Jogo Da Velha Usando Minimax

> José Luiz Corrêa Junior - GitHub - <juninhopc@icloud.com>
>
> Felippe Mangueira da Silva Sposito - GitHub - <felippesposito@hotmail.com>

> > Escrito em: Ruby

Princípio do problema

O problema inicial era criar um Inteligência artificial para jogar um jogo através de uma interface de usuário, afim de cumprir o teste de turing. Para a realização do exercício foi utilizado a estratégia do jogo da veha.

Princípios básicos do funcionamento

O programa utiliza cálculo de possibilidades através do algoritmo de lógica minimax para prever os movimentos básicos do jogo da velha, e assim simular jogadas realistas.

Arquivos e módulos

O programa é composto de apenas um arquivo, `jogo.rb` , que consiste de um código ruby responsável por calcular as alternativas de jogada e interagir com o jogador.

Linguagem

A linguagem utilizada foi Ruby, pela riqueza de métodos e velocidade de escrita de código (dinamicidade). A possibilidade de uso de métodos de bibliotecas prontas também foi considerada.

Código

O código foi dividido em métodos (dentro de classes) que serão colocados abaixo:

Classe `Jogo`

`mostra_fim_jogo`

Imprime os resultados do jogo;

`jogada_humano`

Método que recebe o input das jogadas do usuário; manda depois para o método de cálculo de possibilidades; Também verifica jogadas inválidas.

`mostra_tabuleiro`

O método recebe a casa escolhida pela IA, e imprime na tela para o usuário.

`turno`

Verifica se é final de jogo, e, caso não seja, localiza o dono do turno atual (IA ou usuário).

Classe `resultados_intermediarios`

Classe responsável por realizar os cálculos (que faz o trabalho duro, por assim dizer) pois calcula todas as possíveis jogadas, e diminui as possibilidades à cada jogada do usuário.

Subclasse recursiva

```
def resultado_intermediario
    ranks = movimentos.collect{ |estado_jogo| estado_jogo.rank }
    if jogador_atual == 'X'
      #retorna ranks.max se for o computador
      #para maximizar a jogada dele
      ranks.max
    else
      #retorna ranks.min se for a jogada do humano
      #para minimizar a jogada do humano
      ranks.min
    end
```

Faz os cálculos máximos e mínimos da vitória através de recursão, com o algoritmo minimax

```
class ArvoreJogo
  def generate
    #Ja passa o jogador atual e o tabuleiro no metodo initialize
    estado_inicial = EstadoJogo.new('X', Array.new(9))
    gerador_movimentos(estado_inicial)
    estado_inicial #retorna o tabuleiro criado
  end
```

Classe `EstadoJogo`

Basicamente a classe EstadoJogo é a classe que verifica, aloca e maneja a memória dentro do programa, através das classes `Temporario`

```
  class Temporario
    #criacao do metodo de acesso, leitura e escrita
    attr_accessor :states
    def initialize
      @states = {}
    end
  end
```

e `Initialize`
(esse metodo é acionado automaticamente com uma criação de um objeto instanciando a classe que ele pertence, isso é padrão da linguagem)

```
  def initialize(jogador_atual, tabuleiro)
    #metodo self para chamar o metodo de acesso das variaveis de instancia
    self.jogador_atual = jogador_atual
    self.tabuleiro = tabuleiro
    #inicializa o vetor chamado movimentos que vai receber as jogadas
    self.movimentos = []
  end
```