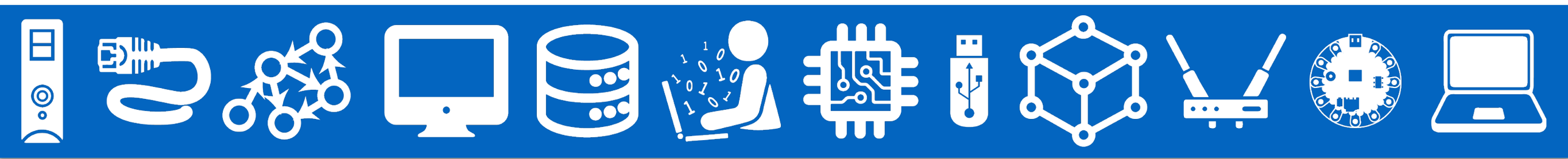
CS134:

Inheritance and Board Class



Announcements & Logistics

- **HW 8** due today @ 10 pm
- **Lab 8** Autocomplete:
  - Focuses on OOP and program design
    - Multiple classes working together
    - Drawing pictures can be helpful!
- **Lab 9** will be a two-week lab: strongly encourage you work in pairs
  - "Mini project" : different from standard labs in length/complexity
  - This week will be a build-up to the ideas used in Lab 9

Do You Have Any Questions?

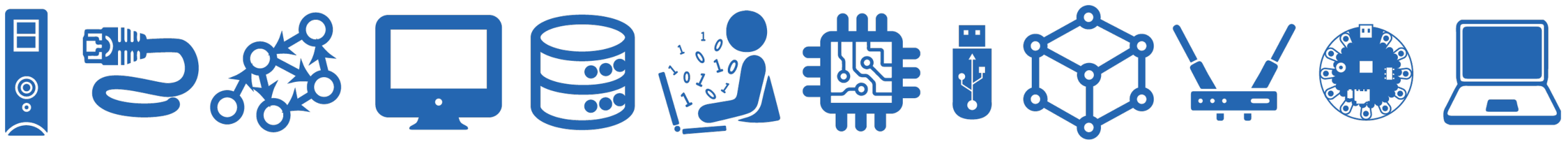
Last Time

- Designed a Library class that stores a sorted shelf of Book objects
- Learnt how to:
    - call `sorted()` function in Python by specifying the `key` function
    - how to *pass a function* as an ***argument*** to another function
    - define/call functions with optional arguments

Today's Plan

- Continue discussing some of the important OOP principles
  - **Abstraction** - handle complexity by ignoring/hiding messy details
  - **Inheritance** - derive a class from another class that shares a set of attributes and methods
  - **Encapsulation** bundling data & methods that work together in a class
  - **Polymorphism** - using a single method or operator for different uses
- Focus on inheritance
- Start implementing a text-based board game

Inheritance Review



Inheritance: Constructor

```
class Rectangle:

    def __init__(self, length, width):

        self._length = length
        self._width = width
```

Parent (super class)

```
class Square(Rectangle):

    def __init__(self, length):

        super().__init__(length, length)
```

Calls **constructor** of super class

Inheritance represents **"is a"** relationship.
A `Square` is a `Rectangle`.

Inheritance: Methods

```
class Rectangle:

    def __init__(self, length, width):
        self._length = length
        self._width = width

    def draw(self):
        print('draws a rectangle')

class Square(Rectangle):

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

    def draw(self):
        print('draws a square')
```

```
sq = Square(12)

sq.draw()
```

calls `draw` of square

```
"draws a square"
```

draw method of `Square` overrides that of `Rectangle`

Inheritance: Methods

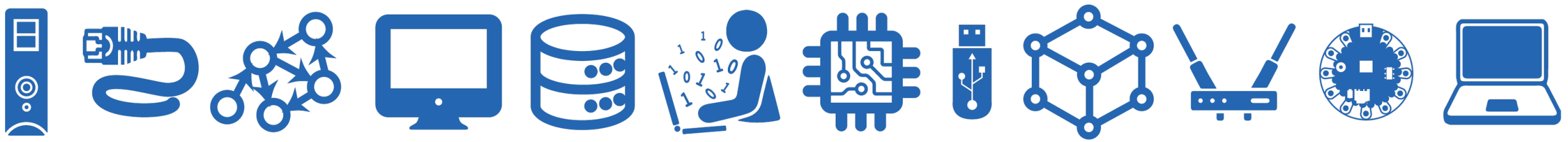
```
class Rectangle:

    def __init__(self, length, width):
        self._length = length
        self._width = width

    def draw(self):
        print('draws a rectangle')

class Square(Rectangle):

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

    def draw(self):
        print('draws a square')
```

```
sq = Square(12)

sq.draw()
```

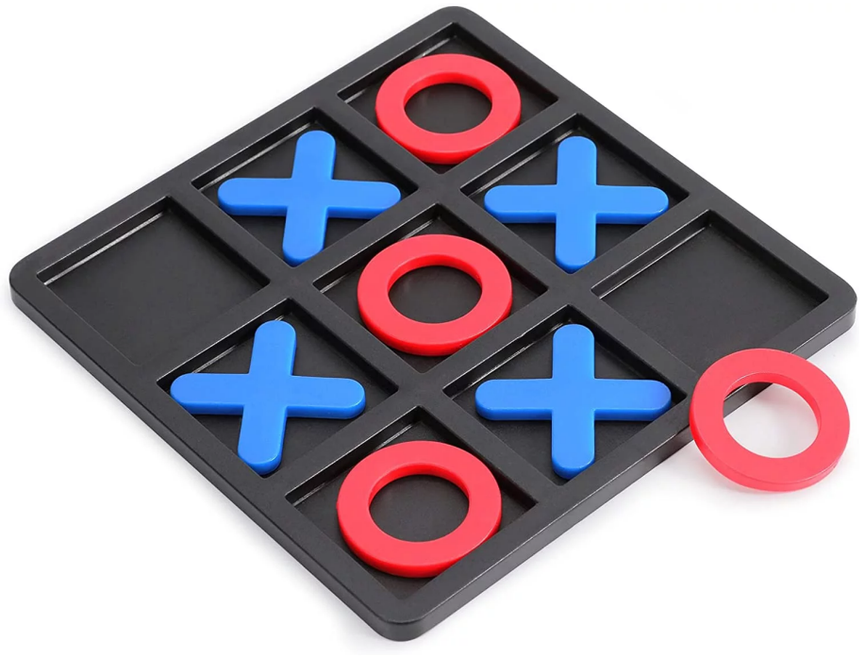
```
"draws a rectangle"
```

If `Square` has no `draw` method, it calls draw of `super` class

Inheritance and OOP: word-based board games

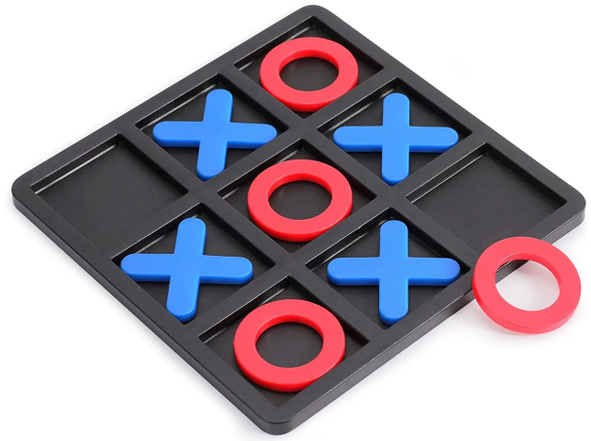


Simple Board Games

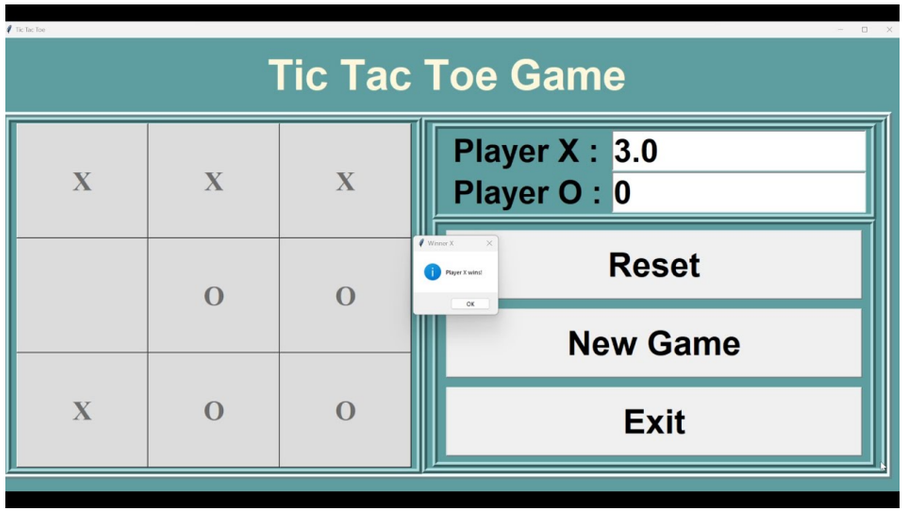


Common Features of Physical Game?

- Often 2 or many player
- Board at the bottom
    - Grid-based (rows and columns)
- Game pieces (tiles/cubes)
    - Go "on top" of the board
    - Have a letter (or many letters) on them
- Some uncertainty is part of the fun
    - Randomness in the configurations
- Winning configuration
- May or may not be timed



