Agente Inteligente para Jogos de Estratégia Baseados em Turnos: Uma Abordagem Utilizando Planejamento Baseado em Casos

Bruno C. de Paula[1,2] Alessandro L. Koerich[1] Fabrício Enembreck[1]

[1]Pontifícia Universidade Católica do Paraná (PUCPR)
Programa de Pós-graduação em Informática (PPGIa)
R. Imaculada Conceição, 1155 80215-901
Curitiba, PR

[2]Instituto de Tecnologia do Paraná (TECPAR)
Departamento de Inteligência Artificial
R. Prof. Algacyr M. Mader, 3775 81350-010
Curitiba, PR

Resumo

Este artigo descreve a criação de uma arquitetura para construção de jogadores automáticos para a classe dos jogos de estratégia baseados em impérios. Tal arquitetura foi construída através de técnicas de raciocínio baseado em casos (RBC), mais especificamente planejamento baseado em casos (PBC).

O objetivo da arquitetura é povoar o mundo do jogo com jogadores automáticos interessantes. Tal arquitetura é composta por três camadas. A primeira camada é responsável pela recuperação dos planos mais semelhantes ao estado atual do jogador automático e na aplicação do plano no ambiente do jogo. Esta base de planos é construída a partir da captura das ações realizadas por jogadores humanos. Caso haja falha no plano, a segunda camada, denominada camada reparadora, permite a correção do plano e a anotação do possível motivo do erro. A terceira camada, por sua vez, é responsável por a partir da informação de erro preenchida pela segunda camada prevenir possíveis erros antes de ocorrerem.

O ambiente de aplicação desta arquitetura são jogos *online multiplayer* devido à necessidade de captura de planos. Os resultados obtidos no teste com o jogo Promisance demonstraram a aplicabilidade da arquitetura explicitando que os hábitos do grupo foram associados ao estilo do agente.

Palavras-chave: inteligência artificial, raciocínio baseado em casos (RBC), planejamento baseado em casos (PBC), agentes.

Contatos dos Autores:
{bruno,alekoe,fabricio}@ppgia.pucpr.br

1. Introdução

Um dos gêneros de jogo de maior sucesso entre os jogos para computador é o jogo de estratégia, principalmente, o sub-gênero dos jogos de estratégia em tempo real ou *real-time strategy games* (*RTS*). O desempenho da inteligência artificial (IA) dos *RTS*, entretanto, é pobre para os padrões humanos, não acompanhando a evolução apresentada na IA de jogos clássicos. Tal atraso se deve a uma série de motivos, segundo Buro [Buro 2003]: o mundo do jogo apresenta muitos objetos, informação imperfeita e ações tanto no nível micro quanto macro; o mercado limita os recursos destinados à CPU; não há competição entre os pesquisadores para desenvolver IAs que competem entre si devido à falta de padronização na definição da arquitetura da IA entre diferentes jogos.

Devido ao primeiro fator citado, a *IA* dos jogos de estratégia utiliza, geralmente, máquinas de estado para representar seu comportamento devido à facilidade de codificação. A aplicação de máquinas de estado, porém, implica em relativa previsibilidade para o sistema.

Técnicas mais avançadas de IA podem ser aplicadas para a construção de jogadores automáticos em jogos de estratégia, como, por exemplo, redes neurais. Persistem, entretanto, alguns problemas quanto à aplicação de técnicas de aprendizagem de máquina no mundo da IA comercial principalmente em um ambiente em tempo real. Os principais problemas são [Tozour 2002]: os sistemas de aprendizagem de máquina podem aprender as lições erradas a partir de jogadores que joguem errado; é difícil fazer o ajuste. As funções de *fitness* não devem se preocupar apenas com a competência do jogador automático, mas também com a diversão de quem está jogando; algumas técnicas são difíceis de modificar, testar e fazer a depuração, como, por exemplo, redes neurais; os personagens em alguns jogos são efêmeros e é difícil (e, talvez, desnecessário) representar uma IA que seja observável.

