



# **PMR3201 Computação para Automação**

## **Aula de Laboratório 7**

PyQt: GridLayout, Jogo da Velha

---

Newton Maruyama  
Thiago de Castro Martins  
Marcos S. G. Tsuzuki  
Rafael Traldi Moura  
20 de maio de 2019

PMR-EPUSP

1. O Jogo da Velha
2. Interface gráfica
3. Para você fazer

## O Jogo da Velha

---

# Interface para o Jogo da Velha

- ▶ Em nossa última aula de laboratório será apresentado uma implementação do Jogo da Velha utilizando uma interface gráfica (Veja figura abaixo).
- ▶ Inicialmente apresenta-se o modo de operação da interface.



- ▶ Programa se encontra no arquivo `ControleJogoDavelha.py`.

# Modo de operação

- ▶ No início, o programa se encontra em estado de **RESET** (primeira figura do lado esquerdo).
- ▶ O programa se inicia ao se pressionar o botão **START**.
- ▶ O jogador 'X' sempre inicia a partida (*label* fica verde). A sua jogada é indicada na janela de texto.
- ▶ o usuário (jogador 'O') deve executar a jogada do jogador 'X' pressionando o botão adequado.
- ▶ Após a jogada de 'X' ser executada o jogador 'O' deve jogar (*label* correspondente fica verde).

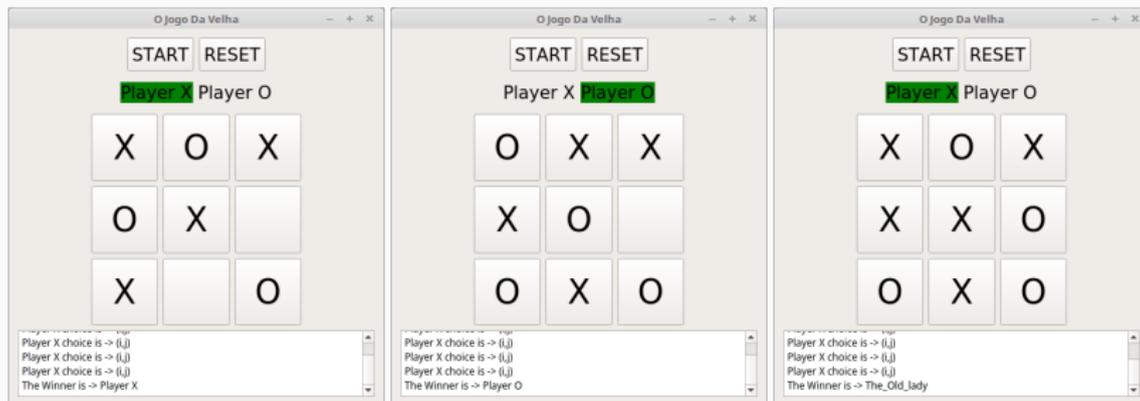


# Modo de operação

- ▶ O jogador 'O' executa sua jogada (primeira figura do lado esquerdo) e novamente passa ser a vez do jogador 'X' que indica sua jogada na janela de texto.
- ▶ Omite-se uma jogada.
- ▶ Eventualmente, algum jogador pode vencer a partida como indicado na última figura da esquerda para direita.



- ▶ Três possíveis finais de jogo são ilustrados nas figuras abaixo:



## **Interface gráfica**

---

- ▶ Duas classes são utilizadas no programa:
  - ▶ TicTacToe: controla a representação do jogo em um arranjo  $A[][]$  de dimensão  $3 \times 3$ .

```
self.A = [['b', 'b', 'b'], ['b', 'b', 'b'], ['b', 'b', 'b']]
```

- ▶ Janela: realiza a construção da interface gráfica e controle do jogo.

## Classe TicTacToe: Estrutura de dados

▶ `self.A = [['b', 'b', 'b'], ['b', 'b', 'b'], ['b', 'b', 'b']]`

Matriz que representa o jogo. Pode conter os valores: 'b', 'X', 'O'.

▶ `self.CurrentPlayer = 'X' # Player 'X' always starts the game`

Indica qual o jogador corrente.

▶ `self.N = 0`

Número de jogadas realizadas.