Computer Variants



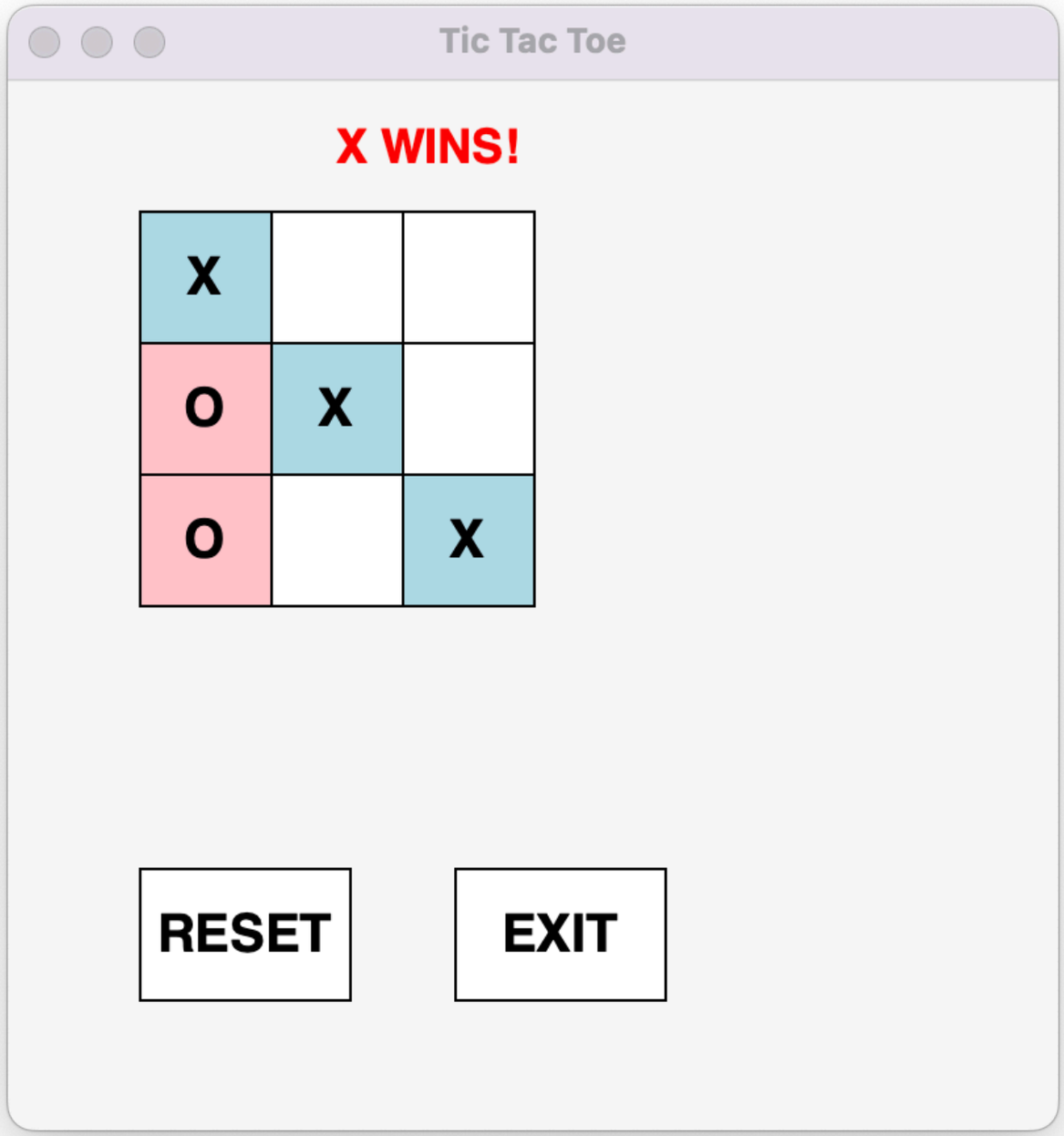
Common Features of Computer Variants?

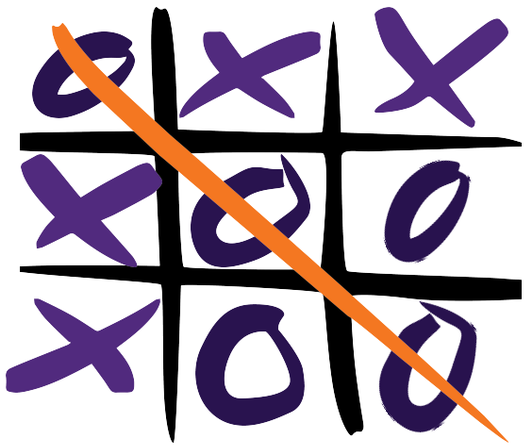
- Often 1 player (or play with computer)
- Game board: now a graphical screen
  - A grid area to place the pieces
  - Text areas on the sides to give game status
  - "Buttons" to reset/exit game
- Some uncertainty is part of the fun
  - Randomness in the configurations
- May or may not be timed
- Other features???

Example: Tic Tac Toe

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

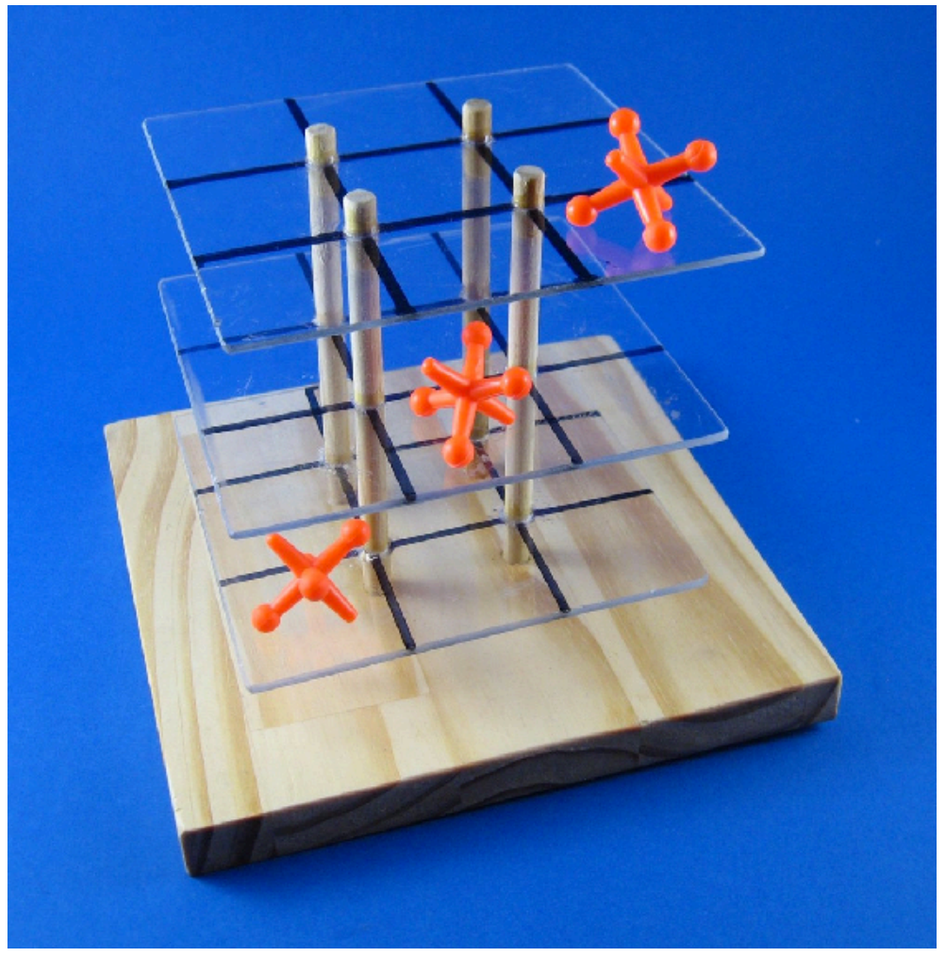
```
>>> python3 tttgame.py
```





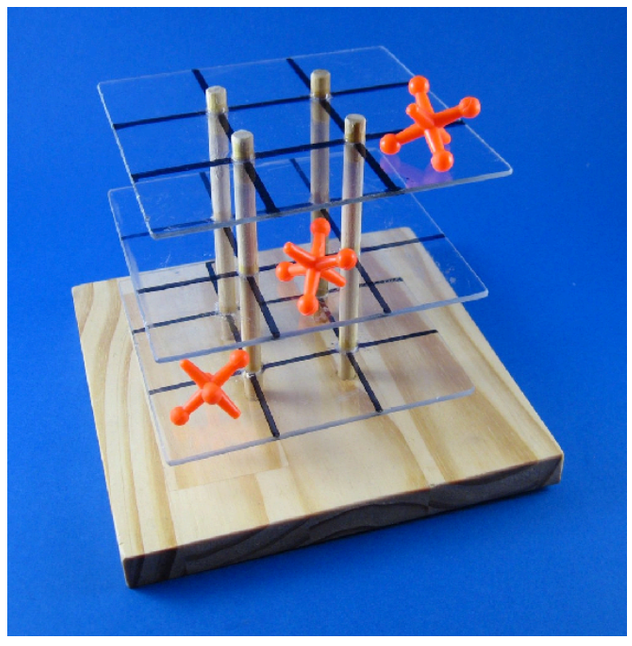
Decomposition

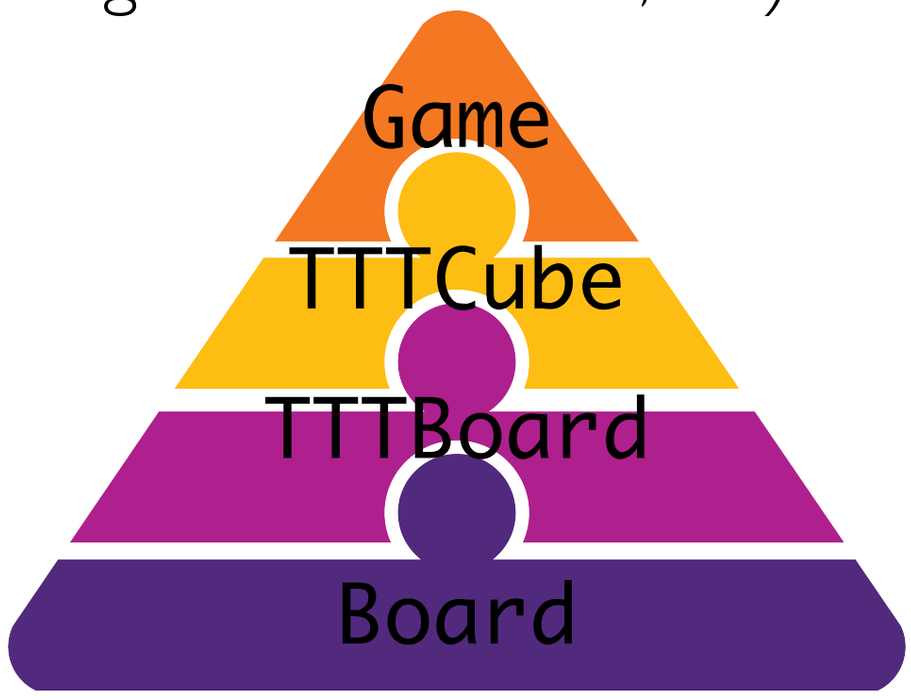
- Let's try to identify the "layers" of this game
- Through abstraction and encapsulation, each layer can ignore what's happening in the other layers
- What are the layers of Tic Tac Toe?



