Artigo
Henry Pereira · Ago. 10, 2021 8min de leitura

Vamos lutar contra as máquinas!

Calma, calma, não estou incentivando uma guerra contra as máquinas no melhor estilo sci-fi para impedir ao dominação mundial do Ultron ou da Skynet.

Ainda não... ainda não

Convido você para desafiarmos as máquinas através da criação de um jogo bem simples utilizando ObjectScript com Python embarcado.

Tenho que dizer que fiquei super empolgado com a feature do Embedded Python no InterSystems IRIS, é incrível o leque de possibilidades que se abre para criar aplicações fantásticas.

Vamos construir um jogo da velha, as regras são bem simples e acredito que todos sabem como jogar.

Era o que me salvava do tédio na minha infância durante viagens longas de carro com a família, antes de crianças terem celulares ou tablets, nada como desafiar meus irmãos a jogar algumas partidas no vidro embaçado.

Então apertem o cinto e vamos lá!

Regras

Como comentado, as regras são muito simples:

- apenas 2 jogadores por partida
- é jogado em turnos em um grid de 3x3
- o jogador humano sempre será a letra X e o computador a letra O
- os jogadores só poderão colocar as letras nos espaços vazios
- o primeiro que completar uma sequência de 3 letras iguais na horizontal, ou na vertical ou na diagonal, é o vencedor
- quando os 9 espaços estiverem ocupados será empate e o fim da partida

Todo o mecanismo e as regras escreveremos em ObjectScript, o mecanismo do jogador do computador será

escrito em Python.

Vamos colocar as mãos na massa

Controlaremos o tabuleiro em uma global, sendo que cada linha estará em um nó e cada coluna em um piece.

Nosso primeiro método é para iniciar o tabuleiro, para facilitar irei iniciar a global já com os nós (linhas A, B e C) e com os 3 pieces:

```
/// Iniciate a New Game
ClassMethod NewGame() As %Status
{
  Set sc = $$$OK
  Kill ^TicTacToe
  Set ^TicTacToe("A") = "^^"
  Set ^TicTacToe("B") = "^^"
  Set ^TicTacToe("C") = "^^"
  Return sc
}
```

neste momento iremos criar o método para adicionar as letras nos espaços vazios, para isto cada jogador irá passar a localização do espaço do tabuleiro.

Cada linha uma letra e coluna um número, para colocar o X no meio, por exemplo, passamos B2 e a letra X para o método.

```
ClassMethod MakeMove(move As %String, player As %String) As %Boolean
{
  Set $Piece(^TicTacToe($Extract(move,1,1)),"^",$Extract(move,2,2)) = player
}
```

Vamos validar se a coordenada passada é válida, a maneira mais simples que vejo é utilizando uma expressão regular:

```
ClassMethod CheckMoveIsValid(move As %String) As %Boolean
{
  Set regex = ##class(%Regex.Matcher).%New("(A|B|C){1}[0-9]{1}")
  Set regex.Text = $ZCONVERT(move,"U")
  Return regex.Locate()
}
```

precisamos garantir que o espaço selecionado esteja vazio

```
ClassMethod IsSpaceFree(move As %String) As %Boolean
{
  Quit ($Piece(^TicTacToe($Extract(move,1,1)),"^",$Extract(move,2,2)) = "")
}
```

Nooice!

Agora vamos verificar se algum jogador venceu a partida ou se o jogo já terminou, para isto vamos criar o método CheckGameResult.

Primeiro verificamos se teve algum vencedor completando pela horizontal, usaremos uma list com as linhas e um simples $Find resolve

```
    Set lines = $ListBuild("A","B","C")
    // Check Horizontal
    For i = 1:1:3 {
      Set line = ^TicTacToe($List(lines, i))
      If (($Find(line,"X^X^X")>0)||($Find(line,"O^O^O")>0)) {
        Return $Piece(^TicTacToe($List(lines, i)),"^", 1)_" won"
      }
    }
```

Com outro for verificamos a vertical

```
For j = 1:1:3 {
      If (($Piece(^TicTacToe($List(lines, 1)),"^",j)'="") &&
        ($Piece(^TicTacToe($List(lines, 1)),"^",j)=$Piece(^TicTacToe($List(lines, 2))
,"^",j)) &&
        ($Piece(^TicTacToe($List(lines, 2)),"^",j)=$Piece(^TicTacToe($List(lines, 3))
,"^",j))) {
        Return $Piece(^TicTacToe($List(lines, 1)),"^",j)_" won"
      }
    }
```

para verificar a diagonal:

```
    If (($Piece(^TicTacToe($List(lines, 2)),"^",2)'="") &&
      (
        (($Piece(^TicTacToe($List(lines, 1)),"^",1)=$Piece(^TicTacToe($List(lines, 2)
),"^",2)) &&
          ($Piece(^TicTacToe($List(lines, 2)),"^",2)=$Piece(^TicTacToe($List(lines, 3
)),"^",3)))||
        (($Piece(^TicTacToe($List(lines, 1)),"^",3)=$Piece(^TicTacToe($List(lines, 2)
),"^",2)) &&
        ($Piece(^TicTacToe($List(lines, 2)),"^",2)=$Piece(^TicTacToe($List(lines, 3))
,"^",1)))
      )) {
      Return ..WhoWon($Piece(^TicTacToe($List(lines, 2)),"^",2))
    }
```

por último, verificamos se teve um empate

```
    Set gameStatus = ""
    For i = 1:1:3 {
      For j = 1:1:3 {
        Set:($Piece(^TicTacToe($List(lines, i)),"^",j)="") gameStatus = "Not Done"
      }
    }
    Set:(gameStatus = "") gameStatus = "Draw"
```

Great!

