Sistemas Multiagentes em Jogos

Sistemas Multiagentes

Prof. Wagner Tanaka Botelho

Gabriel Nobrega de Lima
Yuri David Santos

Jogos de Estratégia em Tempo Real

Os games de estratégia em Tempo Real ou *Real time Strategy Games* (RTS) são jogos em que o progresso ocorre de maneira continua, sem a necessidade de turnos. Estes jogos em sua maioria baseiam-se na construção e desenvolvimento de civilizações onde objetivo é explorar um mapa, coletar recursos e eliminar adversários. Este tipo de gênero de jogo surgiu em 1981 com o jogo Utopia mostrado na Figura 1, onde o objetivo era a administração continua de uma ilha em plena ascensão populacional.

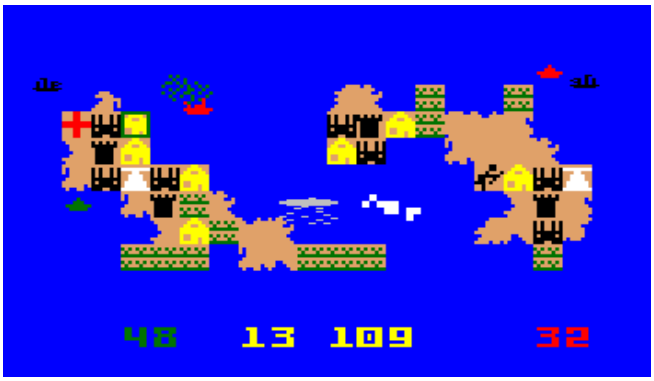

Figura 1 - Jogo Utopia de 1981, considerado o primeiro RTS.

O jogo Dune 2 mostrado na Figura 2 estabeleceu definitivamente o gênero de jogos de estratégia em Tempo Real introduzindo a maior parte das características vistas em jogos da atualidade. Sua interface permitia que o usuário interagisse através do mouse e pudesse explorar mapas, coletar recursos e eliminar adversários.

Figura 2 – Dune 2 de 1992, o primeiro RTS que agregava a maior parte das características presentes nos jogos atuais.

Com o crescimento dessa indústria e orçamentos cada vez mais altos, pesados investimentos começaram a ser feitos para melhorar a IA por trás do controle dos adversários. É muito difícil conceber uma IA para um jogo desse tipo que seja capaz de desafiar os melhores jogadores. A maioria dos sistemas de IA de níveis mais difíceis utilizam de trapaça (cheating), o que pode ser muito decepcionante para os jogadores quando descoberto. Exemplos de cheating em jogos RTS são: IA começa com mais recursos, ou tem conhecimento de toda informação do mundo do jogo, ou pode construir e realizar ações básicas mais rapidamente. O ideal é construir uma IA capaz de competir com jogadores habilidosos em igualdade de condições.

A maioria dos sistemas de IA para esses jogos são feitos com A* para navegação e outras técnicas de IA não distribuída, por razões de simplicidade e melhor performance. No entanto, explorar novos paradigmas se faz necessário frente a essa situação de inferioridade (em relação aos melhores jogadores) em que a IA tradicional se encontra.

Em 2008, no artigo “Using Multi-Agent Potential Fields in Real-Time Strategy Games” (Hagelbäck), é feita uma proposta de utilização de Sistemas Multi-Agentes em jogos RTS. Nesse caso, o uso da IA distribuída seria feito para solucionar os problemas de navegação, estratégias e táticas de ataque e defesa. O artigo foi elaborado a partir da experiência do autor num campeonato de IA aplicada a jogos RTS, no qual ele utilizou a técnica de Campos Potenciais (Potential Fields), obtendo bons resultados.

Navegação com Potencial Fields

Em 1985 Ossama Khatib introduziu o conceito de Potencial Fields para que robôs móveis pudessem percorrer ambientes evitando choques contra obstáculos. Segundo sua idéia, o robô deveria se mover no ambiente considerando um campo de forças, que imprime uma força resultante em sua posição o levando em direção ao destino final. Para isso todo ambiente em que robô irá percorrer deve ser mapeado por um campo de forças como mostrado na Figura 3. As forças atrativas seriam mais predominantes na direção e sentido onde o robô deve seguir para encontrar uma posição final desejada e as forças repulsivas permitem que o robô não esbarre nos obstáculos presentes no cenário.

