Implementação e Avaliação do Algoritmo MCTS-UCT para o jogo Chinese Checkers

Jhonny Moreira

Introdução

Introdução

Na área da inteligência artificial (IA), a motivação é conseguir colocar os computadores a realizar tarefas que só os humanos conseguem fazer.

Dentro desta área, os jogos apresentam um grande desafio, pois conseguir derrotar um jogador humano experiente é uma tarefa difícil.

Jogos de Tabuleiro

Entre os jogos, os de tabuleiro são os mais populares para a IA, isto porque, são facilmente representados em estruturas de dados, existe um número limitado de jogadas possíveis e seguem uma lista de regras bem definidas.

Dentro dos jogo de tabuleiro, os mais estudados são o xadrez e o jogo de Go, já existindo programas para ambos capazes de derrotar os melhores jogadores humanos.

Chinese Checkers

É um jogo de tabuleiro, que não é tão estudado como o xadrez e o jogo de Go, que pode ser jogado por diferentes números de jogadores (2, 3, 4 ou 6).

Monte-Carlo Tree Search

Este algoritmo tem ganhado popularidade recentemente, graças aos seus excelentes resultados no jogo de Go, que tem sido adaptado a outros jogos.

Objetivos

- Implementar o jogo de Chinese Checkers
- Implementar o Monte-Carlo Tree Search
- Estudar como os diferentes parâmetros do MCTS afectam a resposta tanto para dois como para três jogadores

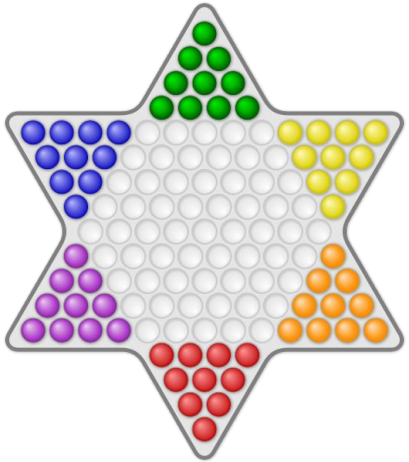
Conceitos Fundamentais

Chinese Checkers

Cada jogador tem 10 peças, que não podem ser removidas do tabuleiro.

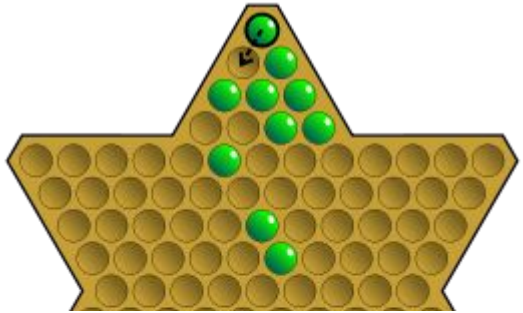
O objetivo do jogo é chegar com todas as peças à posição final, antes que os adversários.

O tabuleiro é uma estrela hexagonal, e as 10 casas dos cantos servem como posições iniciais/finais.



Chinese Checkers - Movimentos Simples

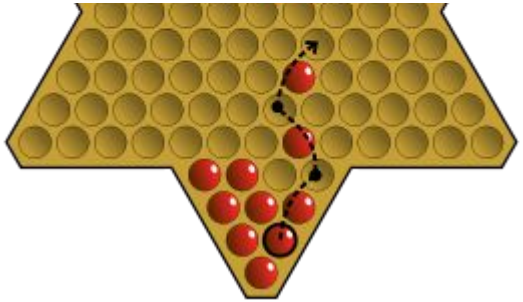
Nos movimentos simples, as peças podem-se mover para posições vazias, que sejam adjacentes a posição atual.



Chinese Checkers - Movimentos Com Saltos

As peças podem saltar por cima de peças adjacentes, desde que caiam numa posição livre.

Os saltos, podem ser concatenados, podendo numa só jogada saltar por cima de várias peças.



Jogos e Algoritmos para Jogos

O que é um jogo?

Formalmente podemos definir um jogo como um problema de procura com os seguintes elementos:

- S0
- Jogador(**S**)
- Jogadas(**S**)
- Jogar(**S**, **A**)
- Terminal(**S**)
- Utilidade(**S**,**P**)

Algoritmos para jogos- MCTS

O MCTS é um algoritmo que resolve o problema de pesquisa com adversários.

