Sistemas Multi-Agentes em jogos

Gabriel Nobrega de Lima

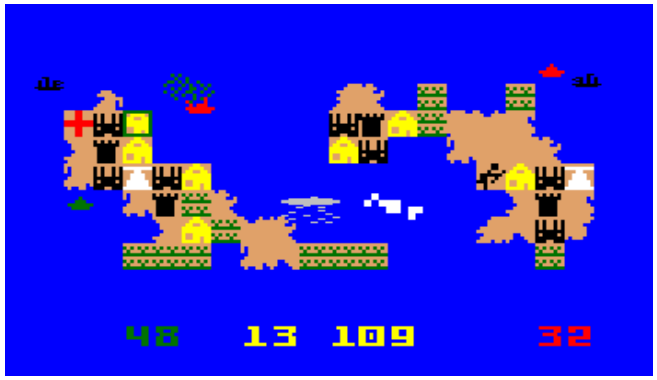
Yuri David Santos

História do gênero RTS

- **Real-time Strategy (RTS) Games** são jogos de estratégia em que o progresso não ocorre através de turnos. Os jogadores tomam ações sobre unidades e constroem estruturas em tempo real com o objetivo de eliminar forças inimigas (unidades e edificações).

História do gênero RTS

- O Utopia (1981) foi o primeiro jogo de RTS produzido.



- O Dune 2 (1992) já agregava a maior parte das características vistas nos jogos de RTS da atualidade.

História do gênero RTS

- Com os anos novos títulos foram lançados aumentado a popularidade do gênero, tais como, Command & Conquer (1995), Red Alert (1996) , Warcraft (1994), Age Of Empires (1997), StarCraft (1998) e etc...

IA NOS JOGOS RTS

- Com o crescimento desta indústria passou-se a investir maiores recursos na criação de IA que proporcionassem melhores experiências em jogo.

- Um dos maiores problemas ao se trabalhar com IA para jogos RTS é a criação de sistemas inteligentes que sejam capazes de derrotar jogadores habilidosos sem a utilização de trapaças (*cheating*).

- Na maioria dos jogos de RTS os níveis de dificuldade mais difíceis são de fato conseguidos utilizando trapaças.
  - Todas as informações do universo estão disponíveis para IA.
  - A IA inicia com maior número de recursos.
  - O tempo de construção e movimentação dos NPCs é maior.

IA NOS JOGOS RTS

- A trapaça é decepcionante quando descoberta pelo jogador. A insatisfação pode ser tão grande que este para de jogar.

- O ideal é construir um sistema inteligente o suficiente que consiga oferecer dificuldade ou mesmo derrotar jogadores habilidosos sob as mesmas condições.

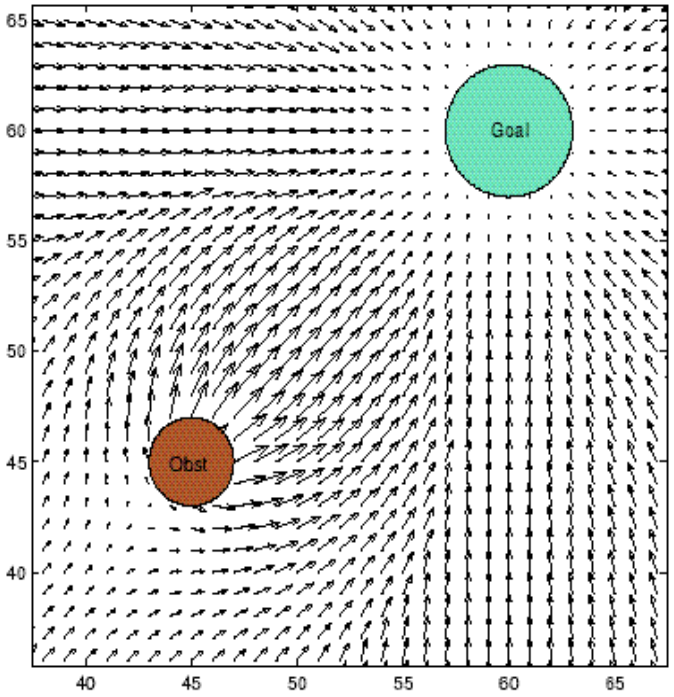
- A maioria dos jogos RTS utilizam sistemas convencionais de navegação para a movimentação de NPCs com a utilização de A* com IA não distribuída.
  - Estas estratégias vêem sendo utilizadas por oferecerem computacionais mais atrativos.
  - Simplicidade.
  - Facilmente implementável.

MULTIAGENTES EM JOGOS RTS

- A proposta em utilizar multiagentes em jogos RTS teve origem no trabalho “Using Multi-agent Potential Fields in Real-time Strategy Games” (2008, Hagelbäck).
  - Tal artigo foi originado a partir das experiência do autor no campeonato de IA em jogos RTS onde estratégia baseada em multiagente acabou derrotando todos os adversários que utilizavam técnicas não distribuídas de IA.
- Outros variações de trabalho surgiram:
  - ” The Rise of Potential Fields in Real Time Strategy Bots”. (Hagelbäck)
  - “A Multiagent Potential Field-Based Bot for Real-Time Strategy Games”.(Hagelbäck)
  - “Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games”(Sandberg, 2011, MasterThesis)