# Classe TicTacToe: funções

- Uma breve explanação sobre as funções dessa classe são ilustradas a seguir:

TicTacToe

**Reset(self):** executa um reset na estrutura de dados.

**CellAlreadyTaken(self,i,j):** verifica se a célula A[i][j] já está ocupada.

**Play(self,ID,i,j):** insere ID na célula A[i][j].

**ChangePlayer(self):** troca a identidade do jogador corrente.

**CheckPlayerWins(self,ID):** checa se o jogador ID ganhou o jogo.

**CheckBlank(self):** checa se ainda existe alguma posição não ocupada.

**GameEnds(self):** checa se o jogo terminou.

**GameState(self):** checa o estado do jogo.

**SendMessage(self,message):** imprime a mensagem no console.

**PrintA(self):** Imprime a matriz A[][] no console.

# Class TicTacToe: código

```
class TicTacToe():
    def __init__(self):
        self.Reset()

    def Reset(self):
        self.A = [['b','b','b'],['b','b','b'],['b','b','b']]
        self.CurrentPlayer = 'X' # Player 'X' always starts the game
        self.N = 0

    #
    # Check if the cell (i,j) is already taken by a player
    def CellAlreadyTaken(self,i,j):
        if self.A[i][j] == 'b':
            return(False)
        else:
            return(True)

    #
    # Player movement is recorded in Matrix A
    def Play(self,ID,i,j):
        self.A[i][j] = ID

    #
    # Change which player will play next
    def ChangePlayer(self):
        if self.CurrentPlayer == 'X':
            self.CurrentPlayer = 'O'
        else:
            self.CurrentPlayer = 'X'
```

# Class TicTacToe: código

```
#
# Check if Player ID='X','O' has won
def CheckPlayerWins(self, ID):
    A = self.A
    # check lines, columns and diagonals
    if ((A[0][0] == ID and (A[0][1] == ID and (A[0][2] == ID)) or \
        ((A[1][0] == ID and (A[1][1] == ID and (A[1][2] == ID)) or \
        ((A[2][0] == ID and (A[2][1] == ID and (A[2][2] == ID)) or \
        ((A[0][0] == ID and (A[1][0] == ID and (A[2][0] == ID)) or \
        ((A[0][1] == ID and (A[1][1] == ID and (A[2][1] == ID)) or \
        ((A[0][2] == ID and (A[1][2] == ID and (A[2][2] == ID)) or \
        ((A[0][0] == ID and (A[1][1] == ID and (A[2][2] == ID)) or \
        ((A[0][2] == ID and (A[1][1] == ID and (A[2][0] == ID))):
        return(True)
    else:
        return(False)

#
# Check if matrix A still have blank spaces, i.e.,
# 'b'
def CheckBlank(self):
    A = self.A
    if (A[0][0] == 'b' or (A[0][1] == 'b' or (A[0][2] == 'b') or \
        (A[1][0] == 'b') or (A[1][1] == 'b') or (A[1][2] == 'b') or \
        (A[2][0] == 'b') or (A[2][1] == 'b') or (A[2][2] == 'b')):
        return(True)
    else:
        return(False)
```

# Class TicTacToe: código

```
#
# Check if the game ended
def GameEnds(self):
    if self.CheckPlayerWins('X'):
        return(True)
    elif self.CheckPlayerWins('O'):
        return(True)
    elif not self.CheckBlank():
        return(True)
    else:
        return(False)

#
# Checks the state of the game
def GameState(self):
    if self.CheckPlayerWins('X'):
        return(True, 'Player X')
    elif self.CheckPlayerWins('O'):
        return(True, 'Player O')
    elif not self.CheckBlank():
        return(True, 'The_Old_lady')
    else:
        return(False, 'Still_Game')

# Imprime a mensagem no console
def SendMessage(self, message):
    print(message)

# Imprime a matrix A[][] no console
def PrintA(self):
    A = self.A
    print('Jogada =', self.N)
    for i in range(3):
        print(A[i][0], ' ', A[i][1], ' ', A[i][2])
    print()
```

# Classe Janela: funções

- ▶ Uma breve explanação sobre as funções dessa classe são ilustradas a seguir:

