Ethan DiPilato, Brett Grossman, Abriana Nash
7/13/17
AI Final Project

Project Update

The current priority will be to get done ASAP the replacement of keyboard inputs with directions from the solution path. The tasks left to follow are finishing the getActions function, writing the greedy agent, writing the BFS search to generate a solution path from the outputs of the various agents. The work will be divided between Brett and Ethan if it is determined that Abriana is not able to contribute for the remaining project time.

This is the getActions function, still a work in progress, it will generate a list of all final positions at the bottom of the board that the piece it is provided can be placed.

```
#generates list of final positions
def getActions():
    positions = []
    if TetrisApp.stone == tetris_shapes[6]:  #shape is square, run code once
        for x in range(10):
            positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,0)), x, 0)
            for y in range(22):
                if not check_collision(board, TetrisApp.stone, (x,y)):
                    position = positions[x]
                    if y > position[2]:
                        positions.pop()
                        positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,y)), x, y)
    if TetrisApp.stone == tetris_shapes[5] or TetrisApp.stone == tetris_shapes[1] or TetrisApp.stone == tetris_shapes[2]: #shape is line, Z or S, run twice (one rotation)
    #ADD IN ROTATION
        for z in range(2):
            for x in range(10):
                positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,0)), x, 0)
                for y in range(22):
                    if not check_collision(board, TetrisApp.stone, (x,y)):
                        position = positions[x]
                        if y > position[2]:
                            positions.pop()
                            positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,y)), x, y)
    else: #run four times (three rotations)
    #ADD IN ROTATION
        for x in range(10):
            positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,0)), x, 0)
            for y in range(22):
                if not check_collision(board, TetrisApp.stone, (x,y)):
                    position = positions[x]
                    if y > position[2]:
                        positions.pop()
                        positions.append(tetris.join_matrices(board, TetrisApp.stone, (x,y)), x, y)

    return positions
```

This is the RandomAgent which randomly chooses a final position for the given piece from the list provided by the getActions function.

```
class RandomAgent():
    #chooses randomly a final position from a list of all available final positions
    def getAction(self, gameState):
        actionList = state.getActions(TetrisApp.stone)
        return actionList[randint(0,len(actionList))]
```

The successor and game state functions.

```python
#returns the base coordinate for the stone, the current state of the game board, and the next stone
def getGameState():
    return TetrisApp.stone_x, TetrisApp.stone_y, board, TetrisApp.next_stone

#generates a list of successors of potential states where the stone has moved left/right or rotated
def generateSuccessor():
    successors = []
    if not tetris.check_collision(board, TetrisApp.stone, (TetrisApp.stone_x - 1, TetrisApp.stone_y)):
        successors.append(tetris.join_matrices(board, TetrisApp.stone, (TetrisApp.stone_x-1, TetrisApp.stone_y)))
    if not tetris.check_collision(board, TetrisApp.stone, (TetrisApp.stone_x + 1, TetrisApp.stone_y)):
        successors.append(tetris.join_matrices(board, TetrisApp.stone, (TetrisApp.stone_x+1, TetrisApp.stone_y)))
    TetrisApp.rotate_stone
    successors.append(tetris.join_matrices(board, TetrisApp.stone, (TetrisApp.stone_x, TetrisApp.stone_y)))
    return successors
```

The expectimax agent that we hope to produce our best results with. Takes max scores at top of tree at first layer (current turn) and second turn (using previewed piece). Then uses averages of max possible scores of each shape at all future layers.

```python
class ExpectimaxAgent():

    def getAction(self, gameState):

        return self.value(gameState, self.depth-1, 1)
    def value(self, state, depthLimit, currentDepth, currentPiece):

        #get successor states if not terminal, returns low negative number if it is terminal.
        if(currentPiece is not 0):
            if(tetris.TetrisApp.isGameOver(state,currentPiece)):
                return -9999999
            successorList = list()
            actionList = state.getActions(currentPiece)
        #if no shape is set, generate list of shapes and get expected value (average) of best move with each shape
        else:
            avg = 0.0
            for x in tetris.tetris_shapes:
                avg = avg + value(state, depthLimit, currentDepth, x)
            return avg/len(tetris.tetris_shapes)

        #return action at top of tree
        if(currentDepth==0):
            scoreHold, action = max([(self.value(state.generateSuccessor(TetrisApp.stone, x), depthLimit,1, 1),x) for x in actionList])
            return action
        #for previewed piece
        if(currentDepth==1):
            return max(self.value(state.generateSuccessor(TetrisApp.next_stone, x), depthLimit, 2, 0 )for x in actionList)

        #return score at max depth
        if(currentDepth==depthLimit and currentPiece is not 0):
            return min(scoreEvaluationFunction(state.generateSuccessor(currentPiece,x) for x in actionList)

        #all other cases (standard)
        return max(self.value(state.generatesuccessor(currentPiece,x) depthLimit, currentDepth+1, 0) for x in actionList)
```

Rudimentary evaluation function that we hope to further tune once it is testable. Currently takes the game's score and divides it by the number of filled squares on the board.

```
def evaluationFunction(state):
    return state.getScore() * (1/len(getPieces.asList())
```

Boolean game over method added to the tetris.py file. The tetris.py we are using did not have this method, and instead checks for a game over as part of its method to start a new turn. We added this method for use in the expectimax agent in order to more easily determine terminal nodes.

```
def isGameOver(self,board1, stone1):
    return check_collision(board1,stone1,(int(cols / 2 - len(stone1[0])/2), 0))
```