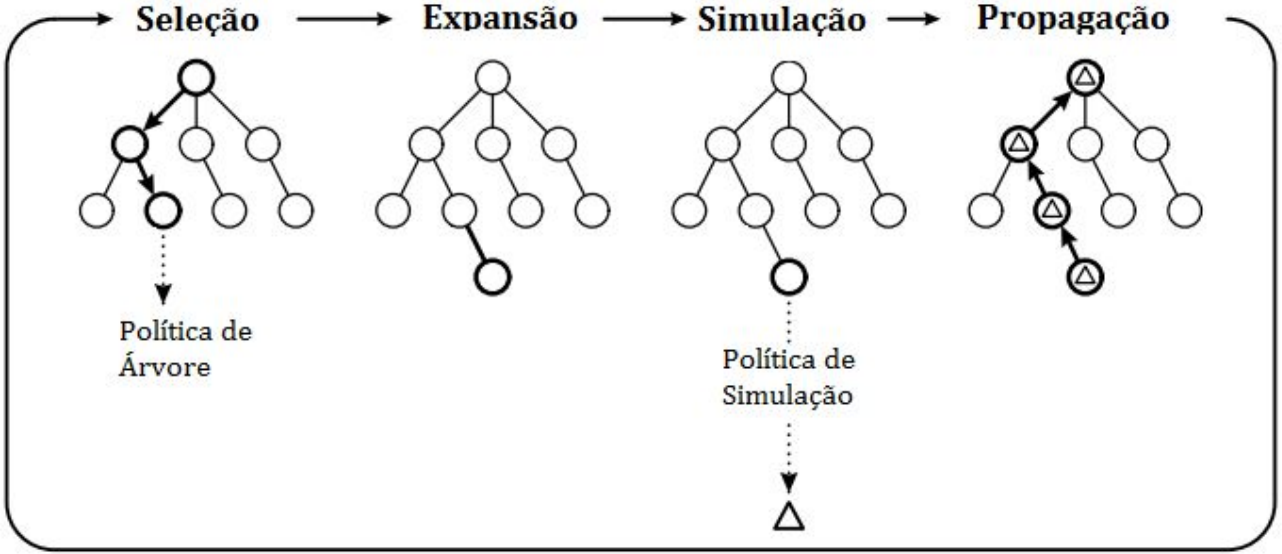
Este algoritmo pode ficar a correr indefinidamente até retornar uma resposta ótima, ou pode-se limitar o tempo de execução ou limitar as iterações do algoritmo, para garantir que retorna uma resposta.

A cada iteração, o algoritmo realiza quatro passos:

Algoritmos para jogos- MCTS

Seleção - utilizando a política de árvore, procura um nó expansível.

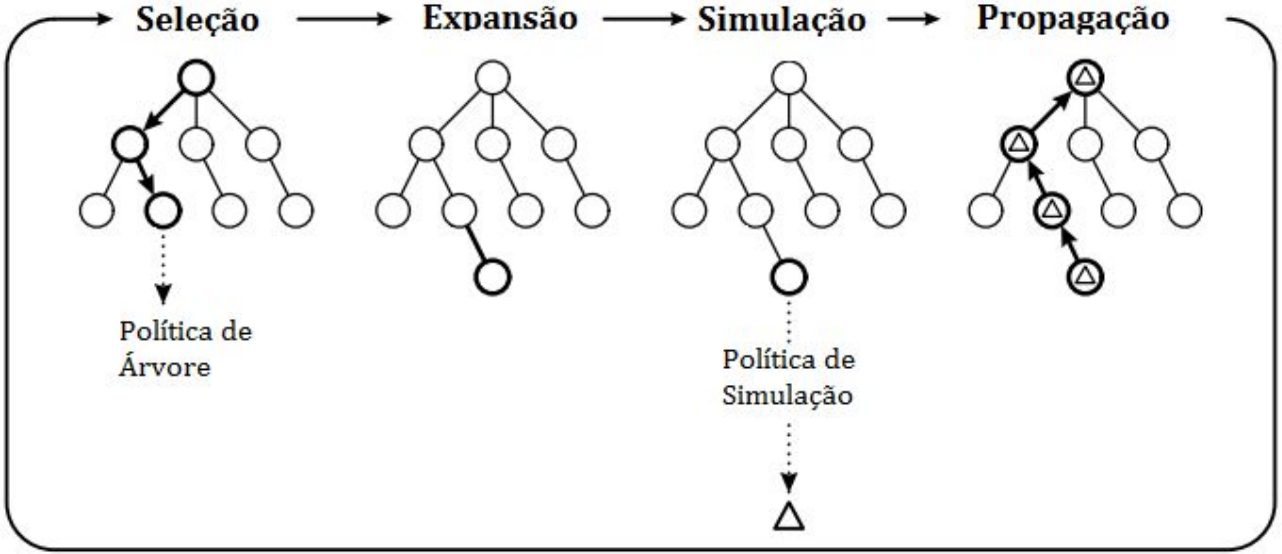
Expansão - são gerados um ou mais filhos, do nó selecionado no passo anterior, e adicionados à árvore.