Janela

**PlayerXMove(self):** decide qual a jogada do jogador 'X'.

**Controle(self,i,j,pressedbutton):** Controle principal das ações do jogo.

**press\_button\_reset(self):** ações realizadas quando o botão RESET é pressionado.

**press\_button\_start(self):** ações realizadas quando o botão START é pressionado.

**press\_button(self,i,j,pressed):** ações realizadas quando um botão do grid é pressionado.

**WriteMessage(self,message):** Escreve uma mensagem na área de texto.

**SetPar(self, x):** parâmetros que configuram os botões.

**SetButtonGrid(self, button, ID):** escreve a string ID no botão do grid.

**ResetAllGridButtons(self):** reseta todos os botões do gridlayout, i.e., escreve ''.

**initUI(self):** constroe a interface gráfica.

# Class Janela: código

```
class Janela(QWidget):
    def __init__(self):
        super(Janela, self).__init__()

        # Variaveis de status do Jogo
        self.Reset = True # You can reset it at any time
        self.GameOn = False # sinalizes that game is not finished
        # The state of the game:
        self.Jogo = TicTacToe()
        # Operacoes de criacao da interface grafica
        self.initUI()

    # Here you must declare your algorithm for Player 'X'
    def PlayerXMove(self):
        self.WriteMessage('Player X choice is -> (i,j)')
```

# Class Janela: código

```
#
# The control of the game !
#
def Controle(self,i,j,presedbutton):
    if not self.Jogo.CellAlreadyTaken(i,j):
        # Player ID selects cell (i,j)
        # Change matrix A
        self.Jogo.Play(self.Jogo.CurrentPlayer,i,j)
        self.Jogo.N = self.Jogo.N + 1
        #
        #
        self.SetButtonGrid(presedbutton,self.Jogo.CurrentPlayer)

    if not self.Jogo.GameEnds(): # The game ended ?
        self.Jogo.ChangePlayer() # Change CurrentPlayer to the Opposite
        if self.Jogo.CurrentPlayer == 'X':
            # change label colors
            self.lbl1.setStyleSheet('background: green')
            self.lbl2.setStyleSheet('background: transparent')
            # Player X chooses which cell(i,j) to play
            self.PlayerXMove()
        else: # self.Jogo.CurrentPlayer = 'O'
            self.lbl1.setStyleSheet('background: transparent')
            self.lbl2.setStyleSheet('background: green')
    else: # The game ended
        endstate = self.Jogo.GameState()
        thewinner = endstate[1] # String of the winner
        self.WriteMessage('The Winner is -> '+ thewinner)
        self.GameOn = False
    else: # the cell (i,j) is taken ! choose another one empty !
        self.WriteMessage('Cell is already taken !')
    self.Jogo.PrintA()
```

# Class Janela: código

```
#
# Actions for the reset button
def press_button_reset(self):
    print('Reset')
    # Reset the Grafical Interface
    self.ResetAllGridButtons()
    self.lbl1.setStyleSheet('background: transparent')
    self.lbl2.setStyleSheet('background: transparent')
    self.textEdit.clear()
    # Reset the actual game, creates a new game
    self.Jogo = TicTacToe()
    self.Reset = True
    self.GameOn = False

#
# Action for the start button
def press_button_start(self):
    if self.Reset == True:
        # set flags
        self.Reset = False
        self.GameOn = True
        # Player X must play
        self.lbl1.setStyleSheet('background: green')
        self.textEdit.append(str('Start - Player X'))
        self.PlayerXMove()
    else: # Reset == False
        self.WriteMessage('Please Reset First !')
```

# Class Janela: código

```
# Action for the buttons of the grid layout
def press_button(self,i,j,pressed):
    # which button of the grid has been pressed
    print('i =',i, ' j =',j)
    if self.GameOn == True:
        # Lets transfer to other function
        self.Controle(i,j,pressed)
    else:
        self.WriteMessage('Game is over ! Please Reset !')
# Write messages in the text area
def WriteMessage(self,message):
    self.textEdit.append(message)

# Set parameters of widgets: button size, font type and size
def SetPar(self, x):
    x.setFixedSize(80,80)
    x.setFont(QFont('SansSerif', 30))

# Set what text goes on button: ' ', 'X' or 'O'
def SetButtonGrid(self, button, ID):
    button.setText(ID)

# Make all buttons of the grid blank text
def ResetAllGridButtons(self):
    for i in range(9):
        self.listb[i].setText(' ')
```