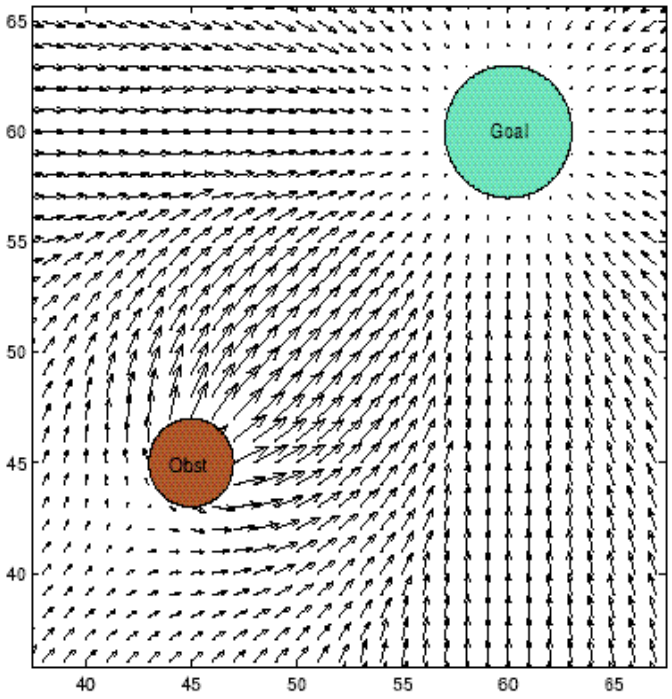

Figura 3 - Mapeamento do campos de força e um ambiente.

O circulo marrom posicionado na parte inferior esquerda da Figura 3 é um obstáculo, sendo assim, deve empregar forças repulsivas representadas pelos vetores apontando para fora. O circulo verde representa uma posição objetivo que o robô deve chegar, oferecendo predominantemente forças atrativas. Quando este tipo de mapeamento é realizado, e um robô é colocado no ambiente sua navegação se torna trivial, já que a direção que o mesmo deve sempre seguir para encontrar seu objetivo é aquela em que o vetor força possui maior módulo. A mesma idéia para a navegação de agentes em jogos RTS foi utilizada nos trabalhos de [Hagelbäck et al.], neste caso os robôs passam a ser personagens do jogo e uma modelagem semelhante é realizada para que não haja uma colisão com obstáculos no cenário do jogo e que se encontre uma rota para uma posição final. A Figura 4 mostra um

agente simbolizado pelo circulo verde, uma posição objetivo determinada pelo quadrado azul e os obstáculos desenhados com cor marrom.

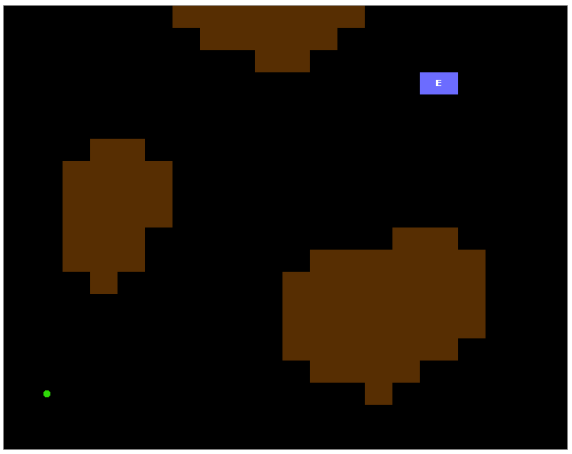

Figura 4 – Cenário de navegação para o agente.

Para que o agente consiga se locomover até o objetivo precisa-se gerar o Campo de Potencial de todo o cenário. O primeiro passo é sempre considerar a posição objetivo como uma fonte atrativa, gerando valores de força cada vez maiores conforme nos aproximamos de sua região. Tal mapeamento pode ser visualizado na Figura 5, as cores mais claras são aquelas que oferecem maior atração e as escuras maiores repulsão.