Assim, é interessante aplicar técnicas que sejam fáceis de modificar e fáceis de ajustar. Uma técnica que se encaixa nessas características, permitindo, principalmente, a representação explícita do conhecimento no formato de casos é a técnica de planejamento baseado em casos (PBC). Esta técnica constitui-se no reuso de planos bem sucedidos com o objetivo de resolver novos problemas de planejamento [Spalazzi 2001]. O PBC utiliza uma abordagem diferente em relação às técnicas de planejamento tradicionais. Ao invés de um problema de busca de uma seqüência de ações que transforma um estado inicial em um estado final, PBC se baseia na adaptação de casos para resolver novos problemas [Cox 2002].

Tal necessidade de modificação dos planos é justificável devido à grande quantidade de possíveis planos [Bergmann et al. 1998].

Um dos primeiros planejadores baseados em casos foi proposto por Hammond [Hammond 1989] para auxiliar na construção e execução de receitas de comida chinesa buscando em uma base de planos (receitas) um plano que resolva um problema similar à situação corrente e adaptando a solução para que encaixe nas novas circunstâncias. Se boa, a nova solução é armazenada.

Como o conhecimento do plano pode ser incompleto um plano falho pode ser gerado. Um ponto de destaque é a possível correção da falha. Para este processo, o planejador deve explicar como a falha ocorreu e usar a explicação para encontrar um conjunto de estratégias de reparo na memória, escolhendo uma para executar o reparo [Cox 2002]. Além da correção, existem também estratégias de recuperação dos planos considerando a antecipação de falhas, evitando situações que gerem falhas no futuro. Em relação aos passos característicos do PBC pode-se considerar ser uma especialização dos aplicados no CBR. Em [BERGMANN 1998] contextualiza-se este ciclo, o qual segue a maioria das abordagens para sistemas que usam PBC. Primeiramente ocorre a recuperação e organização da base de casos a partir de uma base de casos construída a partir de experiências passadas ou com o conhecimento de um especialista. Depois, são realizadas a adaptação e reutilização das soluções anteriores, propondo uma nova solução. Por fim a solução é revista e atualizada e os planos novos que emergiram da iteração são armazenados como novos casos.

Este artigo descreve a criação de um ambiente para a geração de jogadores automáticos para a classe dos jogos de estratégia baseados em impérios ou *empire-based strategy games*. O jogador automático tem como principal objetivo povoar o mundo do jogo com jogadores interessantes de se interagir, pois imitam o estilo de jogo apresentado pelo grupo. Além disso, como características relevantes, o jogador automático pode gerar planos inéditos e também se preocupa com a antecipação e correção de falhas. Para realizar suas jogadas tal jogador automático tem acesso a uma base de planos construída a partir de planos de jogadores humanos. A técnica de IA aplicada é PBC. Dentre as principais inovações deste trabalho destaca-se a verificação da viabilidade de se aplicar PBC para a geração de planos que não apenas maximizam a performance de um jogador automático, mas também imitam o estilo de jogo que está sendo aplicado pelos jogadores. Por fim, PBC herda como característica de RBC a possibilidade de representar de forma compreensível para o ser humano a explicação do porquê da escolha de uma ação podendo ajudar os jogadores novatos.

O artigo está organizado como se segue. A Seção 2 descreve alguns trabalhos relacionados a este quanto à técnica aplicada. Na Seção 3 a arquitetura dos agentes que simulam um jogador humano é apresentada. Finalmente, apresentamos alguns resultados preliminares da aplicação do agente proposto em um jogo de estratégia por turnos jogável via interface *web*.

