Universidade Federal de Santa Catarina
INE5430 - InteligÊncia Artificial
Prof. Elder Rizzon Santos
CiÊncia da Computa ª o
Trabalho sobre MÉtodos de busca
Willian de Souza – 11100901

1. Introdu ª o

Para este trabalho, foi usada a linguagem de programa ª o python para contruir a l gica do jogo. A Interface de acesso ao usuÁrio foi produzida de forma simple baseada no console [*Figura 1*].

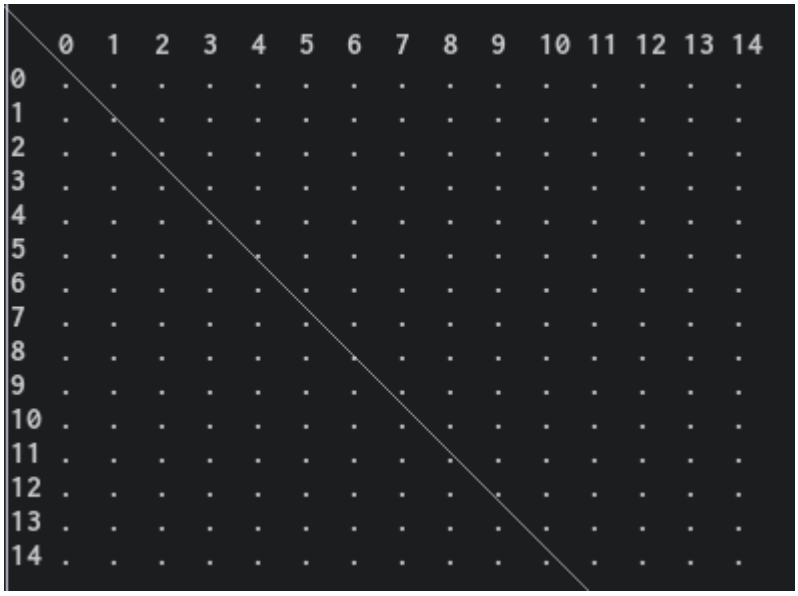

Figura 1. Tabuleiro do jogo

Como visto na *Figura 1,* a representa ª o do tabuleiro foi feita atravÉs de matriz, onde o "." representa um campo em branco, ou seja, um campo ainda nª o consumido pelo jogo. O jogo conta com duas representa ı es de pe as, pe a *preta* representada pela letra "x" e a pe a *branca* representada pela letra "o". O fonte da representa ª o segue abaixo.

```
class Piece:
    """Piece representation
    """
    """Indicate the position in board does not played yet"""
    NONE = '.'
    """Indicate a BLACK piece for game"""
    BLACK = 'x'
    """Indicate a WHITE piece for game"""
    WHITE = 'o'
```

A entrada dos dados para o jogo Øno formato “linha, coluna” como segue na *Figura 2*, onde mostra tambØm as posi 1 es jogadas pelo jogador 1 e jogador 2.

