Building Tic Tac Toe (Part 2)

Nov 12, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
Announcements & Logistics

- **Lab 7** feedback coming soon (sorry)
- **HW 8** posted on Wed, due Monday 11/15 10 pm
- **Lab 9 Boggle** will be released today: multi-week partners lab (counts as a two labs in terms of grade; Lab is decomposed into four logical parts
  - **Parts 1 & 2 (BoggleLetter & BoggleBoard)** due Nov 17/18 10 pm
  - We will run our tests on these and return automated feedback (similar to Lab 4 part 1), but you are allowed to revise it afterwards
  - **Parts 3 & 4 (BoggleWords & Game)** are due Dec 1/2 the week after Thanksgiving break
  - No expectation to work during Thanksgiving break: enjoy your well-deserved time off!
- Thanksgiving week: attending lab is optional but encouraged

Do You Have Any Questions?
Last Time

• Started to discuss an application of object-oriented design
  • Started to build a graphical board game: Board class
  • Used the graphics package as a black box tool for our design
  • Discussed decomposition by breaking tic-tac-toe into layers
Last Time: Board class

- Basic features of our game board:
  - Text areas: above, below, right of grid
  - Grid of squares of set size: rows x cols
  - Reset and Exit buttons
  - React to mouse clicks (we'll discuss this)
- These are all **graphical** (GUI) components
  - Used graphics package to create rectangles/window/text
  - `Obj.draw(win)` draws object on graphical window win
Board Class: All the Pieces

- Grid for the game
- Upper text area
- Right text area
- Lower text area
- Reset/Exit buttons
Today’s Plan

- Look at helper methods in the Board class
- Talk about building the Tic Tac Toe board by inheriting from Board class
  - How can we extend board for a Tic Tac Toe (TTT) game?
  - What TTT-specific new methods/attributes do we need?
- Move up the next layer: TTT Letter
  - What attributes/methods can we use to implement functionality of a single Tic Tac Toe letter?
- Next time: Wrap up Tic Tac Toe by completing the logic of the game
Helper Methods: Board

- Now that we have a board with a grid, buttons, and text areas, it would be useful to define some methods for interacting with these objects.
- Helpful methods?
Now that we have a board with a grid, buttons, and text areas, it would be useful to define some methods for interacting with these objects.

Helpful methods?

- Get grid coordinate of mouse click
- Determine if click was in grid, reset, or exit buttons
- Set text to one of 3 text areas
- ...

Note that none of this is specific to Tic Tac Toe (yet)!

Always good to start general and then get more specific.
Helper Methods

addStringToLowerText(self, text)
    Add text to text area below grid.
    Does not overwrite existing text.

clearLowerText(self)
    Clear text area below grid.

clearTextArea(self)
    Clear text in text area to right of grid.

clearUpperText(self)
    Clear text area above grid.

drawBoard(self, win)

getPosition(self, location)
    Converts a window location (tuple) to a grid position (tuple).
    Window locations are x, y coordinates.
    Note: Grid positions are always returned as col, row.

inExit(self, point)
    Returns true if point is inside exit button (rectangle)

inGrid(self, point)
    Returns True if a Point (point) exists inside the grid of squares.

inReset(self, point)
    Returns true if point is inside exit button (rectangle)

setStringToLowerText(self, text)
    Set text to text area below grid.
    Overwrites existing text.

setStringToUpperText(self, text)
    Clear text area above grid.

setTextArea(self, text)
    Sets text to text area to right of grid.
    Overwrites existing text.
Working with Mouse Clicks

• `win.getMouse()` returns a `Point` object, which has an `x` and `y` coordinate (tuple) determined by the screen coordinate

• We can use helper methods (with simple calculations) to test which grid square or button the click occurred in

• This will be useful in our next step!

• (Run `python3 board.py` in Terminal)
Board Class: Bigger Picture

• Tic Tac Toe is not the only text based board game

• Our **Board** class that can be used for other games as well, such as Boggle!

