CS 134:
Tic Tac Toe (2)
TTTBoard
Announcements & Logistics

• **Lab 7** feedback coming soon

• **HW 8** due Monday Nov 14 10 pm

• **Lab 9 Boggle** will be released later today: multi-week partners lab (counts as two labs in terms of grade; Lab is decomposed into three logical parts)
  
  • **Parts 1 & 2 (BoggleLetter & BoggleBoard)** due Nov 16/17
  
  • We will run our tests on these and return automated feedback (similar to Lab 4 part 1); you are allowed/encouraged to revise it afterwards

  • **Parts 3 (BoggleGame)** (and revised Parts 1 and 2) due Nov 30/Dec 1

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
  - `object.draw(win)` draws object on graphical window `win`
Board Class: All the Pieces

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

• Look at some of the 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 to the next layer: **TTTLetter**
  • 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
Board Helper Methods
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 (aside from getters, setters, __init__, etc)

- 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 (aside from getters, setters, __init__, etc).

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.
class Board(builtins.object):
    Board(win, xInset=50, yInset=50, rows=3, cols=3, size=50)

Methods defined here:

__init__(self, win, xInset=50, yInset=50, rows=3, cols=3, size=50)
    Initialize self. See help(type(self)) for accurate signature.

drawBoard(self)
    Create the board with the grid, text areas, and buttons

getPosition(self, point)
    Converts a window location (Point) to a grid position (tuple).
    Note: Grid positions are always returned as col, row.

getStringFromLowerText(self)
    Get text from text area below grid.

getStringFromTextArea(self)
    Get text from text area to right of grid.

getStringFromUpperText(self)
    Get text from text area above grid.

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.

setStringToTextArea(self, text)
    Sets text to text area to right of grid. Overwrites existing text.

setStringToUpperText(self, text)
    Set text to text area above 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!

```bash
>>> python3 board.py
```
Example: inGrid(point)

- Test to see if the mouse click was in the game grid

```python
# check for click in grid
def inGrid(self, point):
   ...
    Returns True if a Point (point) exists inside the grid of squares.
    ...
    ptX = point.getX()
    ptY = point.getY()
    maxY = self._size * (self._rows + 1)
    maxX = self._size * (self._cols + 1)
    return ptX <= maxX and ptY <= maxY and ptX >= self._xInset and ptY >= self._yInset
```

- There are several other helper methods as well! We’ll keep these in mind as we implement our other classes.

```python
win = GraphWin("Tic Tac Toe", 400, 400)
board = Board(win)
point = win.getMouse()
print("Clicked in grid?", board.inGrid(point))
```
Board Class: Bigger Picture

- Tic Tac Toe is not the only text based board game
- Our **Board** class can be used for other games too, such as Boggle! (Lab 9)
- 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 and methods (getters) to access 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!
TTTBoard Class
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 *inheriting* 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 extra attributes and/or methods might be useful?
• Think of the grid composed of TTTLetters
  • Initially populate grid with TTTLetters that are “empty”
• 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/diag checks
TTTLetter

- To use TTTLetter, we just need to know its documentation (not how it is implemented) — abstraction!
- We will explore the implementation later

```python
class TTTLetter(builtins.object):
    def __init__(self, board, col=-1, row=-1, letter=' '):
        A TTT letter has several attributes that define it:
        * `_row, _col coordinates indicate its position in the grid (ints)
        * `_textObj denotes the Text object from the graphics module,
          which has attributes such as size, style, color, etc
          and supports methods such as getText(), setText() etc.
    
    Methods defined here:
    
    __init__(self, board, col=-1, row=-1, letter=' ')
    Initialize self. See help(type(self)) for accurate signature.

    __repr__(self)
    Return repr(self).

    __str__(self)
    Return str(self).

    getLetter(self)
    Returns letter (text of type str) associated with property textObj

    setLetter(self, char)
```
TTTLetter

- To use TTTLetter, we just need to know its documentation (not how it is implemented) — abstraction!

