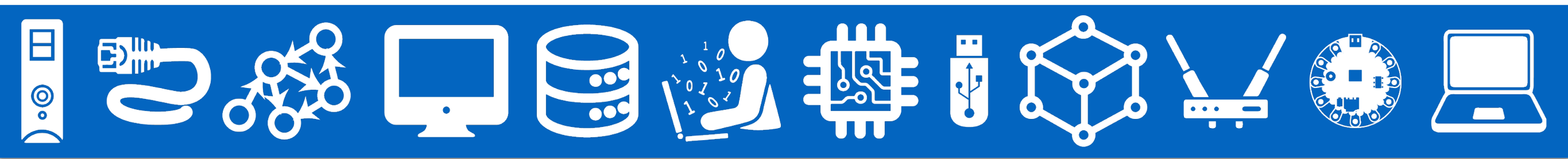
CS134:
Tic Tac Toe 4



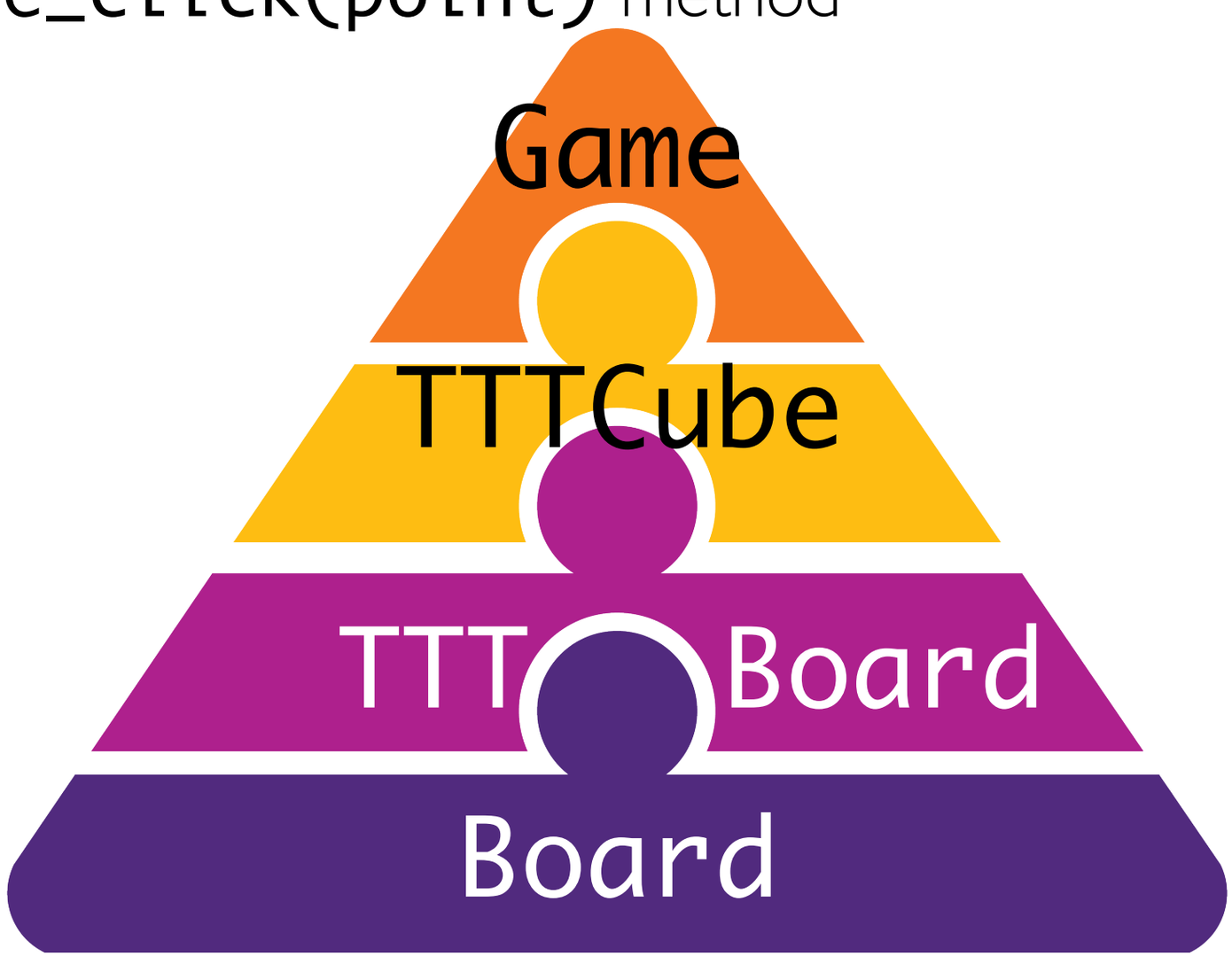
Announcements & Logistics

- **HW 9**: due today
- **Lab 9 Boggle**: two-week lab starts today!
  - **Part 1** due Weds/Thurs 10 pm
  - Should receive automated feedback immediately on Gradescope
  - 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 Nov 20/21 (handout will be posted soon)
  - Part 2 also has a **prelab!**
    - Asks you to draw out the Boggle game logic (similar to TTT logic flow chart)

