Artículo
Henry Pereira · 26 sep, 2021 Lectura de 8 min

¡Luchemos contra las máquinas!



Calma, calma, no estoy promoviendo una guerra contra las máquinas como en las películas de ciencia ficción, para evitar la dominación mundial de Ultron o Skynet.
Todavía no, todavía no

Os invito a retar a las máquinas a través de la creación de un juego muy sencillo usando ObjectScript con Python embebido.

Tengo que deciros que me emocioné mucho con la función de Python integrado en InterSystems IRIS. Es increíble el montón de posibilidades que se abren para crear aplicaciones fantásticas.

Vamos a construir un juego "tres en raya". Las reglas son bastante sencillas y creo que todo el mundo sabe jugar.

Es lo que me salvó del tedio en mi infancia, durante los largos viajes en coche con la familia antes de que los niños tuvieran teléfonos móviles o tabletas. ¡Nada como retar a mis hermanos a jugar unas partidas en el cristal borroso!

Así que... ¡abrochaos el cinturón y vámonos!

Normas

Como he comentado, las reglas son bastante simples:

- solo 2 jugadores por set
- se juega por turnos en una cuadrícula de 3x3
- el jugador humano siempre será la letra X y la computadora la letra O
- los jugadores solo podrán poner las letras en los espacios vacíos
- el primero en completar una secuencia de 3 letras iguales en horizontal, o en vertical o en diagonal, es el ganador
- cuando se ocupen los 9 espacios, será un empate y el final de la partida

Todo el mecanismo y las reglas lo escribiremos en ObjectScript, el mecanismo del jugador de la máquina se

escribirá en Python.

Vamos a trabajar

Controlaremos el tablero en un *global*, en el que cada fila estará en un nodo y cada columna en una pieza.

Nuestro primer método es iniciar el tablero, para que sea fácil iniciaré el global ya con los nodos (filas A, B y C) y con las 3 piezas:

```
/// Iniciar un juego nuevo
ClassMethod NewGame() As %Status
{
  Set sc = $$$OK
  Kill ^TicTacToe
  Set ^TicTacToe("A") = "^^"
  Set ^TicTacToe("B") = "^^"
  Set ^TicTacToe("C") = "^^"
  Return sc
}
```

en este momento crearemos un método para añadir las letras en los espacios vacíos, para esto cada jugador dará la ubicación del espacio en el tablero.

Cada fila una letra y cada columna un número, para poner la X en el medio, por ejemplo, pasamos B2 y la letra X al método.

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

Vamos a comprobar si la coordinación es válida, la forma más simple que veo es usando una expresión 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()
}
```

Necesitamos garantizar que el espacio seleccionado está vacío.

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

¡Muy bien!

Ahora comprobamos si algún jugador ganó el set o si el juego ya está terminado, para esto creemos el método

CheckGameResult.

Primero verificamos si hubo algún ganador completando por la horizontal, usaremos una lista con las filas y un simple $ Find resuelve

```
    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"
      }
    }
```

Con otro *For* comprobamos la 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 comprobar la 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 fin, comprobamos si hubo un 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"
```

¡Genial!

Es hora de construir la máquina

Vamos a crear a nuestro oponente, necesitamos crear un algoritmo capaz de calcular todos los movimientos disponibles y usar una métrica para saber cuál es el mejor movimiento.

Lo ideal es utilizar un algoritmo de decisión llamado MiniMax (Wikipedia: MiniMax)

El algoritmo MiniMax es una regla de decisión utilizada en teoría de juegos, teoría de decisiones e inteligencia artificial.

Básicamente, necesitamos saber cómo jugar asumiendo cuáles serán los posibles movimientos del oponente y coger el mejor escenario posible.

En detalle, tomamos la escena actual y comprobamos de forma recurrente el resultado del movimiento de cada jugador. En caso de que la máquina gane el juego, puntuamos con +1, en caso de que pierda, puntuamos con -1 y con 0 si empata.

Si no es el final del juego, abrimos otro árbol con el estado actual del juego. Después de eso, encontramos la jugada con el valor máximo para la máquina y el mínimo para el oponente.

Mira el siguiente gráfico - hay 3 movimientos disponibles: B2, C1 y C3.

Al elegir C1 o C3, el oponente tiene la oportunidad de ganar en el siguiente turno, pero si elige B2 no importa el movimiento que elija el oponente, la máquina gana la partida.

Es como tener la gema del tiempo en nuestras manos e intentar encontrar la mejor línea de tiempo.

Convirtiendo a 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
}
```

Primero, convierto el *global* en una matriz simple, ignorando columnas y filas, dejándolo plano para facilitar.

En cada movimiento analizado llamamos al método copyGameState que, como su nombre indica, copia el estado del juego en ese momento, donde aplicamos el MiniMax.

El método getBestMove que se llamará repetidamente hasta que finalice el juego encontrando un ganador o un empate.

Primero se mapean los espacios vacíos y verificamos el resultado de cada movimiento, cambiando entre los jugadores.

Los resultados se almacenan en *move* ['puntuación'] para, después de comprobar todas las posibilidades, encontrar el mejor movimiento.

¡Espero que os haya divertido! Es posible mejorar la inteligencia usando algoritmos como Alpha-Beta Pruning (Wikipedia: AlphaBeta Pruning) o redes neuronales. ¡Solo tened cuidado de no darle vida a Skynet!

No dudeis en dejar cualquier comentario o pregunta.

¡Esto es todo, amigos!

Código fuente completo:
InterSystems Iris version 2021.1.0PYTHON

#Early Access Program (EAP) #Inteligencia Artificial #Mejores prácticas #Python #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/%C2%A1luchemos-contra-las-m%C3%A1quinas