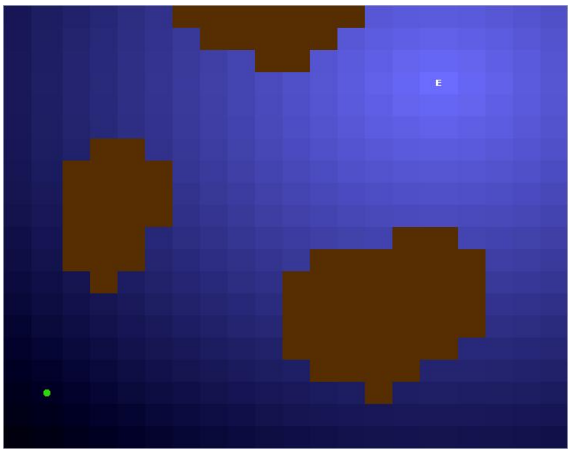

Figura 5 – Campo de Potencial gerado para o objetivo E.

A navegação do agente não pode ser ainda realizada visto que o Campo de Potencial ainda não considerou os obstáculos no cenário, representadas pelas regiões de cor marrom. Para que o agente não esbarre em nenhuma destas áreas deve-se empregar um potencial de repulsão que emana de cada uma destas áreas, como pode ser visto na Figura 6.

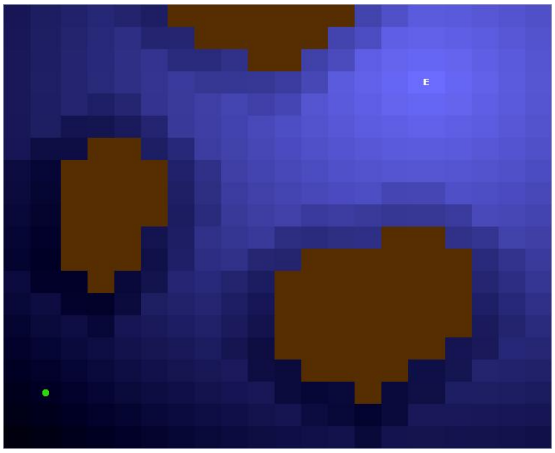

Figura 6 – Campo de Potencial dos obstáculos.

A Figura 6 mostra o Campo de Potencial total gerado para o cenário. O agente agora pode navegar movendo-se sucessivamente para as regiões que apresentam maiores potenciais, ou seja, aquelas com cores mais claras, o que permitirá encontrar o objetivo simbolizado pelo quadrado de cor azul “E”. Caso outras unidades sejam inseridas no cenário, como mostrado na Figura 7 com a intrdução de outros dois agentes simbolizados por círculos brancos deve-se levar em conta um potencial de repulsão nestas regiões.

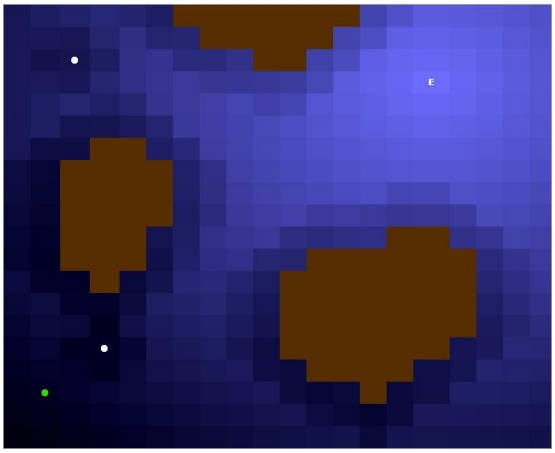

Figura 7 – Adição de dois agentes e seus respectivos Campos de Potencial.

Na Figura 7 pode-se perceber que a inserção dos agentes de cor branca tornou as regiões próximas mais escuras e, por conseguinte de caráter mais repulsivo ao agente de navegação verde, permitindo que o este possa caminhar evitando os novos obstáculos, como mostra a trajetória final na Figura 8.

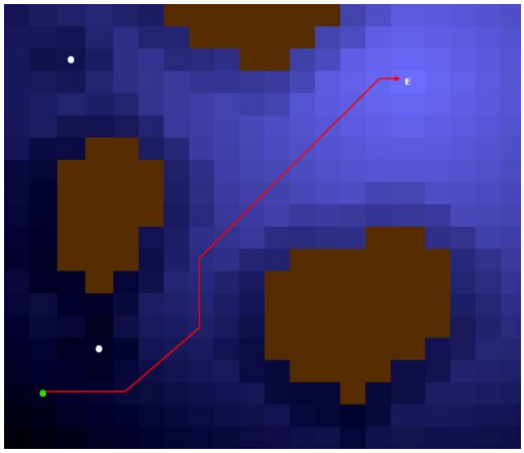

Figura 8 – Percurso final obtido com a navegação sobre o Campo de Potencial.