# Class Janela: código

```
def initUI(self):
    StartButton = QPushButton('START')
    StartButton.setFixedSize(80,40)
    StartButton.setFont(QFont('SansSerif', 16))
    StartButton.clicked.connect((lambda: self.press_button_start()))

    ResetButton = QPushButton('RESET')
    ResetButton.setFixedSize(80,40)
    ResetButton.setFont(QFont('SansSerif', 16))
    ResetButton.clicked.connect((lambda: self.press_button_reset()))
    box1 = QHBoxLayout()
    box1.addWidget(StartButton)
    box1.addWidget(ResetButton)
    box1.setAlignment(QtCore.Qt.AlignCenter)

    self.lbl1 = QLabel('Player X')
    self.lbl1.setFont(QFont('SansSerif',16))
    self.lbl2 = QLabel('Player O')
    self.lbl2.setFont(QFont('SansSerif',16))
    hbox1 = QHBoxLayout()
    hbox1.addWidget(self.lbl1)
    hbox1.addWidget(self.lbl2)
    hbox1.setAlignment(QtCore.Qt.AlignCenter)
```

# Class Janela: código

```
grid = QGridLayout()
button1 = QPushButton("")
self.SetPar(button1)
button1.clicked.connect(lambda: self.press_button(0,0,button1))
button2 = QPushButton("")
self.SetPar(button2)
button2.clicked.connect(lambda: self.press_button(0,1,button2))
button3 = QPushButton("")
self.SetPar(button3)
button3.clicked.connect(lambda: self.press_button(0,2,button3))
button4 = QPushButton("")
self.SetPar(button4)
button4.clicked.connect(lambda: self.press_button(1,0,button4))
button5 = QPushButton("")
self.SetPar(button5)
button5.clicked.connect(lambda: self.press_button(1,1,button5))
button6 = QPushButton("")
self.SetPar(button6)
button6.clicked.connect(lambda: self.press_button(1,2,button6))
button7 = QPushButton("")
self.SetPar(button7)
button7.clicked.connect(lambda: self.press_button(2,0,button7))
button8 = QPushButton("")
self.SetPar(button8)
button8.clicked.connect(lambda: self.press_button(2,1,button8))
button9 = QPushButton("")
self.SetPar(button9)
button9.clicked.connect(lambda: self.press_button(2,2,button9))
```

# Classe Janela: código

```
grid.addWidget(button1, 0,0)
grid.addWidget(button2, 0,1)
grid.addWidget(button3, 0,2)
grid.addWidget(button4, 1,0)
grid.addWidget(button5, 1,1)
grid.addWidget(button6, 1,2)
grid.addWidget(button7, 2,0)
grid.addWidget(button8, 2,1)
grid.addWidget(button9, 2,2)
grid.setAlignment(QtCore.Qt.AlignCenter)

textArea = QHBoxLayout()
self.textEdit = QTextEdit()
textArea.addWidget(self.textEdit)

mainbox = QVBoxLayout()
mainbox.addLayout(box1,10)
mainbox.addLayout(hbox1,10)
mainbox.addLayout(grid,60)
mainbox.addLayout(textArea,20)

self.setLayout(mainbox)

self.setGeometry(300, 300, 450, 450)
self.setWindowTitle('O Jogo Da Velha')
self.show()
```

# Construção da interface gráfica

- ▶ Como já visto na aula anterior podemos inserir *containers* dentro de *containers*.
- ▶ A interface vista abaixo contém quatro elementos básicos:
  - ▶ Um *container* QHBoxLayout que contém os botões START e RESET.
  - ▶ Um outro *container* QHBoxLayout que contém dois *labels* 'PLAYER X' e 'PLAYER O'.
  - ▶ Um terceiro *container* do tipo QGridLayout contém nove botões formando um tabuleiro de  $3 \times 3$ .
  - ▶ Um quarto *container* do tipo QVBoxLayout contém um *Widget* do tipo QTextEdit.
- ▶ Esses quatro elementos descritos acima são colocados dentro de um *container* do tipo QVBoxLayout.