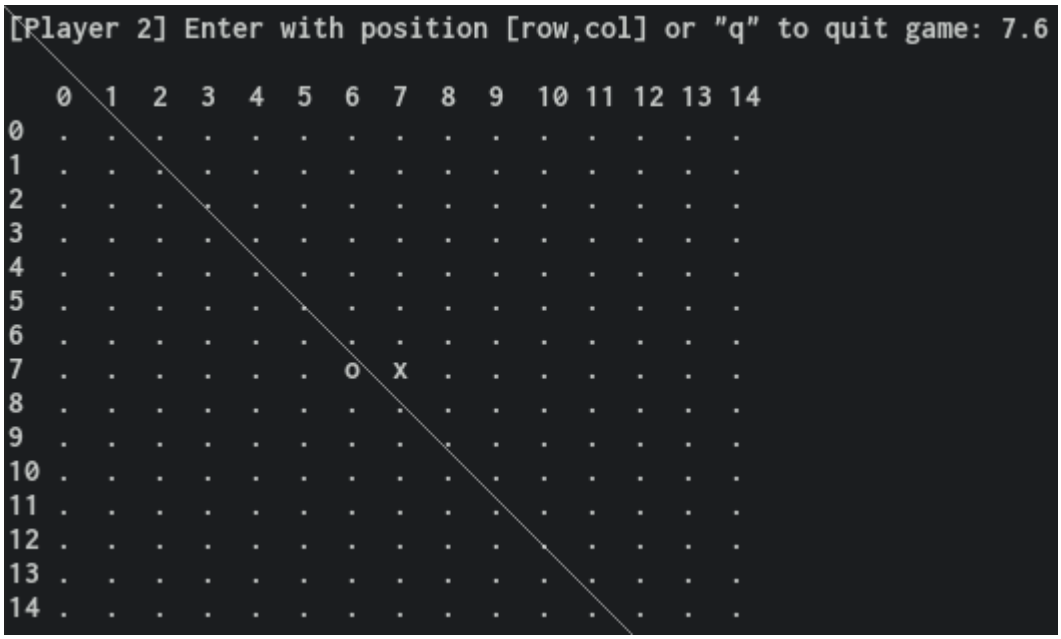

Figura 2. Estado do tabuleiro ap a jogada do Player 1 seguida pelo Player 2, na posi ª o [7,6]

2. Implementa ª o

2.1. Player

Para representar os Players (Jogadores), foi usada uma representa ª o abstrata chamada *AIPlayer* como segue:

```
class AIPlayer:
    """ Abstract AI players.
        To construct an AI player:
        Construct an instance (of its subclass) with the game Board
    """
    def __init__(self, board, piece):
        self._board = board
        self.my_piece = piece
        self.opponent = Piece.WHITE if piece == Piece.BLACK else Piece.BLACK
    def play(self, row, col):
        raise NotImplemented
```

2.2. Human

Humano Øum tipo de player, essa Øa representa ª o mais simples, ela Ømanipulada atravØs das entradas de uma pessoa pelo console como visualizado na *Figura 2*.

Um Humano herda de *AIPlayer* define e sobreesqueve a fun ª o play, que recebe a linha e a coluna e insere a pe a correspondente ao jogador no tabuleiro. Essa fun ª o tambØm ØresponsÆvel por gerenciar se a joagada pode ser efetuada ou n ª o. De maneira simples pode exemplicar da seguinte

forma, se um *jogador 1* jogou sua pe a "*x*" na linha 7 e coluna 7 e o *jogador 2* tentar jogar sua pe a "*o*" na mesma posi a o, temos uma situa a o de conflito e a regra do jogo diz que n a o podemos sobrescrever uma jogada colocando outra pe a na mesma posi a o em que uma pe a j se encontra, dessa forma lna ada uma *exception* (exe a o), para ser tratada no n vel de aplica a o.

```
class HumanPlayer(AIPlayer):
    """ Human Player
    """
    def __init__(self, board, piece, first=True):
        super(HumanPlayer, self).__init__(board, piece)
        self.first = not first
    def play(self, row, col):
        if self._board.get_piece(row, col) != Piece.NONE:
            raise OverwritePositionException
        return self._board.play_piece(self.my_piece, row, col)
    def __repr__(self):
        player_number = int(self.first) + 1
        return 'Player {}'.format(player_number)
```

2.3. Border

Border neste momento talvez seja a principal implementa a o, ou seja, nela que se encontra atualmente a maior.

O tabuleiro recebe como par metros dois n meros, o primeiro "`size`" usado para construir a tabela, que uma matriz quadrada (size X size) e o segundo "`sequence_victory`" usado para determinar a sequ ncia de pe as iguais que constitui em uma vit ria.

Nesta classe temos a API necess ria para que outras classes possam manipular o tabuleiro com facilidade, dentra ela est a fun a o `play_piece` mostrada abaixo.

```
def play_piece(self, piece, row, col):
    self._table[row, col] = piece
    winner = self.has_winner(piece, row, col)
    if (self._table == '.').sum() == 0 and winner == Piece.NONE:
        raise NotBlankSpaceException
    return winner
```

Esta fun a o respons vel por gerenciar o que pode ocorrer durante uma jogada, ou seja, ap s a jogada verificado se possui algum vencedor na rodada atual, se possuir o mesmo retornado, por ela tamb m respons vel por verificar se o jogo terminou sem ganhador, atrav s da *exception* `NotBlankSpaceException` lan ada para o n vel de cima da aplica a o. Essa exception pode ser interpretada como uma caso de empate, onde acabam as casas (posi 1 es) poss veis para o jogo.

