

textGridworldDisplay.py ([original](#))

```
# textGridworldDisplay.py
# -----
# Licensing Information: Please do not distribute or publish solutions to this
# project. You are free to use and extend these projects for educational
# purposes. The Pacman AI projects were developed at UC Berkeley, primarily by
# John DeNero (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# For more info, see http://inst.eecs.berkeley.edu/~cs188/sp09/pacman.html

import util

class TextGridworldDisplay:

    def __init__(self, gridworld):
        self.gridworld = gridworld

    def start(self):
        pass

    def pause(self):
        pass

    def displayValues(self, agent, currentState = None, message = None):
        if message != None:
            print message
            values = util.Counter()
            policy = {}
            states = self.gridworld.getStates()
            for state in states:
                values[state] = 0
                policy[state] = None
            prettyPrintValues(self.gridworld, values, policy, currentState)

    def displayNullValues(self, agent, currentState = None, message = None):
        if message != None:
            print message
            prettyPrintNullValues(self.gridworld, agent, currentState)

    def displayQValues(self, agent, currentState = None, message = None):
        if message != None:
            print message
            qValues = util.Counter()
            states = self.gridworld.getStates()
            for state in states:
                for action in self.gridworld.getPossibleActions(state):
                    qValues[(state, action)] = agent.getQValue(state, action)
            prettyPrintQValues(self.gridworld, qValues, currentState)

    def prettyPrintValues(self, gridworld, values, policy=None, currentState = None):
        grid = gridworld.grid
        maxLen = 11
        newRows = []
        for y in range(grid.height):
            newRow = []
            for x in range(grid.width):
                state = (x, y)
                value = values[state]
                action = None
                if policy != None and state in policy:
                    action = policy[state]
                actions = gridworld.getPossibleActions(state)
                if action not in actions and 'exit' in actions:
                    action = 'exit'
                valString = None
                if action == 'exit':
                    valString = border('%.2f' % value)
```

```

else:
    valString = '\n\n%.2f\n\n' % value
    valString += ' '*maxLen
if grid[x][y] == 'S':
    valString = '\n\nS: %.2f\n\n' % value
    valString += ' '*maxLen
if grid[x][y] == '#':
    valString = '\n#####\n#####\n#####\n'
    valString += ' '*maxLen
pieces = [valString]
text = ("\n".join(pieces)).split('\n')
if currentState == state:
    l = len(text[1])
    if l == 0:
        text[1] = '*'
    else:
        text[1] = "|" + ' '*int((l-1)/2-1) + '*' + ' '*int((l)/2-1) + "|"
if action == 'east':
    text[2] = ' ' + text[2] + '>'
elif action == 'west':
    text[2] = '<' + text[2] + ' '
elif action == 'north':
    text[0] = ' '*int(maxLen/2) + '^' + ' '*int(maxLen/2)
elif action == 'south':
    text[4] = ' '*int(maxLen/2) + 'v' + ' '*int(maxLen/2)
newCell = "\n".join(text)
newRow.append(newCell)
newRows.append(newRow)
numCols = grid.width
for rowNum, row in enumerate(newRows):
    row.insert(0, "row"+str(rowNum))
newRows.reverse()
collabels = [str
collabels.insert
finalRows = [col
print indent(fin
justify='center', hasHeader=True)

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

def prettyPrintNullValues(gridworld, current
    grid = gridworld.grid
    maxLen = 11
    newRows = []
    for y in range(grid.height):
        newRow = []
        for x in range(grid.width):
            state = (x, y)

            # value = values[state]

            action = None
            # if policy != None and state in policy:
            #     action = policy[state]
            #
            actions = gridworld.getPossibleActions(state)

            if action not in actions and 'exit' in actions:
                action = 'exit'

            valString = None
            # if action == 'exit':
            #     valString = border('%.2f' % value)
            # else:
            #     valString = '\n\n%.2f\n\n' % value
            #     valString += ' '*maxLen

            if grid[x][y] == 'S':
                valString = '\n\nS\n\n'
                valString += ' '*maxLen