# Construção da interface gráfica

- ▶ Os *widgets*, botões, *labels*, etc., podem ter suas dimensões especificadas além de outros atributos como o tipo da fonte, cor da fonte, etc.
- ▶ Por exemplo, o botão START é especificado como:

```
StartButton.setFixedSize(80,40)  
StartButton.setFont(QFont('SansSerif', 16))
```

- ▶ Para cada botão é associado uma ação ou uma função que deve ser executada caso o botão seja pressionado.
- ▶ Para o botão START por exemplo a ação é especificada da seguinte forma:

```
StartButton.clicked.connect((lambda: self.press_button_start()))
```

- ▶ Para o botão START por exemplo a ação é especificada da seguinte forma:

```
ResetButton.clicked.connect((lambda: self.press_button_reset()))
```

## Construção da interface gráfica

- ▶ Os botões que representam o tabuleiro do Jogo da Velha são inseridos num *container* do tipo *grid*

```
grid = QGridLayout()
```

- ▶ Os botões são inseridos no *container* da seguinte forma:

```
grid.addWidget(button1, 0,0)  
grid.addWidget(button2, 0,1)  
grid.addWidget(button3, 0,2)  
grid.addWidget(button4, 1,0)  
grid.addWidget(button5, 1,1)  
grid.addWidget(button6, 1,2)  
grid.addWidget(button7, 2,0)  
grid.addWidget(button8, 2,1)  
grid.addWidget(button9, 2,2)
```

- ▶ O *container* principal do tipo vertical é denominado `mainwindow`.
- ▶ Os outros *containers* são inseridos em `mainwindow` como ilustrado a seguir:

```
mainwindow = QVBoxLayout()  
mainwindow.addLayout(box1, 10)  
mainwindow.addLayout(hbox1, 10)  
mainwindow.addLayout(grid, 60)  
mainwindow.addLayout(textArea, 20)
```

- ▶ Note que os números 10, 10, 60 e 20 representam os percentuais que cada *container* ocupa da janela.

- ▶ Os botões do *grid* estão associados à função `press_button()`
- ▶ A função verifica se o jogo está terminado ou não, se estiver terminado ignora este evento.
- ▶ Caso o jogo esteja ativo passa a execução para a função `Controle()`.

```
# Action for the buttons of the grid layout
def press_button(self, i, j, pressed):
    # which button of the grid has been pressed
    print('i =', i, ' j =', j)
    if self.GameOn == True:
        # Lets transfer to other function
        self.Controle(i, j, pressed)
    else:
        self.WriteMessage('Game is over ! Please Reset !')
```

- ▶ A função `Controle()` realiza as principais funções de controle da interface e do estado do jogo.

**Para você fazer**

---

- ▶ Modifique a função `PlayerXMove()` inserindo um código que realize as jogadas do jogador 'X'.

- ▶ A função PlayerXMove() do arquivo ControlejogoDaVelhaV3.py pode utilizar duas implementações de jogadores que são implementadas através da classe Players(): NaivePlayerP() e SmartPlayer().

```
def PlayerXMove(self):
    B = list(map(list, self.Jogo.A)) # realiza uma copia do conteudo da matriz A
                                     # na matriz
    # Voce pode escolher entre um jogador Naive e um Jogador Smart
    #
    # O jogador Naive joga aleatoriamente
    (i,j) = self.play.NaivePlayerP(B)
    #
    # O jogador Smart joga realizando simulacoes de jogadas possiveis
    # escolhendo um caminho que pode levar a vitoria no menor numero de
    # jogadas
    # (i,j) = self.play.SmartPlayerP(B)
    m = '(' + str(i) + ',' + str(j) + ')'
    self.WriteMessage('Player X choice is -> (i,j)=' + m)
```

► Resumo da classe `Players`

Players

**`ChangePlayerP(self, ID)`**: troca o jogador corrente.

**`CheckBlankP(self, A)`**: checa se existem posições ainda não jogadas.

**`CheckPlayerWinsP(self, A, ID)`**: checa se o jogador identificado por ID ganhou o jogo.

**`GameEndsP(self, A)`**: checa se o jogo atingiu seu final.

**`NaivePlayerP(self, A)`**: algoritmo que implementa um jogador que decide as posições aleatoriamente.

**`BestXMove(self, A, level, LastPlayer)`**: Função recursiva do algoritmo Minimax.

**`SmartPlayerP(self, A)`**: implementação de um jogador 'X' que utiliza uma versão de algoritmo Minimax.

- ▶ A função NaivePlayerP apresenta um algoritmo que escolhe aleatoriamente uma posição vazia.

```
def NaivePlayerP(self,A):
    listblank=[]          # armazena posicoes (i,j) que contem 'b'
                        # ou seja posicoes vazias ou nao utilizadas
    for i in range(3):  # varre a matrix A
        for j in range(3):
            if A[i][j] == 'b': # descobre posicoes vazias
                listblank.append((i,j)) # coloca a posicao vazia (i,j)
                                       # na lista
    nblank = len(listblank) # numero de posicoes vazias
    if nblank > 0:
        index = random.randint(0,nblank-1) # escolhe aleatoriamente uma
                                           # posicao da lista
    return(listblank[index]) # retorna (i,j) escolhido
```