A trajetória encontrada na Figura 8 foi gerada de maneira automática mesmo após a adição de unidades inicialmente não presentes no cenário. Tal característica mostra que os Campos de Potencial permitem que percursos sejam traçados em ambientes dinâmicos sem qualquer custo adicional. As técnicas de navegação clássicas como A* e Dijkstra exigiriam re-calculo da rota toda a vez que um objeto se movesse o que geralmente introduz alguns problemas nestas implementações.

Os comportamentos que os agentes em jogos assumem podem ser modelados totalmente através do tipo de Campos de Potencial gerados pelos obstáculos ou quaisquer elementos do jogo. Por exemplo, para que um agente mantenha sempre uma distância determinada de um destino final deve-se gerar um Campo de Potencial progressivamente maior a partir do ponto objetivo até que tal distância seja atingida, como mostrado na Figura 9.

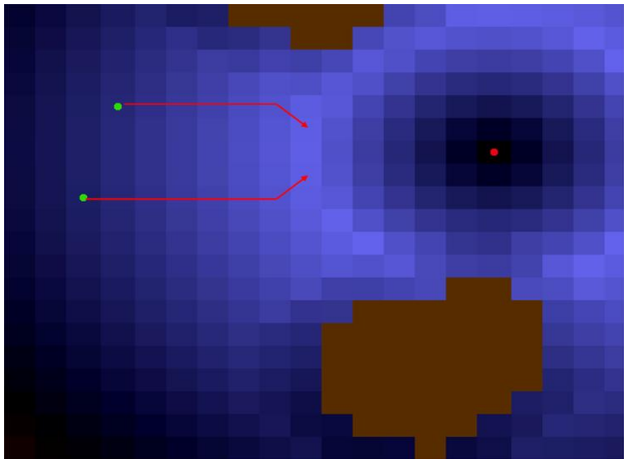

Figura 9 – Comportamento de tiro a distância implementado através da especificação do Campo de Potencial do agente adversário vermelho.

Na Figura 9 os agentes verdes que estão em direção ao ataque de um agente vermelho manterão uma distância adequada do inimigo, tal comportamento é conseguindo simplesmente variando a forma como o Campo de Potencial é criado pelo inimigo.

A utilização dos Campos de Potencial permite a navegação automática de agentes em ambientes dinâmicos de maneira automática, contudo, sua implementação não é simples e o processo para a geração do Campo de potencial pode ser computacionalmente custosa. Tais características fazem com este tipo de navegação possua poucas aplicações em jogos comerciais, onde o desempenho de processamento e memória são requisitos muito críticos.

Arquitetura Multiagentes para um jogo de Tanques

No jogo de tanques, onde o objetivo é eliminar os tanques inimigos e suas bases, ao mesmo tempo protegendo as próprias bases, Hagelbäck et al utiliza uma abordagem de Potential Fields para a navegação, controlando ao mesmo tempo outras funções.
Em alto nível, a estratégia se dividia em duas partes: atacar apenas os tanques e atacar tanques e bases. A primeira é a estratégia padrão. A segunda começa a ser executada caso o inimigo tenha metade do número de tanques ou menos, no máximo 6 tanques, e apenas uma base restante. No caso geral, as bases e os inimigos geram atração no campo, porém quando a estratégia inicial está sendo usada, as bases geram campos que repelem as unidades, pois não devem ainda ser atacadas, mas atuam como obstáculos.
Em relação as táticas de ataque, um agente coordenador decide qual unidade deve atacar qual inimigo ou base inimiga. O objetivo disso é concentrar fogo para eliminar unidades que estejam mais danificadas (diminuindo o poder de fogo do inimigo).
Essa coordenação do ataque é feita inicialmente através de uma matriz, construída a partir dos dados que os agentes passam para o agente coordenador (sua posição, se está pronto para atacar, se tem inimigos ao alcance, etc.). Com isso o coordenador constrói uma matriz A, onde as linhas representam os inimigos e as colunas, as unidades. Cada elemento ixk da matriz indica que o inimigo i pode ser atacado pela unidade k (isto é, está dentro do raio de ataque) quando tem valor 1, ou 0 se não for possível. Outra matriz H com os pontos de vida ou hit points (HP) dos inimigos é construída e ordenada, ditando também a ordem da matriz A. O coordenador então verifica quais inimigos podem ser eliminados nesse turno, concentrando os ataques nesses inimigos, e zerando os outros valores das colunas das unidades que estão participando do ataque. Em seguida, para assegurar que cada unidade só irá atacar um inimigo, o coordenador zera todos os valores de cada coluna exceto o primeiro 1. Dessa forma é definida a organização dos ataques.
Para verificar quais agentes devem ser inseridos na matriz A antes da realização desse processo, para evitar calcular a distância euclidiana entre cada unidade e todos os inimigos, e calcular também se os inimigos estão na linha de fogo (isto é, sem obstáculos à frente), o coordenador faz uma checagem em 3 níveis: primeiro calcula a distância Manhattan, soma da distância em X e em Y (cálculo simples e rápido), entre eles; se for menor que um certo limiar, calcula a distância euclidiana (cálculo mais demorado) para

