assim, para declarar a ocorrência de uma novidade, verifica-se se EA > MME + (DPE * *numSD*), onde *numSD* é fixo e estabelecido empiricamente.

6 Experimentos

Para validar a metodologia para detecção de alterações de comportamento, foi implementado um sistema com dois agentes no jogo UT2004, utilizando a IDE Pogamut [26]. Uma das modalidades do UT2004, denominada de *Deathmatch*, consiste em partidas entre diversos jogadores, onde o objetivo é marcar o maior número de pontos possível em um período de tempo determinado. Jogadores ganham pontos cada vez que matam um jogador inimigo. Essa foi a modalidade adotada nestes experimentos. Com a IDE Pogamut, é possível desenvolver agentes (personagens) para o UT2004 utilizando a linguagem de programação Java. Os clientes Pogamut interagem com o servidor do UT2004 por meio do *middleware* GameBots. Esse *middleware* permite que clientes criados usando a IDE se conectem via *socket* a uma instância do UT2004.

Os agentes foram implementados por meio de uma Máquina de Estados Finitos (MEF). O primeiro agente implementado representa o jogador humano e o segundo representa o inimigo controlado pela IA do jogo. O experimento coloca os agentes para jogar uma partida no modo *Deathmatch* por um período de tempo determinado. O mapa de jogo escolhido para o experimento foi o *DMTrainingDay*, por ser o mapa mais simples disponível. O agente inimigo implementa um único comportamento do início ao fim do experimento. Esse comportamento foi denominado de *Hunter*, pois ele simula um comportamento de caçador. O agente jogador possui comportamento variável, podendo transitar entre dois comportamentos distintos, sejam eles um comportamento *Hunter*, idêntico ao do agente inimigo, e um comportamento *Prey*, bastante distinto do anterior em que o agente apenas procura itens pelo cenário e foge quando é atacado. O agente jogador altera seu comportamento em instantes de tempo pré-determinados, de modo a facilitar a análise dos resultados.

O primeiro passo foi construir o fluxo de dados que contém as informações da interação do agente jogador com o jogo. Estatísticas são armazenadas a cada novo evento que ocorre no jogo e, a cada 15 segundos, são utilizadas para definir um exemplo a ser processado pelo algoritmo de agrupamento. Nesse experimento, são utilizados vários dados estatísticos. Optou-se por separar os atributos por categorias de modo a poder, em experimentos futuros, investigar a adequabilidade de determinadas categorias de atributos para representar variações de comportamento. Devido a essa separação, foi necessário criar diferentes instâncias do algoritmo de agrupamento, ou seja, cada agrupador é responsável por quantizar exemplos segundo uma categoria de atributos. As categorias de atributos foram denominadas de *Life*, *State* e *Armory*. *Life* compreende atributos que representam a quantidade de pontos do agente. *State* contém os atributos que representam a frequência de visita a cada estado da FSM que modela o comportamento. *Armory* contém os atributos referentes às armas que o agente pode utilizar no jogo. A Tabela 1 apresenta os atributos contidos em cada categoria. As Tabelas 2, 3 e 4 apresentam os valores de parâmetros utilizados pelos algoritmos implementados.

Tabela 1 - Atributos utilizados por categoria.

Categoria	**Atributo**
Armory	Número de trocas de armas
Armory	Número de tiros disparados
Armory	Frequência de utilização de primeiro modo de tiro
Armory	Frequência de utilização de segundo modo de tiro
Armory	Número de vezes que acionou arma sem estar em movimento
Life	Número de vezes que morreu
Life	Número de inimigos mortos
State	Frequência de permanência por estado da MEF, totalizando 6 atributos.

Tabela 2 - Conjunto de parâmetros utilizados, por categoria, pelo algoritmo de agrupamento *Leader-Follower*.

Parâmetro	**Descrição**	**Valor**		
		Armory	***Life***	***State***
σ	Dispersão das RBFs	2	2	0,15
$threshold$	Limiar para aceitação de exemplos	0,1	0,2	0,1
α	Fator de média móvel para adaptação de centros	0,05	0,05	0,05

Tabela 3 - Conjunto de parâmetros utilizados, por categoria, pelo algoritmo de agrupamento *DenStream*.