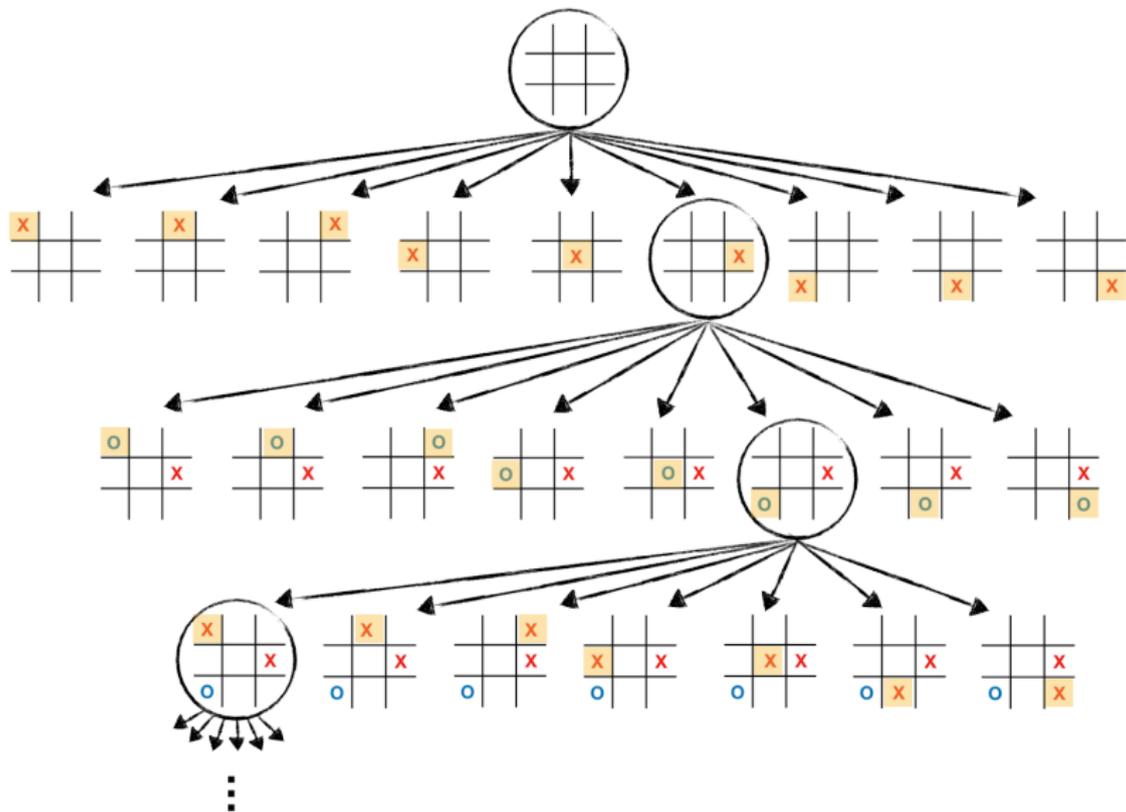
- ▶ Uma abordagem ingênua sobre o jogo seria considerar que existem 9 posições no tabuleiro que podem ser preenchidas pelos símbolos 'X', 'O' e 'b' (blank) o que resultaria em  $19.683 (= 3^9)$  estados do tabuleiro e  $362.880 (= 9!)$  possíveis jogos diferentes.
- ▶ Entretanto uma análise mais cuidadosa deve considerar os seguintes aspectos:
  - ▶ O jogo termina assim que um jogador consegue estabelecer uma sequência de 3 símbolos consecutivos,
  - ▶ Se o jogador 'X' começa o jogo o número de símbolos 'X' é sempre maior ou igual o número de símbolos 'O'.
- ▶ Se forem considerados ainda simetrias (rotações e reflexões) somente existem 138 estados terminais do jogo. Assumindo que o jogador 'X' sempre inicia o jogo, obtém-se:
  - ▶ 91 posições distintas com vitória do jogador 'X'
  - ▶ 44 posições distintas com vitória do jogador 'O'
  - ▶ 3 posições distintas com empate.