Decomposition

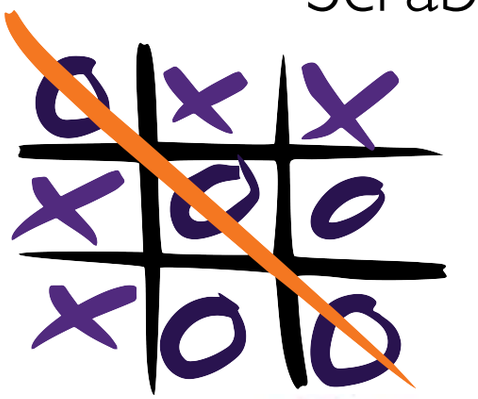
- Bottom layer: **Basic board** w/buttons, text areas, mouse click detection (not specific to Tic Tac Toe!)
- Lower middle layer: Extend the **basic board with Tic Tac Toe specific features** (3x3 grid, of TTTCubes, initial board state: all letters start blank)
- Upper middle layer: **Tic Tac Toe "cubes" or "letters"** (9 in total!); set text to X or O
- Top layer: **Game logic** (alternating turns, checking for valid moves, etc)

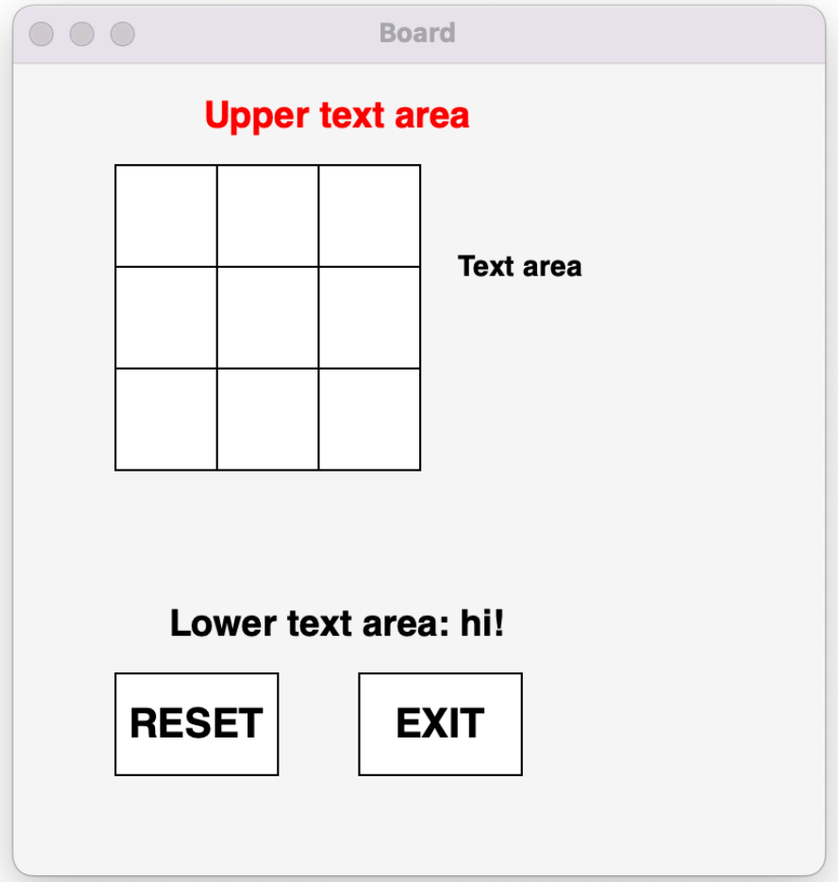




Board class

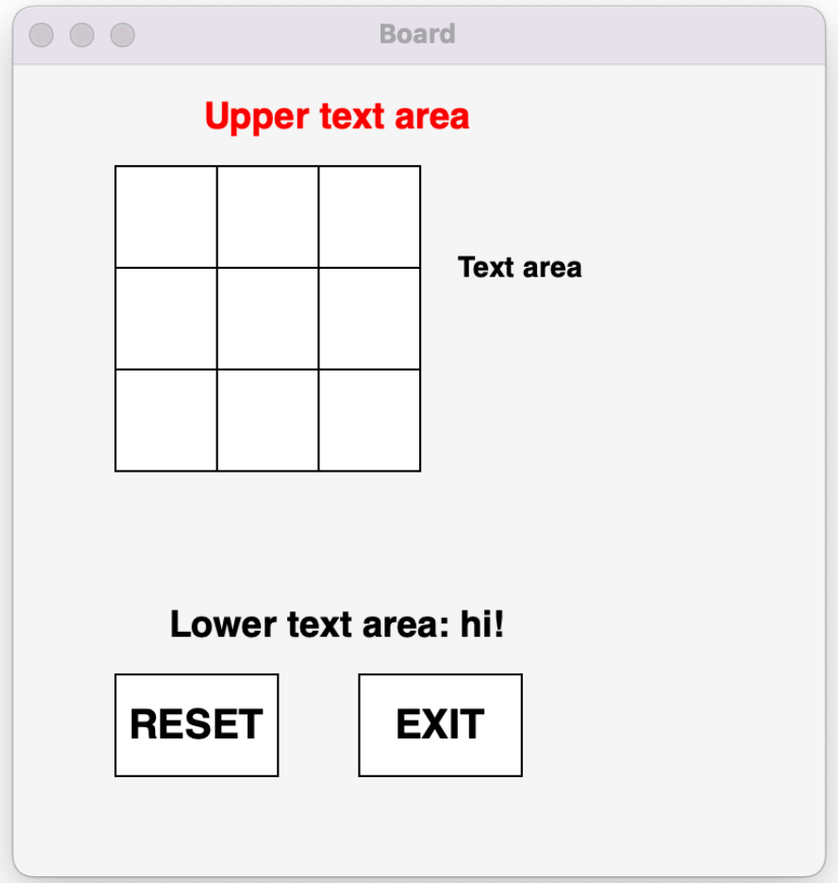
- Let's start at the bottom: Board class
- What are basic features of all game boards?
    - Think generally…many board-based games have the similar basic requirements
    - (For example, Boggle, TicTacToe, Scrabble, etc)





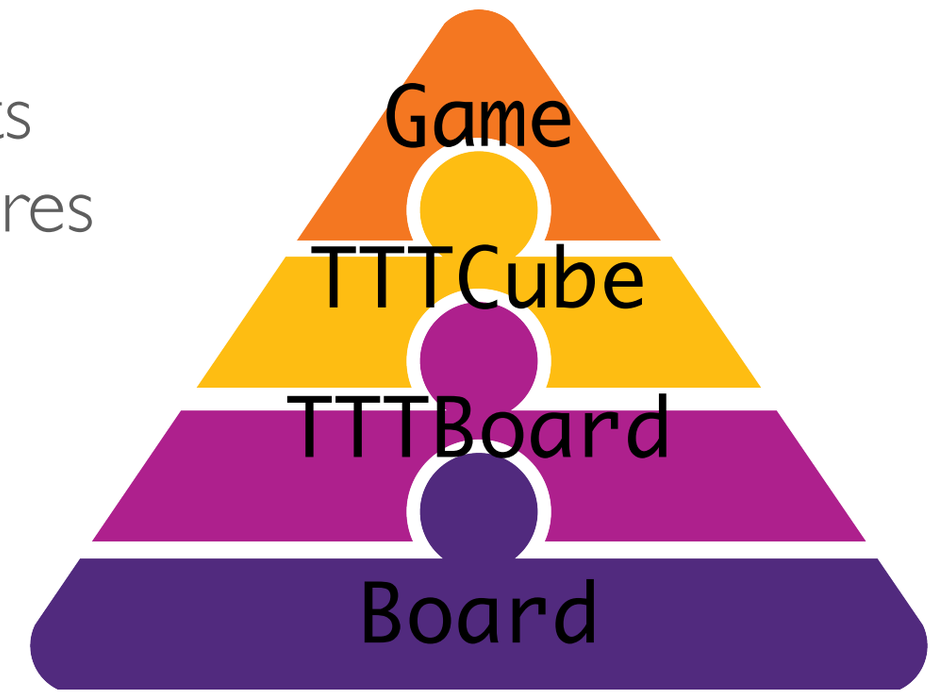
Board class

- 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

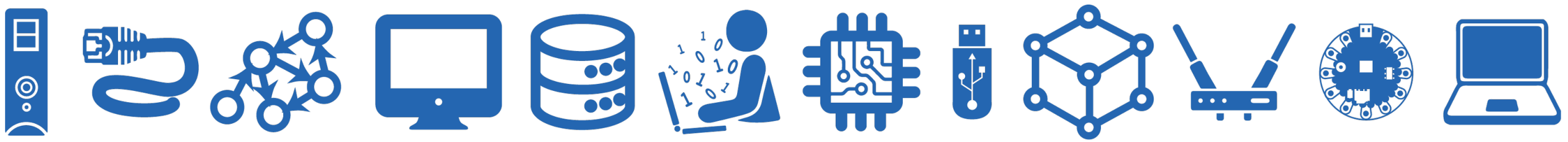


Inheritance

- Board (super class)
  - **Generic board** w/buttons, text areas, mouse click detection
- Tic Tac Toe Board (sub class)
  - Inherits from Board and extends it to TTT specific features and methods
    - Doesn't have to recreate a Board from scratch
- Looking ahead: Boggle (Lab 9)
  - Similar grid-based board game, also inherits from Board and extends it to Boggle features and methods



Graphics Module

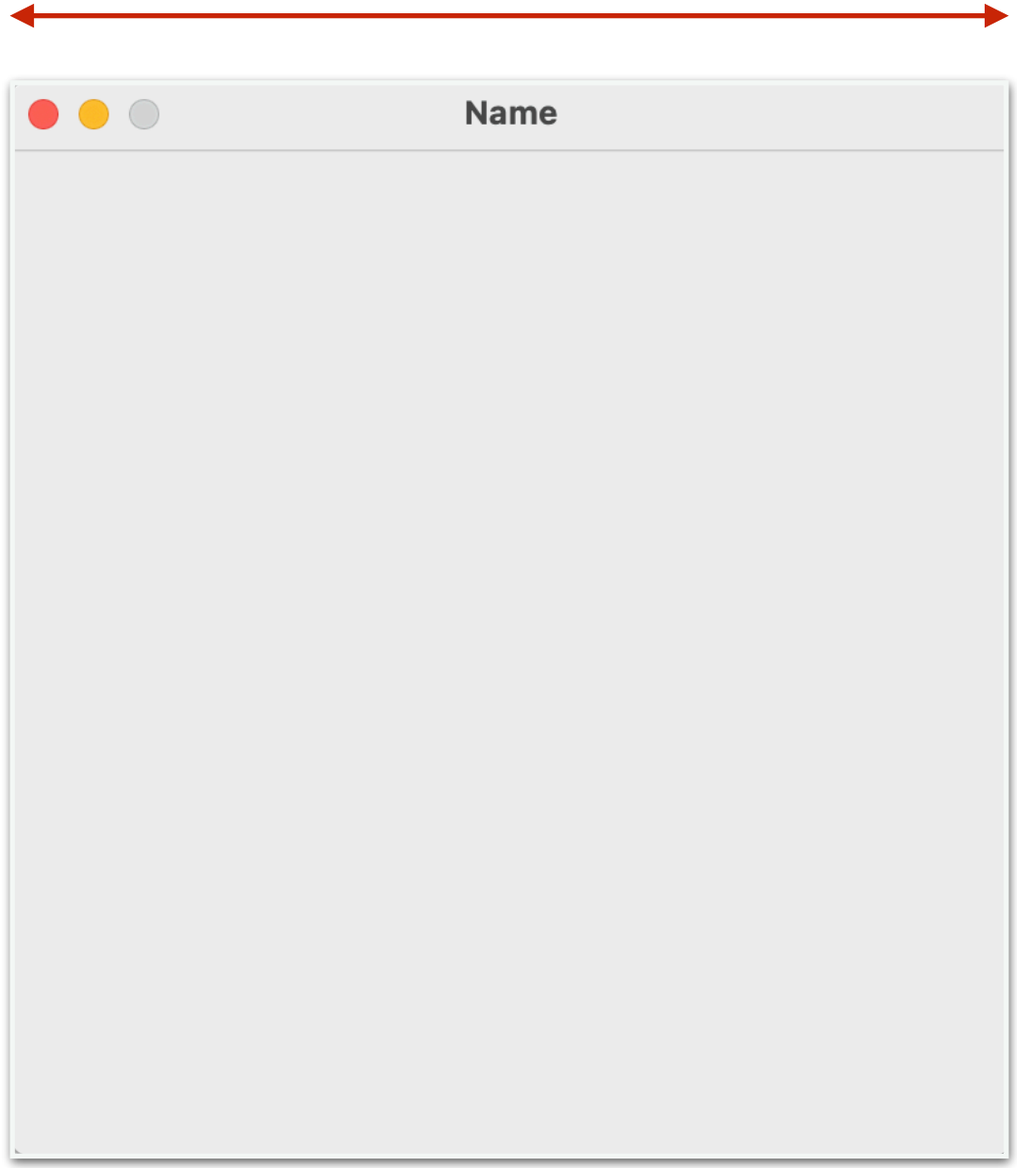


Graphics Package for Board

```
>>> from graphics import *
>>> # takes title and size of window
>>> win = GraphWin("Name", 400, 400)
```

