Autonomous agent for the game Tetris

Inteligência Artificial 2021/2022

Bernardo Kaluza - 97521

Pedro Lima - 97860

Nuno Cunha - 98124

Diogo Gomes

Luís Seabra Lopes

Piece Detection

- Para detetarmos a peça atual (state['piece']), usamos um dicionário, onde as keys são as posições onde as peças dão spawn (sendo estas estáticas) e os itens a arrays das peças / rotações correspondentes
- É retornado a Shape(classe da peça) e o numero de rotações correspondente

```
    "[[2, 2], [3, 2], [4, 2], [5, 2]]" :  I,
    "[[4, 2], [5, 2], [4, 3], [4, 4]]" :  J,
    "[[4, 2], [4, 3], [4, 4], [5, 4]]" :  L,
    "[[3, 3], [4, 3], [3, 4], [4, 4]]" :  O,
    "[[4, 2], [4, 3], [5, 3], [5, 4]]" :  S,
    "[[4, 2], [4, 3], [5, 3], [4, 4]]" :  T,
    "[[4, 2], [3, 3], [4, 3], [3, 4]]" :  Z
}

rotacoes= {
    "[[2, 2], [3, 2], [4, 2], [5, 2]]" :  2,
    "[[4, 2], [5, 2], [4, 3], [4, 4]]" :  4,
    "[[4, 2], [4, 3], [4, 4], [5, 4]]" :  4,
    "[[3, 3], [4, 3], [3, 4], [4, 4]]" :  1,
    "[[4, 2], [4, 3], [5, 3], [5, 4]]" :  2,
    "[[4, 2], [4, 3], [5, 3], [4, 4]]" :  4,
    "[[4, 2], [3, 3], [4, 3], [3, 4]]" :  2
}

def identify_piece(piece):
    current= peca.get(piece, None)
    rotation= rotacoes.get(piece, None)
    if current and rotation is not None:
        return Shape(current), rotation
    return None
```

Simulation

- Nesta função é onde vamos calcular todas as posições para a peça atual e associar-lhe um score, a combinação de keys com maior score é depois retornado.

```
def simulation(piece, rotations, game, width, height):
    keys = []
    inputs = []
    score = -10000
    for i in range(0, rotations):
        for j in range(1, width - 1):
            new_inputs = []
            new_inputs, test_piece = generateinputs(piece,
            for position in test_piece.positions:
                game.append([position[0], position[1]])

            new_move = Move(game, height,width)
            new_score = new_move.get_score()

            for position in test_piece.positions:
                game.remove([position[0], position[1]])

            if new_score > score:
                score=new_score
                keys = new_inputs

        piece.rotate()
        inputs.append("w")

    return keys
```

Generateinputs

- Nesta função é onde é calculada a posição no game de uma copia da peça para a posição pretendida
- É retornado o as keys e a posição da nova peça

```
def generateinputs(piece, position, game,inputs, new_inputs, width, height):
    new_piece=deepcopy(piece)
    new_inputs.extend(inputs)

    while isvalid(new_piece, game, width, height):
        new_piece.translate(-1, 0)
        new_inputs.append("a")
    new_piece.translate(1, 0)

    if "a" in new_inputs: new_inputs.remove("a")

    for i in range(1,position):
        if isvalid(new_piece, game, width, height):
            new_piece.translate(1, 0)
            if "a" in new_inputs : new_inputs.remove("a")
            else: new_inputs.append("d")
        else:
            new_piece.translate(-1, 0)
            if "d" in new_inputs : new_inputs.remove("d")
            else: new_inputs.append("a")
            break

    new_inputs.append("s")

    while isvalid(new_piece, game,width,height):
            new_piece.translate(0, 1)
    new_piece.translate(0, -1)

    return (new_inputs, new_piece)
```

Classe Move

- Nesta classe é onde são calculados os scores para cada posição da peça no game
- O score é dado consoante o estado atual do jogo e os pesos que nós atribuímos a cada característica do jogo
- É retornado o score calculado

```python
class Move:
    def __init__(self, game, game_height, game_width) -> None:
        self.game = game
        self.game_height = game_height
        self.game_width = game_width
```

```python
def get_score(self):

    lines = self.clear_rows()
    heights=self.get_all_columns_height()

    a=-0.51
    b=0.76
    c=-0.35
    d=-0.18

    score =( (a *self.get_aggregate_height(heights))
            + (b*lines)
            + (c * self.get_holes(heights))
            + (d * self.get_bumpiness(heights)))
    return score
```

Classe Move - Pesos

- Os pesos usados foram calculados por tentativa erro e estes foram os melhores valores obtidos para 10 seeds diferentes com os mesmos pesos


```
"6342": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.27, 'd': -0.18}", "results": [512, 509, 645, 326, 485, 311, 309, 639, 566, 518], "media": 482.0},
"6343": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.28, 'd': -0.18}", "results": [662, 585, 695, 445, 564, 212, 366, 352, 579, 404], "media": 486.4},
"6344": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.29, 'd': -0.18}", "results": [615, 550, 567, 315, 588, 594, 574, 255, 437, 102], "media": 459.7},
"6345": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.30, 'd': -0.18}", "results": [511, 431, 373, 389, 411, 662, 334, 480, 403, 368], "media": 436.2},
"6346": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.31, 'd': -0.18}", "results": [621, 376, 350, 256, 547, 685, 349, 685, 569, 349], "media": 479.4},
"6347": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.32, 'd': -0.18}", "results": [515, 671, 547, 642, 595, 531, 533, 392, 352, 103], "media": 488.1},
"6348": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.33, 'd': -0.18}", "results": [595, 676, 361, 523, 345, 573, 494, 427, 671, 316], "media": 498.1},
"6349": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.34, 'd': -0.18}", "results": [301, 498, 601, 627, 558, 489, 268, 609, 506, 473], "media": 493.0},
"6351": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.35, 'd': -0.18}", "results": [425, 415, 352, 558, 400, 696, 378, 614, 595, 561], "media": 499.4},
"6352": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.37, 'd': -0.18}", "results": [383, 488, 398, 536, 423, 226, 458, 395, 480, 291], "media": 407.8},
"6353": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.38, 'd': -0.18}", "results": [463, 366, 369, 358, 428, 450, 462, 315, 392, 217], "media": 382.0},
"6354": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.39, 'd': -0.18}", "results": [193, 299, 497, 366, 384, 245, 350, 535, 183, 302], "media": 335.4},
"6355": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.40, 'd': -0.18}", "results": [502, 489, 450, 243, 257, 407, 530, 520, 297, 547], "media": 424.2},
"6356": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.41, 'd': -0.18}", "results": [384, 528, 264, 176, 476, 361, 200, 426, 259, 401], "media": 347.5},
"6357": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.42, 'd': -0.18}", "results": [174, 176, 218, 401, 481, 387, 308, 360, 274, 443], "media": 322.2},
"6358": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.43, 'd': -0.18}", "results": [288, 231, 370, 154, 305, 407, 398, 247, 377, 301], "media": 307.8},
"6359": {"stats": "{'a': -0.51, 'b': 0.76, 'c': -0.44, 'd': -0.18}", "results": [401, 472, 216, 389, 374, 406, 167, 332, 341, 223], "media": 332.1},
```