Algoritmos para jogos- MCTS

Simulação - utilizando a política de simulação, é encontrada uma utilidade para os nós adicionados.

Propagação - os resultados das simulações são propagados pela árvore



Políticas de árvore e simulação

Política de árvore - determina que nó vai ser escolhido e expandido, durantes os passos de seleção e expansão.

Política de simulação - escolhe um caminho a partir de um nó não terminal até um nó terminal (simula um jogo)

As políticas mais simples são a escolha aleatória dos nós, contudo o MCTS apresenta melhores resultados com políticas ajustadas ao problema.

MCTS-UCT

O MCTS tenta aproximar o valor de cada nó ao seu valor real, para isso vai construindo uma árvore parcial. A árvore construída depende da maneira como os nós são selecionados.

O UCT (Upper Confidence Bound for Trees) é uma política de árvore que trata o problema da escolha de um nó, como um problema Multi-Armed Bandit.

O valor dado pelo UCT, para cada nó, corresponde a um valor aproximado da recompensa, calculado a partir das simulações.

MCTS-UCT

Assim sendo o UCT procura um nó j, que maximize: $UCT = \overline{X}_j + C_p\sqrt{\frac{2\ln n}{n_j}}$

- *n* corresponde ao número de vezes que o nó atual foi visitado
- n_j corresponde ao número de vezes que o nó-filho j foi visitado
- X_j é a recompensa média do nó-filho j
- C_p é uma constante

O princípio está em que se n_j tender para ∞ o seu valor vai tender para 0.

Outros Algoritmos para Jogos

Para dois jogadores:

- Min-Max
- Corte Alpha-Beta

Para múltiplos jogadores:

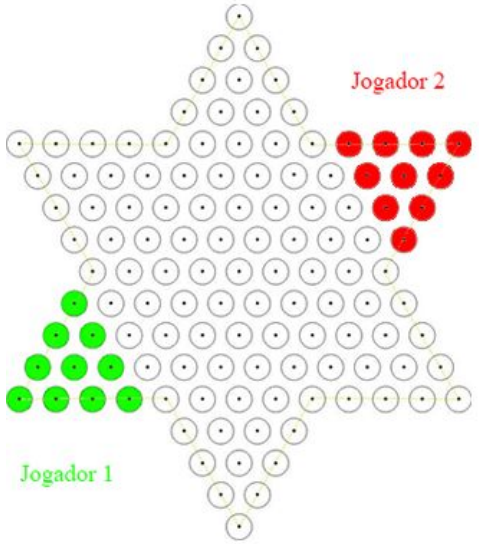
- Max^n
- Paranoid

Implementação - Chinese Checkers

Implementação - Chinese Checkers

Estado de Jogo:

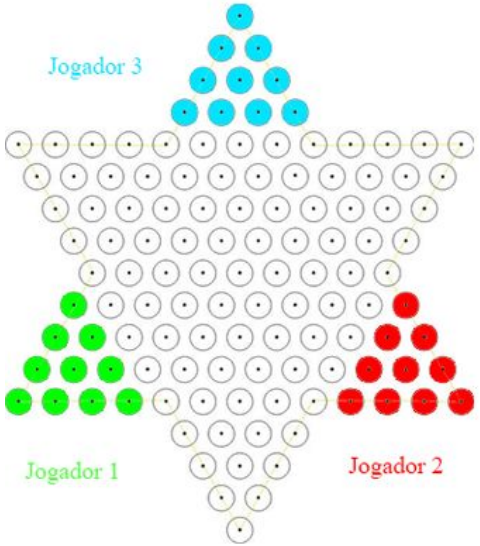
- Tabuleiro
- Jogadores
- Jogador da vez
- Estado anterior



Implementação - Chinese Checkers

Um estado de jogo também é capaz de gerar as jogadas possíveis, para o jogador da vez.