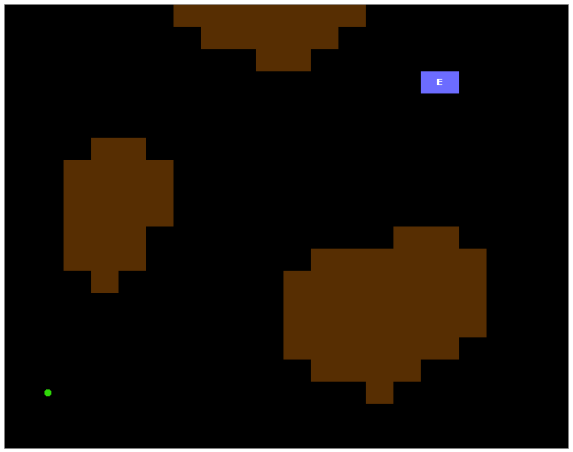
Navegação com Pontecial Fields

- Em 1985 Ossama Khatib introduziu o conceito de Potencial Fields para que robôs móveis pudessem percorrer ambientes evitando choques contra obstáculos

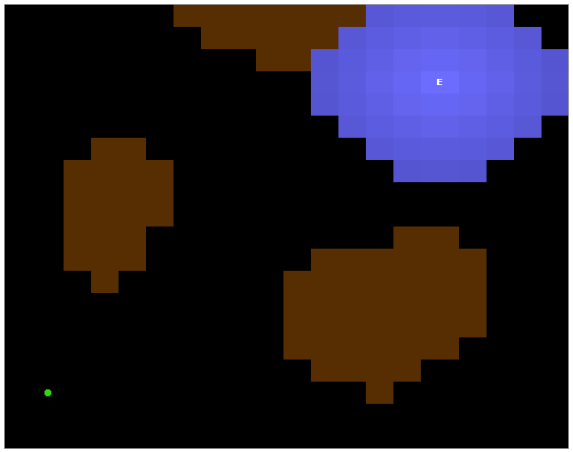


Navegação com Pontecial Fields

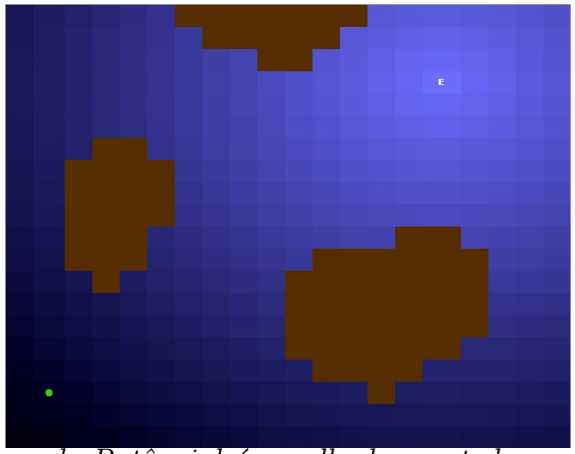
Primeiro Passo:



O objetivo para o agente é posicionado em E.



Uma carga de atração é colocada em E.

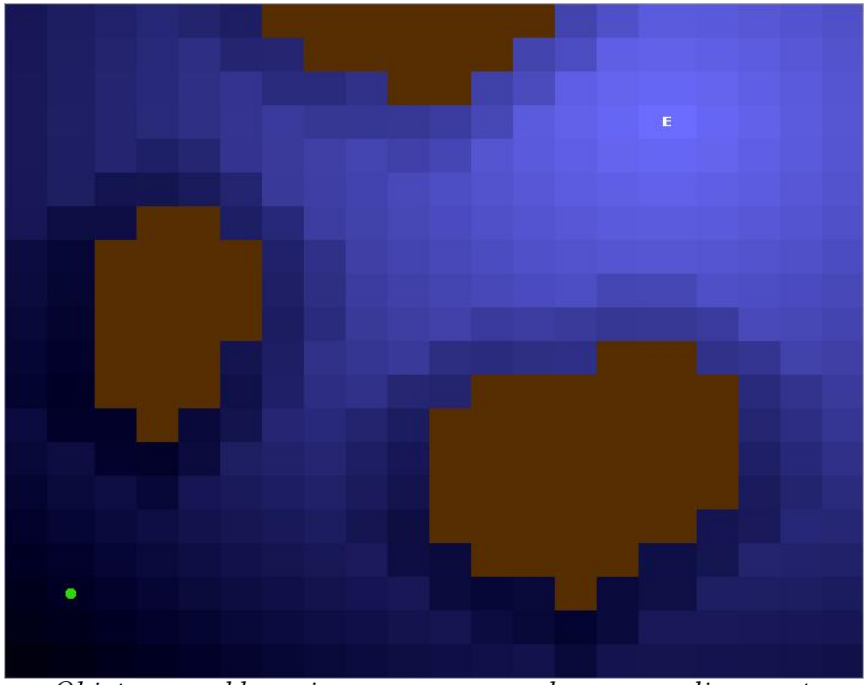


O Mapa de Potêncial é espalhado por todo o cenário.

*Cores mais claras representam células mais atrativas, enquanto aquelas mais escuras repulsivas.