A maior implementa a o desta classe se d por conta de uma busca no tabuleiro, para encontrar um vitorioso, essa busca feita atrav s da fun a o `has_winner`, que faz a procura do vencedor pelas diferentes op 1 es que se pode alcan ar a vit ria.

```
def has_winner(self, piece, row, col):
    if self._search_line(piece, row, col):
        """ Faz verificação por linha
            se ocorrer uma vitória na linha atual
            escreve a peça ganhadora e retorna verdadeiro
```

```
            """
            return piece
        if self._search_column(piece, row, col):
            """ Faz verificação por linha
                se ocorrer uma vitória na linha atual
                escreve a peça ganhadora e retorna verdadeiro
            """
            return piece
        if self._search_diagonal(piece, row, col):
            """ Faz verificação da diagonal no sentido da diagonal principal
                se ocorrer uma vitória na diagonal atual
                escreve a peça ganhadora e retorna verdadeiro
            """
            return piece
        if self._search_opposite_diagonal(piece, row, col):
            """ Faz verificação da diagonal no sentido da diagonal secundária
                se ocorrer uma vitória na diagonal atual
                escreve a peça ganhadora e retorna verdadeiro
            """
            return piece
        return Piece.NONE
```

Essa fun ª o usa outras quatro fun 1 es que auxiliam na busca da vit ria `_search_line`, `_search_column`, `_search_diagonal` e `_search_opposite_diagonal`. E essas fun 1 es utilizam a fun ª o chamada `_victory_match`, que gera a forma vencedora para fazer a avali ª o das posi 1 es (ex. "*xxxxx*" que Øuma sequ Gncia vÆlida para a vit ria do jogador com a pe a "*x*") segue a implementa ª o.

```
def _victory_match(self, piece):
    """ A generator to match victory
    :param piece: for check the victory sequence
    :return: the sequence for victory
    """
    return piece * self._sequence_victory
```

As fun 1 es de busca da vit ria pela linha e pela coluna s ª o um pouco semelhantes. A busca por linha pega atravØs da posi ª o da jogada atual, cinco colunas anterior a atual e cinco colunas ap s, respeitante os limites do tabuleiro, transforma essa lista retornada em uma *string* e com o auxilio da fun ª o `_victory_match` verifica se a *string* necessÆria para garantir a vit ria estÆcontida na na *string* transformada. De formar semelhante a busca por coluna Øfeita, considerando que as movimenta 1 es para a verifica ª o do intervalo s ª o feitas nas linha e da mesma forma a lista retornada Øconvertida para *string* e comparada com a `match_str`.

```
def _search_line(self, piece, row, col):
    """ Search has victory in line
            Check in matrix row if has match value to victory
            it get the previous five column position from current position played and
            the next five column position from current and check if has victory
    :param piece: current piece played
    :param row: position in matrix
    :param col: position in matrix
    :return: if has a victory in current line, True
                otherwise is False
    """
    match_str = self._victory_match(piece)
    start_col = 0 if (col - self._sequence_victory) < 0 else (col - self._sequence_victory)
    size = len(self._table)
    end_col = size if (col + self._sequence_victory + 1) > size else (col + self._sequence_victory + 1)
```

```
    return match_str in ''.join(self._table[row, start_col:end_col])

def _search_column(self, piece, row, col):
    """ Search has victory in line
            Check in matrix column if has match value to victory
            it get the previous five row position from current position played and
            the next five row position from current and check if has victory
    :param piece: current piece played
    :param row: position in matrix
    :param col: position in matrix
    :return: if has a victory in current row, True
                otherwise is False
    """
    match_str = self._victory_match(piece)
    start_row = 0 if (row - self._sequence_victory) < 0 else (row - self._sequence_victory)
    size = len(self._table)
    end_row = size if (row + self._sequence_victory + 1) > size else (row + self._sequence_victory + 1)
    return match_str in ''.join(self._table[start_row:end_row, col])
```