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)



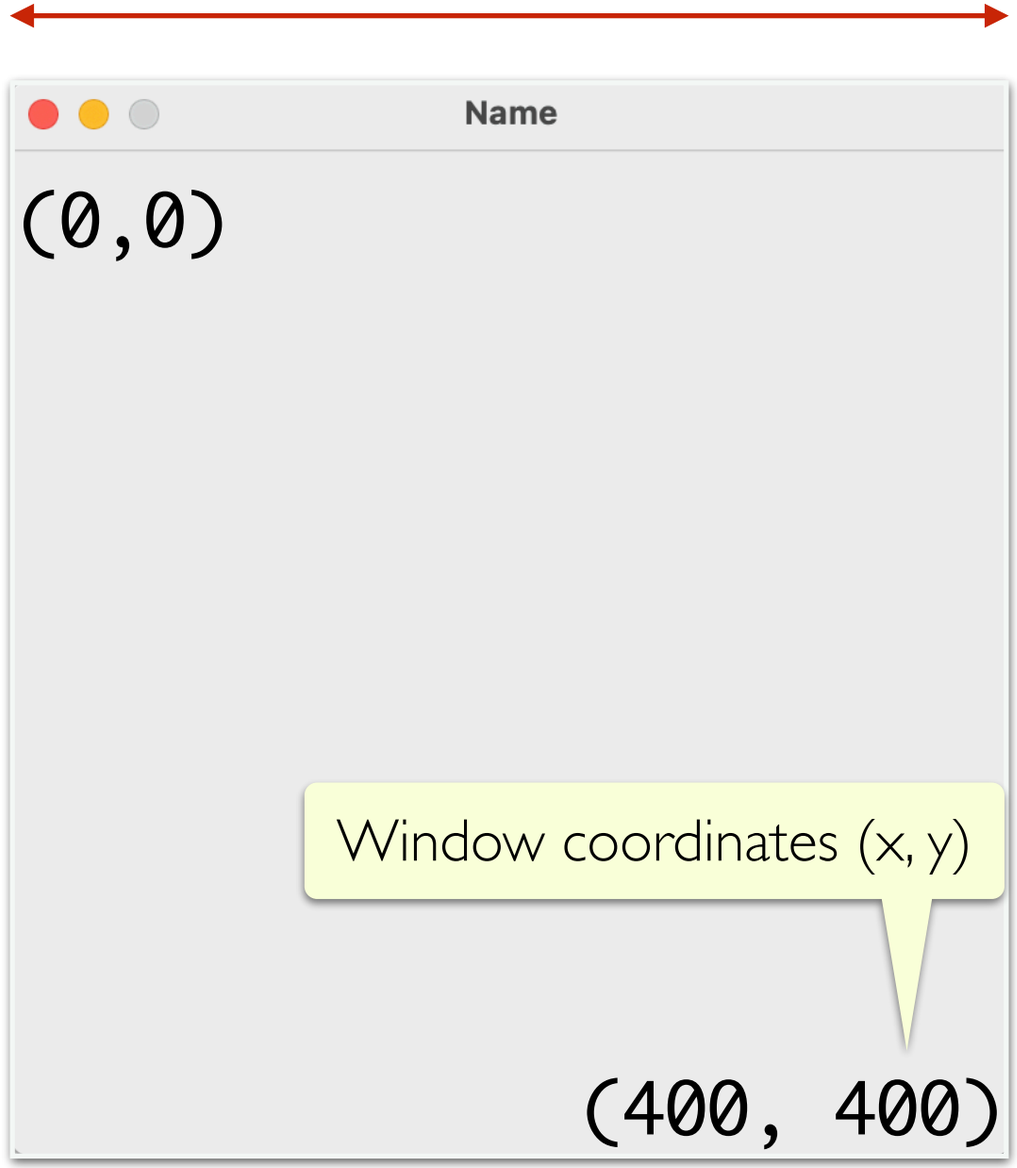
A **pixel** is one of the small dots or squares that make up an image on a computer screen.

Graphics Package for Board

```
>>> from graphics import *
>>> # takes title and size of window
>>> win = GraphWin("Name", 400, 400)
```

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

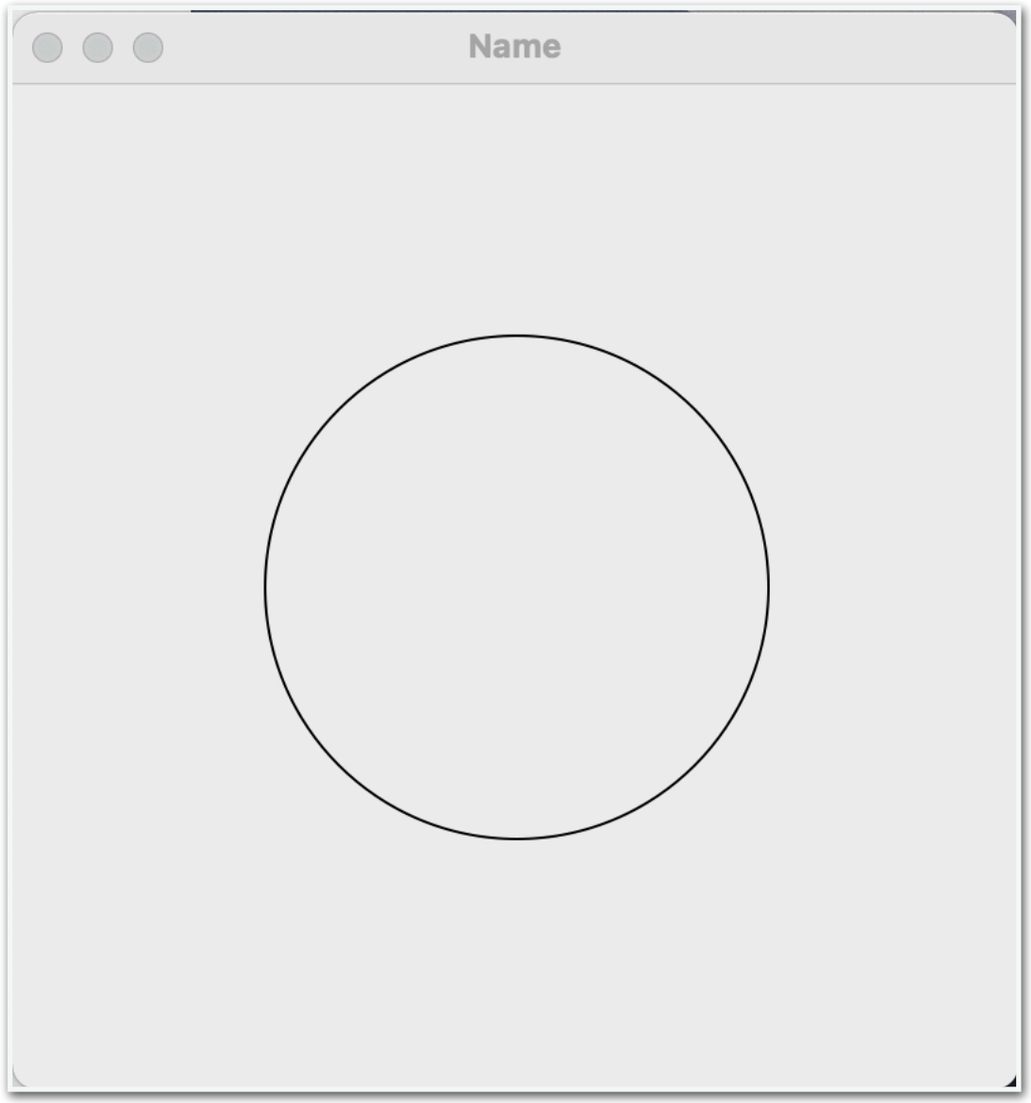
Create a window with title "Name" and size 400x400 (measured in pixels)



A **pixel** is one of the small dots or squares that make up an image on a computer screen.

Graphics Package for Board

```
>>> # create point obj at x,y coordinate in window
>>> pt = Point(200, 200)
>>> # create circle w center at pt and radius 100
>>> c = Circle(pt, 100)
>>> # draw the circle on the window
>>> c.draw(win)
Circle(Point(200.0, 200.0), 100)
```