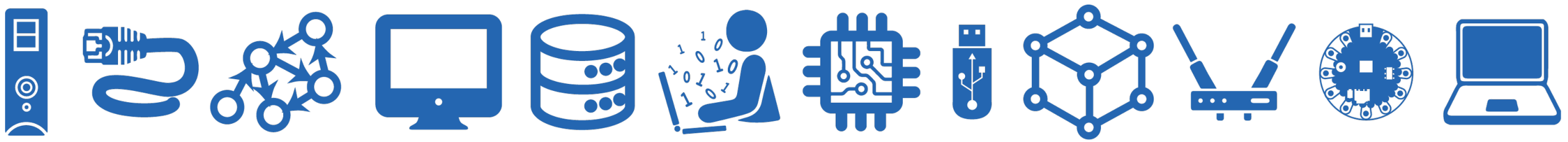
Do You Have Any Questions?

Last Time and Today

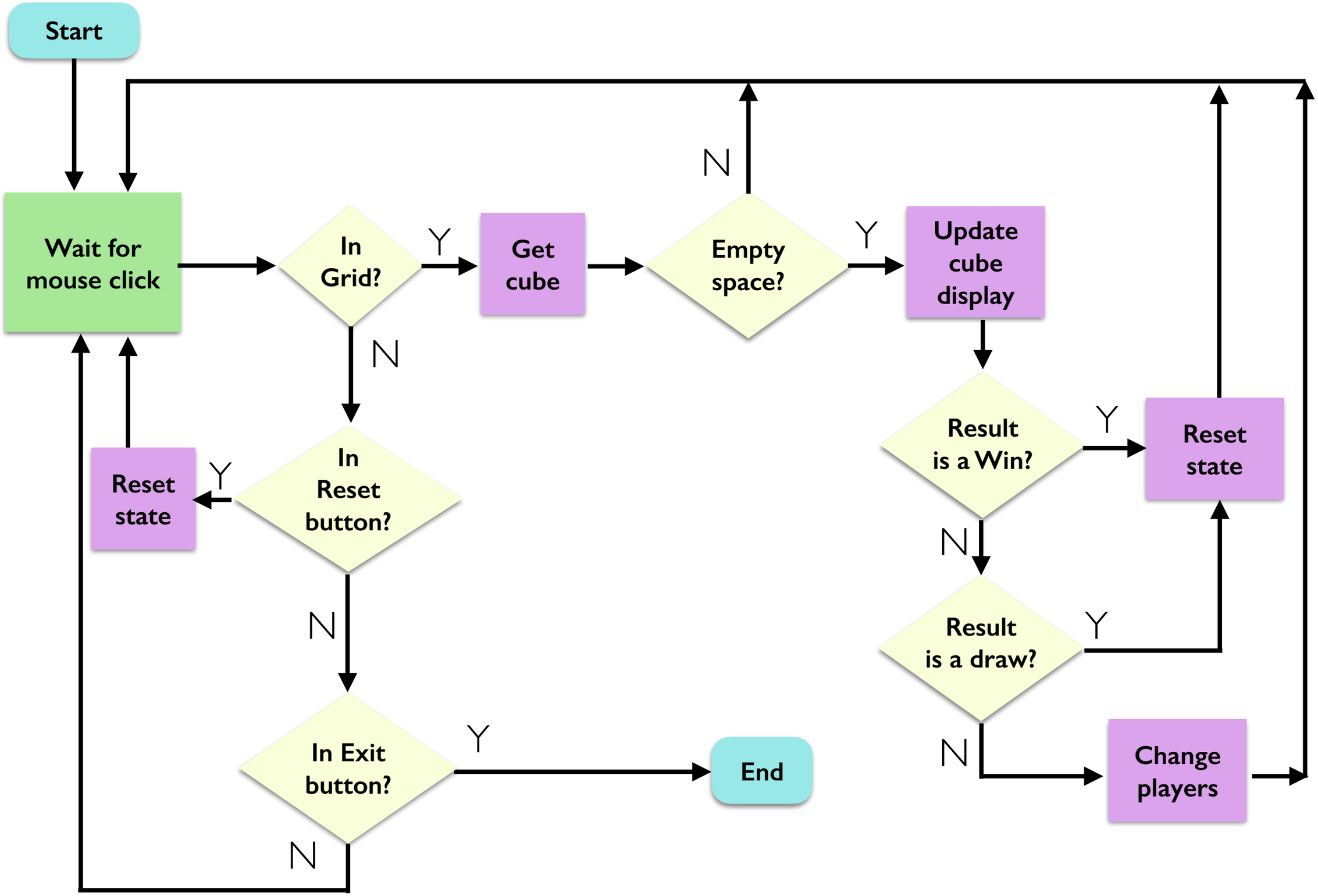
- Implemented guessinggame.py and designed TTTGame class
- Today:
  - Wrap up the game
    - Finish TTTGame `do_one_click(point)` method
  - Brief discussion of Tuples
  - TTT vs Boggle discussion