```

```

elif grid[x][y] == '#':
    valString = '\n#####\n#####\n#####\n'
    valString += ' '*maxLen
elif type(grid[x][y]) == float or type(grid[x][y]) == int:
    valString = border('%.2f' % float(grid[x][y]))
else: valString = border(' ')
pieces = [valString]

text = ("\n".join(pieces)).split('\n')

if currentState == state:
    l = len(text[1])
    if l == 0:
        text[1] = '*'
    else:
        text[1] = "|" + ' '*int((l-1)/2-1) + '*' + ' '*int((l)/2-1) + "|"

    if action == 'east':
        text[2] = ' ' + text[2] + '>'
    elif action == 'west':
        text[2] = '<' + text[2] + ' '
    elif action == 'north':
        text[0] = ' '*int(maxLen/2) + '^' + ' '*int(maxLen/2)
    elif action == 'south':
        text[4] = ' '*int(maxLen/2) + 'v' + ' '*int(maxLen/2)
    newCell = "\n".join(text)
    newRow.append(newCell)
newRows.append(newRow)
numCols = grid.width
for rowNum, row in enumerate(newRows):
    row.insert(0, "\n" + str(rowNum))
newRows.reverse()
collabels = [s
collabels.inse
finalRows = [c
print indent(f
fix = '|', postfix='|',
justify='center', hasHeader=True)

def prettyPrintQValues(gridWorld, qValues, c
grid = gridWorld.grid
maxLen = 11
newRows = []
for y in range(grid.height):
    newRow = []
    for x in range(grid.width):
        state = (x, y)
        actions = gridWorld.getPossibleActions(state)
        if actions == None or len(actions) == 0:
            actions = [None]
        bestQ = max([qValues[(state, action)] for action in actions])
        bestActions = [action for action in actions if qValues[(state, action)] ==
bestQ]

    # display cell
    qStrings = dict([(action, "%.2f" % qValues[(state, action)]) for action in
actions])
    northString = ('north' in qStrings and qStrings['north']) or ' '
    southString = ('south' in qStrings and qStrings['south']) or ' '
    eastString = ('east' in qStrings and qStrings['east']) or ' '
    westString = ('west' in qStrings and qStrings['west']) or ' '
    exitString = ('exit' in qStrings and qStrings['exit']) or ' '

    eastLen = len(eastString)
    westLen = len(westString)
    if eastLen < westLen:
        eastString = ' '*(westLen-eastLen)+eastString
    if westLen < eastLen:
        westString = westString+' '*(eastLen-westLen)

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

if 'north' in bestActions:
    northString = '/' + northString + '\\'
if 'south' in bestActions:
    southString = '\\' + southString + '/'
if 'east' in bestActions:
    eastString = '+' + eastString + '>'
else:
    eastString = '+' + eastString + ' '
if 'west' in bestActions:
    westString = '<' + westString + ' '
else:
    westString = ' ' + westString + ' '
if 'exit' in bestActions:
    exitString = '[' + exitString + ']'

ewString = westString + "      " + eastString
if state == currentState:
    ewString = westString + " * " + eastString
if state == gridWorld.getStartState():
    ewString = westString + " S " + eastString
if state == currentState and state == gridWorld.getStartState():
    ewString = westString + " S:* " + eastString

text = [northString, "\n"+exitString, ewString, ' '*maxLen+"\n", southString]

if grid[x][y] == '#':
    text = ['', '\n#####\n#####\n#####', '']

```

```

newCell = " " + direction + " "
newRow.append(newCell)
newRows.append(newRow)
numCols = grid
for rowNum, row in enumerate(newRows):
    row.insert(0, ' ')
newRows.reverse()
colLabels = [str(colNum) for colNum in range(numCols)]
colLabels.insert(0, ' ')
finalRows = [colLabels] + newRows

```

```

print indent(finalRows, separateRows=True, delim='|', prefix='', postfix='',
justify='center', hasHeader=True)

```

```

def border(text):
    length = len(text)
    pieces = ['-' * (length+2), '|' + ' ' * (length+2) + '|', ' | '+text+ ' | ', '|' + ' ' *
(length+2) + '|', '-' * (length+2)]
    return '\n'.join(pieces)

```

INDENTING CODE

*# Indenting code based on a post from George Sakkis
(http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/267662)*

```
import cStringIO, operator
```

```

def indent(rows, hasHeader=False, headerChar='-', delim='|', justify='left',
           separateRows=False, prefix='', postfix='', wrapfunc=lambda x:x):
    """Indents a table by column.
    - rows: A sequence of sequences of items, one sequence per row.
    - hasHeader: True if the first row consists of the columns' names.
    - headerChar: Character to be used for the row separator line
      (if hasHeader==True or separateRows==True).
    - delim: The column delimiter.
    - justify: Determines how are data justified in their column.
      Valid values are 'left', 'right' and 'center'.
    - separateRows: True if rows are to be separated by a line
      of 'headerChar's.
    - prefix: A string prepended to each printed row.
    """

```

```

- postfix: A string appended to each printed row.
- wrapfunc: A function f(text) for wrapping text; each element in
  the table is first wrapped by this function."""
# closure for breaking logical rows to physical, using wrapfunc
def rowWrapper(row):
    newRows = [wrapfunc(item).split('\n') for item in row]
    return [[substr or '' for substr in item] for item in map(None,*newRows)]
# break each logical row into one or more physical ones
logicalRows = [rowWrapper(row) for row in rows]
# columns of physical rows
columns = map(None,*reduce(operator.add,logicalRows))
# get the maximum of each column by the string length of its items
maxWidths = [max([len(str(item)) for item in column]) for column in columns]
rowSeparator = headerChar * (len(prefix) + len(postfix) + sum(maxWidths) + \
                             len(delim)*(len(maxWidths)-1))

# select the appropriate justify method
justify = {'center':str.center, 'right':str.rjust, 'left':str.ljust}
[justify.lower()]
output=cStringIO.StringIO()
if separateRows: print >> output, rowSeparator
for physicalRows in logicalRows:
    for row in physicalRows:
        print >> output, \
            prefix \
            + delim.join([justify(str(item),width) for (item,width) in
zip(row,maxWidths)]) \
            + postfix
        if separateRows or hasHeader: print >> output, rowSeparator; hasHeader=False
    return output.getvalue()

import math
def wrap_always(text, width):
    """A simple wo
    It doesn't
    return '\n'.jo

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

# TEST OF DISPLAY CASE
if __name__ == '__main__':
    import gridworld, util

    grid = gridworld.getCliffGrid3()
    print grid.getStates()

    policy = dict([(state, 'east') for state in grid.getStates()])
    values = util.Counter(dict([(state, 1000.23) for state in grid.getStates()]))
    prettyPrintValues(grid, values, policy, currentState = (0,0))

    stateCrossActions = [(state, action) for action in grid.getPossibleActions(state)]
    for state in grid.getStates():
        qStates = reduce(lambda x,y: x+y, stateCrossActions, [])
        qValues = util.Counter(dict([(state, action), 10.5) for state, action in
qStates]))
        qValues = util.Counter(dict([(state, action), 10.5) for state, action in
reduce(lambda x,y: x+y, stateCrossActions, [])]))
        prettyPrintQValues(grid, qValues, currentState = (0,0))

```