2. Trabalhos Relacionados

Não se conhece jogo comercial que tenha aplicado alguma técnica de planejamento além de F.E.A.R [ORKIN, 2006]. Tal jogo, porém, não aplica técnica de planejamento baseado em casos. Em jogos clássicos, por sua vez, a aplicação foi bastante efetiva. Entretanto, uma série de iniciativas acadêmicas testou a aplicabilidade de técnicas de planejamento em jogos. Quanto a RBC, mais especificamente PBC, duas iniciativas destacam-se por ter influência direta no trabalho proposto.

Em uma das primeiras aplicações de RBC na IA de um jogo comercial de destaque, Fasciano [Fasciano 1996] criou um jogador automático para o jogo *SimCity*. *SimCity* é jogo de simulação de cidades em tempo real que permite que o jogador assuma o papel de um prefeito. Fasciano construiu, um planejador baseado em casos chamado *MAYOR* cujo foco era trabalhar com os seguintes problemas, típicos de um ambiente complexo e dinâmico, por exemplo: conciliar tarefas de resposta imediata e tarefas de melhorias da cidade, gerenciar interrupções, emergências e oportunidades, priorizar atividades que sejam mais importantes, e resolver os problemas sem ter um conhecimento completo do mundo. Assim como um jogador humano, o agente não possui, a priori, uma noção da influência exata de cada ação. O entendimento do mundo vai ficando mais acurado conforme aumenta a experiência do jogador.

A arquitetura de *MAYOR* consiste em um conjunto de módulos independentes que monitoram independentemente e trabalham para alcançar certas condições para o mundo. Tais módulos geram tarefas que são fornecidas a um agendador de tarefas centralizado. O agendador prioriza as tarefas e seleciona uma para execução. A utilização de planejamento baseado em casos no sistema *MAYOR* aproveita uma das características mais marcantes desta técnica que é trabalhar com a utilização de planos que se provaram bem sucedidos em domínios de entendimento incompleto. Na falha, o sistema analisa os motivos desta e como corrigir o plano para o domínio do *SimCity* baseando-se em uma rede de dependências de parâmetros.

Em outra iniciativa que merece destaque, Fagan e Cunninghan [Fagan et al. 2003] remetem aos problemas da antecipação da jogada de um jogador humano, para um sistema que prevê as próximas ações de um jogador no clássico jogo *Space Invaders*, no qual uma nave deve eliminar os alienígenas que se apresentam na tela de maneira descendente. O objetivo principal do trabalho destes autores é dar suporte a *non-player characters* (*NPCs*) que tenham um modelo do comportamento do jogador para se antecipar e se adaptar às ações deste [KERKEZ 2003].

Os trabalhos descritos anteriormente se relacionam com o proposto neste artigo no que diz respeito às

técnicas e estratégias aplicadas. Segue-se a estratégia de Fagan e Cunninghan quanto à representação dos planos como conjuntos de pares estado-ação. Entretanto, o sistema destes pesquisadores caracteriza-se por trabalhar com um tamanho fixo de plano por ser destinado ao reconhecimento de planos. Propomos neste trabalho um agente que utiliza um tamanho de plano variável e dependente das ações efetuadas em uma seção de jogo. Da arquitetura de *MAYOR* aproveita-se a divisão modular com uma camada de correção de falhas. Entretanto, ao invés de utilizar uma rede de dependências entre os parâmetros construídos por um especialista (necessária quando da correção de falhas), optou-se pela construção de tal rede com o conhecimento obtido durante o próprio jogo.

3. Arquitetura dos Agentes

Nesta seção descrevemos a arquitetura de agentes que está sendo proposta. Inicialmente, comenta-se sobre a motivação e objetivos desta arquitetura. O próximo passo é a descrição das especificidades de cada etapa do *PBC* aplicada em relação à arquitetura proposta.

O objetivo do jogador automático proposto não é apenas maximizar sua posição no ranking. Deve-se destacar que o objetivo da IA para jogos de entretenimento não é a geração de um jogador que vença sempre. A diversão pode ser maximizada, portanto, fazendo algumas falhas do *NPC* serem intencionais, mas plausíveis. Para alcançar tal credibilidade o agente imitará os comportamentos e estilos de jogo dos outros jogadores e agentes a partir de uma base de planos gerada através da captura das ações dos próprios jogadores.