• Summary of our basic **Board** class implementation:
  
  • Create a grid of a certain **size** (e.g., 3 by 3 for Tic Tac Toe)
  
  • Define attributes/properties (getters) for **rows, cols, size**, etc
  
  • Provide helper methods to recognize and interpret a mouse click on the board
  
  • Provide other basic features (and methods for manipulating them) such as text areas for indicating whose turn it is, printing who wins, etc

• Through the **power of inheritance** we can use the same board class for TicTacToe and Boggle!
Moving up: TTTBoard

- Although our Board class provides a lot of useful functionality, there are some Tic Tac Toe specific features we need to support
- We can do this by **inherit**ing from the Board class
- We can take advantage of all of the methods and attributes defined in **Board** and add any (specific) extras we may need for **TTT**
- What extras (attributes and/or methods) might be useful?
TTT Board Design

• Think of the grid composed of **TTTLetters**
  • Initially populate grid with **TTTLetters** that are empty
• But let's think about the Board state in the "middle of the game"
• What are some helper methods that can help get/set the game state?
  • Check individual **TTTLetters** for X or O
  • Setting individual **TTTLetters** to X or O
  • Check for win (how?)
• Need helper methods for row/column checks
To use TTTLetter, we just need to know its documentation (not how it is implemented)

```python
class TTTLetter(builtins.object):
    TTTLetter(col=-1, row=-1, letter='')

    Methods defined here:
    __init__(self, col=-1, row=-1, letter='')
        Initialize self. See help(type(self)) for accurate signature.

    resetLetter(self)
        Resets the letter of TTTLetter to empty string

Data descriptors defined here:

col
    Property: col on grid of TTTLetter

letter
    Property: letter (str) associated with TTTLetter

row
    Property: row on grid of TTTLetter

textObj
    Property: textObj attribute of TTTLetter
```
To use TTTLetter, we just need to know its documentation (not how it is implemented).

To use TTT letters we need to know that they have:

- \((\text{col, row})\) position on board
- a textObj which is a Text object that is further composed of
  - a Point object (from graphics package) that determines where it appears on the board (depends on col, row and inset values: preset by default so we don't have to worry about it)
  - a letter (string) which is what we care about
    - Going to be "X" or "O" in this game
- Note: Can always use \texttt{getText()} of Text object to find string value of text
Initializing the TTT Board

• What (data) attributes do we need?
  • Everything inherited from Board class
  • A grid: a list of lists of TTTLetters

```python
class TTTBoard(Board):
    __slots__ = ['_grid']

    def __init__(self):
        super().__init__()  # Inherit from Board

        # create a 3x3 grid of TTTLetters (list of lists)
        self._grid = []
        for col in range(self.cols):
            # create a list of col letters
            colLetters = [TTTLetter(col, row) for row in range(self.rows)]
            self._grid.append(colLetters)
```

Initialize parent's attributes e.g., rows/cols
Populate grid with TTTLetters
Drawing the TTT Board

- Initializing TTTLetters does not draw a board
- To draw the TTTLetters on the board, we can write a drawTTTBoard method
  - But should we draw the board from scratch?

```python
def drawBoard(self, win):
    # draw generic grid first
    super().drawBoard(win)

    # draw TTT letters on grid
    for col in range(self.cols):
        for row in range(self.rows):
            tttLet = self._grid[col][row]
            tttLet.textObj.draw(win)
```

Overrides drawBoard in Board class

Call parent's draw method first

Then draw the TTTLetters on the window by calling Text objects draw method
Setting Letters on the Board

- Right now our board is blank. To put some letters on the board, what do we need to do?
  - Change the `TTTLetter` object from "" (empty) to "X" or "O"
- Before we write any setter methods, we should also write accessor (getter) methods to help us

```python
def getLetterObj(self, pos):
    """Returns the TTTLetter at position pos""
    pass # TODO

def getLetter(self, pos):
    """Return the string associated with TTTLetter at position pos""
    pass # TODO

def setLetter(self, pos, alph):
    """Let the string of TTTLetter at position pos to alph""
    pass # TODO
```
def getLetterObj(self, pos):
    """Returns the TTTLetter at position pos""
    col, row = pos
    return self._grid[col][row]

def getLetter(self, pos):
    """Return the string associated with TTTLetter at position pos""
    letObj = self.getLetterObj(pos)
    # calls getter method of TTTLetter class
    return letObj.letter

def setLetter(self, pos, alph):
    """Let the string of TTTLetter at position pos to alph""
    letObj = self.getLetterObj(pos)
    # calls setter method of TTTLetter class
    letObj.letter = alph