Navegação com Pontecial Fields

Segundo Passo:

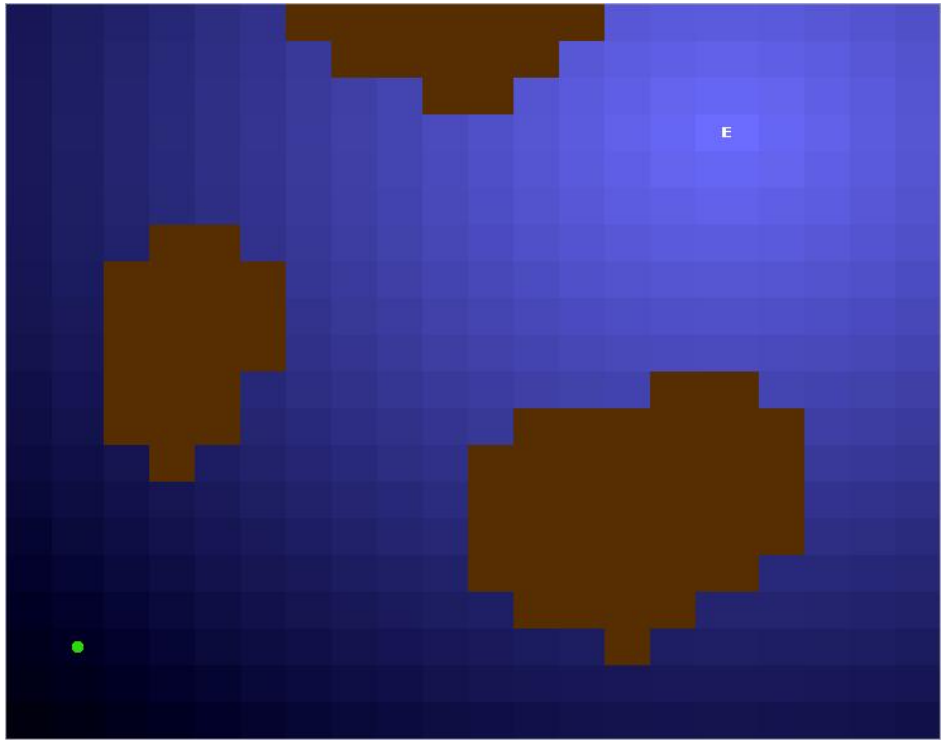


Objetos que bloqueiam a passagem devem repelir agentes.

- Cada objeto colidível no cenário gera uma pequena força de repulsão que é somada ao Mapa de Potencial inicialmente gerado.

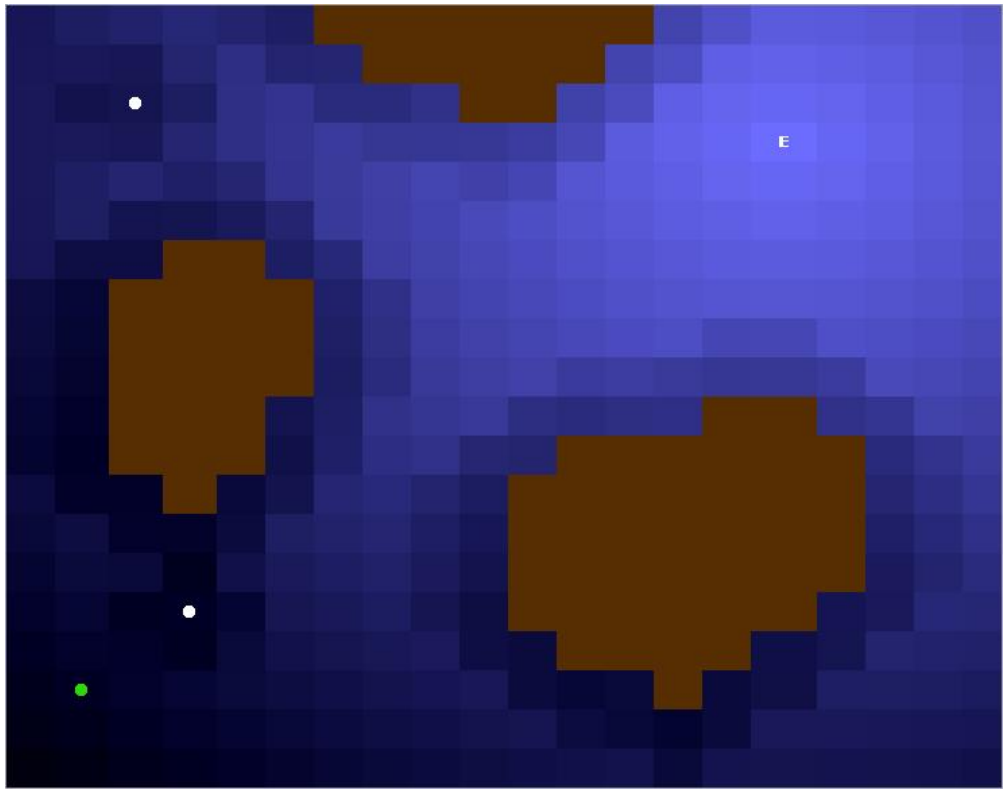
Navegação com Pontecial Fields

- O agente(circulo verde) deve chega na posição E. Ele fará isso de maneira automática sempre caminhando para célula de maior potencial.