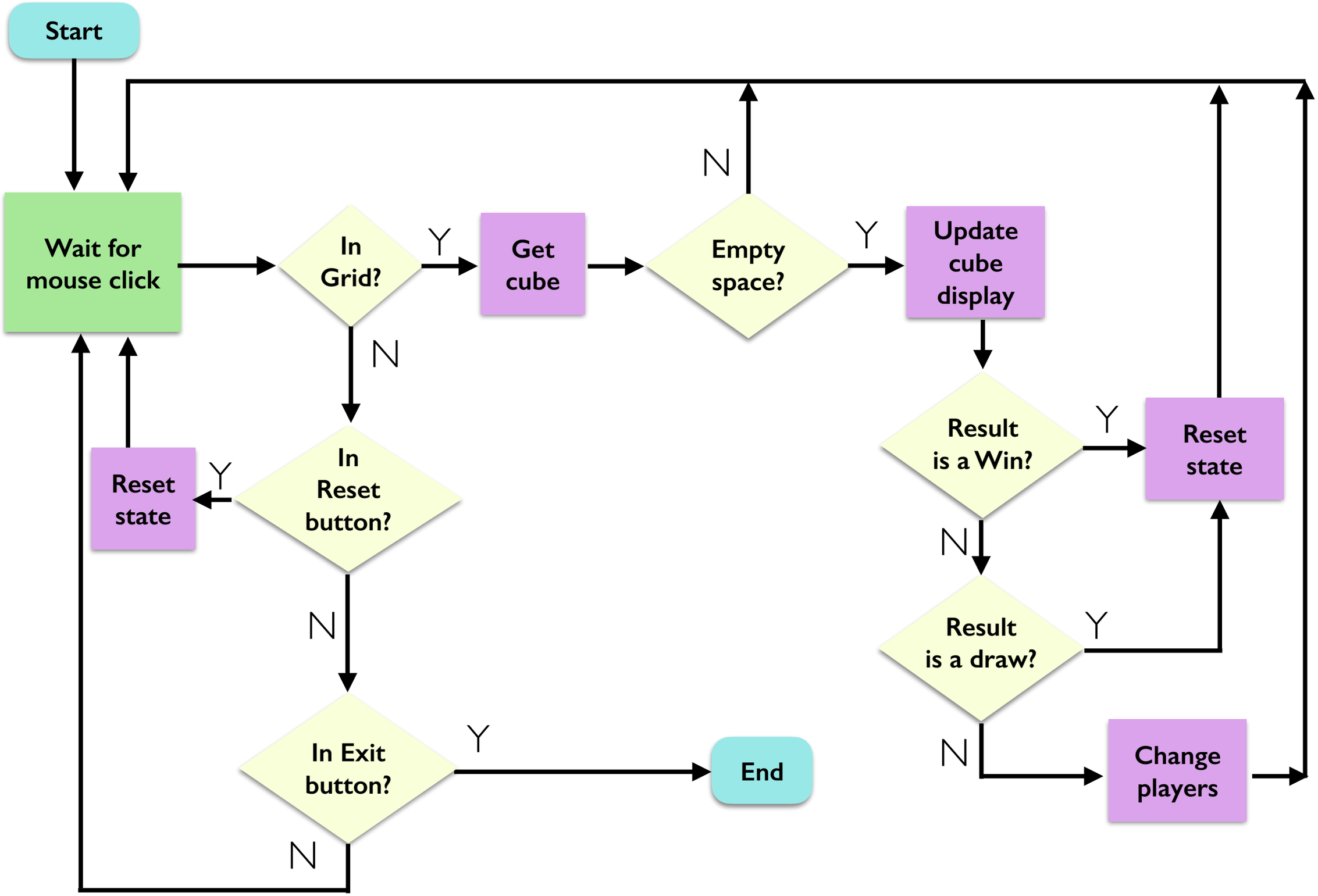
TTT Game Logic



TTT Game Logic



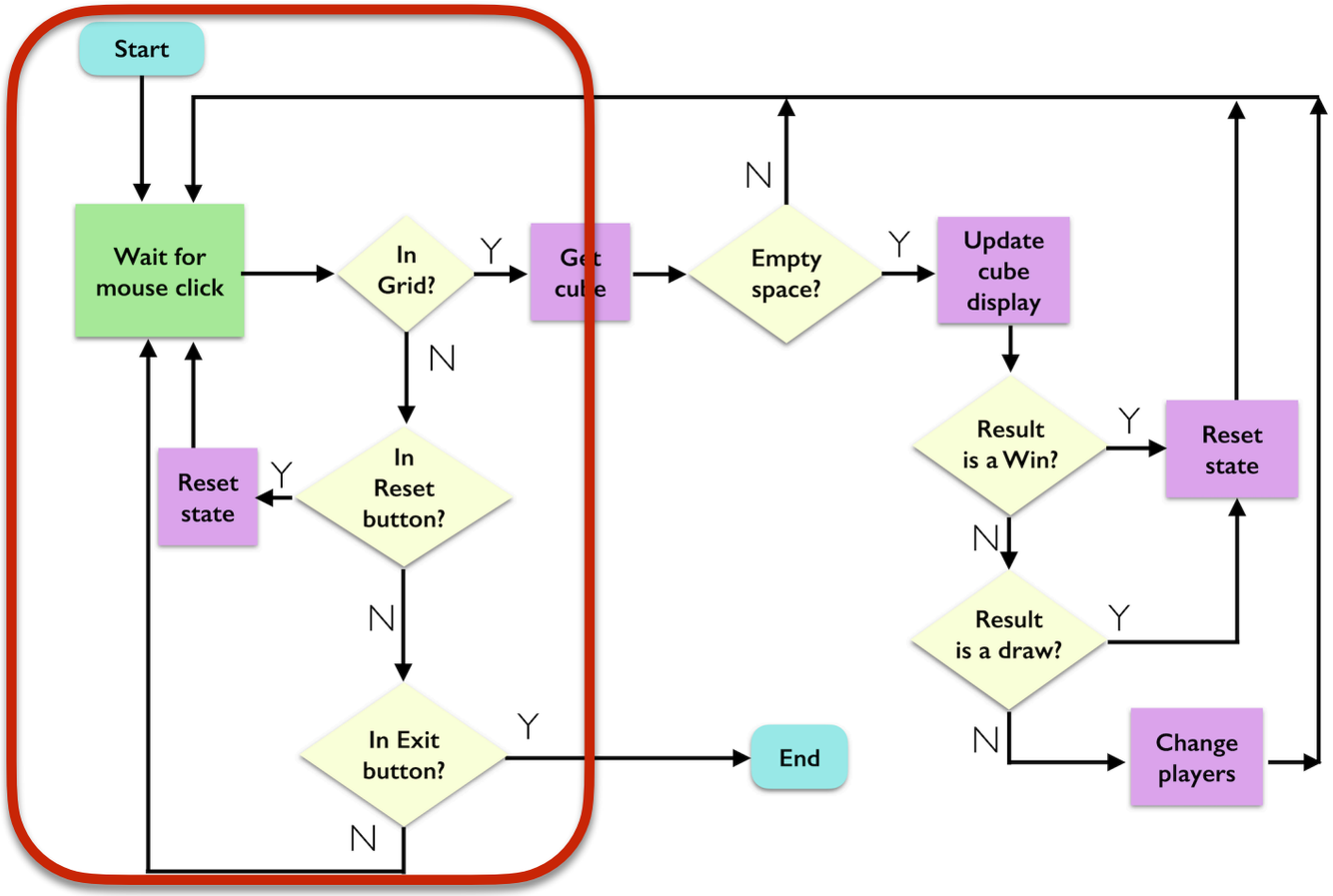
TTT Game Logic



Translating our Logic to Code

- Last class, we started our 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