## Uma comparação com outros jogos

- ▶ Existem estimativas do número de jogos possíveis para um jogo de xadrez da ordem de  $1.0 \times 10^{37}$  o que torna bastante difícil o cálculo da melhor jogada em tempo real.
- ▶ Em 1996 o supercomputador da IBM Deep Blue com 256 processadores projetado especialmente para jogar xadrez e capaz de analisar 200 milhões de posições por segundo realizou um confronto com Gary Kasparov. Placar: Kasparov 3 vitórias, 2 empates e 1 derrota. Um novo confronto em 1997 resultou em: Deep Blue 2 vitórias, 3 empates e 1 derrota.
- ▶ Para um jogo de GO é estimado um número de jogos em torno de  $1.74 \times 10^{172}$ .

- ▶ Na teoria dos jogos a Árvore de Jogo é uma estrutura do tipo grafo direcionado que representa o jogo.
- ▶ Os nós (*nodes*) representam estados do jogo enquanto que os arcos (*arcs*) representam ações.
- ▶ Uma árvore de jogo completa contém todos os possíveis 'jogos' (todos os possíveis caminhos ou sequências de ações e estados) que podem ocorrer.
- ▶ Os nós-terminais (*leaf-nodes*) representam um estado correspondente ao final de um jogo, ou seja, vitória de um dos jogadores ou empate.
- ▶ Uma árvore de jogo pode ser interpretada como uma simulação que gera todos os possíveis jogos.
- ▶ Partindo-se de um determinado estado, a utilização de uma Árvore de Jogo permite estabelecer uma ação (jogada) ótima para um determinado jogador analisando-se idealmente todos os possíveis caminhos a serem percorridos.

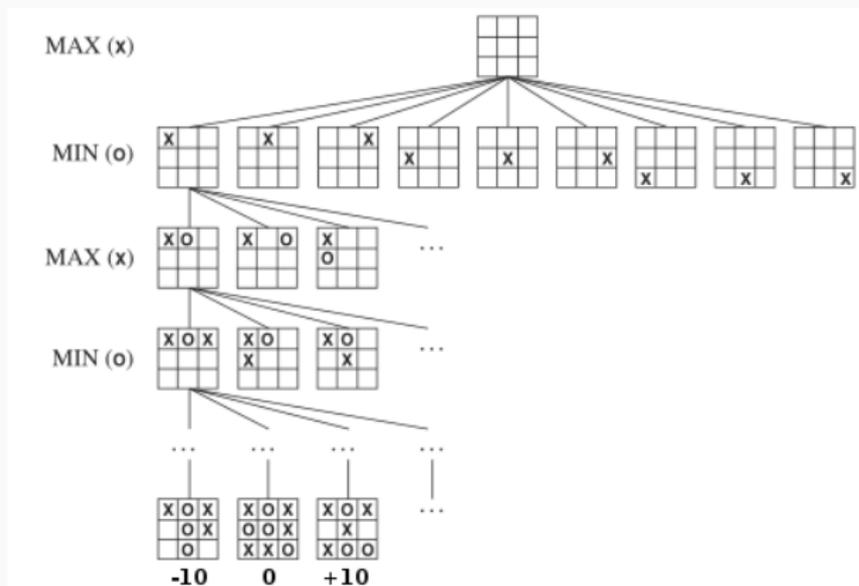
# Árvore de jogo parcial do Jogo da Velha



- ▶ Na Teoria dos Jogos, o algoritmo minimax calcula ações que possam minimizar a perda máxima de um jogador. O cálculo da melhor jogada requer a construção de uma árvore de jogo.
- ▶ Por exemplo, no Jogo da Velha existem dois jogadores que alternam jogadas. O jogador 1 será identificado aqui pelo símbolo 'X' que utiliza no tabuleiro e o jogador 2 será identificado pelo símbolo 'O'.
- ▶ Por hipótese vamos considerar que o jogador 'X' sempre inicia o jogo. O algoritmo é executado dentro da perspectiva de que 'X' deve minimizar a perda máxima.
- ▶ Obviamente a perspectiva do jogador 'O' é de estabelecer jogadas para maximizar a perda máxima do outro jogador.
- ▶ O algoritmo proposto não utiliza uma árvore explicitamente mas sim uma função recursiva onde cada instancia da função representa um possível estado do jogo.