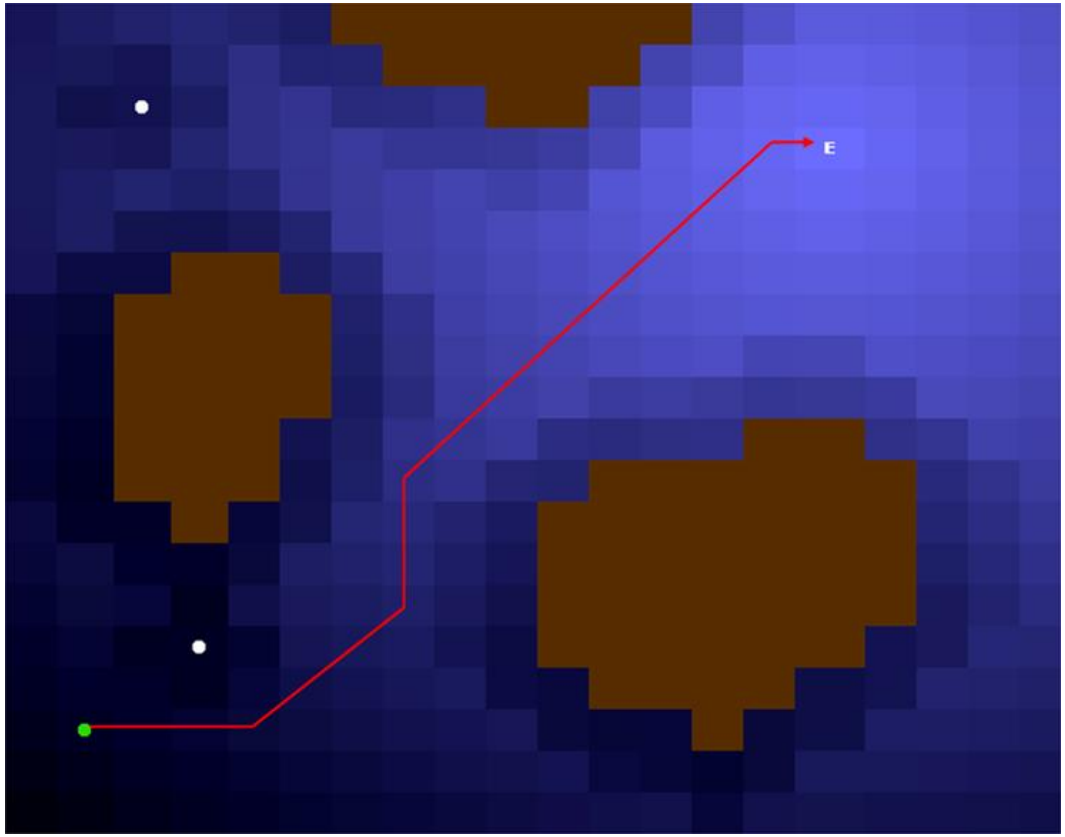
Navegação com Pontecial Fields

- Adicionando novas unidades uma mesma modelagem deve ser realizada.



Navegação com Pontecial Fields

- Rota realizada pelo agente utilizando o Potencial Fields.



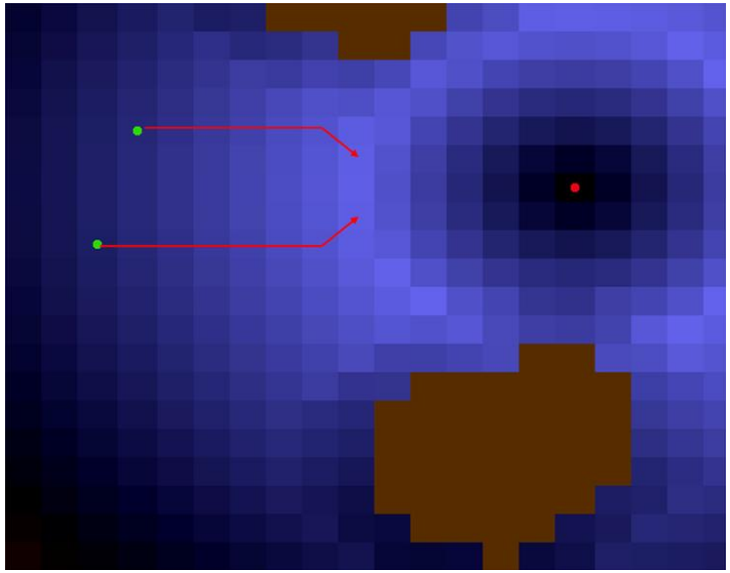
Navegação com Pontecial Fields - Benefícios

- Um dos benefícios mais importantes em se utilizar Potencial Fields para navegação é sua habilidade em se adequar muito bem a cenários dinâmicos.
  - Técnicas de navegação clássicas exigiriam re-calculo da rota toda a vez que um objeto se movesse.

- A criação de comportamento dos agentes na navegação pode ser modelada simplesmente modificando a forma dos Potencial Fields gerados por um objeto.