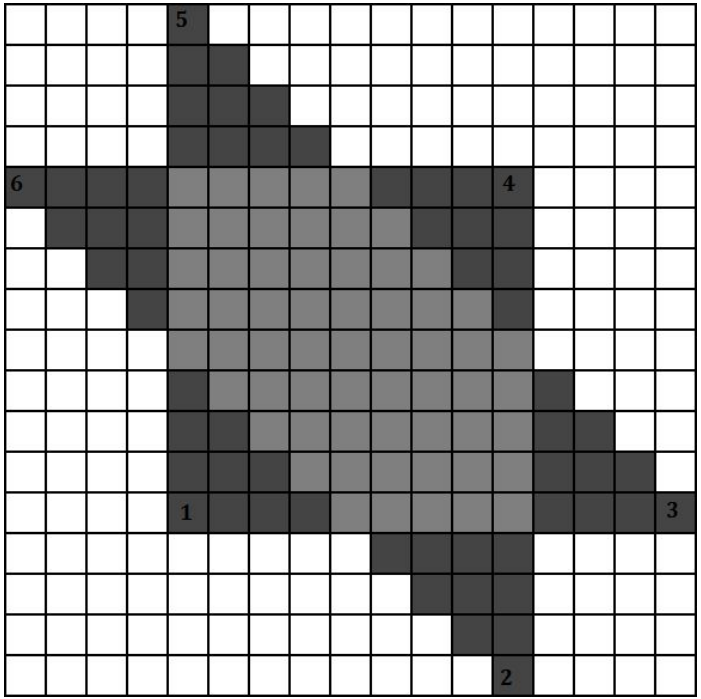
Quando o jogo é inicializado, gera-se o estado inicial.



Tabuleiro

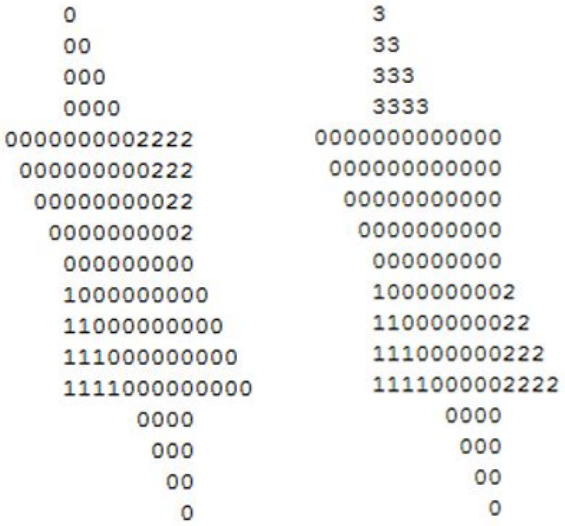
O Tabuleiro é representado numa matriz 17x17.

Para definir quais são os espaços jogáveis é utilizado um conjunto de regras.



Tabuleiro

Quando um jogo é inicializado, na matriz, os espaços em branco são posições ilegais, os espaços com 0 são casas vazias, e os números 1-6 representam uma peça do jogador do seu respectivo número.



Jogadores

Os Jogadores são definidos:

- Número
- Posição inicial no tabuleiro
- Peças e respectivas coordenadas.

Implementação - MCTS-UCT

Implementação MCTS-UCT

- O limite computacional é o número de nós explorados
- Durante a expansão só os 10 melhores nós são adicionados à árvore.

Algoritmo 2: MCTS-UCT

```
function MCTS-UCT(S0)
    Cria nó-raiz V0 a partir do estado S0
    while Dentro dos limites computacionais do
        V1 ⇐ Política_de_Árvore(V0)
        Δ ⇐ Política_de_Simulação(S(V1))
        Propagar(Δ, V1)
    return Melhor_Filho(V0)
end function
```

Implementação Política de Árvore

- A recompensa média corresponde a percentagem de vitórias
- A constante C_p usada foi $1/\sqrt{2}$

Algoritmo 3: Política de Árvore

```
function POLÍTICA_DE_ÁRVORE(V)
    while V tem descendentes do
        V ⇐ UCB1(V)
    return V
end function
function UCB1(V)
    return arg_max_{V' ∈ FilhosdeV}  Q(V')/N(V') + (1/√2) √(2 ln N(V) / N(V'))
end function
```

Implementação Política de Simulação

Durante as simulações, são escolhidas as jogadas que jogam as peças mais atrás 95% das vezes, e jogadas aleatórias 5%.

Em qualquer decisão do algoritmo, se existir empate o desempate é realizado aleatoriamente.