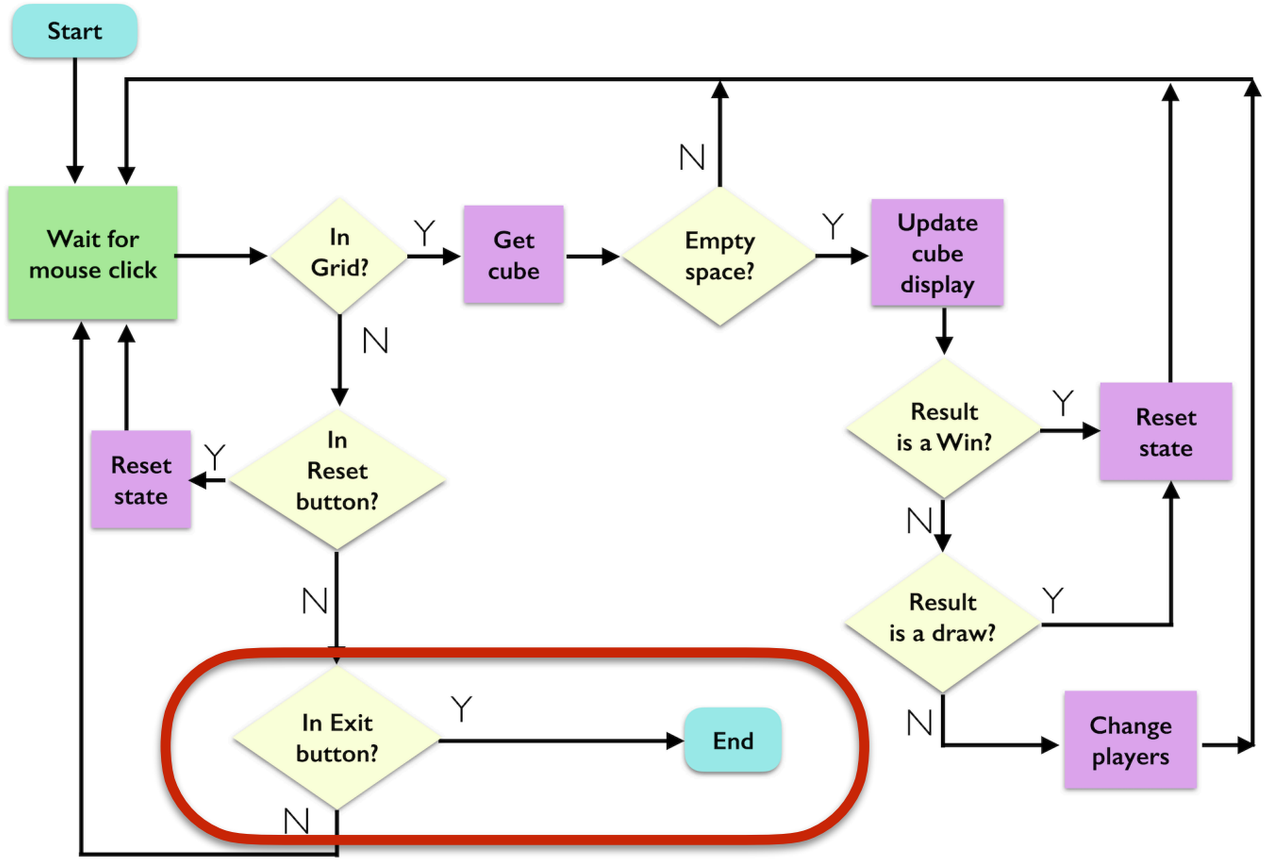
```
def do_one_click(self, point):

  # 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)

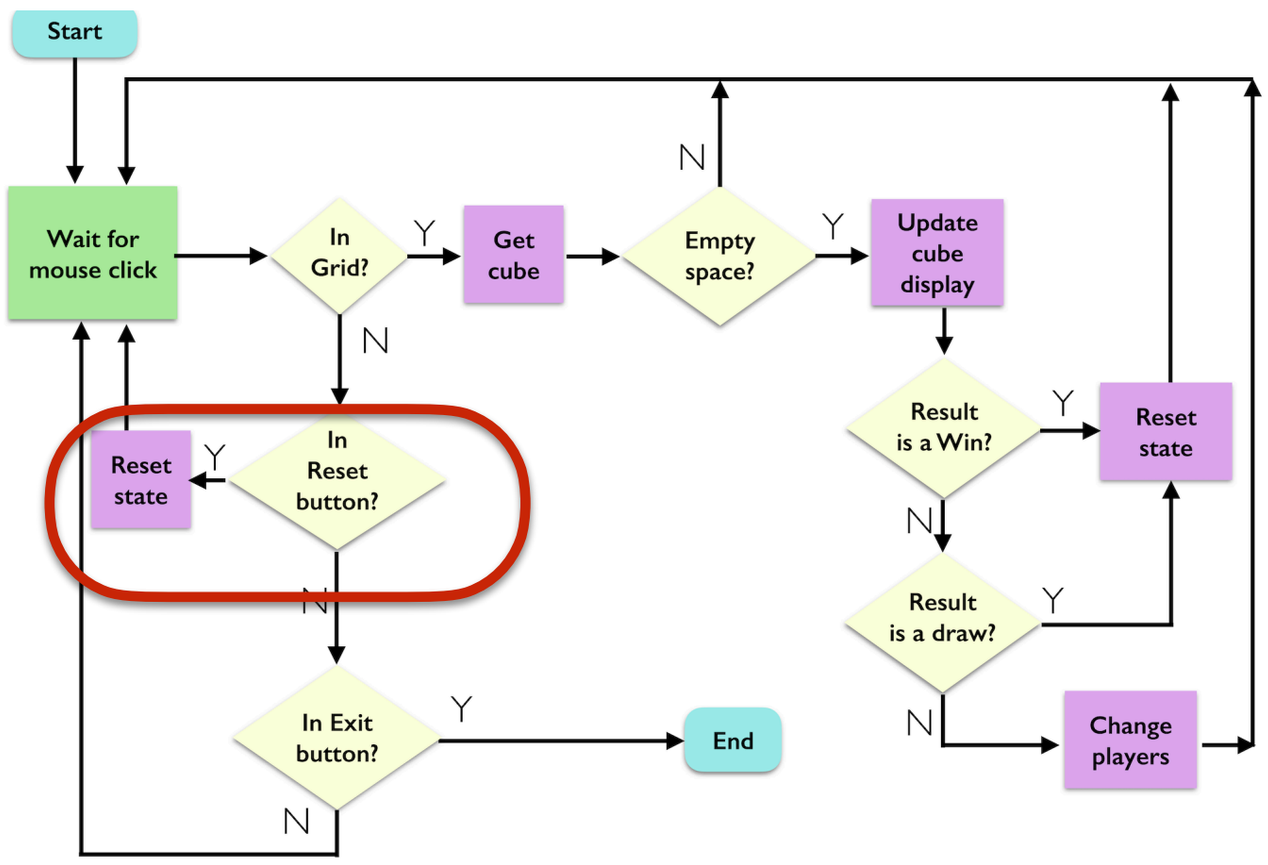
```
if self._board.in_exit(point):
    print("Exiting...")
    # game over
    return False
```



Translating our Logic to Code

- Now let's handle reset