Navegação com Pontecial Fields - Benefícios

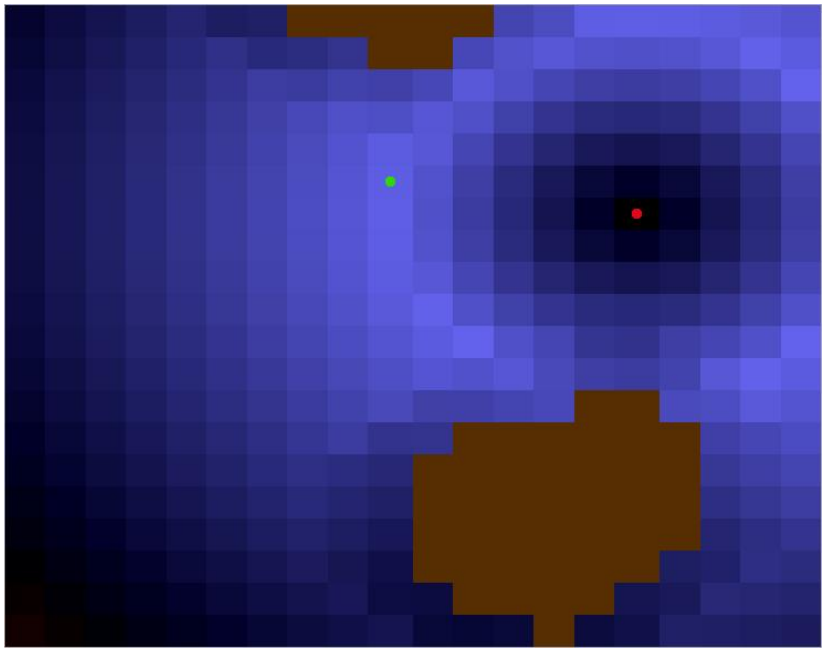
- Criação de um Potencial Field não linear para a obtenção de comportamentos específicos.
  - Tiro de um canhão deve ser realizada sob uma maior distância possível.

