Building Tic Tac Toe (Part 3)

Nov 15, 2021

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

- **Lab 7** feedback coming VERY soon
- **HW 8** due tonight @ 10 pm
- **Lab 9 Boggle** released on Friday: 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 Wed/Thur 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
  - Make sure to watch pre-lab video which goes over **shakeCubes** algorithm
  - Windows users: anyone try the graphics package yet?
- Shikha's office hours moved from Wed afternoons to **Tues 3-5 pm**

Do You Have Any Questions?
Git Workflow Reminder

- Starting a work session:
  - Always **pull most recent version** before making any edits (**clone** if using a new machine)

- Middle of a work session:
  - **Commit changes** to all files first (git commit -am "message") commits changes to all files already on evolene
  - After commit, **pull again** to get your partner's edits
  - If an editor opens up saying files were merged: that's okay, just save & exit ("Ctrl+x" and then "y")
  - Then **push your edits** to evolene (can check evolene to make sure it worked)

- Do the above steps (commit, pull, push) frequently
- Can check status anytime by typing **git status**
- Let us know if you face any issues!

---

Do You Have Any Questions?
Last Time

• Looked at important helper methods in the **Board** class
• Discussed how to build the **TTTBoard** class
  • Added a grid of **TTTLetters** to the **Board** class
  • Discussed logic to check for win on **TTTBoard**
    • (Will review today)
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
```

Why initialize `count` here?

What next?
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
Checking the 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
```

Just reverse row and col
Check Diagonals

Primary diagonal has row/col same

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

```python
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
```
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 will make this method public as it will be 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
```

O WINS!

Click anywhere to continue
Today’s Plan

• Finish our game!

• Implement TTTLetter
  • We already have a good sense of what it should do after our last class, but let’s look at the details

• Implement the game logic
  • Keep track of mouse clicks
  • Keep track of players ("X" and "O" alternate)
  • Use methods in TTTLetter and TTTBoard to check for win after each move
We have already seen a glimpse of what TTTLetters needs to do.

In fact it has to support this functionality in order for TTTBoard!

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
TTTLetter: __init__

• Let’s think about __init__ first

• A TTTLetter is just a “wrapper” around a Text object

• Using passed in parameters (col, row, letter), initialize __slots__ attributes

```python
class TTTLetter:
    __slots__ = ['_row', '_col', '_textObj']

    def __init__(self, col=-1, row=-1, letter=''):  
        # global variables needed for graphical testing
        xInset = 50; yInset = 50; size = 50

        # call setter
        self.textObj = Text(Point(xInset + size * col + size / 2,  
                                 yInset + size * row + size / 2), letter)

        self.col = col
        self.row = row
```

Calling setters for these attributes
TTTLetter: Getters

• Now let’s implement the necessary getter (@property) methods
• We need getters for attributes row, col, letter (str), and textObj

```python
@property
def textObj(self):
    """Property: textObj attribute of TTTLetter""
    return self._textObj

@property
def letter(self):
    """Property: letter (str) associated with TTTLetter""
    return self.textObj.getText()

@property
def col(self):
    """Property: col on grid of TTTLetter""
    return self._col

@property
def row(self):
    """Property: row on grid of TTTLetter""
    return self._row

Method of Text Object in graphics package
```
TTTLetter: Setters

- Now let’s implement the necessary setter methods
- We need setters for attributes row, col, letter (str), and textObj

```python
@textObj.setter
def textObj(self, textObj, size=20, style="bold"):  
    self._textObj = textObj
    self._textObj.setSize(size)
    self._textObj.setStyle(style)

@col.setter
def col(self, newCol):
    """Set column""
    self._col = newCol

@row.setter
def row(self, newRow):
    """Set row""
    self._row = newRow

@letter.setter
def letter(self, char):
    self._textObj.setText(char)
```
TTTLetter: resetLetter

- Do we need any other convenient helper methods?
  - Always good to implement `__str__` and `__repr__`
  - Maybe one to “reset” the state of a TTTLetter
  - In this case, that just means setting the letter attribute back to ""

```python
def resetLetter(self):
    """Resets the letter of TTTLetter to empty string""
    self.textObj.setText("")

def __str__(self):
    l, col, row = self.letter, self.col, self.row
    return "{} at Board position ({{}, {{}}}".format(l, col, row)

def __repr__(self):
    l, col, row = self.letter, self.col, self.row
    return "TTTLetter({}, {}, '{}")".format(col, row, l)
```
Testing TTTLetter

• It's always a good idea to test our class and methods in isolation

