Jogador para Starcraft baseado em Redes de Comportamentos

Anderson Pontes Vieira - 023108

***Resumo*—Este trabalho propõe o uso de uma implementação de Redes de Comportamentos para controlar um jogador de *Starcraft*. A interface do jogador com o jogo é feita através de uma API que o controlador tenha acesso às mesmas informações e ações que um jogador humano. Detalhamos os comportamentos implementados e discutimos os resultados e possíveis trabalhos futuros.**

***Index Terms*—estratégia, jogos, redes de comportamentos, starcraft**

I. Introduction

EXISTEM duas motivações principais para se trabalhar com jogos de computador no desenvolvimento de algoritmos de tomada de decisão. O primeiro é que os mecanismos de seleção de ação dos jogos atuais levam, muitas vezes, a comportamentos repetitivos, o que provoca a perda de interesse do jogador uma vez que ele entenda a estratégia da máquina. O segundo é que os jogos podem prover um ambiente dinâmico e de complexidade limitada, ideal para testar propostas da área de sistemas inteligentes.

Um jogador automático deve ser capaz de realizar uma sequência de ações de forma a atingir um objetivo. Deve também identificar certas condições no ambiente e agir de forma oportunista. Como balancear oportunismo e persistência na busca de um objetivo, agindo rapidamente em um ambiente ruidoso e dinâmico? Para tratar esta questão a pesquisadora do MIT, Pattie Maes, propõe a idéia de Redes de Comportamentos [1]. Neste trabalho, procuramos usar esta idéia para desenvolver um jogador para o jogo de estratégia *Starcraft*, em que um jogador precisa criar uma colônia, explorar recursos naturais e construir um exército para disputar o território com outro jogador [2].

A. Redes de Comportamentos

Uma Rede de Comportamentos é formada por diversos módulos de competência interligados. As conexões entre os módulos permitem que eles influenciem uns aos outros, e compitam pela possibilidade de serem ativados. Um módulo x é composto por:

- Lista de pré-condições (c_x): proposições que devem ser verdadeiras antes da execução do comportamento.
- Lista de adições (a_x): proposições que devem ser verdadeiras após a execução do comportamento.
- Lista de deleções (d_x): proposições que devem ser falsas após a execução do comportamento.
- Ação: ação ou ações executadas pelo módulo.

Existem três tipos de ligação entre os módulos definidas em função das listas que os compõem:

- Sucessor: uma ligação do módulo x ao módulo y para cada proposição $p \in a_x \cap c_y$.
- Predecessor: uma ligação do módulo x ao módulo y para cada proposição $p \in c_x \cap a_y$.
- Conflitante: uma ligação do módulo x ao módulo y para cada proposição $p \in c_x \cap d_y$.

Estas ligações são usadas para espalhar ativação pela rede. Os módulos se estimulam através das ligações do tipo sucessor e predecessor e se inibem através das ligações conflitantes. Três fontes são responsáveis por ceder e retirar ativação da rede:

- Estado: cada proposição do estado atual envia ativação para os módulos que os têm em suas listas de pré-condições.
- Objetivos: cada objetivo envia ativação para os módulos que os têm em suas listas de adição.
- Objetivos protegidos: cada objetivo protegido retira ativação dos módulos que os têm em suas listas de deleção.

Um módulo é dito executável quando todas as proposições de sua lista de pré- condições são satisfeitas. Para que um módulo seja ativado, ou seja, realize sua ação, três critérios devem ser satisfeitos:

1) o módulo deve ser executável,
2) o nível de ativação do módulo deve ser maior que um limite θ,
3) o módulo deve ter a maior ativação dentre todos os que satisfazem 1 e 2.

Quando um módulo é ativado, sua ação é executada, seu nível de ativação é reiniciado e θ volta ao valor original. Caso nenhum módulo se torne ativo na iteração atual, o valor de θ é decrementado e uma nova iteração é iniciada.

II. Metodologia

Para examinar a atuação da rede durante uma partida, quatro comportamentos foram implementados:

- BUILD_BARRACKS: constrói um quartel
  - pré-condições: worker
  - adições: barracks
  - deleções:
  - ação: build_barrack
- BUILD_SUPPLY: constrói um depósito de suprimentos
  - pré-condições: worker
  - adições: not_population_limit
  - deleções:
  - ação: build_supply
- BUILD_MARINE: treina um soldado
  - pré-condições: barracks, not_population_limit
  - adições: fire_power

  - deleções:
  - ação: train_marine
- DEFEND: ataca os inimigos que aparecem na área de visão
  - pré-condições: fire_power, under_attack
  - adições: defense
  - deleções:
  - ação: defend

Os comportamentos interagem da maneira descrita na Seção I-A. O jogador possui dois objetivos: *fire_power* e *defense*.

Uma API foi usada para realizar a conexão do controle com o jogo. A *Brood War Application Programming Interface* (*BWAPI*) é um *framework* em C++ que possibilita a criação de módulos de inteligência artificial para o Starcraft [3]. O *wrapper* pybw provê interfaces na linguagem Python para todas as estruturas da BWAPI [4]. Este *wrapper* foi necessário pois a rede de comportamentos foi implementada em Python. Além disso, foi usada a extensão pybwsal, que provê uma cama a mais de abstração para tarefas como construir estruturas, coletar recursos, etc.