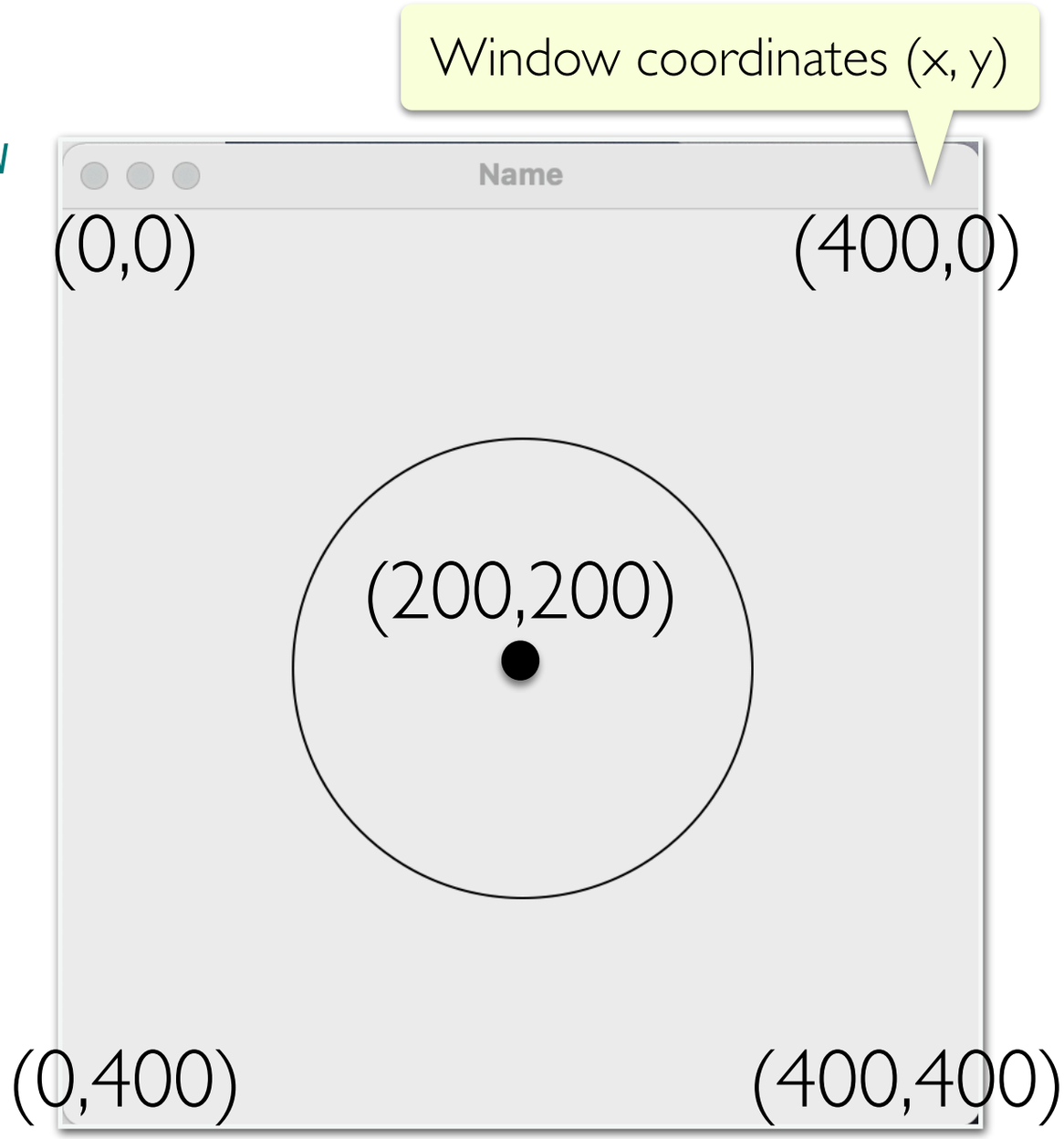


Graphics Package for Board

```
>>> # create point obj at x,y coordinate in window
>>> pt = Point(200, 200)
>>> # create circle w center at pt and radius 100
>>> c = Circle(pt, 100)
>>> # draw the circle on the window
>>> c.draw(win)
Circle(Point(200.0, 200.0), 100)
```

We can draw other shapes as well.

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

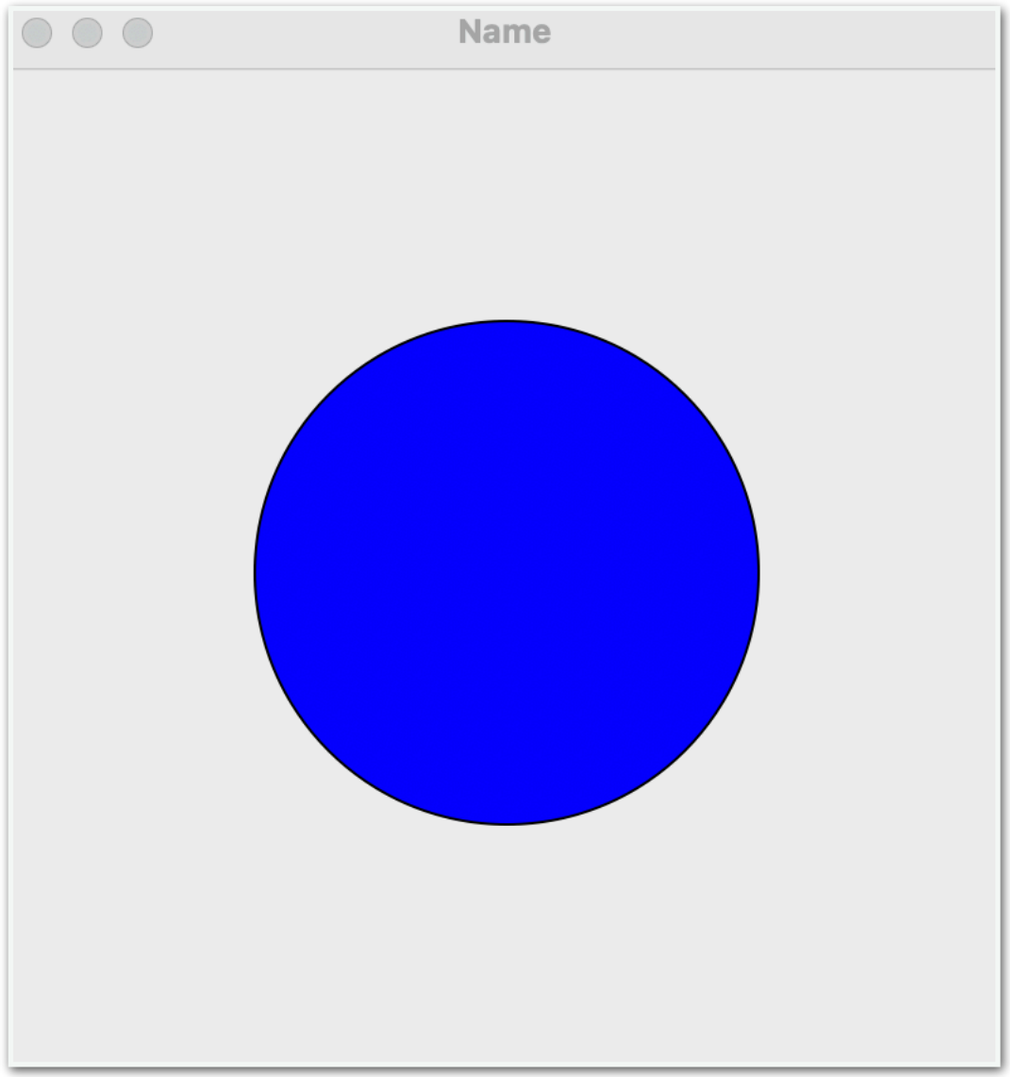


Graphics Package for Board

```
>>> # set color to blue
>>> c.setFill("blue")
>>> # Pause to view result
>>> win.getMouse()
Point(76.0, 322.0)
>>> # close window when done
>>> win.close()
```

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

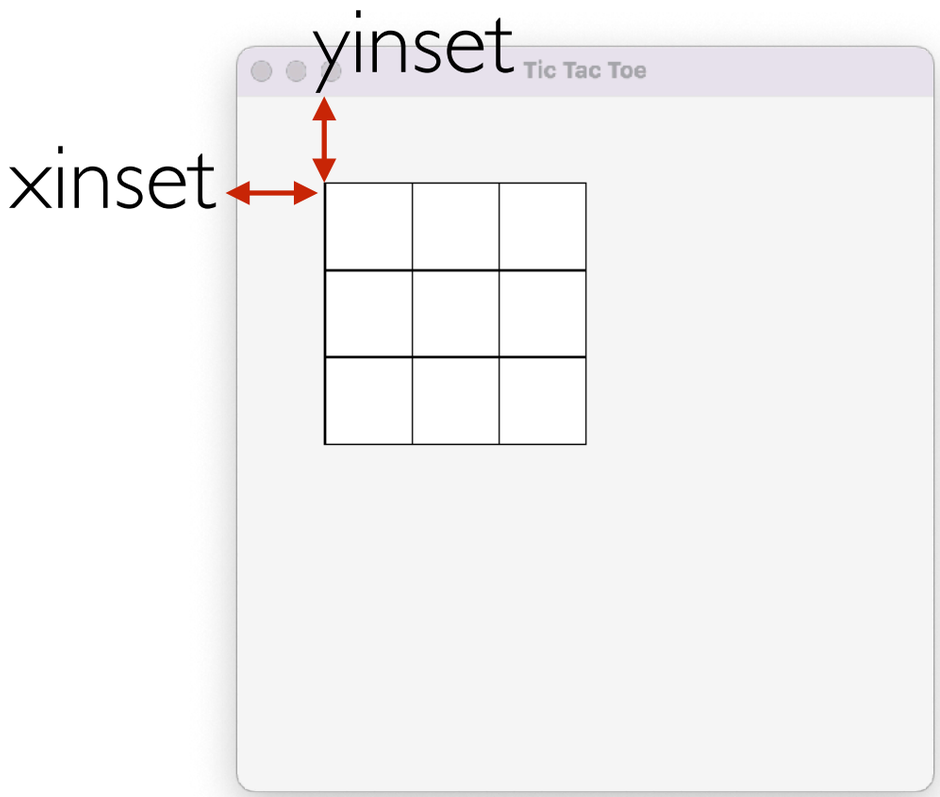
Board Class: Getting Started

- Attributes:
  - Graphical stuff:

```
# _win: graphical window on which we will draw our board
# _xinset: avoids drawing in corner of window
# _yinset: avoids drawing in corner of window
# _size: edge size of each square
```

  - Grid stuff:

```
# _rows: number of rows in grid of squares
# _cols: number of columns in grid of squares
```



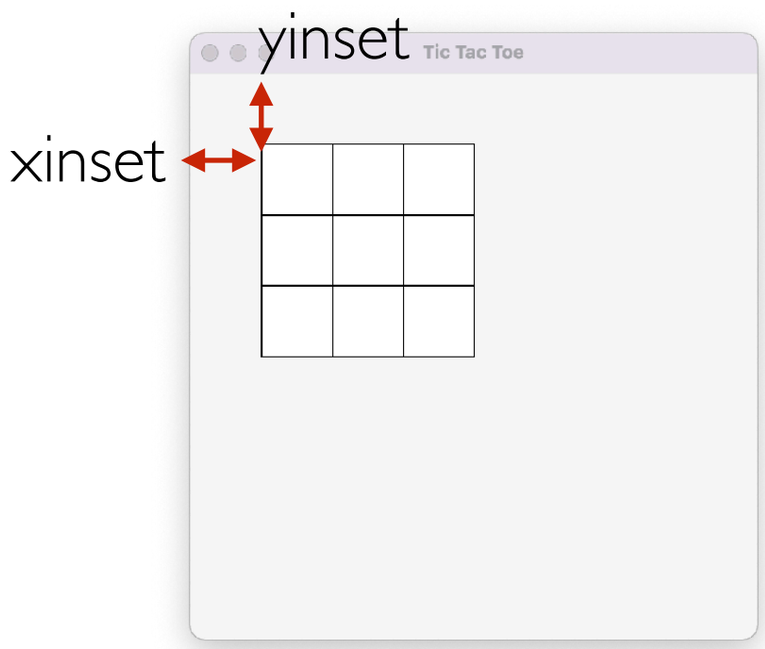
- (We will add a few more attributes later)
- Need constructor and getters
- Need to make the **grid**, **text areas**, and **buttons**
- Might need some helper methods to organize our code

Board Class: __init__ and getters

```
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
    # _grid: _ros x _cols grid of text rectangles for displaying letters
    # _size: edge size of each square

    __slots__ = [ '_xinset', '_yinset', '_rows', '_cols', '_size', \
                  '_grid', '_win', '_exit_button', '_reset_button', \
                  '_text_area', '_lower_word', '_upper_word']

    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
        self._grid = []
        self.make_board()
```