```python
if __name__ == "__main__":
    win = GraphWin("Tic Tac Toe", 400, 400)

    # create and draw board
    board = Board()
    board.drawBoard(win)

    letter = TTTLetter(1, 1, "X")
    letter2 = TTTLetter(1, 2, "O")
    letter3 = TTTLetter(2, 1, "X")

    # draw them on the board
    letter.textObj.draw(win)
    letter2.textObj.draw(win)
    letter3.textObj.draw(win)

    # pause and wait for mouse click
    point = win.getMouse()
```
Finally... TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Let’s think about the “common” case: a valid move in the middle of the game
Finally... TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Now let’s consider the case of a win, draw, or invalid move
Finally…TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Now’s let suppose a player chooses reset
Finally…TTT Game Logic

• Let’s create a TTT flowchart to help us think through the state of the game at various stages

Now’s let suppose a player chooses exit
Finally…TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Finally, let’s handle the click that may be outside of any of the “valid” regions
Finally… TTT Game Logic

• Let’s create a TTT flowchart to help us think through the state of the game at various stages
Translating our Logic to Code

- We are going to need a (mostly infinite) while loop to wait for clicks
- Then we need a few if-elif-else checks to handle the grid/reset/exit check
- Let’s start with that logic and fill the rest in later

```python
# set letters and initial game state
exit = False
reset = False
firstPlayer = True
numMoves = 0

while not exit:
    # get point clicked by mouse
    point = win.getMouse()

    # check for exit button
    if board.inExit(point):=

    # check for reset button
    elif board.inReset(point):=

    # check for a grid button
    elif board.inGrid(point):=
```

![Diagram of the logic flow in the code]

Y: Yes, N: No
Translating our Logic to Code

Let's handle the “exit” button first (since it’s the easiest)

```python
while not exit:
    # get point clicked by mouse
    point = win.getMouse()
    
    # check for exit button
    if board.inExit(point):
        exit = True
```

![Flowchart diagram](image)
Translating our Logic to Code

- Now let’s handle reset

```python
# check for reset button and reset board state
elif board.inReset(point):
    reset = True

# check for a grid button
elif board.inGrid(point):
    # start over, clear initial state, reset board
    if reset:
        firstPlayer = True
        numMoves = 0
        board.reset()
        board.clearUpperText()
        reset = False
```

![Flowchart diagram](https://via.placeholder.com/150)
Finally, let's handle a “normal” move. Start by getting position and TTTLetter

```python
# check for a grid button
elif board.inGrid(point):
    # convert point to a grid location
    position = board.getPosition((point.getX(), point.getY()))

# get TTTLetter at that grid location
tlet = board.getLetterObj(position)
```
Translating our Logic to Code

- The rest of our code checks for a valid move, a win, a draw, and updates state accordingly.
- At the end, if the move was valid, we swap players.

```python
# make sure this square is vacant
if tlet.letter == "":
    if firstPlayer:
        l = "X"
    else:
        l = "0"
    tlet.letter = l

# valid move, increment numMoves
numMoves += 1

# check for win or draw
winFlag = board.checkForWin(l)
if winFlag or numMoves == 9:
    if winFlag:
        board.setStringToUpperText(l + " WINS!")
    else:
        board.setStringToUpperText("DRAW!")
    board.setStringToLowerText("Click anywhere to continue")
    reset = True
    # pause and wait for click
    point = win.getMouse()

    # game continues, swap player
    else:
        firstPlayer = not firstPlayer
```
TTT Summary

• Basic strategy
  • **Board**: start general, don’t think about game specific details
  • **TTTBoard**: extend generic board with TTT specific features
    • Inherit everything, overwrite attributes/methods as needed
  • **TTTLetter**: isolate functionality of a single TTTLetter on board
    • Think about what features are necessary/helpful in other classes as well
  • **Game**: think through logic conceptually before writing any code
    • Translate logic into code carefully, testing along the way
Boggle Strategies

• At a high level, Tic Tac Toe and Boggle have a lot in common, but the game state of Boggle is more complicated

• For simplicity, in Lab 9 we are asking you to start from the top (BoggleLetter) and work down to BoggleBoard (rather than the bottom up as we did in TTT)

• *Don’t forget the bigger picture as you implement individual methods*

• Think holistically about how the objects/classes work together

• Isolate functionality and test often (use `__str__` and `__repr__` to print values as needed)

• *Discuss logic with partner before writing any code*

• Worry about common cases first, but don’t forget the “edge” cases

• Come see instructors/TAs for clarification

GOOD LUCK and HAVE FUN!