Liden [LIDEN 2004] trata desta questão da "estupidez artificial" para a classe dos jogos de tiro. Para os jogos de estratégia como o Promisance, alguns dos truques citados pelo autor poderiam ser aplicáveis embora a "vida" de um império seja maior que a de um *NPC* de um jogo de tiro.

3.1 Geração da base de planos

Para a geração da base de planos, inicialmente, pode-se, por exemplo, utilizar um ambiente com apenas jogadores humanos interagindo. Todos os jogadores serão monitorados e seus planos armazenados como uma seqüência de pares estado-ação [KERKEZ 2003]. O tamanho máximo de cada plano será o correspondente a uma seqüência de jogo do jogador, ou seja, cada plano será formado por todos os pares ação-estado desde o *login* até o *logout* do jogador em uma seção. O problema desta abordagem é descobrir o tamanho ideal dos planos gerados. Fagan e Cunningan [FAGAN et al. 2003], por exemplo, utilizam sub-planos de tamanho quatro. Os planos de um jogo de estratégia, porém, são mais complexos e devem envolver mais ações. Uma janela maior, porém, implica em dificultar o *matching* de um sub-plano que seja semelhante a outro. Embora o tamanho do sub-plano possa ser considerado um parâmetro a ser considerado, testes subseqüentes foram feitos com o tamanho de sub-plano igual a uma seção completa de jogo e a aceitação de todos os sub-planos independentemente do suporte.

3.2 Processo de recuperação dos planos

Inicialmente o sistema não possui nenhum jogador automático. Em alguns dias de jogo (parâmetro dependente do tipo de jogo), conforme os jogadores forem interagindo com o ambiente, os planos vão sendo incluídos na base de planos e chega o momento de utilizar estes planos através da geração de uma população de agentes que imite os planos que foram acrescentados a base de planos até o momento.

A IA destes agentes possui três camadas distintas para decidir a ação executada pelo agente. Tal arquitetura guarda semelhança com a arquitetura de *subsumption* de Brookes [BROOKS, 1991] na qual uma camada inibe e tem precedência sobre a outra.

A camada planificadora, com maior precedência, faz uma busca em todos os estados componentes dos planos da base de planos. Para o estado mais semelhante encontrado, a ação associada a este estado é executada. O próximo passo do agente é seguir a próxima ação do plano. Todavia, antes de executar a próxima ação, o agente verifica se a ação gerou o estado previsto ou semelhante. No caso de falha, o processo de busca de um estado semelhante é executado novamente. Nova falha implica na inexistência de um estado em um plano que se adapte a situação. Neste caso, a segunda camada da IA (camada reparadora) fica responsável por decidir o comportamento do agente. Tal camada é baseada no módulo *Neighborhood Repair* proposto por Fasciano [FASCIANO, 1996] para o jogo *SimCity*. O agente verifica as diferenças entre o estado atual e o estado esperado com a execução do plano aplicando uma estratégia de reparo para as diferenças significativas e armazenando a estratégia de correção relacionada ao plano se o reparo for bem sucedido.

3.3 Prevenção de falhas e realimentação da base de casos

O algoritmo de prevenção de falhas introduz a preocupação em antecipar as falhas nos planos antes de sua ocorrência. Conforme comentado na seção anterior, caso uma rotina de correção seja bem sucedida, e um problema tenha sido detectado, ela pode ser chamada antes da execução do plano para garantir o seu sucesso.

Deve-se esclarecer o significado e método de detecção de um problema. Um problema, conforme explanado na seção anterior, é detectado pela camada reparadora através dos valores das diferenças entre as variáveis de estado entre o estado desejado e o estado atual. Cada diferença será candidata a ser combatida por uma estratégia de reparo.