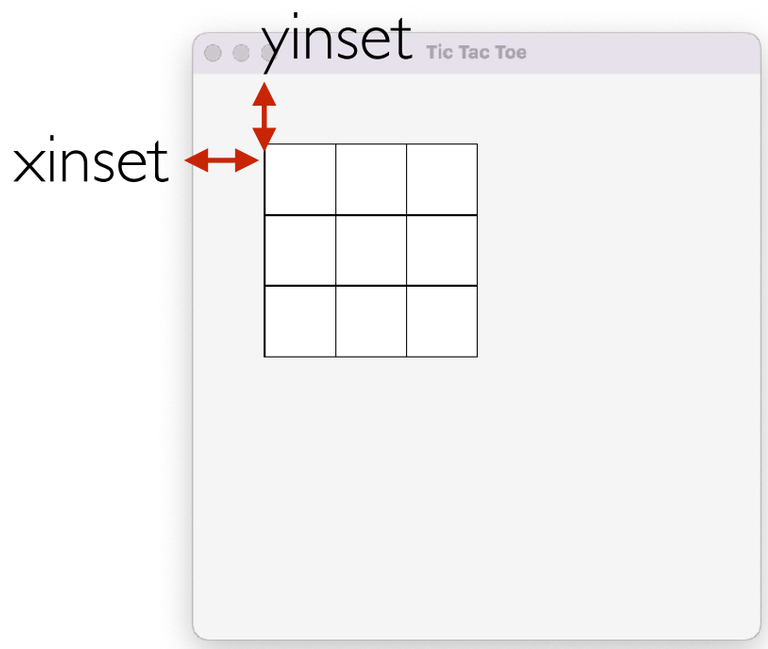
Notice the default values

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

    def get_xinset(self):
        return self._xinset

    def get_yinset(self):
        return self._yinset

    def get_rows(self):
        return self._rows

    def get_cols(self):
        return self._cols

    def get_size(self):
        return self._size

    def get_board(self):
        return self
```



Board class: Making the grid

- What are the features of the grid?
    - rows x columns num of rectangles of a certain size
    - offset from the top-left corner
- Each rectangle in the grid:
    - may need to have some text on it for specific games
    - but we'll leave it blank for now...



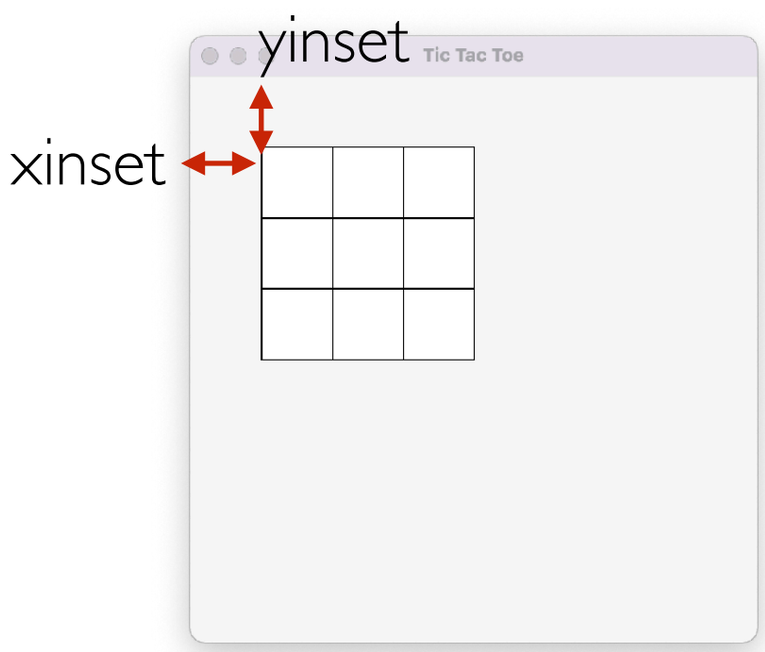
Board class: Making the grid

```
def _make_rect(self, point1, point2, fillcolor="white", text="", textcolor="black"):
    """Creates a rectangle with text in the center"""
    rect = Rectangle(point1, point2, fillcolor)

    text = Text(rect.getCenter(), text)
    text.setTextColor(textcolor)

    return TextRect(rect, text)
```

makes one TextRect at coordinates

```
def __make_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(TODO):
        for c in range(TODO):
            # create first point
            p1 = TODO
            # create second point
            p2 = TODO
            # create rectangle and add to graphical window
            self._grid[r][c] = self._make_textrect(p1, p2)
```

how many rectangles for the grid?

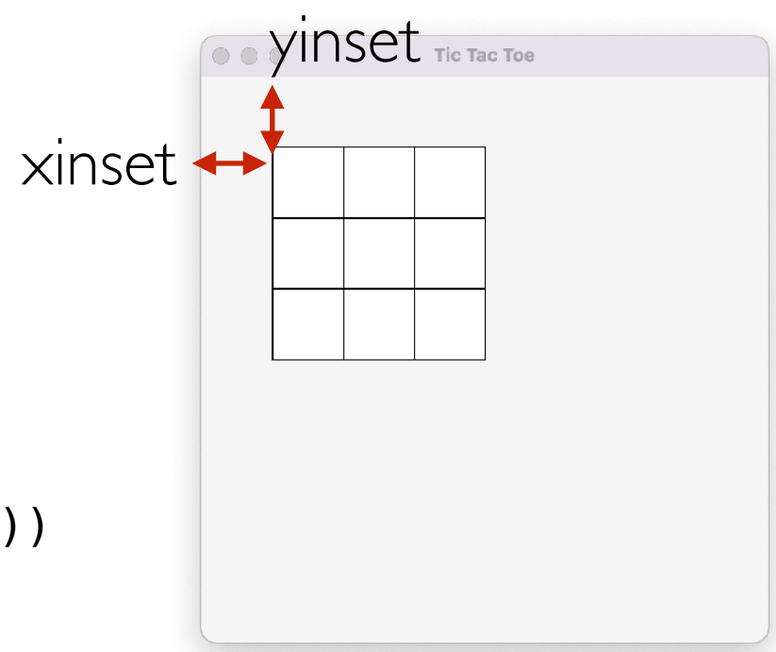


Board class: Making the grid

```
def _make_rect(self, point1, point2, fillcolor="white", text="", textcolor="black"):
    """Creates a rectangle with text in the center"""
    rect = Rectangle(point1, point2, fillcolor)

    text = Text(rect.getCenter(), text)
    text.setTextColor(textcolor)

    return TextRect(rect, text)
```

makes one TextRect at coordinates

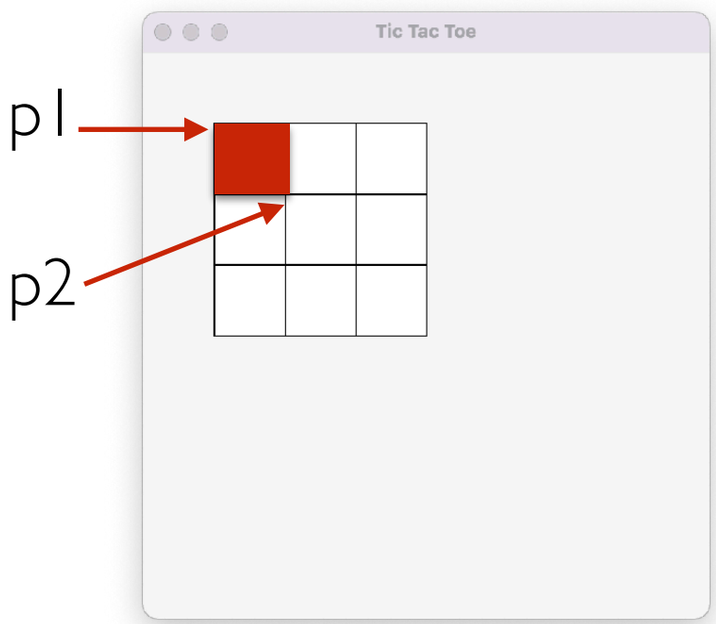
```
def __make_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(self._rows):
        self._grid.append([])
        for c in range(self._cols):
            # create first point
            p1 = Point(self._xinset + self._size * c,
                       self._yinset + self._size * r)
            # create second point
            p2 = Point(self._xinset + self._size * (c + 1),
                       self._yinset + self._size * (r + 1))
            # create rectangle and add to graphical window
            self._grid[r].append(self._make_textrect(p1, p2))
```



Board class: Making the grid

```
def __draw_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(self._rows):
        self._grid.append([])
        for c in range(self._cols):
            # create first point
            p1 = Point(self._xinset + self._size * c,
                       self._yinset + self._size * r)
            # create second point
            p2 = Point(self._xinset + self._size * (c + 1),
                       self._yinset + self._size * (r + 1))
            # create rectangle and add to graphical window
            self._grid[r].append(self._make_textrect(p1, p2))
```

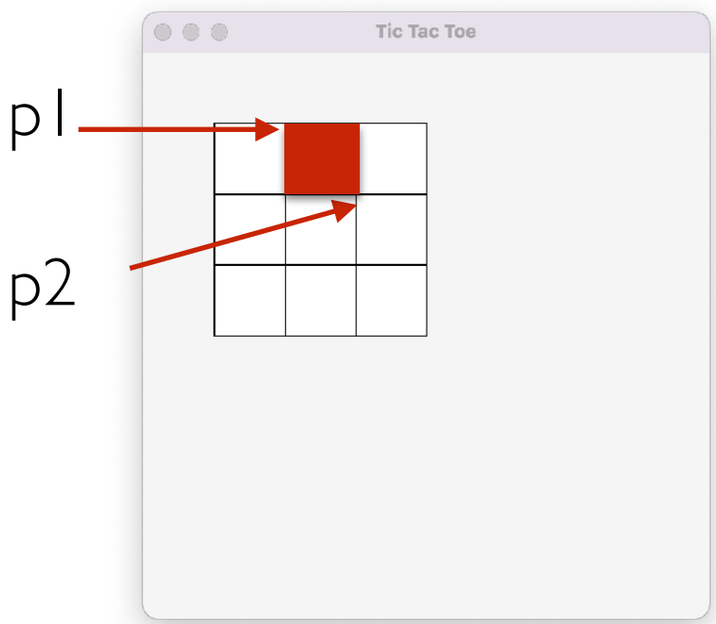
```
r=0, c=0:
  p1:
  xInset + (size * c) = xInset
  yInset + (size * r) = yInset
  p2:
  xInset + (size * (c+1)) = xInset + size
  yInset + (size * (r+1)) = yInset + size
```



Board class: Making the grid

```
def __draw_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(self._rows):
        self._grid.append([])
        for c in range(self._cols):
            # create first point
            p1 = Point(self._xinset + self._size * c,
                       self._yinset + self._size * r)
            # create second point
            p2 = Point(self._xinset + self._size * (c + 1),
                       self._yinset + self._size * (r + 1))
            # create rectangle and add to graphical window
            self._grid[r].append(self._make_textrect(p1, p2))
```

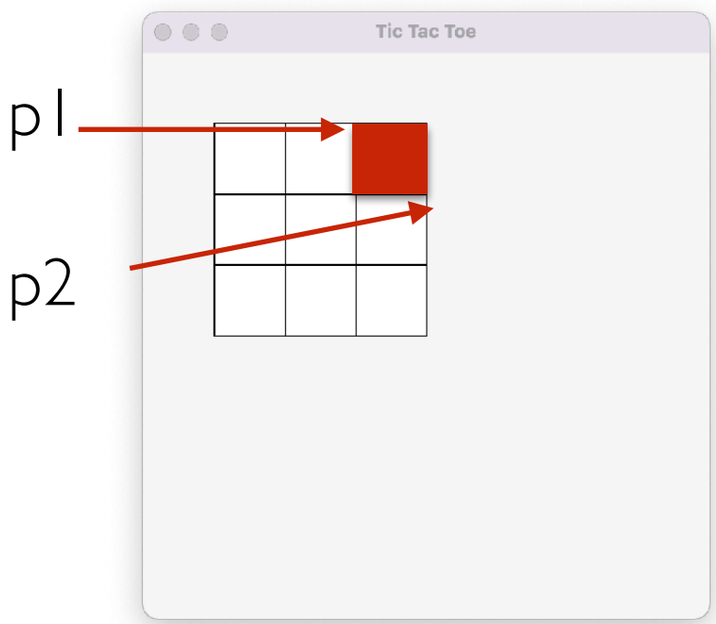
```
r=0, c=1:
  p1:
  xInset + (size * c) = xInset + size
  yInset + (size * r) = yInset
  p2:
  xInset + (size * (c+1)) = xInset + 2 * size
  yInset + (size * (r+1)) = yInset + size
```



Board class: Making the grid

