CS134 Lecture 28:
Tic Tac Toe 4
Announcements & Logistics

- **Lab 9 Boggle**: two-week lab now in progress!
- **Part 1** due tonight/tomorrow 10 pm
- Will return auto-tester feedback on it on Friday
- You can fix anything broken before turning in Part 2
- Must turn in *something* to get Part 2 grade apply to both
- **Part 2** due May 1/2 (handout will be posted soon)
- Part 2 also has a **prelab**!
  - Asks you to draw out the Boggle game logic (similar to TTT logic we will discuss today)

**Do You Have Any Questions?**
Last Time and Today

- Implemented TTTCube and TTTBoard classes
- Today: wrap up the game
  - Implement TTTGame class
  - Talks to each of the classes and calls appropriate methods to implement game logic
- TTT vs Boggle discussion
TTTGame Logic
TTT Game Logic

Start

Wait for mouse click

In Grid?

Y → Get cube

N → Reset state

In Reset button?

Y → In Exit button?

N → End

Empty space?

Y → Update cube display

N → Get cube

Result is a Win?

Y → Reset state

N → Result is a draw?

Y → Change players

N → End

Result is a draw?

Y → Change players

N → End

Get cube

Empty space?
Translating our Logic to Code

• Let’s think about __init__:
  • What do we need?
    • a **board**, **player**, and maybe **num_moves** (to detect draws easily)
Translating our Logic to Code

Now let’s write a method for handling a single mouse click (point)
• The game continues (waits for more clicks) if this method returns True
• If this method returns False, game ends

```python
def do_one_click(self, point):
    # step 1: check for exit button
    if self._board.in_exit(point):
        # TODO

    # step 2: check for reset button
    elif self._board.in_reset(point):
        # TODO

    # step 3: check if click on the grid
    elif self._board.in_grid(point):
        # TODO

    # keep going!
    return True
```
Translating our Logic to Code

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

```python
if self._board.in_exit(point):
    print("Exiting...")
    # game over
    return False
```
Translating our Logic to Code

- Now let's handle reset

```python
elif self._board.in_reset(point):
    print("Reset button clicked")
    self._board.reset()
    self._board.set_string_to_upper_text"
    self._num_moves = 0
    self._player = "X"
```
Finally, let's handle a “normal” move. Start by getting point and TTTCube

```python
elif self._board.in_grid(point):
    tcube = self._board.get_ttt_cube_at_point(point)
    # get the cube at the point the user clicked
```

![Flowchart diagram](image)
elif self._board.in_grid(point):
    # get the cube at the point the user clicked
    tcube = self._board.get_ttt_cube_at_point(point)

    # make sure this square is vacant
    if tcube.get_letter() == "":
        tcube.set_letter(self._player)
        tcube.place_cube(self._board)

    # valid move, so increment num_moves
    self._num_moves += 1

    # check for win or draw
    win_flag = self._board.check_for_win(self._player)
    if win_flag:
        self._board.set_string_to_upper_text(self._player + " WINS!")
    elif self._num_moves == self._board.get_rows() * self._board.get_cols():
        self._board.set_string_to_upper_text("DRAW!")
    else:
        # toggle player!
        self._player = "0" if self._player == "X" else "X"

    # keep going!
    return True

• 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

Translating our Logic to Code
TTT Summary

• Basic strategy
  • **Board**: start general, don’t think about game specific details
  • **TTTBoard**: extend generic board with TTT specific features
    • Inherit everything, update attributes/methods as needed
  • **TTTCube** isolate functionality of a single TTT cube on board
    • Think about what features are necessary/helpful in other classes
  • **TTTGame**: think through logic conceptually before writing any code
    • Translate logic into code carefully, testing along the way
Class Discussion:
Boggle vs TTT Design Differences
Special Methods/Magic Methods
Special Methods

• Start and end with __ (double underscore)
  • Called magic methods (or informally dunder methods)
  • Often not called explicitly using dot notation and called by other means

• What special methods have we already used seen/used so far?

• __init__(self, val)
  • When is it called?
    • Automatically when we create an instance (object) of the class
    • Can also be invoked as obj.__init__(val) (where obj is an instance of the class)
Special Methods

• **__str__(self)**
  • When is it called?
    • When we *print* an instance of the class using `print(obj)`
    • Also called whenever we call `str` function on it: `str(obj)`
    • Can also be invoked as `obj.__str__()`

• **__repr__(self)**
  • Also returns a string but its format is very specific (can be used to recreate the object of the class)
  • Useful for debugging
  • Don't worry about any more specifics for this class
Special Methods for Operators

• We can use mathematical and logical operators such as `==/+` to compare/add two objects of a class by defining the corresponding special method.

• Example of polymorphism (using a single method or operator for different uses):
  
  - `__eq__ (self, other):`  
  - `__ne__ (self, other):`  
  - `__lt__ (self, other):`  
  - `__gt__ (self, other):`  
  - `__add__ (self, other):`  
  - `__sub__ (self, other):`  
  - `__mul__ (self, other):`

• There are many others!

`x == y`  
`x != y`  
`x < y`  
`x > y`  
`x + y`  
`x - y`  
`x * y`  

`__add__`: why we can concatenate sequences with `+` as well as add ints with `+`
Special Method: \texttt{\_\_len\_\_}

- \texttt{\_\_len\_\_(self)}
  - Called when we use the built-in function \texttt{len()} in Python on an object \texttt{obj} of the class: \texttt{len(obj)}
  - We can call \texttt{len()} function on any object whose class has the \texttt{\_\_len\_\_()} special method implemented
  - All built-in collection data types we saw (string, list, range, tuple, set, dictionaries) have this special method implemented
  - This is why we are able to call \texttt{len} on them
  - What is an example of a built-in type that we can't call \texttt{len} on?
  - \texttt{int}, \texttt{float}, \texttt{Bool}, \texttt{None}
Other Special Methods for Sequences

- What other sequence operators have we used in this class?
- They each have a special method that is called whenever they are used
  - **Get** an item at an index a sequence using `[ ]`: calls `__getitem__`
    - e.g., `word_lst[2]` implicitly calls `word_lst.__getitem__(2)`
  - **Set** an item at an index to another `val` using `[ ]`: calls `__setitem__`
    - e.g., `word_lst[0] = "hello"` implicitly calls `word_lst.__setitem__(0, "hello")`
**in** Operator: **__contains__**

- **__contains__(self, val)**
  - When we say `if elem in seq` in Python:
    - Python calls the **__contains__** special method on `seq`
    - That is, `seq.__contains__(elem)`
  - If we want the **in** operator to work for the objects of our class, we can do so by implementing the **__contains__** special method
Iteration Special Methods

• What if we want to "iterate" over an object of our class in a for loop?
• We can achieve this by implementing appropriate special methods:
  • A for loop in Python can iterate over any object whose class has the special methods `__iter__` and `__next__` defined
  • Such objects are called *iterables*
• We can make objects of our class iterable by defining these methods appropriately
[Extra] For loop: Behind the Scenes

```python
# a simple for loop to iterate over a list
for item in num_lst:
    print(item)

• Behind the scenes, the for loop is simply a while loop in disguise, driving iteration within a `try-except` statement. The above loop is really:

```python
try:
    it = iter(num_lst)
    while True:
        item = next(it)
        print(item)
except StopIteration:
    pass
```

Call the `iter` method on object

Access the `next` item if it exists, then print it

This is a way to “hide” the error
Takeaway

- We can implement any of these functionalities that built-in types enjoy for objects of our own class by defining the appropriate special methods.