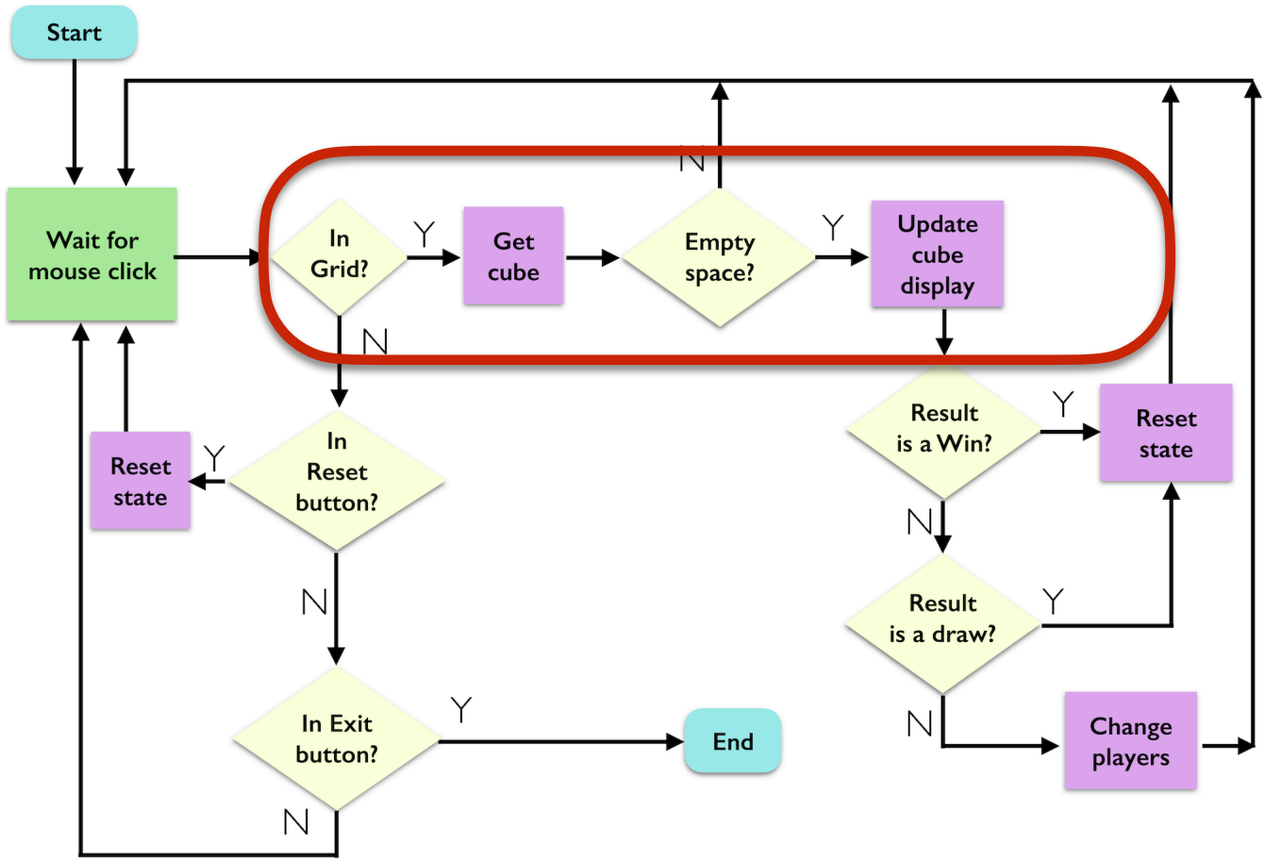
```
elif self._board.in_reset(point):
    print("Reset button clicked")
    self._board.reset()
    self._board.set_string_to_upper_text("")
    self._num_moves = 0
    self._player = "X"
```



Translating our Logic to Code

- Finally, let's handle a "normal" move. Start by getting point and TTTCube