Parâmetro	Descrição	Valor		
		Armory	***Life***	***State***
ε	Distância máxima para um ponto ser considerado pertencente a um *microcluster*	1,45	0,75	0,2
μ	Peso mínimo de um *cluster*	10	10	10
λ	Fator de decaimento	0,00001	0,000005	0,000003
ω	Proporção do peso mínimo (μ) para um ponto ser considerado um *microcluster* potencial.	0,11	0,105	0,11

Tabela 4 - Conjunto de parâmetros utilizados pelo método para detecção de novidades.

Parâmetro	Descrição	Valor
β	Fator de média móvel para atualização de $P_{i,j}$	0,20
γ	Fator de média móvel para entropia	0,125
δ	Fator de média móvel para o desvio padrão	0,25
numSD	Número de desvios padrão a considerar	3

7 Resultados

A seguir, são apresentados resultados obtidos utilizando a abordagem para detecção de novidade no comportamento do agente jogador. As figuras apresentam resultados para experimentos em que o agente jogador inicia o jogo com um comportamento do tipo *Hunter*, troca para o comportamento tipo *Prey* no exemplo de número 60, retornando ao comportamento inicial no exemplo de número 120. As Figuras 1, 2 e 3 apresentam os resultados obtidos utilizando o algoritmo para agrupamento de dados *Leader-Follower*, respectivamente para as categorias de atributos *Armory*, *Life e State*. As Figuras 4, 5 e 6 apresentam os resultados obtidos utilizando o algoritmo *DenStream*.

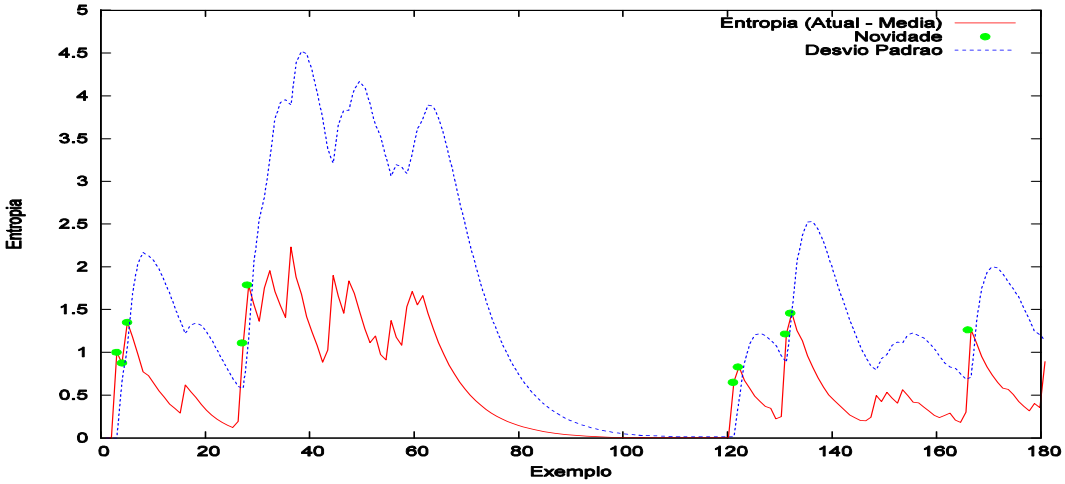

Figura 1 - Detecção de novidades utilizando o algoritmo de agrupamento Leader-Follower (*Armory*).