# Algoritmo Minimax

- ▶ O algoritmo deve atribuir uma pontuação positiva, por exemplo +10, para todo o jogo em que 'X' é vencedor, e atribuir a mesma pontuação mas negativa, -10, para todo o jogo que 'O' for vencedor. Obviamente, todos os jogos que resultam em empate recebem a pontuação nula 0.
- ▶ A figura abaixo ilustra um trecho de uma árvore do Jogo da Velha.



## Solução proposta

- ▶ A função SmartPlayerP() decide a jogada do jogador 'X' baseado no algoritmo Minimax.
- ▶ Inicialmente descobre as posições ainda não jogadas da matriz A e depois executa a função BestXMove() para cada uma das jogadas possíveis.
- ▶ A função BestXMove() devolve um valor numérico que avalia cada uma das jogadas possíveis.
- ▶ Escolhe-se a opção com maior valor numérico.

```
def SmartPlayerP(self,A): # Decide a jogada de 'X'
                           # baseado no algoritmo minimax
    listblank=[]
    for i in range(3): # varre a matrix A procurando posicoes vazias
        for j in range(3):
            if A[i][j] == 'b':          # descobre posicoes vazias
                listblank.append((i,j)) # coloca a posicao vazia (i,j)
                                         # na lista
    nblank = len(listblank) # numero de posicoes vazias
    value = []
    for k in range(nblank): # realiza todas as jogadas possiveis
        (i,j) = listblank[k]
        A[i][j] = 'X'
        value.append(self.BestXMove(A,1,'X')) # value[] contem o valor de cada
                                                # uma das jogadas
    index = value.index(max(value)) # descobre o indice de listblank
                                     # que contem a melhor jogada (i,j)
    return(listblank[index])
```

# Solução proposta

- ▶ A função BestXMove também gera todas as jogadas possíveis considerando o jogador adequado: 'X' ou 'O'.
- ▶ A função realiza recursões até atingir um final de jogo, atribuindo os valores +10-level se o ganhador for o jogador 'X', -10 se o ganhador for o jogador 'O' e o caso seja um empate.
- ▶ a variável level indica o nível de profundidade da recursão. Assim as jogadas que levam a uma vitória mais rápida são favorecidas.

```
def BestXMove(self,A,level,LastPlayer): # Algoritmo recursivo que
                                         # simula jogadas e atribui pontos
                                         # dependendo do jogador vencedor

    if not self.GameEndsP(A): # se o jogo nao terminou
        listblank=[]
        for i in range(3): # varre a matrix A
            for j in range(3):
                if A[i][j] == 'b': # descobre posicoes vazias
                    listblank.append((i,j)) # coloca a posicao vazia (i,j)
                                           # na lista

        nblank = len(listblank) # numero de posicoes vazias
        value = []
        for k in range(nblank):
            (i,j)=listblank[k]
            CurrentPlayer = self.ChangePlayerP(LastPlayer)
            A[i][j] = CurrentPlayer
            value[k]=self.BestXMove(A,level+1,CurrentPlayer)
        return(max(value))
    else: # O jogo terminou mas quem ganhou ?
        if self.CheckPlayerWinsP(A,'X'):
            return(10-level) # desconta-se na pontuacao de 'X' a profundidade
                              # da jogada
        elif self.CheckPlayerWinsP(A,'O'):
            return(-10) # se 'O' vence esse caminho nao pode ser escolhido
        else:
            return(0) # pontuacao para situacao de empate
```

## SmartPlayerP( ) é um algoritmo invencível ?

- ▶ O algoritmo SmartPlayerP( ) não é invencível.
- ▶ Uma das deficiências é que várias jogadas podem devolver o mesmo valor numérico e a função `max( )` escolhe a primeira posição encontrada com o maior valor.
- ▶ A escolha de caminho que possam conter maior número de vitórias possíveis também seria mais eficiente.