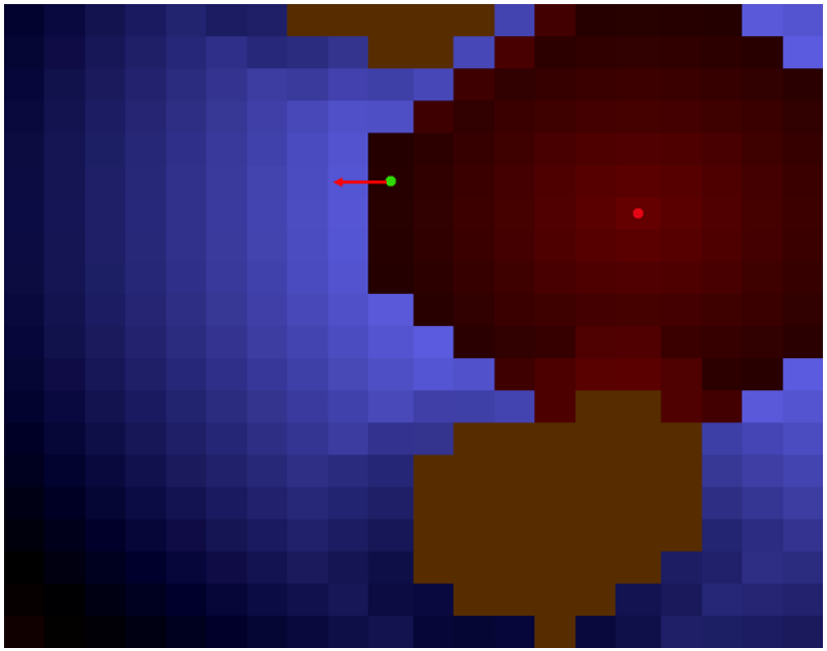


Exemplo: http://www.youtube.com/watch?v=VbQGzr5hiRw

Navegação com Pontecial Fields - Benefícios

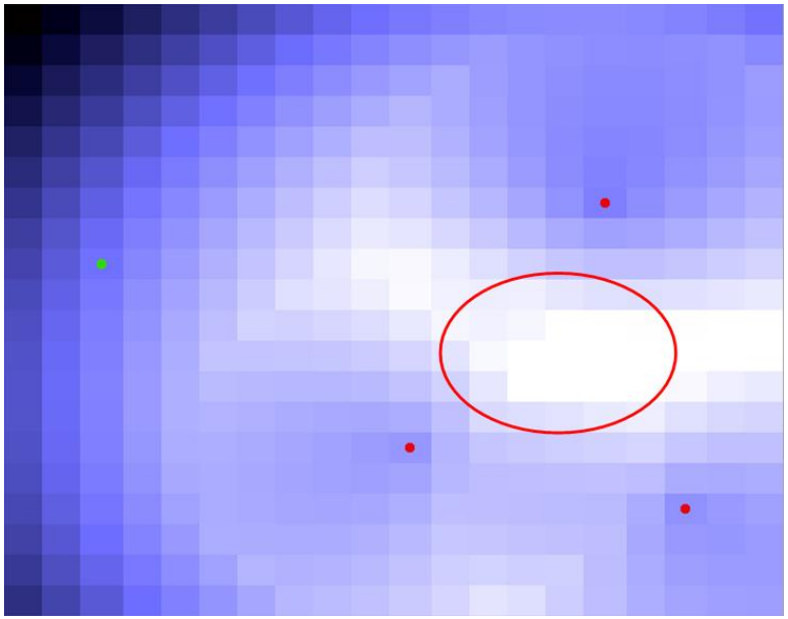
- Ataque e retirada

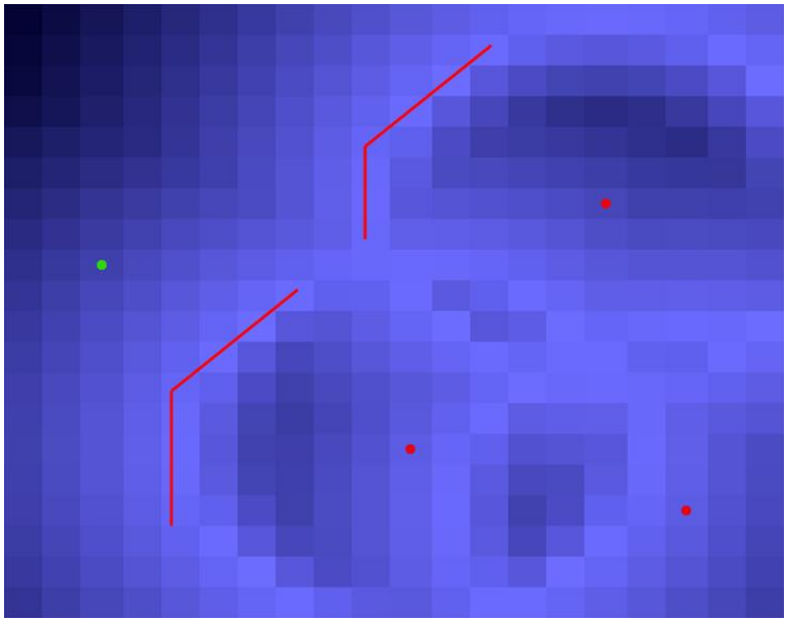




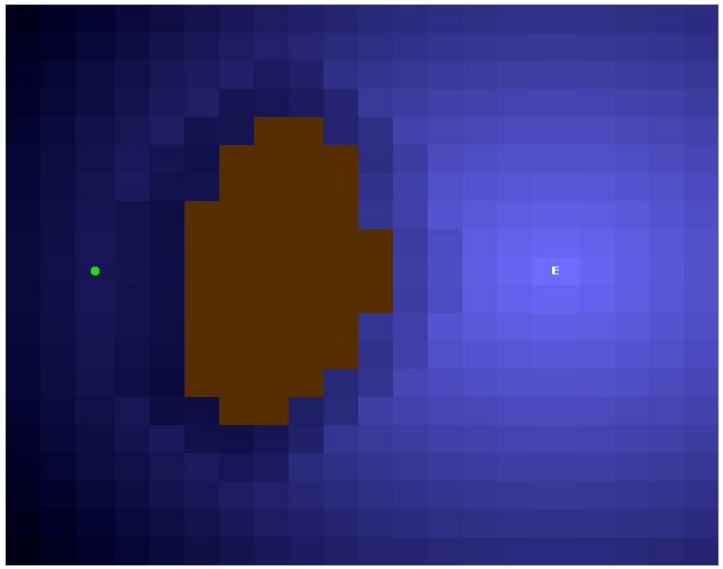
Navegação com Pontecial Fields - Problemas

- Acumulação de Potencial Fields em agentes inimigos.

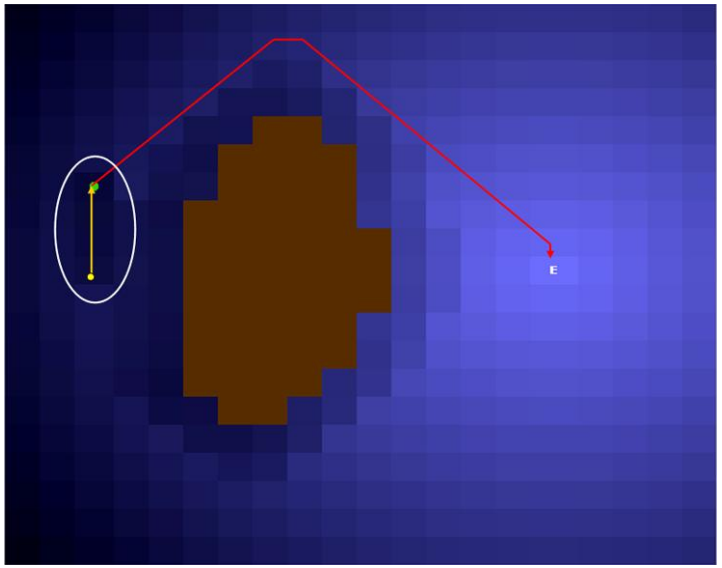




Potencial Fields - Problemas

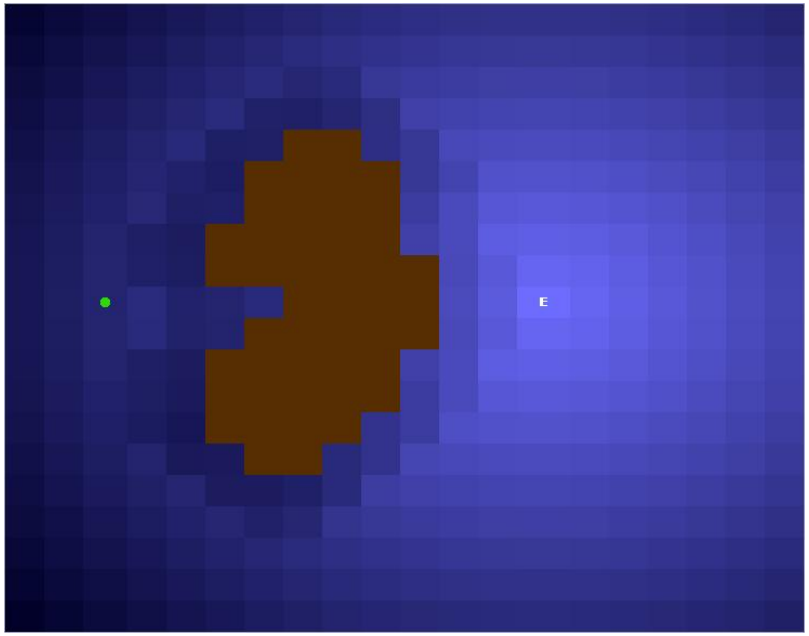


O agente alcança uma célula com maior campo de potencial e fica bloqueada.