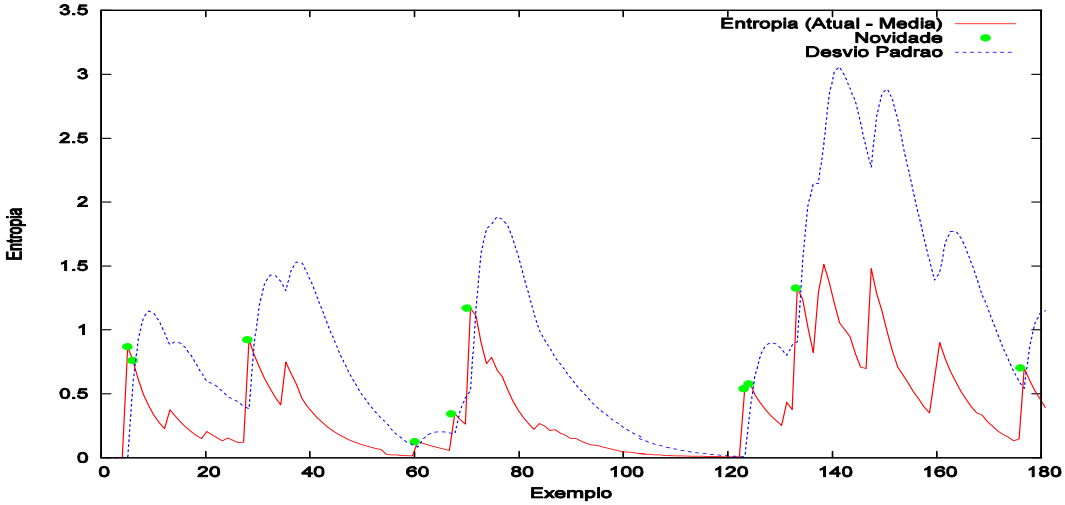

Figura 2 - Detecção de novidades utilizando o algoritmo de agrupamento Leader-Follower (*Life)*.

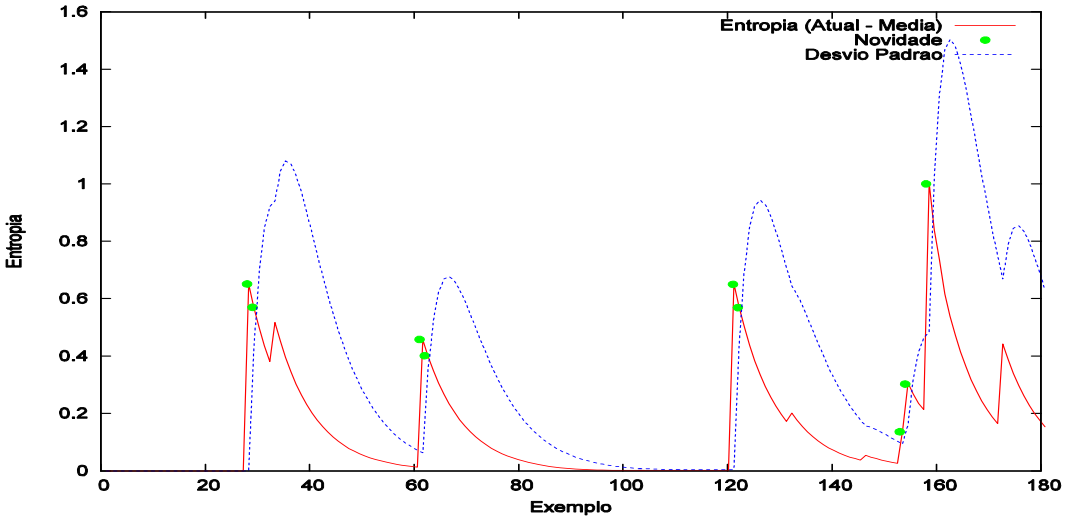

Figura 3 - Detecção de novidades utilizando o algoritmo de agrupamento Leader-Follower (*State)*.

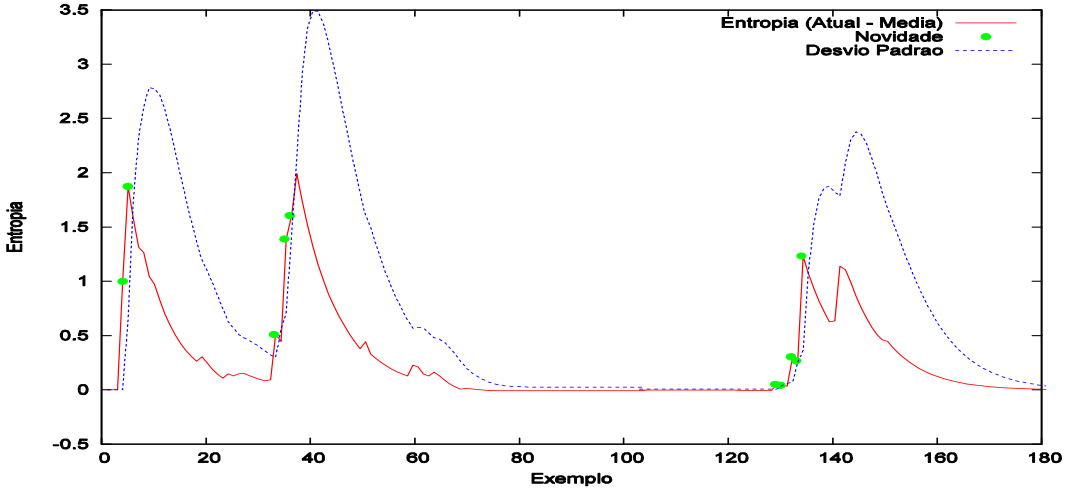

Figura 4 - Detecção de novidades utilizando o algoritmo de agrupamento *DenStream* (*Armory*).