É hora de criarmos a máquina!

Vamos criar o nosso adversário, precisamos criar um algoritmo capaz de calcular todos os movimentos disponíveis e usar uma métrica para saber qual é o melhor movimento.

O ideal é utilizar um algoritmo de decisão chamado de MiniMax (Wikipedia: MiniMax)

O algoritmo MiniMax é uma regra de decisão utilizado em teoria dos jogos, teoria de decisão e inteligência artificial.

Basicamente, precisamos saber como jogar assumindo quais serão os possíveis movimentos do oponente e pegar o melhor cenário possível.

Em detalhes, pegamos o cenário atual e recursivamente verificamos o resultado do movimento de cada jogador, caso o computador ganhar a partida pontuamos com +1, caso perder então pontuamos como -1 e 0 como um empate.

Caso não for o final do jogo, abrimos outra árvore a partir do estado atual. Feito isto encontramos a jogada com o valor máximo para o computador e a mínima para o adversário.

Veja o diagrama abaixo, existem 3 movimentos disponíveis: B2, C1 e C3.

Escolhendo C1 ou C3, o oponente tem uma chance de ganhar no próximo turno, já escolhendo B2 independente do movimento do adversário ganhamos a partida.

É como ter a joia do tempo em suas mãos para encontrar a melhor linha do tempo.

Convertendo para python

```
ClassMethod ComputerMove() As %String [ Language = python ]
{
  import iris
  from math import inf as infinity
  computerLetter = "O"
  playerLetter = "X"

  def isBoardFull(board):
    for i in range(0, 8):
      if isSpaceFree(board, i):
        return False
    return True

  def makeMove(board, letter, move):
    board[move] = letter

  def isWinner(brd, let):
    # check horizontals
    if ((brd[0] == brd[1] == brd[2] == let) or \
      (brd[3] == brd[4] == brd[5] == let) or \
      (brd[6] == brd[7] == brd[8] == let)):
        return True
    # check verticals
    if ((brd[0] == brd[3] == brd[6] == let) or \
```

```
        (brd[1] == brd[4] == brd[7] == let) or \
        (brd[2] == brd[5] == brd[8] == let)):
        return True
    # check diagonals
    if ((brd[0] == brd[4] == brd[8] == let) or \
        (brd[2] == brd[4] == brd[6] == let)):
        return True
    return False

def isSpaceFree(board, move):
    #Retorna true se o espaco solicitado esta livre no quadro
    if(board[move] == ''):
      return True
    else:
      return False

def copyGameState(board):
    dupeBoard = []
    for i in board:
      dupeBoard.append(i)
    return dupeBoard

def getBestMove(state, player):
    done = "Done" if isBoardFull(state) else ""
    if done == "Done" and isWinner(state, computerLetter): # If Computer won
      return 1
    elif done == "Done" and isWinner(state, playerLetter): # If Human won
      return -1
    elif done == "Done":    # Draw condition
      return 0

    # Minimax Algorithm
    moves = []
    empty_cells = []
    for i in range(0,9):
      if state[i] == '':
        empty_cells.append(i)

    for empty_cell in empty_cells:
      move = {}
      move['index'] = empty_cell
      new_state = copyGameState(state)
      makeMove(new_state, player, empty_cell)

      if player == computerLetter:
          result = getBestMove(new_state, playerLetter)
          move['score'] = result
      else:
          result = getBestMove(new_state, computerLetter)
          move['score'] = result

      moves.append(move)

    # Find best move
    best_move = None
    if player == computerLetter:
        best = -infinity
        for move in moves:
            if move['score'] > best:
```

```
                best = move['score']
                best_move = move['index']
    else:
        best = infinity
        for move in moves:
            if move['score'] < best:
                best = move['score']
                best_move = move['index']

    return best_move

  lines = ['A', 'B', 'C']
  game = []
  current_game_state = iris.gref("^TicTacToe")

  for line in lines:
    for cell in current_game_state[line].split("^"):
      game.append(cell)

  cellNumber = getBestMove(game, computerLetter)
  next_move = lines[int(cellNumber/3)]+ str(int(cellNumber%3)+1)
  return next_move
}
```

Primeiro converto a global em um array simples, ignorando colunas e linhas deixando flat para facilitar.

A cada movimento analisado chamamos o método copyGameState, que como o nome diz, copia o estado do jogo naquele momento, onde aplicamos o MinMax.

O método getBestMove que será chamado recursivamente até finalizar o jogo encontrando um vencedor ou o empate.

Primeiro os espaços vazios são mapeados e verificamos o resultado de cada movimento alternando entre os jogadores.

Os resultados são armazenados em move['score'] para depois de verificar todas as possibilidades encontrar o melhor movimento.

Espero que você tenha se divertido, é possível melhorar a inteligência utilizando algoritmos como Alpha-Beta Pruning (Wikipedia: AlphaBeta Pruning) ou redes neurais, só cuidado para não dar vida a Skynet.

Fique livre para deixar comentários ou perguntas

That's all folks

Código completo:
InterSystems Iris versão 2021.1.0PYTHON

#Early Access Program (EAP) #IA #Python #InterSystems IRIS