- To use TTT letters we need to know that they have:
  - \((\text{col}, \text{row})\) position on \text{game grid}
  - a \text{letter} (string): Going to be "X" or "O" in TTT
  - methods for getting and setting \text{letter}
Initializing the TTT Board

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

```
class TTTBoard(Board):
    """TTT Board class implements the functionality of a Tic Tac Toe board. It inherits from the Board class and extends it by creating a grid of TTTLetters."""
    __slots__ = ['_grid']

    def __init__(self, win):
        super().__init__(win)

        # initialize a grid of TTTLetters (list of lists)
        self._grid = []
        for col in range(self._cols):
            grid_col = []
            # next part could be a list comprehension!
            for row in range(self._rows):
                # create new TTTLetter, specifying grid coord
                letter = TTTLetter(self.getBoard(), col, row)
                # add TTTLetter to column
                grid_col.append(letter)
            # add column to grid
            self._grid.append(grid_col)
```

Inherit from Board

Call parent’s __init__ method

Populate grid with empty TTTLetters
• Think of the grid composed of **TTTLetters**
  
  • Initially populate grid with **TTTLetters** that are “empty”
  
  • 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/diag checks
Accessing Letters on the Board

• Right now our board is blank. To put some characters on the board, what do we need to do?
  • Change the **TTTLetter** object from "" (empty) to "X" or "O"
• Let’s write a getter method to help us get **TTTLetter** objects from our grid

```python
def getTTTLetterAtPoint(self, point):
    """Returns the TTTLetter at point (a screen coord tuple)""
    if self.inGrid(point):
        (col, row) = self.getPosition(point)
        return self._grid[col][row]
    else:
        return None
```

Works with screen “Points” from mouse clicks (such as (100, 200))
Setting Letters on the Board

• Once we have a `TTTLetter` object, we can use the `setLetter()` method to change the character to an “X” or “O”

```python
    tttboard.getTTTLetterAtPoint(Point(75, 75)).setLetter("X")
    tttboard.getTTTLetterAtPoint(Point(150, 150)).setLetter("O")
```
Setting Letters on the Board

```python
win = GraphWin("Tic Tac Toe", 400, 400)
tttboard = TTTBoard(win)

tttboard.getTTTLetterAtPoint(Point(75, 75)).setLetter("X")
tttboard.getTTTLetterAtPoint(Point(150, 150)).setLetter("O")
```
Resetting 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 (and clear colors)

```python
# reset all letter objects in grid
def reset(self):
    """Clears the TTT board by clearing letters."""
    for x in range(self._cols):
        for y in range(self._rows):
            # get letter out of grid and reset it
            let = self._grid[x][y]
            let.setLetter(""")
```
TTTBoard Helper Methods:
Checking for Wins
• Think of the grid composed of *TTTLetters*
  • Initially populate grid with *TTTLetters* that are “empty”
• 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/diag checks
Getting Closer

• What other helper methods do we need?
  • Checking for win of a player "X" or "O"

• A player ("X" or "O") wins if:
  • There exists a column filled with their letter, OR
  • There exists a row filled with their letter, OR
  • There exists a diagonal that is filled with their letter

• Let's break that down into separate private helper methods
  • _checkRows
  • _checkCols
  • _checkDiagonals
Checking the Rows

- For a given letter (“X” or “O”), we need to find if there is ANY row that is made of only letter
- How can we approach this?

```python
def _checkRows(self, letter):
    pass
```

Grid positions are (col, row)

checkRows checks the board horizontally
Checking the Rows

- For a given letter ("X" or "O"), we need to find if there is ANY row that is made of only letter
- Fix a row, go through each column

Why initialize `count` here?

```python
# checking for win methods:
def _checkRows(self, letter):
    """Check rows for a win (3 in a row).""
    for row in range(self._rows):
        count = 0
        for col in range(self._cols):
            tttLet = self._grid[col][row]
            pass
# no winning row found
return False
```
Checking the Rows

• For a given letter ("X" or "O"), we need to find if there is ANY row that is made of only letter

• Fix a row, go through each column

```python
# checking for win methods:
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
            if tttLet.getLetter() == 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 a win (3 in a row)."""
    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.getLetter() == letter:
                count += 1

            # if this is a winning row
            if count == self._cols:
                return True

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

Primary diagonal has row/col same

```python
def _checkDiagonals(self, letter):
    """Check diagonals for a win (3 in a row)."""
    # counts for primary and secondary diagonal
    countPrimary, countSecond = 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.getLetter() == letter):
                countPrimary += 1

    # return true if either return in win
    return countPrimary == self.getRows()
```

Check Diagonals

```python
def _checkDiagonals(self, letter):
    """Check diagonals for a win (3 in a row)."""
    # counts for primary and secondary diagonal
    countPrimary, countSecond = 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.getLetter() == letter):
                countPrimary += 1

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

    # return true if either return in win
    return countPrimary == self.getRows() or
           countSecond == self.getRows()
```

Secondary diagonal has row + col = 2

Secondary diagonal:
(0, 2), (1, 1), (2, 0) for a 3x3 board
Putting it all together: the board is in a winning state if any of the three winning conditions are true.

We will 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, so we’ll look at it briefly
  • 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" must alternate)
  • Use TTTLetter and TTTBoard to check for win
The end!