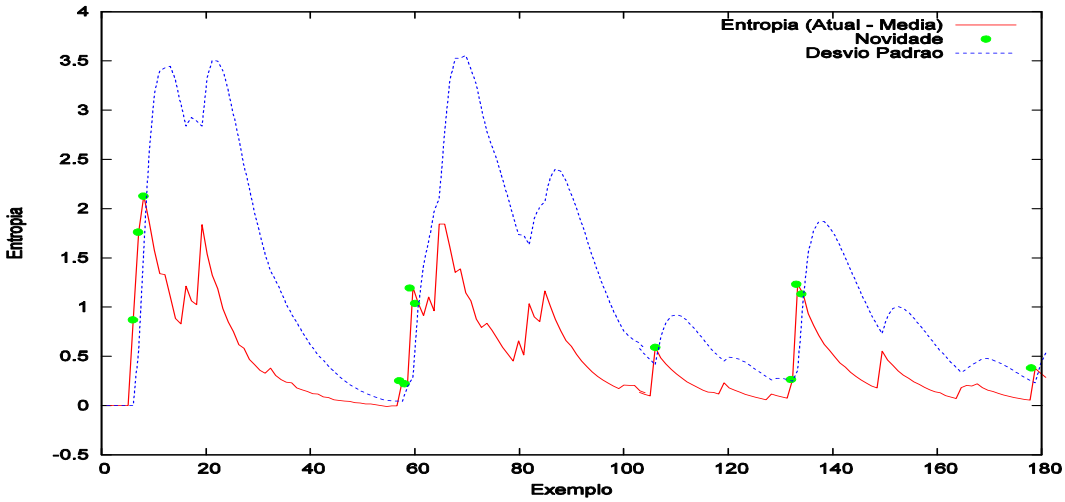

Figura 5 - Detecção de novidades utilizando o algoritmo de agrupamento *DenStream* (*Life*).

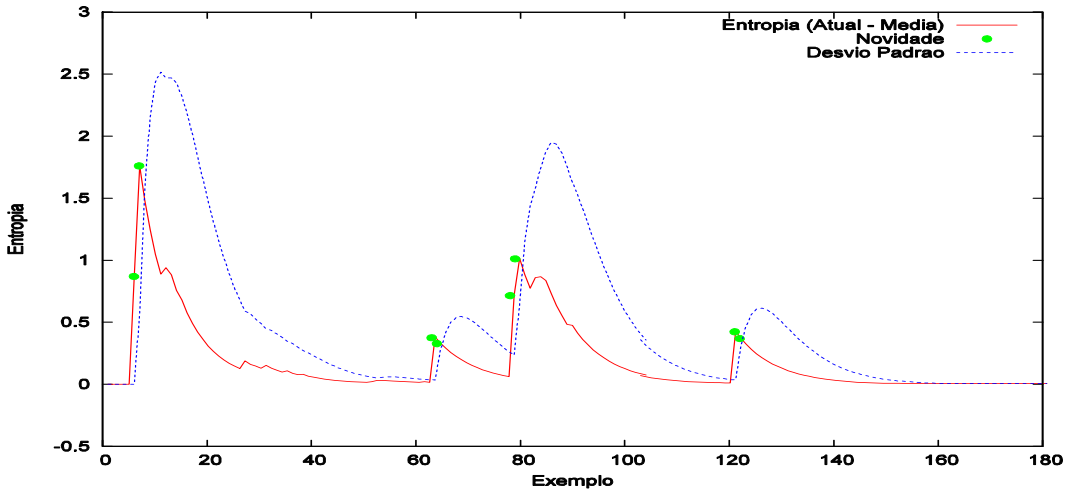

Figura 6 - Detecção de novidades utilizando o algoritmo de agrupamento *DenStream* (*State*).

