CS 134:
Tic Tac Toe
Announcements & Logistics

- **Lab 8 today/tomorrow** (due Wed/Thur)
  - When working with a partner, remember to take turns “driving”
  - Questions?
- **HW 7** due tonight at 11 pm
Last Time

• Learned a bit more about classes and special __ (double underscore) methods
  • __str__ : print representation of objects
  • __init__ : initialize objects
• Began talking about inheritance
Today’s Plan

• Discuss inheritance and object oriented design for Tic Tac Toe
  • Think about how to decompose a game into multiple pieces
    • Board, TTTBoard, TTTLetters, and Game
Tic Tac Toe
Implementing Tic Tac Toe

- Suppose we want to implement Tic Tac Toe
- Teaser demo…

```bash
>>> python3 game.py
```

![Tic Tac Toe game](image)
Decomposition

• Let’s try to identify the “layers” of this game
• Through abstraction, each layer can ignore what’s happening in the other layers
• What are the layers of Tic Tac Toe?
Decomposition

- Let’s try to identify the “layers” of this game
- Through abstraction and encapsulation, each layer can ignore what’s happening in the other layers
- What are the layers of Tic Tac Toe?
  - Bottom layer: **Basic board** w/buttons, text areas, mouse click detection (not specific to Tic Tac Toe!)
  - Lower middle layer: Extend the **basic board with Tic Tac Toe specific features** (3x3 grid, of TTTLetters, initial board state: all letters start blank)
  - Upper middle layer: **Tic Tac Toe “spaces” or “letters”** (9 in total!); set text to X or O
  - Top layer: **Game logic** (alternating turns, checking for valid moves, etc)
Board class

• Let’s start at the bottom: Board class
• What are basic features of all game boards?
  • Think generally…many board-based games have the similar basic requirements
  • (For example, Boggle, TicTacToe, Scrabble, etc)
Board class

- Let’s start at the bottom: Board class
- What are basic features of all game boards?
  - Text areas: above, below, right of grid
  - Grid of squares of set size: rows x cols
  - Reset and Exit buttons
  - React to mouse clicks (less obvious!)
- These are all **graphical** (GUI) components
  - Code for graphics is a little messy at times
  - Lot’s of things to specify: color, size, location on screen, etc
A pixel is one of the small dots or squares that make up an image on a computer screen.

We are going to use a simple graphics package to implement our game board.

Create a window with title “Name” and size 400x400 (measured in pixels).

```python
from graphics import *

# takes title and size of window
win = GraphWin("Name", 400, 400)
```
We can draw other shapes as well.

We'll want to draw Rectangles in our Board class.
Graphics Package for Board

In [3]: # create point obj at x, y coordinate in window
pt = Point(200, 200)

In [4]: # create circle with center at pt and radius 100
   c = Circle(Point(200,200), 100)

In [5]: # draw the circle on the window
   c.draw(win)

Out[5]: Circle(Point(200.0, 200.0), 100)
Detecting “events” like mouse clicks are an important part of a graphical program.

`win.getMouse()` is a blocking method call that “blocks” or waits until a click is detected.
Board class: Getting Started

- Attributes:
  - We will add a few more attributes later
  - We need to draw the grid, text areas, and buttons
  - Might need some helper methods to organize our code
  - Let’s start by **drawing the grid** on our board

```python
# _win: graphical window on which we will draw our board
# _xInset: avoids drawing in corner of window
# _yInset: avoids drawing in corner of window
# _rows: number of rows in grid of squares
# _cols: number of columns in grid of squares
# _size: edge size of each square
```
from graphics import *

class Board:
    # _win: graphical window on which we will draw our board
    # _xInset: avoids drawing in corner of window
    # _yInset: avoids drawing in corner of window
    # _rows: number of rows in grid of squares
    # _cols: number of columns in grid of squares
    # _size: edge size of each square
__slots__ = [ '_xInset', '_yInset', '_rows', '_cols', '_size', '_win', ]

def __init__(self, win, xInset=50, yInset=50, rows=3, cols=3, size=50):
    # update class attributes
    self._xInset = xInset
    self._yInset = yInset
    self._rows = rows
    self._cols = cols
    self._size = size
    self._win = win

    # getter methods for attributes
    def getWin(self):
        return self._win

    def getXInset(self):
        return self._xInset

    def getYInset(self):
        return self._yInset

    def getRows(self):
        return self._rows

    def getCols(self):
        return self._cols

    def getSize(self):
        return self._size
Board class: Drawing the grid

```python
def __makeGrid(self):
    '''Creates a row x col grid, filled with squares'''
    for x in range(self._cols):
        for y in range(self._rows):
            # create first point
            p1 = Point(self._xInset + self._size * x, self._yInset + self._size * y)
            # create second point
            p2 = Point(self._xInset + self._size * (x + 1), self._yInset + self._size * (y + 1))
            # create rectangle
            r = Rectangle(p1, p2)
            r.setFill("white")
            # add rectangle to graphical window
            r.draw(self._win)
```

We need a window (_win) on which to **draw**.