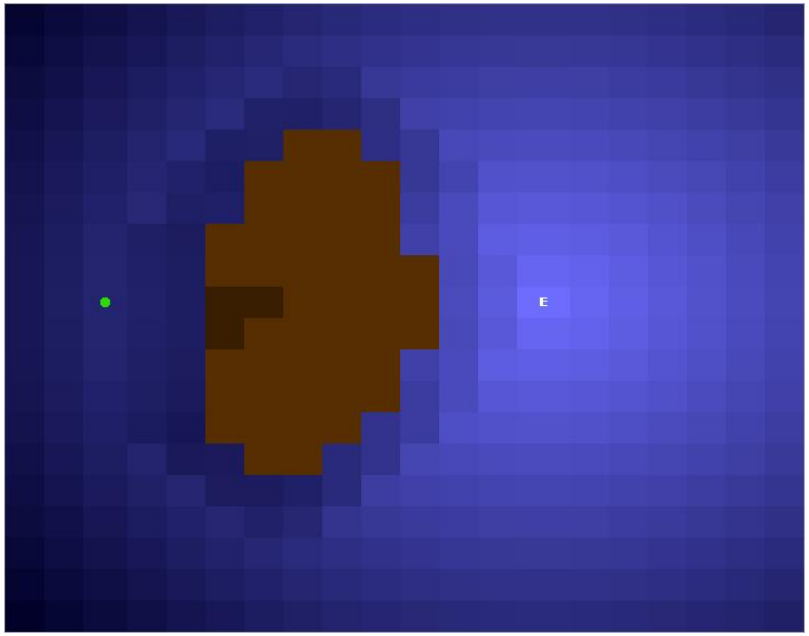


Utilização do Trail [Thurau et al.] para solucionar o problema

Potencial Fields - Problemas com Trail (Regiões não Convexas)



A região não convexa oferece ao agente células na vizinhança com maior potencial. O agente pode caminhar em direção de bloqueio e ficar travado com o trail.



Solução é realizar o preenchimento dos gaps que tornam a região não convexa.

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Utilizado em conjunto com o esquema de navegação baseado em Potencial Fields.

- O jogo é baseado em um combate de tanques. O objetivo é a eliminação de tanques inimigos e bases imóveis.

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- São propostos 2 níveis de táticas gerais (alto nível).

- Ataque simultâneo de bases e tanques.
  - Deve se ter o dobro de tanques que o adversário.
  - O adversário deve ter no máximo 6 tanques.
  - O oponente deve ter somente uma base restante.

- Caso contrário só ataca tanques adversários.
  - Neste caso as bases inimigas devem gerar um Potencial que repele os tanques. Previne que os tanques não colidam em bases inimigas.

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Um agente coordenador, determina quais unidades inimigas devem ser atacadas, priorizando aquelas unidades que já foram atingidas.
  O agente coordenador atua fazendo alguns cálculos, e após isso sabemos quais unidades irão atacar quais outras unidades/bases.

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Inicialmente o coordenador usa uma matriz de possibilidade de ataque, onde a posição ixk indica se o inimigo i pode ser atacado pela unidade k, isto é, se ele está dentro do alcance máximo de ataque (maximum shooting distance – MSD); e um vetor com os hit points (HP) dos inimigos.

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_3 = 4 \\ HP_4 = 4 \\ HP_5 = 3 \end{bmatrix}$$

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Em seguida ele ordena a matriz de possibilidade de acordo com os menores Hps.

- Depois o coordenador organiza os ataques das unidades de forma a eliminar o maior número de unidades possível nesse turno.

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix}$$

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Com isso as outras posições das colunas são zeradas.

- Os valores em branco são irrelevantes a esse ataque.

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix}$$

Arquitetura MultiAgente – Para um RTS de tanques [Hagelbäck et Al.]

- Depois para garantir que cada unidade ataca apenas um inimigo, os outros valores da coluna são preenchidos com zeros.

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix}$$

- Os agentes 0 e 3 atacam 0; 1, 2 e 6 atacam 1; 7 ataca 5; 4 e 5 atacam 4.

Navegação com Pontecial Fields

- São pouco controláveis, isso deve ser levado em conta no planejamento de um jogo RTS.

- São considerados difíceis de se implementar e depurar.

A Multi-Agent Architecture for Game Playing – General Game Playing

- Os sistemas inteligentes em jogos geralmente são criados para apresentar um comportamento de maneira a solucionar problemas específicos.

- General Game Playing (GCP)
  - Aceitam a descrição de qualquer jogo.
  - São capazes de se adaptar e jogar bem o jogo descrito.
  - A inteligência é adquirida sem que o sistema tenha sido programado para atuar sobre um determinado tipo de jogo.