verificar se estão no raio de fogo; somente caso esteja, calcula se o inimigo está na linha de fogo (cálculo muito demorado).

A Multi-Agent Architecture for Game Playing

Os sistemas inteligentes em jogos geralmente são criados para apresentar um comportamento de maneira a solucionar problemas específicos. Em um jogo de corrida os agentes que controlam os carros sabem exatamente as regras da corrida, devem evitar colisões e efetuar ultrapassagens sempre que possível com o objetivo de finalizar a prova em primeiro lugar. Quando um jogo de outro domínio, por exemplo, guerra é tratado um sistema inteligente deve ser totalmente remodelado para que se adéqüe as novas restrições e todo o trabalho realizado para a criação de um sistema inteligente para o jogo de corrida teria que ser re-escrito. Isto seria a tarefa normalmente empregada utilizando os modelos clássicos não distribuídos de Inteligência Artificial, o campo de pesquisa em *General Game Playing* (GCP) estuda a implementação de sistemas jogam quaisquer tipos de jogos sem uma definição pré-concebida das regras em sua implementação. O sistema permite que descrições posteriores das regras sejam disponibilizadas e a arquitetura tenha o poder de aprender sozinha como jogar de maneira eficiente. Como uma GCP é uma área ainda de estudos em andamento, por ser ainda muita ampla, o trabalho de [Kobti et al.] trata somente da classe de jogos de tabuleiro, definindo estes como “Positional Games”. Em seu trabalho é proposto um sistema que permite a criação IA eficientes para jogos de tabuleiro com a descrição posterior das regras do jogo.

O sistema GCP padrão geralmente conta com dois agentes, o Game Manager e o Game Player, como mostra a Figura 10.

Figura 10 - Comunicação entre os agentes Game Manager (GM) e Game Player (GP).

Game Manager (GM): Responsável por enviar inicialmente ao GP as regras do jogo e subseqüentemente os movimentos realizados pelo adversário.

Game Player(GP): Recebe mensagens do GM e envia os movimentos apropriadaos.

**A Universidade de Standford mantém em seu site de GGP um GM que aceita conexões de GP's para jogar jogos. A Linguagem de descrição de regras de jogos utilizaada é a Knowledge Interchange Format (KIF).*

Na Figura 11, pode ser visto todos os agentes e grupos estipulados para a implementação do sistema.

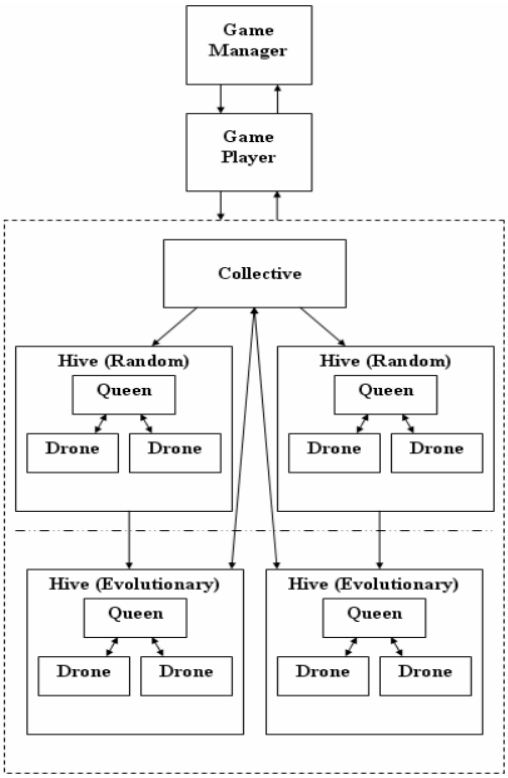

Figura 11 – Arquitetura do Multi-Agent Architecture for Game Playing.