A fun ª o de busca pela diagonal (que Øno mesmo sentido da diagonal principal), foi feita com a ajuda de uma fun ª o da biblioteca matemÆtica *numpy*. A ideia Øgerar offset que a diagonal tem em rela ª o diagonal principal e com a ajuda da fun ª o *diagonal* da biblioteca, pegamos a diagonal solicitada no formato de lista e da mesma forma convertemos para *string* e ent ª o comparamos com a `match_str`.

```
def _search_diagonal(self, piece, row, col):
    """ Search has victory by diagonal
            Check in matrix the diagonal if has match value to victory
            it get the previous five diagonal (row x col) position from current position played and
            the next five diagonal (row x col) position from current and check if has victory
    :param piece: current piece played
    :param row: position in matrix
    :param col: position in matrix
    :return: if has a victory in current diagonal, True
                otherwise is False
    """
    match_str = self._victory_match(piece)
    offset = col - row
    diagonal = np.diag(self._table, k=offset).tolist()
    diagonal_formatted = ''.join(diagonal)
    return match_str in diagonal_formatted
```

JÆna fun ª o de busca pela diagonal oposta, a ideia foi girar o tabuleiro 90 graus no sentido anti-horÆrio, fazer a conver ª o das posi ı es (de 15. 14. …. 1. 0 para 0. 1. …. 14. 15) que ficaram correspondentes s linhas. Essa altera ª o Øapenas no formato de representa ª o dos dados, n ª o afeta a tabela original e facilita pegarmos a diagonal oposta usando os metodos usados na captura da diagonal principal.

```
def _search_opposite_diagonal(self, piece, row, col):
    """ Search has victory by diagonal
            Check in matrix the diagonal if has match value to victory
            it get the previous five diagonal (row x col) position from current position played and
            the next five diagonal (row x col) position from current and check if has victory
    :param piece: current piece played
    :param row: position in matrix
    :param col: position in matrix
    :return: if has a victory in current diagonal, True
```

```
            otherwise is False
    """
    new_col = row
    size = len(self._table)
    new_row = (size - 1) - col
    match_str = self._victory_match(piece)
    offset = new_col - new_row
    column_inverted = self._table[:, ::-1]
    transposed = column_inverted.transpose()
    diagonal = np.diag(transposed, k=offset).tolist()
    diagonal_formatted = ''.join(diagonal)
    return match_str in diagonal_formatted
```

2.4. GomokuState

GomokuState Øa representa ªo dos estados do jogo, ele possui um tabuleiro, uma parametro chamado `level_tree` que indica o nivel da Ætvore, um estado pai (parent) e um conjunto de estados filhos (children). Possui tambØm algumas fun ıes de manipula ªo de conteœdo como `children` que pega o conjunto de nodos filho, `add_child` que adiciona um nodo ao conjunto de nodos filho, `add_children_states` que faz a uniªo do conjunto de nodos folha com o conjunto de nodos recebido por par metro e `is_leaf` que faz a verifica ªo de nodo folha.

```
class GomokuState:
    level_tree = 0
    def __init__(self, board, parent_state):
        self.level_tree += 1
        self.board = cp.copy(board)
        self.parent = parent_state
        self._children = set()
    @property
    def children(self):
        return self._children
    def add_child(self, child_state):
        self._children.add(child_state)
    def add_children_states(self, children_state):
        self._children.update(children_state)
    def is_leaf(self):
        return len(self.children) == 0
```

3. Fun ıes Heur sticas e de Utilidade

Para a fun ªo de Uitlidade serÆconstruida uma formula com base nas Heur sticas, ou seja, temos uma classe usada para separar o n veis de pontua ªo chamada *Score:*

```
class Score:
    SIZE = 10
    ONE = 1
    TWO = SIZE * ONE
    THREE = SIZE * TWO
    FOUR = SIZE * THREE
    FIVE = SIZE * FOUR
```

Como visto no c digo acima, a ideia Øatribuir pontos sequŒncias de pe as com a quantidade de casas decimais correspondente ao numero da sequŒncia.

Na minha Utilidade entra a seguinte formula: `0.95 ** nivel + soma_das_pecas_do_tabuleiro + heuristica`

Onde `nivel`, Øo n vel em que a Ævore se encontra ao termino do jogo, `soma_das_pecas_do_tabuleiro` Øa soma resultante das pe as do tabuleiro, levando em conta as pontas abertas. Para calculo das pontas abertas entra a seguinte formula: partir de duplas Øsomado o valor de 20% da sequŒncia, ou seja, se for uma trinca serÆ `1.2 * THREE`.

JÆna Heur tica serÆacrecido 30% do valor da sequencia por ponta aberta, segue a formula: `1.3 * sequencia`, para uma ponta aberta e `((1.3 ** 2) * sequencia)`, para duas pontas abertas.