Os resultados indicam comportamento bastante semelhante da abordagem de DN em relação ao método de agrupamento utilizado. De modo geral, a abordagem de DN utilizando DenStream produziu melhores resultados em termos de detecção de falsos positivos, ou seja, detectar eventos como novidade quando na verdade não são. Nesse cenário, algumas novidades são provocadas pela interação com o ambiente de jogo. Por exemplo, no intervalo de tempo de 15 segundos utilizado para criar um exemplo, pode ocorrer de o agente jogador não encontrar o agente inimigo no cenário de jogo. Uma situação como essa cria um exemplo com valores de alguns atributos bastante distintos da média, o que provoca que o detector de novidades assinale esse evento como algo novo, embora nesses casos, não tenha havido uma mudança real de comportamento por parte do jogador.

Os dois métodos possuem uma complexidade de ordem linear com o número de centros e microclusters, embora a quantidade de centros possa variar muito em diferentes aplicações, conforme a similaridade entre os dados que são fornecidos. Além disso, há uma dependência de parâmetros iniciais em ambos os métodos, sendo que o DenStream exige uma calibragem maior desses parâmetros. Isso pode ser observado na Tabela 3, onde o valor de ε é distinto para cada categoria utilizada.

Ambos os métodos são capazes de detectar as alterações reais de comportamento ocorridas, com exceção da primeira mudança de comportamento segundo a categoria de atributos *Armory*, onde ambas abordagens falharam em detectar a mudança. Isso se dá devido a detecção de um falso positivo alguns exemplos antes do exemplo de número 60, elevando o limiar para detecção de novidades. A abordagem de DN utilizando *Leader-Follower* apresentou, de modo geral, menor atraso para detecção de verdadeiros positivos.

Os resultados obtidos demonstram que a abordagem de DN utilizada necessita de uma fase inicial de calibragem, que desconsidere as primeiras novidades ocorridas. Para tanto, pode ser utilizada uma janela de tempo inicial em que todas as novidades detectadas são desconsideradas. A abordagem utilizada apresentou resultados promissores, sendo um indicativo de que a utilização de agrupamento *online* aliado a um método para detecção de novidades pode ser utilizado para modelar o comportamento de um jogador a partir de sua interação com o *software* de jogo. Além disso, é possível detectar quando esse comportamento se altera durante o tempo.

8 Conclusões

Este trabalho apresentou uma metodologia para modelagem e detecção de alterações comportamentais de um jogador durante sua interação com um jogo computacional. Para tanto, foram utilizados conceitos de detecção de novidades a partir de agrupamento de dados *online*, baseando-se nos trabalhos de [23] e [24]. A metodologia apresentada foi implementada no FPS UT2004 e os resultados obtidos demonstram seu potencial. Estudos futuros devem considerar experimentos complementares, utilizando uma maior variedade de comportamentos por parte dos agentes. Além disso, outros algoritmos para agrupamento *online* de dados deverão ser investigados em conjunto com a abordagem para detecção de novidades, de modo a considerar possíveis *bias* de cada algoritmo. A modelagem de comportamento de jogadores durante a interação com o jogo, possibilitando detectar mudanças comportamentais, permite ajustar o comportamento dos agentes inimigos de acordo com o comportamento atual do jogador, tornando a experiência de jogo personalizada e adequada ao estilo e características de cada jogador.

9 Agradecimentos

Os autores gostariam de agradecer à Fapesp e ao CNPq pelo apoio financeiro para realização desse projeto.

10 Referências

[1] S. M. Lucas, G. Kendall, Evolutionary computation and games, **IEEE Computational Intelligence Magazine**, 1 (2006), 10-18.

[2] J. Gama, P. P. Rodrigues, An overview on mining data streams, **Studies in Computational Intelligence**, **Springer Berlin/Heidelberg**, (2009), 29-45.

[3] L. Galli, D. Loiacono, P. Lanzi, Learning a context-aware weapon selection policy for Unreal Tournament III, **IEEE Proceedings of the Symposium on Computational Intelligence and Games**, (2009), 310-361.

[4] B. G. Weber, M. Mateas, A data mining approach to strategy prediction, **IEEE Proceedings of the International Conference on Computational Intelligence and Games**, (2009), 140-147.