```
elif self._board.in_grid(point):

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



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

```
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)
        self._board.place_cubes_on_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!")
        # not a win or draw, swap players
        else:
            # toggle player!
            self._player = "O" if self._player == "X" else "X"

# keep going!
return True
```

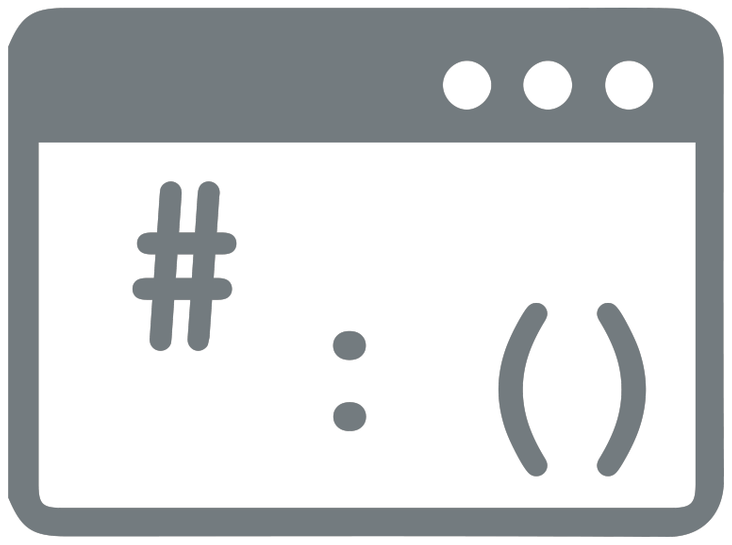
TTT Summary

- 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

Boggle Demo



Boggle: The same but different

- What things are different about Boggle?
  - single player not 2-player
  - pieces are randomly placed on grid, selected from among a set of pre-determined cubes
  - grid size is different (4x4 instead of 3x3)
  - no notion of a "win" "tie" or "draw" --- game continues with new words being created until button is reset
  - ...

Representing Cubes

- The same 16 cubes, each with 6 faces, come in every Boggle Game box

```
CUBE_FACES = [("A", "A", "C", "I", "O", "T"),  # cube 0
              ("T", "Y", "A", "B", "I", "L"),  # cube 1
              ("J", "M", "O", "QU", "A", "B"), # cube 2
              ("A", "C", "D", "E", "M", "P"),  # cube 3
              ("A", "C", "E", "L", "S", "R"),  # cube 4
              ("A", "D", "E", "N", "V", "Z"),  # cube 5
              ("A", "H", "M", "O", "R", "S"),  # cube 6
              ("B", "F", "I", "O", "R", "X"),  # cube 7
              ("D", "E", "N", "O", "S", "W"),  # cube 8
              ("D", "K", "N", "O", "T", "U"),  # cube 9
              ("E", "E", "F", "H", "I", "Y"),  # cube 10
              ("E", "G", "I", "N", "T", "V"),  # cube 11
              ("E", "G", "K", "L", "U", "Y"),  # cube 12
              ("E", "H", "I", "N", "P", "S"),  # cube 13
              ("E", "L", "P", "S", "T", "U"),  # cube 14
              ("G", "I", "L", "R", "U", "W")]  # cube 15
```

