CSCI 134 Fall 2021:
More Classes and Inheritance in
Tic Tac Toe

Nov 10, 2021

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

- **Lab 8 due today/tomorrow**
  - Nice job using multiple classes!
  - When working with a partner, remember to take turns “driving”
  - Questions?
- **Lab 7** feedback coming soon
- **HW 8** posted later today, due Monday 11/15
- **Lab 9** - multi-week partners lab (more info coming soon)
  - Fill out Lida’s form
- Thanksgiving week
  - Attending lab is optional but encouraged

Do You Have Any Questions?
Last Time

- Learned about more special `__ (double underscore) methods`
  - `__str__` : print representation of objects
  - `__repr__` : string representation of objects
- Began talking about inheritance
Today’s Plan

• (Briefly) Review getters and setters (and their funky syntax)
• 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
Reviewing Getters and Setters

```
In [7]: class Example:
   ...:     __slots__ = ['_name']
   ...:
   ...:     def __init__(self, exName):
   ...:         self.name = exName
   ...:
   ...:     @property
   ...:     def name(self):
   ...:         print("in getter")
   ...:         return self._name
   ...:
   ...:     @name.setter
   ...:     def name(self, val):
   ...:         print("in setter")
   ...:         self._name = val

In [8]: ex = Example("example")

in setter

In [9]: ex.name  #this is a method call

in getter

Out[9]: 'example'
```
Tic Tac Toe
Implementing Tic Tac Toe

- Suppose we want to implement Tic Tac Toe
- Teaser demo…
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?
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?
  - Top layer: **Game logic** (alternating turns, checking for valid moves, etc)
  - Upper middle layer: **Tic Tac Toe “spaces” or “letters”** (9 in total!); set text to X or O
  - Lower middle layer: Extend the **basic board with Tic Tac Toe specific features** (3x3 grid of TTTLetters, initial board state: all letters start blank)
  - Bottom layer: **Basic board** w/buttons, text areas, mouse click detection (not specific to Tic Tac Toe!)
Board class

• Let’s start at the bottom: Board class
• What are basic features of a game board?
  • Think generally...many board-based games have the similar basic requirements
  • (Like Boggle and TicTacToe, for example)
Board class

• Let’s start at the bottom: Board class
• What are 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 (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, etc
Graphics Package for Board

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

```python
In [1]: from graphics import *
```

```python
In [2]: # takes title and size of window
    win = GraphWin("Name", 400, 400)
```

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.
We can draw other shapes as well.

We'll want to draw Rectangles in our grid.
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 pauses and waits until a click is detected.
Board class: Getting Started

• Attributes: rows, cols, size, xInset, yInset (since we want to have a margin around our grid)
  • We might add others later

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

• Might need some helper methods

• Let’s start by drawing the grid on our board
Board class: Drawing the grid

```python
from graphics import *

class Board:
    __slots__ = ['_xInset', '_yInset', '_rows', '_cols', '_size',
                 '_exitButton', '_resetButton',
                 '_textArea', '_lowerWord', '_upperWord']

    def __init__(self, xInset=50, yInset=50, rows=3, cols=3, size=50):
        self._xInset = xInset  # x inset, avoids drawing in corner of window
        self._yInset = yInset  # y inset, avoids drawing in corner of window
        self._rows = rows      # rows in grid
        self._cols = cols      # columns in grid
        self._size = size      # edge size of each square

        self.__initTextArea()

    def __makeGrid(self, self, win):
        """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))
                r = Rectangle(p1, p2)  # create rectangle
                r.setFill("white")
                r.draw(win)  # add rectangle to graphical window
```

We have defined all of our getter @property methods as well (but they aren't shown on this slide).

We need a window (win) before we can draw. We'll create that separately. We want to make sure we only create a single window.
Board class: Getting Started

- Attributes: `rows, cols, size, xInset, yInset` (since we want to have a margin around our grid)
  - Also need text areas: `textArea, upperWord, lowerWord`
- We need to draw the grid, text areas, and buttons
- Might need some helper methods
- 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

```python
def __initTextAreas__(self):
    self._textArea = Text(Point(self.xInset * self.rows + self.size * 2,
                                self.yInset + 50), "right")
    self._textArea.setSize(14)
    self._lowerWord = Text(Point(160, 275), "lower")
    self._lowerWord.setSize(18)
    self._upperWord = Text(Point(160, 25), "upper")
    self._upperWord.setSize(18)
    self._upperWord.setTextColor("red")

def __drawTextAreas__(self, win):
    """Draw the text area to the right/lower/upper side of main grid""
    self._textArea.draw(win)
    #draw the text area below grid
    self._lowerWord.draw(win)

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

- Attributes: `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**!
Initializing and Drawing the Buttons

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

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

def drawBoard(self, win):
    # this creates a row x col grid, filled with squares
    win.setBackground("white smoke")
    #self.__makeGrid(win)
    self.__makeResetButton(win)
    self.__makeExitButton(win)
    #self.__drawTextAreas(win)
```
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
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! (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 Tic Tac Toe 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!