[5] N. V. Hoorn, J. Togelius, J. Schmidhuber, Hierarchical controller learning in a first person shooter, **IEEE Proceeding of the Symposium on Computational Intelligence and Games**, (2009).

[6] J. Westra, F. Dignum, Evolutionary neural networks for non-player characters in Quake III, **IEEE Proceeding of the Symposium on Computational Intelligence and Games,** (2009), 302-309.

[7] S. Priesterjahn, O. Kramer, A, Weimer, E, Weimer, A. Goebels, Evolution of reactive rules in multi-player computer games based on imitation, **Proceeding of the International Conference on Natural Computation, Springer-Verlag,** (2005), 744-755.

[8] A. Drachen, A. Canossa, G. N. Yannakakis, Player modeling using self-organization in Tomb Raider: Underworld, **IEEE Proceedings of the International Conference on Computational Intelligence and Games**, (2009), 1-8.

[9] H. P. Martinez, K. Hullett, G. N. Yannakakis, Extending neuro-evolutionary preference learning through player modeling, **IEEE Proceedings of the Symposium on Computational Intelligence and Games**, 2010, 313-320.

[10] P. Spronck, Adaptive Game AI, Tese de doutoramento, (2005).

[11] M. Smith, S. Lee-Urban, H. Muñoz-Avila, Relaliate: learning winning policies in first person shooter games, **AAAI Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence**, (2007).

[12] A. S. Hefny, A. A. Hatem, M. M. Shalaby, A. F. Atiya, Cerberus: applying supervised and reinforcement learning techniques to capture the flag games, **AAAI Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference**, (2008), 179-184.

[13] S. Priesterjahn, Online imitation and adaptation in modern computer games, Tese de doutoramento, (2009).

[14] M. K. Crocomo, M. Miazaki, E. V. Simões, Algoritmos evolutivos para a produção de NPCs com comportamentos adaptativos, **Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital**, (2007), 44-53.

[15] P. Bradley, U. Fayyad, C. Reina, Scaling clustering algorithms to large databases, **AAAI Proceeding of the 4th International Conference on Knowledge Discovery and Data Mining**, (1998), 9-15.

[16] L. O´Callaghan, A, Meyerson, R. Motwani, N. Mishra, S. Guha, Streaming-data algorithms for high-quality clustering, **IEEE Proceeding of the Eighteenth Annual International Conference on Data Engineering**, (2002), 685-696.

[17] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, **Wiley-Interscience**, 2000.

[18] T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: an efficient data clustering method for very large databases, **ACM Proceedings of the International Conference on Management of Data**, (1996), 103-114.

[19] C. Aggarwal, J. Han, J. Wang, P.Yu, A framework for clustering evolving data streams, **Proceeding of the Twenty-Ninth International Conference on Very Large DataBases, Morgan Kauffmann**, (2003), 81-92.

[20] W. Wang, J. Yang, R. R. Muntz, STING: a statistical information grid approach to spatial data mining, **Proceeding of the Twenty-Third International Conference on Very Large Data Bases, Morgan Kauffman**, (1997), 186-195.

[21] denstream

[22] L. Itti, P. Baldi, A principled approach to detecting surprising events in video, **IEEE Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition**, (2005), 631-637.

[23] M. K. Albertini, R. F. Mello, A Self-organizing neural network to approach novelty detection, **Intelligent Systems for Automated Learning and Adaptation: Emerging Trends and Applications, IGI Global**, (2010), 49-71.

[24] C. M. M. Pereira, Detecção de faltas: uma abordagem baseada no comportamento de processos, Dissertação de Mestrado, Universidade de São Paulo, (2011).

[25] C. E. Shannon, A mathematical theory of communication, **Bell System Technical Journal**, 27 (1948), 379-423 e 623-656.

[26] J. Gemrot, R. Kadlec, M. Bida, O. Burket, R. Pibil, J. Havlicek, L. Zemcak, J. Simlovic, R. Vansa, M. Stolba, T. Plch, C. Brom, Pogamut 3 can assist developers in building AI (not only) for their videogame agents, **Agents for Games and Simulations**, (2009), 1-15.

[27] Epic Games, Digital Extremes, Unreal Tournament 2004, **Atari (Linux/Windows), Macsoft (MacOS), Midway,** (2004).