\(x=0, y=0:\)

\(\text{p1:}\)
- \(x\text{Inset} + (\text{size} \times x) = x\text{Inset}\)
- \(y\text{Inset} + (\text{size} \times y) = y\text{Inset}\)

\(\text{p2:}\)
- \(x\text{Inset} + (\text{size} \times (x+1)) = x\text{Inset} + \text{size}\)
- \(y\text{Inset} + (\text{size} \times (y+1)) = y\text{Inset} + \text{size}\)
Board class: Drawing the grid

```python
def __makeGrid(self):
    """Creates a row x col grid, filled with squares"""
    for x in range(self._cols):
        for y in range(self._rows):
            # create first point
            p1 = Point(self._xInset + self._size * x, self._yInset + self._size * y)
            # create second point
            p2 = Point(self._xInset + self._size * (x + 1), self._yInset + self._size * (y + 1))
            # create rectangle
            r = Rectangle(p1, p2)
            r.setFill("white")
            # add rectangle to graphical window
            r.draw(self._win)
```

We need a window (_win) on which to **draw**.

x=0, y=1:

p1:
\[
\begin{align*}
    x & \inset + (\text{size} \times x) = x\inset \\
    y & \inset + (\text{size} \times y) = y\inset + \text{size} 
\end{align*}
\]

p2:
\[
\begin{align*}
    x & \inset + (\text{size} \times (x+1)) = x\inset + \text{size} \\
    y & \inset + (\text{size} \times (y+1)) = y\inset + 2 \times \text{size}
\end{align*}
\]
def __makeGrid(self):
    """Creates a row x col grid, filled with squares"""
    for x in range(self._cols):
        for y in range(self._rows):
            # create first point
            p1 = Point(self._xInset + self._size * x, self._yInset + self._size * y)
            # create second point
            p2 = Point(self._xInset + self._size * (x + 1), self._yInset + self._size * (y + 1))
            # create rectangle
            r = Rectangle(p1, p2)
            r.setFill("white")
            # add rectangle to graphical window
            r.draw(self._win)

x=0, y=2:
    p1:
    xInset + (size * x) = xInset
    yInset + (size * y) = yInset + 2 * size
    p2:
    xInset + (size * (x+1)) = xInset + size
    yInset + (size * (y+1)) = yInset + 3 * size

We need a window (_win) on which to draw.
def __makeGrid(self):
    """Creates a row x col grid, filled with squares"""
    for x in range(self._cols):
        for y in range(self._rows):
            # create first point
            p1 = Point(self._xInset + self._size * x, self._yInset + self._size * y)
            # create second point
            p2 = Point(self._xInset + self._size * (x + 1), self._yInset + self._size * (y + 1))
            # create rectangle
            r = Rectangle(p1, p2)
            r.setFill("white")
            # add rectangle to graphical window
            r.draw(self._win)

x=1, y=0:
p1:
xInset + (size * x) = xInset + size
yInset + (size * y) = yInset
p2:
xInset + (size * (x+1)) = xInset + 2 * size
yInset + (size * (y+1)) = yInset + size

And so on...
Board class: Getting Started

- Attributes:
  
  ```
  # _win: graphical window on which we will draw our board
  # _xInset: avoids drawing in corner of window
  # _yInset: avoids drawing in corner of window
  # _rows: number of rows in grid of squares
  # _cols: number of columns in grid of squares
  # _size: edge size of each square
  ```

- (We will add a few more attributes later)

- We need to draw the grid, text areas, and buttons

- Might need some helper methods to organize our code

- Now let's **draw the text areas** (we need 3!)
  
  - Text areas are just called Text objects in our graphics package
  
  - We can customize the font size, color, style, and size and call “setText” to add text
Initializing and Drawing the Text Areas

• We’ll add attributes for the text areas: _textArea, _lowerWord, _upperWord

```python
def __initTextAreas(self):
    # initialize text areas
    self._textArea = Text(Point(self._xInset * self._rows + self._size * 2,
                               self._yInset + 50), "")
    self._textArea.setSize(14)
    self._lowerWord = Text(Point(160, 275), "")
    self._lowerWord.setSize(18)
    self._upperWord = Text(Point(160, 25), "")
    self._upperWord.setSize(18)
    self._upperWord.setTextColor("red")

def __drawTextAreas(self):
    """Draw the text area to the right/lower/upper side of main grid""
    
    #initialize before drawing
    self.__initTextAreas()

    # draw main text area (right of grid)
    self._textArea.draw(self._win)

    #draw the text area below grid
    self._lowerWord.draw(self._win)

    #draw the text area above grid
    self._upperWord.draw(self._win)
```
Board class: Getting Started

- Attributes: 
  _win, _rows, _cols, _size, _xInset, _yInset, _textArea, _upperWord, _lowerWord
  - Also _resetButton, _exitButton

- We need to draw the grid, text areas, and buttons
- Might need some helper methods
- Finally, let’s **draw the buttons**!
  - Buttons are just more rectangles…
Initializing and Drawing the Buttons

```python
def __makeResetButton(self):
    """Add a reset button to board""
    self._resetButton = Rectangle(Point(50, 300), Point(130, 350))
    self._resetButton.setFill("white")
    self._resetButton.draw(self._win)
    Text(Point(90, 325), "RESET").draw(self._win)

def __makeExitButton(self):
    """Add exit button to board""
    self._exitButton = Rectangle(Point(170, 300), Point(250, 350))
    self._exitButton.draw(self._win)
    self._exitButton.setFill("white")
    Text(Point(210, 325), "EXIT").draw(self._win)

def drawBoard(self):
    # this creates a row x col grid, filled with squares, including buttons
    self._win.setBackground("white smoke")
    self.__makeGrid()
    self.__makeResetButton()
    self.__makeExitButton()
    self.__drawTextArea()```
Putting it all together
Helper 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?
Helper 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
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.

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

    addStringToTextArea(self, text)
        Add text to text area to right of grid.
        Does not overwrite existing text.

    addStringToUpperText(self, text)
        Add text to text area above grid.
        Does not overwrites 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.

    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!
- (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! (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 properties (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!
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 extras (attributes and/or methods) might be useful?
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 extras (attributes and/or methods) might be useful?
  - Populate grid with **TTTLetters**
  - Check individual **TTTLetters** for X or O
  - Setting individual **TTTLetters** to X or O
  - Check for win (how?)
More next time!