A Multi-Agent Architecture for Game Playing – General Game Playing

- Sistemas “General Game Playing” puros até hoje nunca foram inteiramente implementados.
- O artigo de [Kobti et al.] trata somente da classe de jogos de tabuleiro, definindo estes como “Positional Games”.
- Propõe um sistema capaz de interpretar regras de jogos e com prática, aprender como jogar de maneira eficiente.

A Multi-Agent Architecture for Game Playing – Arquitetura

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 as ações apropriadas.

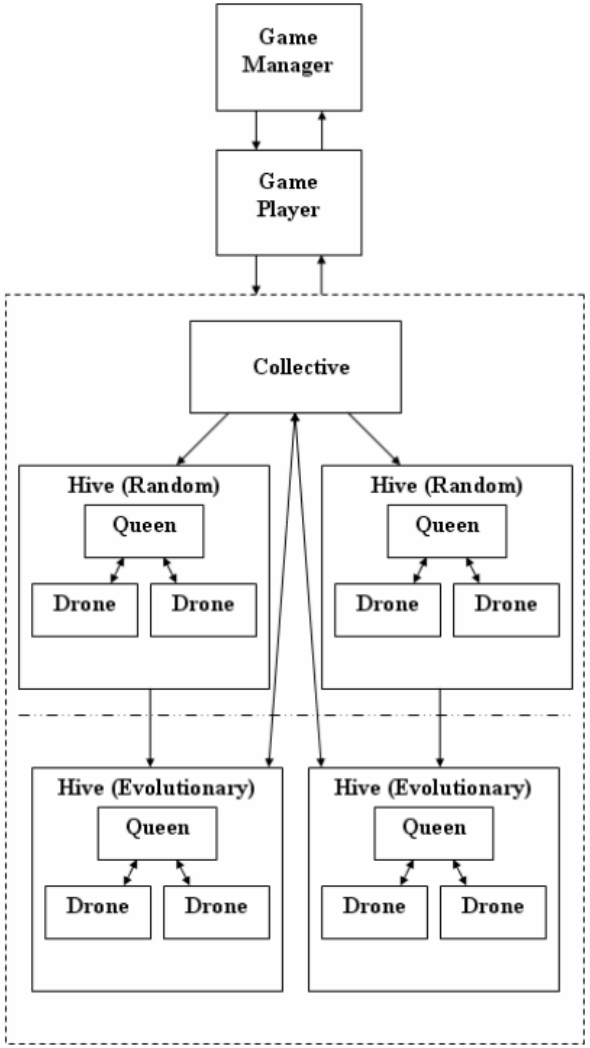
**A Universidade de Standford mantém em seu site de GGP um GM que aceita conexões de GP's para jogar jogos. A Linguagem de descrição de regras de jogos utilizaada é a Knowledge Interchange Format (KIF).

A Multi-Agent Architecture for Game Playing – Como construir o GP?

- Utilizar arvores de busca.
  - Arvores que consideram todas as possibilidades.(ex. Inviável para xadrez).
  - Arvores alfa-beta e Arvores MinMax (minimizam a dimensão de busca com heurísticas).

- A nova proposta é construir um Game Player (GP) com uma arquitetura multiagente.

A Multi-Agent Architecture for Game Playing – Arquitetura



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 Queen e Drones proporcionam a sequencia de passos seguintes no jogo. Efetivamente jogam o jogo.

*Dois tipos,

Random: Cada drone faz jogadas aleatórias mas que respeitam as regras. Geram as seqüência de treinamento.

Evolutionary: 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.

A Multi-Agent Architecture for Game Playing

- Criação de Hives pelo Collection

```
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
```

A Multi-Agent Architecture for Game Playing

- O artigo não deixa claro o tipo de heurística que o Collections utiliza para selecionar uma melhor seqüência. Poderia se utilizar as seqüência retornada pelas Queens que obteve maior quantidade de padrões semelhantes ou menor numero de passos até encontrar a vitória.

A Multi-Agent Architecture for Game Playing

- Cada Evolutionary Hives escolhe aleatoriamente uma seqüência na base de conhecimento.

- Então ele passa a ler a seqüência selecionada a partir do movimento em que o jogo atual está.
  - Se o movimento não poder ser feito ele pula e tenta o próxima.
  - Se o numero de movimentos da seqüência terminar, continue inserindo movimentos legais aleatórios.
- Se houver adição de movimentos à sequencia selecionada, então deve-se adicioná-la na base de conhecimento.

A Multi-Agent Architecture for Game Playing

- Realização de crossover(20%) deve sempre ser realizada para que se gere evolução na base do conhecimento.
  - Seleciona-se aleatoriamente duas seqüência de movimento da base de conhecimento.
  - Defini-se aleatoriamente um ponto de crossover e realiza uma troca de segmentos.
  - As duas novas seqüência são armazenadas na base de conhecimento.
- Isto permite que padrões de jogos vitoriosos comecem a aparecer com freqüência, permitindo que o sistema aprenda a jogar melhor.

A Multi-Agent Architecture for Game Playing

- Evolutionary Hives

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

A Multi-Agent Architecture for Game Playing

- Resultados de uma implementação para o jogo da velha contra uma estratégia clássica de Minimax.

Player	*Wins*	*Draws*
Collective with Evolutionary Hives	30	0
Minimax	0	0

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.