As estratégias de reparo bem sucedidas provocam a associação da condição para que a estratégia seja chamada ao plano cujo reparo foi necessário. Com

menos força de associação, também, propõe-se que a condição e suas estratégias de reparo também sejam associadas às ações. Tais estratégias de reparo são inferidas a partir de uma já citada rede de dependências construída através de hipóteses geradas pela diferença entre o estado esperado e o estado obtido após a execução de uma ação em um plano que não foi bem sucedido.

Destaca-se que as ações realizadas para o reparo do plano e o plano associado, se bem sucedido, também podem ser acrescentados à base de planos. Dessa forma, planos novos não são gerados apenas pela intervenção humana, mas também pela aprendizagem dos jogadores automáticos. A arquitetura descrita foi testada com um jogo *on-line* descrito na próxima seção.

4. Resultados Obtidos

Esta seção apresenta alguns resultados obtidos através da aplicação da arquitetura descrita em um jogo *online multiplayer* real com um grupo de teste. Descreve-se, inicialmente, o jogo utilizado e suas características. Por fim são apresentados alguns dos resultados obtidos.

4.1 O jogo

Promisance [PROMISANCE] é um jogo *multiplayer* de estratégia em turnos que permite o gerenciamento de um império persistente. Sua interface gráfica é essencialmente textual e baseado em web.

O objetivo principal do jogo é fazer com que o valor do império cresça. Esse valor, denominado *networth* é calculado a partir do tamanho do império e de quanto dinheiro o império possui. Quanto ao gerenciamento do império, diversas ações estão disponíveis as quais custam turnos. Tais ações estão relacionadas, principalmente, à construção da infra-estrutura da civilização, à exploração e prospecção de novas terras e a geração de tropas.

Destaca-se que para o desenvolvimento do agente o jogo foi simplificado no sentido de retirar as interações entre os jogadores.

4.2 Teste Realizado

No teste realizado, com um grupo limitado a 7 jogadores humanos, focalizou-se o processo de captura dos planos. Foram obtidos 5.133 pares estado-ação em um período de jogo de três semanas. A indexação do estado do estado, objetivando a recuperação do plano foi feita utilizando-se as seguintes características: dinheiro, *networth* (valor do império), quantidade de terra, magia e comida. Com estas características pode-se extrair 2.357 estados diferentes os quais os jogadores passaram. Quanto às ações associadas ao estado, foram capturadas dezesseis ações diferentes dentre ações de gerenciamento e tipos de ataque. A tabela abaixo sumariza as ações que mais foram executadas pelo grupo, refletindo a necessidade de interação entre os jogadores.

Tabela 1: Ações mais executadas pelo grupo

# de execução	Ação
938	Ataque padrão a outro império
793	Construção
729	Geração de dinheiro
474	Exploração do terreno
247	Agricultura

Quanto aos estados mais observáveis (Tabela 2), surpreende o domínio de um estado que se caracteriza por ter valor 0 na quantidade de dinheiro. Tal estado ocorreu para mais de um jogador humano ou agente e aparentou inicialmente ser uma falha do jogo. Posteriormente, verificou-se que este estado (e outros cujo valor de dinheiro chega a 0) seria gerado pela inexperiência dos jogadores que gastavam demais seus recursos sem se preocupar com a geração. Por fim, também foi possível a utilização destes recursos como estratégia de jogo, ou seja, houve casos em que deliberadamente o jogador não tentou eliminar a falta de dinheiro.