Always unpack a small tuple; don't do pos[0], pos[1]
def getLetterObj(self, pos):
    """Returns the TTTLetter at position pos""
    col, row = pos
    return self._grid[col][row]

def getLetter(self, pos):
    """Return the string associated with TTTLetter at position pos""
    letObj = self.getLetterObj(pos)
    # calls getter method of TTTLetter class
    return letObj.letter

def setLetter(self, pos, alph):
    """Let the string of TTTLetter at pos be alph""
    letObj = self.getLetterObj(pos)
    # calls setter method of TTTLetter class
    letObj.letter = alph
Get/Set Letters on the Board

```python
def getLetterObj(self, pos):
    """Returns the TTTLetter at position pos""
    col, row = pos
    return self._grid[col][row]

def getLetter(self, pos):
    """Return the string associated with TTTLetter at position pos""
    letObj = self.getLetterObj(pos)
    # calls getter method of TTTLetter class
    return letObj.letter

def setLetter(self, pos, alph):
    """Let the string of TTTLetter at pos be alph"
    letObj = self.getLetterObj(pos)
    # calls setter method of TTTLetter class
    letObj.letter = alph
```
Draw Board with Letters

```python
In [22]:
tttboard = TTTBoard()
tttboard.setLetter((1, 2), "X")
tttboard.setLetter((2, 0), "O")
tttboard.setLetter((0, 1), "X")

In [23]:
win = GraphWin("Tic Tac Toe", 400, 400)
tttboard.drawBoard(win)
```
Reseting the TTTBoard

• As we are building the Board it would be helpful for us to have a way to reset the state of the board to be blank

• This, of course, is also helpful during play (if we hit the reset button or the game ends in Win/Draw and we want to restart)

• What do we need to change to reset the board?
  • Reset every TTTLetter to empty string
  • `resetLetter()` method exists in TTTLetter Class

```python
def reset(self):
    for x in range(self.cols):
        for y in range(self.rows):
            # get letter out of grid and reset it
            tttLet = self._grid[x][y]
            # calls reset letter in TTT Letter
            tttLet.resetLetter()
```
Getting Closer

- What other helper methods do we need?
  - Checking for win of a player "X" or "O"
- A letter ("X" or "O") wins if:
  - There exists a column filled with letter; OR
  - There exists a row filled with letter; OR
  - There exists a diagonal that is filled with letter
- Let's break that down into separate private helper methods
  - _checkRows
  - _checkCols
  - _checkDiagonals
Checking the Rows

- We need to find if there is ANY row that is made of only letter.
- How can we approach this?

Grid positions are (col, row)

def _checkRows(self, letter):
    pass

checkRows checks the board horizontally
Checking the Rows

- We need to find if there is ANY row that is made of only letter
- Fix a row, go through each column

```python
def _checkRows(self, letter):
    for row in range(self.rows):
        count = 0
        for col in range(self.cols):
            tttlet = self._grid[col][row]
            pass  # todo

    return False
```
Checking the Rows

- We need to find if there is ANY row that is made of only letter
- Fix a row, go through each column

```python
def _checkRows(self, letter):
    for row in range(self.rows):
        count = 0
        for col in range(self.cols):
            tttLet = self._grid[col][row]

            # check how many times letter appears in row
            if tttLet.letter == letter:
                count += 1
            # if this is a winning row
            if count == self.rows:
                return True

        # no winning row found
    return False
```

If all letters match, return True
If no winning row, return False
Similarly Check Columns

- We can similarly check a column for a win

```python
def _checkCols(self, letter):
    # check columns
    for col in range(self.cols):
        count = 0
        for row in range(self.rows):
            tttLet = self._grid[col][row]

            # check how many times letter appears
            if tttLet.letter == letter:
                count += 1

        # if this is a winning col
        if count == self.cols:
            return True

    # if no winning cols
    return False
```
Check Diagonals

Primary diagonal has row/col same

```
def _checkDiagonals(self, letter):
    # counts for primary and secondary diagonal
    countA, countB = 0, 0

    for col in range(self.cols):
        for row in range(self.rows):
            tttLet = self._grid[col][row]

            # update count for primary diagonal
            if (row == col and
                tttLet.letter == letter):
                countA += 1

            # update count for secondary diagonal
            if (row + col == self.rows - 1 and
                tttLet.letter == letter):
                countB += 1

    # return true if either return in win
    return countA == self.rows or countB == self.rows
```

Secondary diagonal: (0, 2), (1, 1), (2, 0) for a 3x3 board

Secondary diagonal has row + col = n -1
Final Check for Win

• Putting it all together: the board is in a winning state if any of the three winning conditions are true

• We'll make this method **public** as it will needed outside of this class

```python
def checkForWin(self, letter):
    rowWin = self._checkRows(letter)
    colWin = self._checkCols(letter)
    diagWin = self._checkDiagonals(letter)

    return rowWin or colWin or diagWin
```
Leftovers: Next time

- We don't have a working Tic Tac Toe game yet
  - But we're getting close!
- What's left?
  - We have been using TTTLetter (but we will look at its implementation on Monday)
  - We need to implement the game logic
- What do we need to do to put this all together?
  - Keep track of mouse clicks
  - Keep track of players ("X" and "O" alternate)
  - Use TTTLetter and TTTBoard to check win