Agente Drone: Assume uma único tipo de movimento permitido na descrição das regras do jogo.

Agente Queen: Possui uma série de agentes Drone. Sua responsabilidade é aceitar ou não o movimento realizando por um Drone checando sua validade e atualizando o estado de jogo para seus Drones.

Hive: Um grupo de um Queens e Drones proporcionam a seqüência de passos seguintes no jogo. Efetivamente jogam o jogo. Existem dois tipos de *Hives*, o Random, onde Cada drone faz jogadas

aleatórias mas que respeitam as regras, gerando as seqüência de treinamento e o evolutionary que utilizam as seqüência de treinamento para criar movimentos mais inteligentes a partir de um dado movimento.

Collective: Cria um número finito de *Hives* que jogam de maneira independente e utiliza o resultado de cada um destes Hives para decidir o melhor movimento o enviando ao GP.

O algoritmo proposto para a criação de *Hives* segue abaixo:

```
ALGORITHM: CREATE-HIVES
INPUT: current Game State

FOR i ← 1 to m do
  create Random-Hive
  Move-Sequence-R ← sequence of random moves
                    returned from Random-Hive
  if (Move-Sequence-R is a winning sequence)
    store Move-Sequence-R in Knowledge-Base
END-FOR

FOR i ← 1 to n do
  create Evolutionary-Hive
  Move-Sequence-E ← sequence of intelligent moves
                    returned from Evolutionary-Hive
  if (Move-Sequence-E is a winning sequence)
    store Move-Sequence-E in Knowledge-Base
END-FOR

select best sequence from Knowledge-Base and return
the first move from that sequence
```

]

O *Hives* evolucionários permitem que as sequencias disponíveis na base de dados orginem jogos mais eficientes através da utilização de um algoritmo evolucionista, como mostrado abaixo:

ALGORITHM: EVOLUTIONARY-HIVES
INPUT: current Game State and Knowledge-Base containing
random sequences of moves

FOR each Evolutionary-Hive, **do**
Pick a sequence K_i from Knowledge-Base. K_i has l moves.
FOR each move from m_1 to m_l in K_i, **do**
if (move m_k is legal)
make it
else
make move m_{k+1}
if (end of sequence is reached)
make random move
END-FOR
if (sequence played is different from K_i)
store it in Knowledge-Base
if (probability of crossover is met)
Pick two sequences from Knowledge-Base
Find matching point, and crossover the two sequences at that point.
Store the new sequences in Knowledge-Base
Examine Knowledge-Base to find patterns in the sequences and store them in Pattern-Base
END-FOR

O sistema proposto em “A Multi-Agent Architecture for Game Playing” permite a criação de um sistema que consegue aprender a jogar de maneira eficiente jogos de tabuleiro com o uso de sistemas multiagentes e algoritmos evolucionários. Este trabalho é parte dos esforços para a criação de sistemas inteligentes globais, tendo obtido excelentes resultados contra estratégias clássicas como a busca em árvore utilizando Minimax.

Bibliografia

Using Multi-agent Potential Fields in Real-time Strategy Games, Johan Hagelbäck and Stefan J. Johansson.
The Rise of Potential Fields in Real Time Strategy Bots ,Johan Hagelbäck and Stefan J. Johansson. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.
A Multiagent Potential Field-Based Bot for Real-Time Strategy Games, Johan Hagelbäck and Stefan J. Johansson *International Journal of Computer Games Technology*, 2009.
Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games, Thomas Willer Sandberg, Master Thesis, 2011.
Learning Human-like Movement Behavior for Computer Games C. Thurau, C. Bauckhage, and G. Sagerer *International Conference on the Simulation of Adaptive Behavior (SAB)*, 2004.
A Multi-Agent Architecture for Game Playing, Ziad Kobti, Shiven Sharma, Computational Intelligence and Games, 2007.