Tabela 2: Estados mais observados e características do estado

#	Dinheiro	Networth	Terra	Mana	Comida
55	0	12 milhões	9 mil	1,6 milhão	91 milhões
54	960 milh.	19 milh.	11 mil	7 milh.	130 milh.
38	270 mil	200 mil	330	500	47 mil
33	100 mil	150 mil	250	500	10 mil
31	580 milh	20 milh	1,2 mil	7,7 milh	140 milh

Quanto à seleção do plano a ser escolhido, no teste realizado com o Promisance optou-se pela escolha do plano que maximizasse o *networth*. Em relação à correção de falhas, apenas armazenou-se a ocorrência das falhas sendo necessária a utilização posterior desta informação para corrigir a falha. Deve-se destacar, também, a dependência entre os parâmetros, ou seja, o aumento de um parâmetro pode implicar na diminuição de outro.

5. Conclusões

Conclui-se que a arquitetura proposta é útil para povoar jogos *online multiplayer* baseados em impérios. Além disso, também poderia ser utilizada como auxílio aos jogadores novatos. Destaca-se, neste ponto, o uso da técnica de PBC. Tal técnica permitiu que se percebesse o porquê explícito da escolha pelo agente de uma certa ação.

Outro ponto relevante a se considerar é a necessidade de validar, quando um round de jogo for finalizado, a credibilidade dos agentes. O último problema a ser contextualizado é relacionado à avaliação da criação da IA e sua efetividade. As técnicas de IA para jogos de computador caracterizam-se por objetivar a maximização da diversão. Como medir o quanto um jogador está se divertindo? Poder-se-ia convencionar que o jogador se diverte quando se sente desafiado e se estiver jogando com um adversário humano a sua altura. Assim, para saber se um jogador está se divertindo com um jogador automático seria suficiente a aplicação do Teste de Turing e conseqüente verificação se o jogador crê que seu adversário seja humano.

Referências

BERGMANN, RALPH; AVILA, HÉCTOR MUÑOZ; VELOSO, MANUELA. 1998. Case-Based Reasoning applied to Planning Tasks in Case-Based Reasoning Technology from Foundations to Applications.

BROOKS, R.A., 1991. *"How to build complete creatures rather than isolated cognitive simulators,"* in K. VanLehn (ed.), Architectures for Intelligence, pp. 225-239, Lawrence Erlbaum Assosiates, Hillsdale, NJ.

BURO, M. 2003. *RTS Games as a Test-Bed for Real-Time AI Research, Proceedings of the 7th Joint Conference on Information Science (eds. K. Chen et al.)*, p. 481–484.

COX, MICHAEL T.; MUNOZ-AVILA, HECTOR; BERGMANN, RALPH. 2002. *Planning in Case-Based Reasoning: Commentary*.

FASCIANO, MARK J. 1996. *Real Time Case Based Reasoning in a Complex World.* Technical Report TR-. 96-05, Computer Science Department, University of Chicago

FAGAN , MICHAEL; CUNNINGHAM, PÁDRAIG. 2003, *Case-Based Plan Recognition in Computer Games.* ICCBR, 2003. Disponível em: http://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-01.pdf

HAMMOND, K., 1989. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press Inc.

KERKEZ, BORIS. 2003. *Incremental Case-Based Plan Recognition With Incomplete Plan Libraries*. disponível em: http://personal.ashland.edu/~bkerkez/Research/public/BKerkez_PhD_Dissertation_WSU_2003.pdf

LIDEN, LARS. 2004. *Artificial Stupidity: The Art of Intentional Mistakes in AI Game Programming Wisdom II*. Charles River Media. Rockland. 2004.

PROMISANCE. Página do Jogo Promisance. http://sourceforge.net/projects/promisance

ORKIN, J. (2006), 3 States & Plan: The AI of F.E.A.R., Game Developers Conference Proceedings

SPALAZZI, LUCA.A 2001. *Survey on Case-Based Planning*. Artificial Intelligence Review, Vol. 16, Num. 1. Disponível em: http://www.diiga.univpm.it/~spalazzi/reports/airev-2001.ps

TOZOUR, PAUL. 2002. *The Evolution of Game AI. AI Programming Wisdom*, Charles River Media, p. 3-15.