III. Resultados

Devido a quantidade limitada de comportamentos, a estratégia de jogo resultante foi extremamente simples. Os trabalhadores ficam recolhendo recursos permanentemente até que, por algum motivo, é dada a ordem para realizar uma construção. Digamos, por exemplo, que o comportamento *BUILD_MARINE* tenha uma alta ativação, mas não possa ser executado porque chegou-se ao limite da população. Como uma das pré-condições para este comportamento é o estado *not_population_limit*, uma parte da sua ativação é passada para o comportamento *BUILD_SUPPLY*, que possui *not_population_limit* na sua lista de adições. Um exemplo deste caso pode ser visto na Figura 1, onde as mensagens na tela indicam quais comportamentos foram ativados.

A Figura 2 traz o gráfico dos níveis de ativação de cada comportamento em relação ao limiar (*threshold*). Os momentos em que a ativação de um comportamento encontra o limiar e cai subtamente indicam que aquele comportamento foi executado. Podemos perceber que em alguns casos, como no do comportamento *DEFEND,* o nível de ativação chega a ser maior que o limiar, mas como nem todas as pré-condições são atendidas, o comportamento não é disparado.

IV. Conclusão

Apesar de simples, o jogador implementado foi capaz de construir um exército numeroso de *marines* e se defender de ataques durante algum tempo. A rede de comportamentos pareceu adequada para realizar o tipo de tomada de decisão que é necessário durante o jogo. Um aprimoramento do trabalho atual envolveria uma modelagem mais completa, adicionando-se novos comportamentos e definindo objetivos mais claros. Uma questão que deve ser investigada, neste caso, é se um número muito maior de comportamentos implicaria na lentidão da rede, prejudicando o desempenho do jogador.

Duas outras questões que poderiam ser abordadas são o aprendizado e a automatização. Ao longo de várias partidas, o agente poderia criar novas ligações entre comportamentos ou até mesmo novos comportamentos de acordo com a experiência. A automatização tornaria mais eficiente a execução de sequências de tarefas que se repetem muito, deixando o processamento livre para outras atividades.

Mais uma limitação em potencial seria o fato de se usar uma camada de abstração sobre as tarefas simples. Seria necessário investigar o quanto a forma como as ações realizadas por esta camada influencia no desempenho da rede.

Referências

[1] Maes, P., *How To Do the Right Thing*. Connection Science Journal, 1989, Vol. 1, pp 291-323.

[2] Blizzard Enternainment, “Starcraft,” [Online]. Disponível: http://us.blizzard.com/en-us/games/sc/. [Accessed: 05/07/2010].

[3] "BWAPI - An API for interacting with Starcraft: Broodwar (1.16.1)". [Online]. Disponível: http://code.google.com/p/bwapi/. [Accessedo: 04/07/2010].

[4] "pybw - Python wrapping for BWAPI". [Online]. Disponível: http://code.google.com/p/pybw/. [Accessedo: 04/07/2010].

Figura 1. *Screenshot* do momento em que o comportamento *BUILD_MARINE* é executado logo após *BUILD_SUPPLY*.

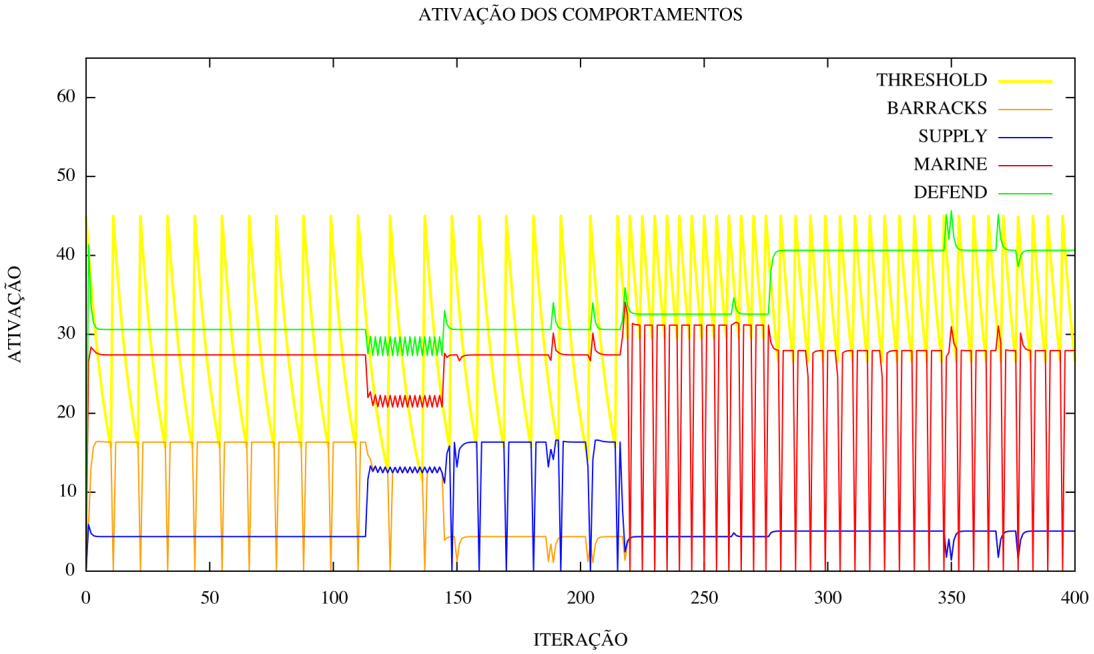

Figura 2. Nível de ativação dos comportamentos em relação ao limiar ao longo do tempo.