```
def __draw_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(self._rows):
        self._grid.append([])
        for c in range(self._cols):
            # create first point
            p1 = Point(self._xinset + self._size * c,
                       self._yinset + self._size * r)
            # create second point
            p2 = Point(self._xinset + self._size * (c + 1),
                       self._yinset + self._size * (r + 1))
            # create rectangle and add to graphical window
            self._grid[r].append(self._make_textrect(p1, p2))
```

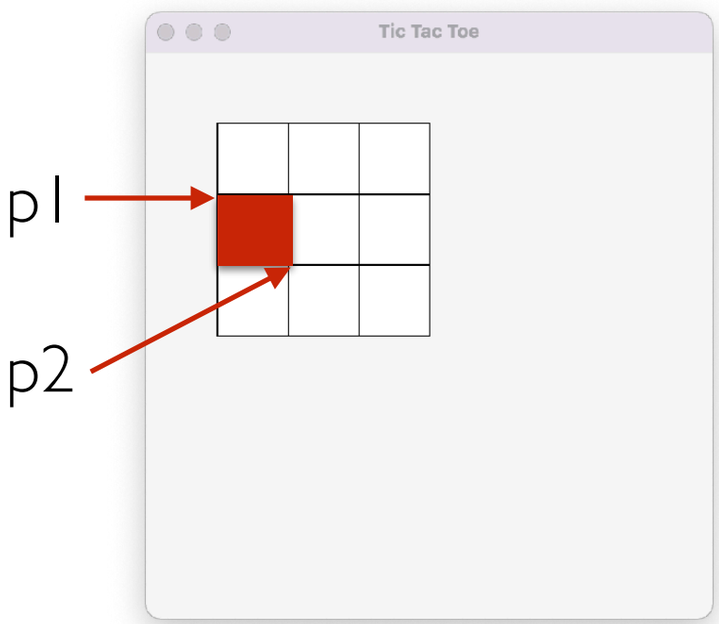
```
r=0, c=2:
  p1:
  xInset + (size * c) = xInset + 2 * size
  yInset + (size * r) = yInset
  p2:
  xInset + (size * (c+1)) = xInset + 3 * size
  yInset + (size * (r+1)) = yInset + size
```



Board class: Making the grid

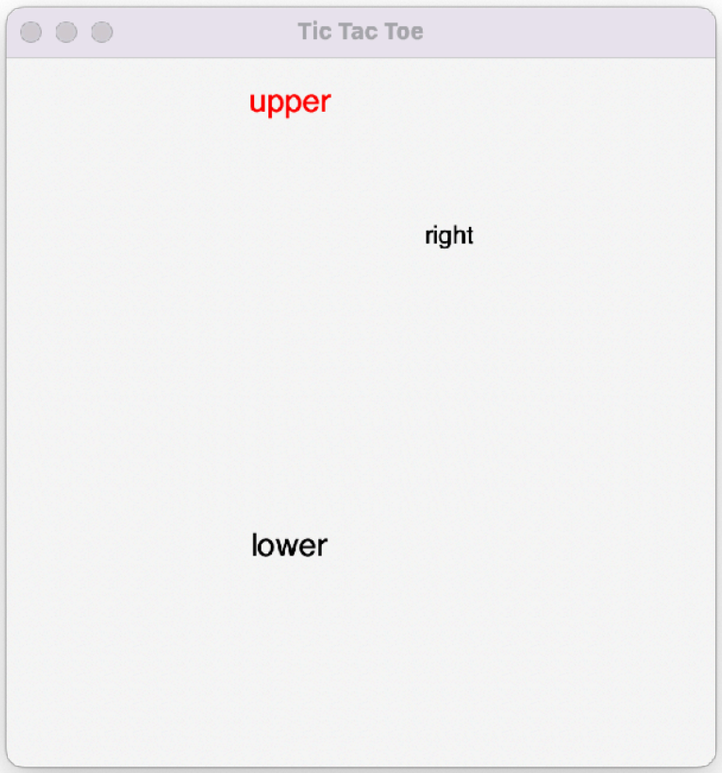
```
def __draw_grid(self):
    """Creates a row x col grid, filled with empty squares"""
    for r in range(self._rows):
        self._grid.append([])
        for c in range(self._cols):
            # create first point
            p1 = Point(self._xinset + self._size * c,
                       self._yinset + self._size * r)
            # create second point
            p2 = Point(self._xinset + self._size * (c + 1),
                       self._yinset + self._size * (r + 1))
            # create rectangle and add to graphical window
            self._grid[r].append(self._make_textrect(p1, p2))
```

```
c=0, r=1:
  p1:
  xInset + (size * c) = xInset
  yInset + (size * r) = yInset + size
  p2:
  xInset + (size * (c+1)) = xInset + size
  yInset + (size * (r+1)) = yInset + 2 * size
```



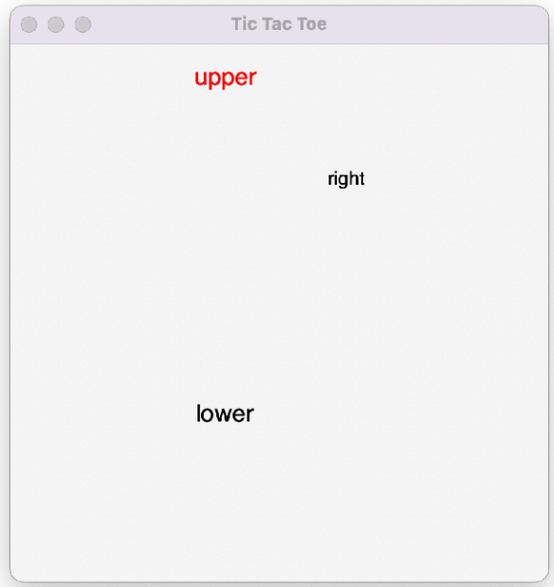
Board class: Text Areas

- We need to make 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
    - Can customize the font size, color, style, and size and call "`setText`" to add text



Board class: Making the Text Areas

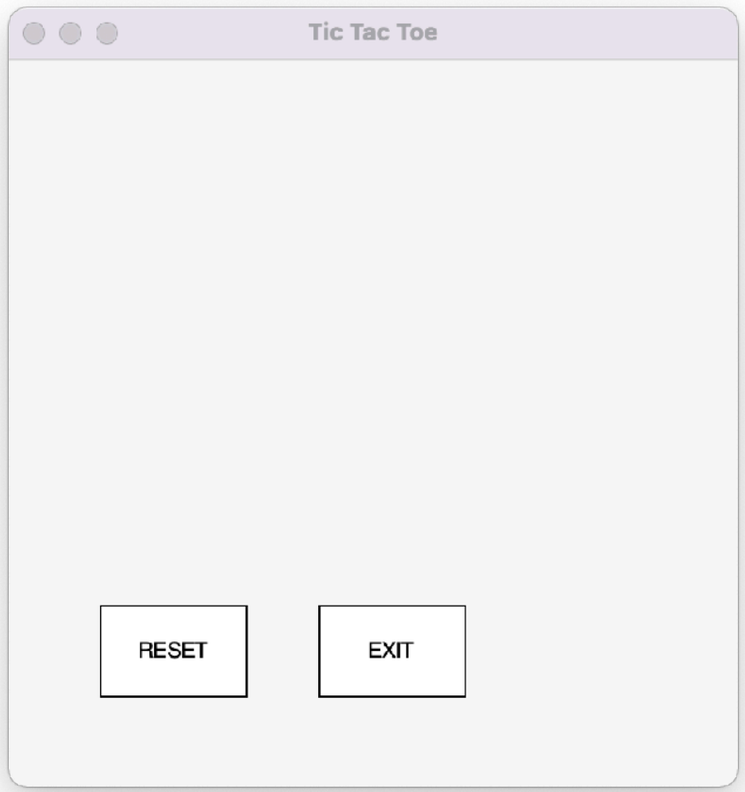
- We'll add attributes for the text areas:
  `_text_area, _lower_word, _upper_word`




```
def __make_text_area(self, point, fontsize=18, color="black", text=""):
    """Creates a text area"""
    text_area = Text(point, text)
    text_area.setSize(fontsize)
    text_area.setTextColor(color)
    text_area.setStyle("normal")
    return text_area