Algoritmo 4: Política de Simulação

```
function POLÍTICA_DE_SIMULAÇÃO(V)
    while V não for terminal do
        V ⇐ arg_min_{V'∈S(V)} Farthest_First(V')
    return V
end function
```

Testes

Parâmetros

Para dois e três jogadores, foram realizados testes nos seguintes parâmetros :

- Políticas de Seleção de nós
- Fator de Ramificação
- Nós Explorados
- Importância da primeira Jogada

Resultados para dois jogadores

Resultados - Política de Seleção dos nós

A versão Greedy venceu, no total, 73,3% dos jogos.

E consistentemente venceu mais jogos para qualquer número de nós explorados.

Nós Explorados	**Greedy**	Epsilon-Greedy
1000	76,7%	23,3%
2000	60%	40%
4000	83,3%	16,7%
Total	**73,3%**	26,7%

Resultados - Fator de Ramificação

A versão que adiciona 10 nós venceu, no total, 55,6% dos jogos.

Contudo, a versão que adiciona 5 nós venceu mais jogos, quando só podia explorar 1000 nós.

Nós Explorados	5 Nós Adicionados	**10 Nós Adicionados**
1000	56,7%	43,3%
2000	33,3%	66,7%
4000	43,3%	56,7%
Total	44,4%	**55,6%**

Resultados - Fator de Ramificação

A versão que adiciona 10 nós venceu, no total, 80% dos jogos.

E consistentemente venceu mais jogos que a versão que adicionava 20 nós.

Nós Explorados	**10 Nós Adicionados**	20 Nós Adicionados
1000	83,3%	16,7%
2000	86,7%	13,3%
4000	70%	30%
Total	**80%**	20%

Resultados - Nós Explorados

A versão que explora 4000 nós ganhou, no total, 70% dos jogos.

Nós Explorados	1000	2000	**4000**
1000	-	73,3%	73,3%
2000	26,7%	-	66,7%
4000	26,7%	33,3%	-
Total	26,7%	53,3%	**70%**

Resultados - Primeira Jogada

No total dos resultados obtidos nos testes anteriores, analisamos a percentagem de vitórias do jogador com direito à primeira jogada (primeiro jogador). O primeiro jogador venceu 55,2% dos jogos.

Resultados para três jogadores

Resultados - Política de Seleção dos nós

A versão Greedy venceu, no total, 61,1% dos jogos. E consistentemente venceu mais jogos.

Tanto para dois como para três jogadores vemos a versão Greedy a apresentar melhores resultados.

Nós Explorados	**Greedy**	Epsilon-Greedy
1000	63,3%	36,7%
2000	66,7%	33,3%
4000	53,3%	46,7%
Total	**61,1%**	38,9%

Resultados - Fator de Ramificação

A versão que adiciona 10 nós venceu, no total, 62,2% dos jogos.

Tanto para dois como para três jogadores a versão que adiciona 10 nós venceu mais jogos.

Nós Explorados	5 Nós Adicionados	**10 Nós Adicionados**
1000	46,7%	53,3%
2000	26,7%	73,3%
4000	40%	60%
Total	37,8%	**62,2%**

Resultados - Fator de Ramificação

A versão que adiciona 10 nós venceu, no total, 78,9% dos jogos.E consistentemente venceu mais jogos.

Tanto para dois como para três jogadores a versão que adiciona 10 nós venceu mais jogos.

Nós Explorados	**10 Nós Adicionados**	20 Nós Adicionados
1000	76,7%	23,3%
2000	66,7%	33,3%
4000	93,3%	6,7%
Total	**78,9%**	21,1%

Resultados - Nós Explorados

A versão que explora 4000 nós ganhou, no total, 73,3% dos jogos.

A versão que adiciona mais nós ganha mais jogos, tanto para dois como três jogadores.

Nós Explorados	1000	2000	**4000**
1000	-	76,7%	80%
2000	23,3%	-	66,7%
4000	20%	33,3%	-
Total	21,6%	55%	**73,3%**

Resultados - Primeira Jogada

Para dois jogadores, o primeiro jogador ganhou 55,2% dos jogos. Analisando a percentagem de vitórias do primeiro jogador nos testes de três jogadores, vemos que o primeiro jogador vence 35,3% dos jogos.

Conclusão

Conclusão

Com os resultados obtidos, concluímos que:

- Para a seleção dos nós, a política Greedy ganhou mais jogos.
- Com o fator de ramificação a 10, o algoritmo ganhou mais jogos. Contudo os resultados, nos levam a crer, que este valor pode ser ajustado ao número de nós explorados.
- Ao aumentarmos o número de nós explorados, estamos a maximizar as chances de vitória.
- Existe uma vantagem para o jogador que tem a primeira jogada.

Obrigado pela Atenção