def __make_text_areas(self):
    """Make text areas in the right/lower/upper side of main grid"""
    # main text area (right of grid)
    self._text_area = self.__make_text_area(Point(self._xinset * self._rows + self._size * 2,
                                                  self._yinset + 50), 14)
    # the text area below grid
    self._lower_word = self.__make_text_area(Point(160, 275))
    # the text area above grid
    self._upper_word = self.__make_text_area(Point(160, 25), color="red")
```

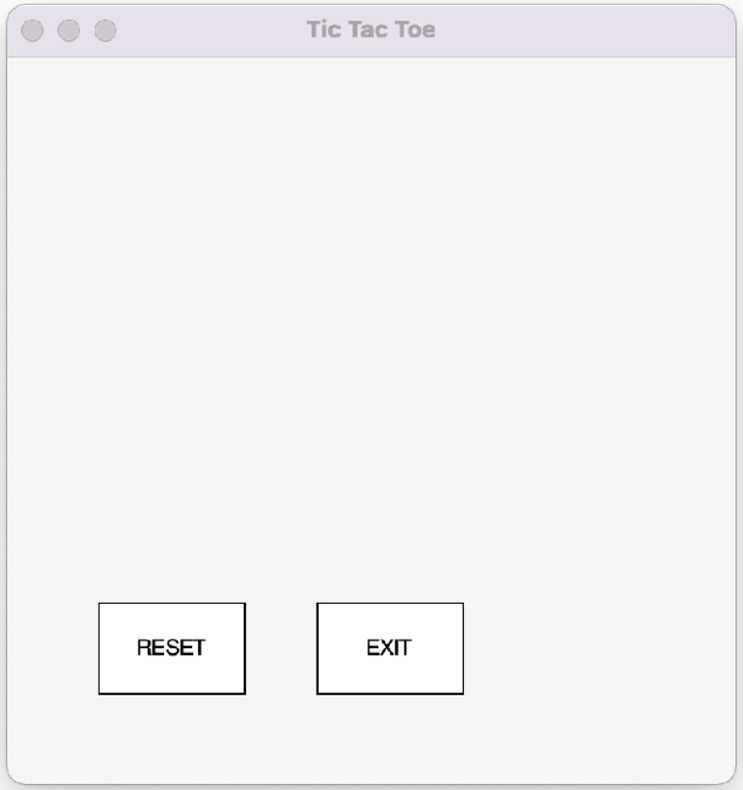
Board class: Buttons

- We need to make the **grid**, **text areas**, and **buttons**
- Might need some helper methods to organize our code
- Finally, let's **draw the buttons**!
  - Buttons are just more text rectangles…



Board class: Making Buttons

- Buttons are just rectangles with text on them



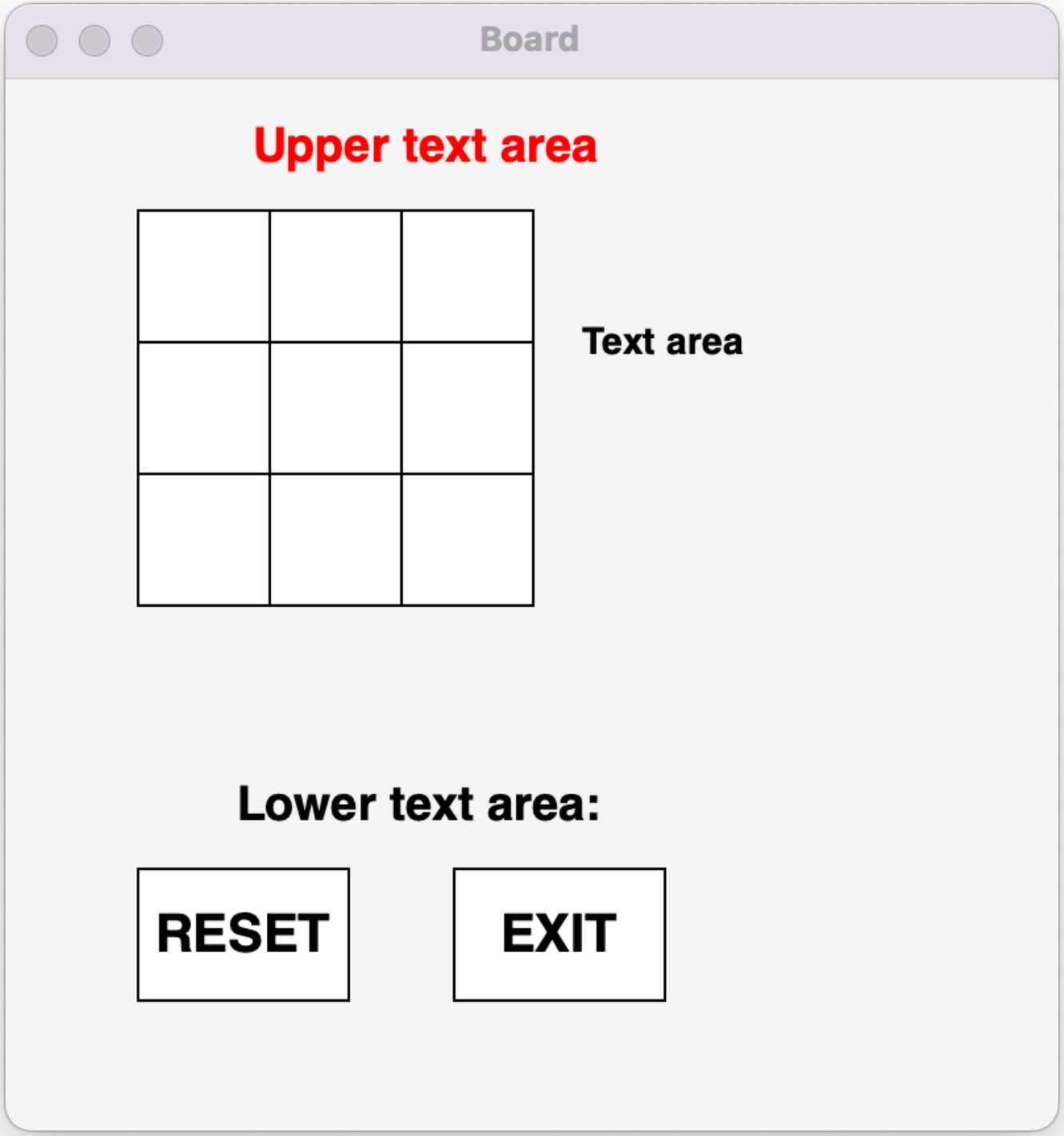

```
def __make_buttons(self):
    """Create reset and exit buttons"""
    p1 = Point(50, 300); p2 = Point(130, 350)
    self._reset_button = self._make_textrect(p1, p2, text="RESET")
    p3 = Point(170, 300); p4 = Point(250, 350)
    self._exit_button = self._make_textrect(p3, p4, text="EXIT")
```

Board class: Drawing Graphics

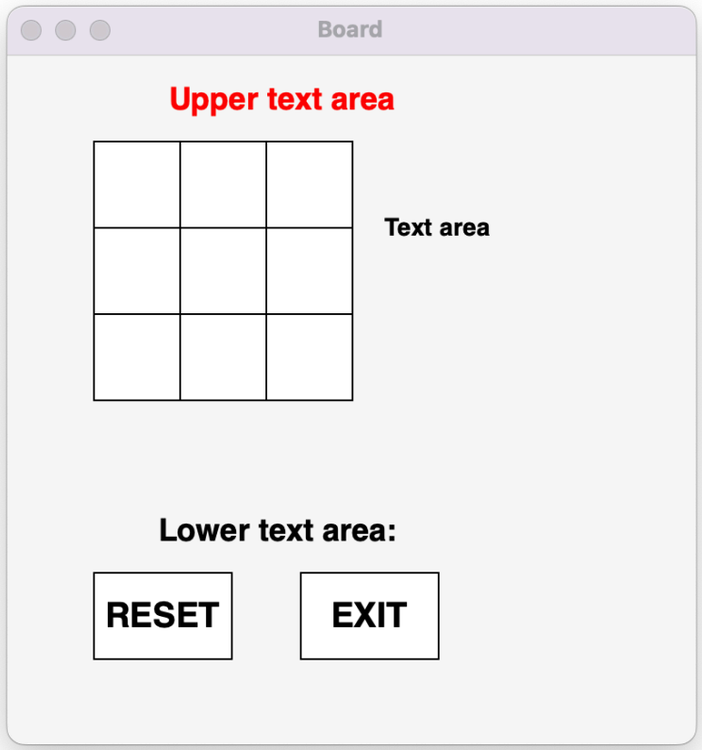
- Graphics objects do not appear until they are drawn on the window
  - Each Graphics class has a draw() method that takes a single argument: the Graphics window

```
def __draw_buttons(self):
    """Draw reset and exit buttons"""
    self._reset_button.draw(self._win)
    self._exit_button.draw(self._win)

def __draw_text_areas(self):
    """Draw text areas"""
    self._text_area.draw(self._win)
    self._lower_word.draw(self._win)
    self._upper_word.draw(self._win)

def __draw_grid(self):
    """Draw grid squares"""
    for r in range(self._rows):
        for c in range(self._cols):
            self._grid[r][c].draw(self._win)
```

Putting it all together

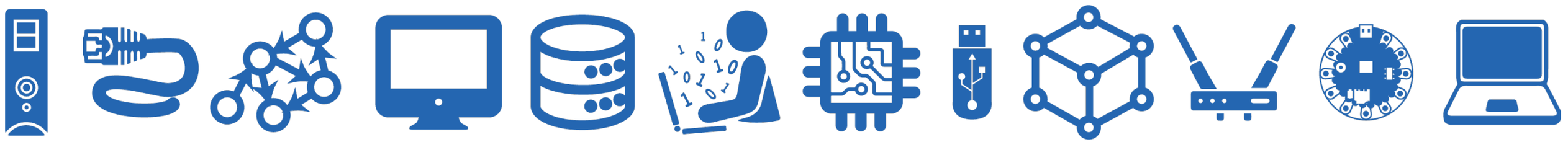


Putting it all together




```
def make_board(self):
    """Create the board with grid, text areas, and buttons"""
    self.__make_grid()
    self.__make_text_areas()
    self.__make_buttons()

def draw_board(self):
    """Create the board with the grid, text areas, and buttons"""
    self._win.setBackground("white smoke")
    self.__draw_grid()
    self.__draw_text_areas()
    self.__draw_buttons()
```

Board Helper 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?

Helper Methods

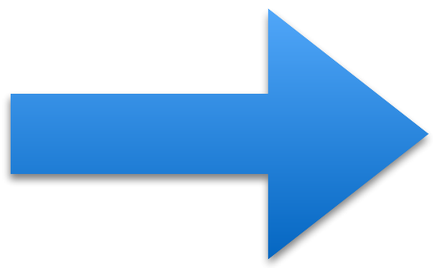
- Now that we have a board with a grid, buttons, and text areas, it would be useful to define some methods for interacting with these objects
- Helpful methods?
    - Get grid coordinate of mouse click
    - Determine if click was in grid, reset, or exit buttons
    - Set text to one of 3 text areas
    - …

- Note that none of this is specific to Tic Tac Toe (yet)!
- Always good to start general and then get more specific

Helper Methods

```
>>> pydoc3 board
```

Public methods!



```
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.
 |
 |  draw_board(self)
 |      Create the board with the grid, text areas, and buttons
 |
 |  get_board(self)
 |
 |  get_cols(self)
 |
 |  get_grid_cell(self, row, col)
 |
 |  get_position(self, point)
 |      Converts a window location (Point) to a grid position (tuple).
 |      Note: Grid positions are always returned as row, col
 |      Negative row or column values may be returned, indicating
 |      that point falls outside of the grid area
 |
 |  get_rows(self)
 |
 |  get_size(self)
 |
 |  get_string_from_lower_text(self)
 |      Get text from text area below grid.
 |
 |  get_string_from_text_area(self)
 |      Get text from text area to right of grid.
 |
 |  get_string_from_upper_text(self)
 |      Get text from text area above grid.
 |
 |  get_win(self)
 |      # getter methods for attributes
 |
 |  get_xinset(self)
 |
 |  get_yinset(self)
 |
 |  in_exit(self, point)
 |      Returns true if point is inside exit button (rectangle)
 |
 |  in_grid(self, point)
 |      Returns True if a Point (point) exists inside the grid of squares.
 |
 |  in_reset(self, point)
 |      Returns true if point is inside exit button (rectangle)
 |
 |  make_board(self)
 |      Create the board with the grid, text areas, and buttons
 |
 |  reset_grid_graphics(self)
 |      Resets the text color and fill color of all cells to their
 |      default values.
 |
 |  set_grid_cell(self, row, col, text, text_color='black', fill_color='whit
 |
 |      Update the graphical representation on a single grid cell
 |
 |  set_string_to_lower_text(self, text)
 |      Set text to text area below grid.  Overwrites existing text.
 |
 |  set_string_to_text_area(self, text)
 |      Sets text to text area to right of grid. Overwrites existing text.
 |
 |  set_string_to_upper_text(self, text)
 |      Set text to text area above grid. Overwrites existing text.
```

The end!