A list of 16 **`tuple`** objects

New Sequence Type: Tuple

- Tuples are an **immutable** **sequence of values** (almost like immutable lists) separated by commas and enclosed within parentheses **( )**

```
# tuple of strings
>>> names = ("Charlie", "Lucy", "Snoopy")

# tuple of ints
>>> primes = (2, 3, 5, 7, 11)

# singleton (tuple with length 1)
>>> num = (5, )

# empty tuple
>>> empt = ()

# tuples can have mixed types
>>> values = (5, True, "abc")
```

Why Use a Tuple to represent a Cube?

- Cube game pieces are physical objects: they are fixed and unchangeable
  - Tuples are immutable
- Cube game pieces have 6 "faces" each, with a letter* per face
  - Tuples are a sequence type; they can store any number of string values, including 6
- Exactly one of a cube's face is "visible" when placed on the grid
  - We can index into a Tuple using the [] operator, so a single integer can be used to represent which of the 6 faces is visible

Why does BoggleBoard store a `list` of Tuples?

- The cube faces are fixed, but the cube locations are not---shaking the boggle game board rearranges the cubes
  - Lists are mutable, so we can rearrange the list's contents to simulate "shaking"
- Cubes are placed onto the 4x4 grid (a list of list of `TextRect` objects)
  - We need a way to map a position in our 1-dimensional list of cubes to a 2-dimensional list of `TextRect` objects and back
  - This is the point of the pre-lab assignment!

Mapping Cubes in BoggleBoard

```
CUBE_FACES = [("A", "A", "C", "I", "O", "T"),  # cube 0
              ("T", "Y", "A", "B", "I", "L"),  # cube 1
              ("J", "M", "O", "QU", "A", "B"), # cube 2
              ("A", "C", "D", "E", "M", "P"),  # cube 3
              ("A", "C", "E", "L", "S", "R"),  # cube 4
              ("A", "D", "E", "N", "V", "Z"),  # cube 5
              ("A", "H", "M", "O", "R", "S"),  # cube 6
              ("B", "F", "I", "O", "R", "X"),  # cube 7
              ("D", "E", "N", "O", "S", "W"),  # cube 8
              ("D", "K", "N", "O", "T", "U"),  # cube 9
              ("E", "E", "F", "H", "I", "Y"),  # cube 10
              ("E", "G", "I", "N", "T", "V"),  # cube 11
              ("E", "G", "K", "L", "U", "Y"),  # cube 12
              ("E", "H", "I", "N", "P", "S"),  # cube 13
              ("E", "L", "P", "S", "T", "U"),  # cube 14
              ("G", "I", "L", "R", "U", "W")]  # cube 15
```

Row 0

Row 1

Row 2

Row 3

```
_grid = [[ 0 , 1 , 2 , 3 ],
         [ 4 , 5 , 6 , 7 ],
         [ 8 , 9 , 10 , 11 ],
         [ 12 , 13 , 14 , 15 ]]
```

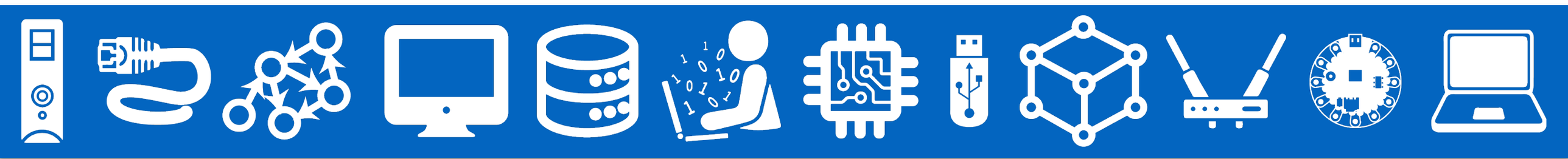
```
def _which_row(self, cube_number) :
        """
        The row of the board's grid that corresponds
        to cube cube_number
        """

def _which_col(self, cube_number) :
        """
        The row of the board's grid that corresponds
        to cube cube_number
        """

def _which_cube(self, row, col) :
        """
        The index of the cube within the cube list
        that corresponds to the cube appearing at
        coords (row,col) of the board's grid
        """
```

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
- ***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__` to print values as needed)
- **Discuss logic with partner/instructor before writing any code**
- Worry about common cases first, but don't forget the "edge" cases
- Come see instructors/TAs for clarification

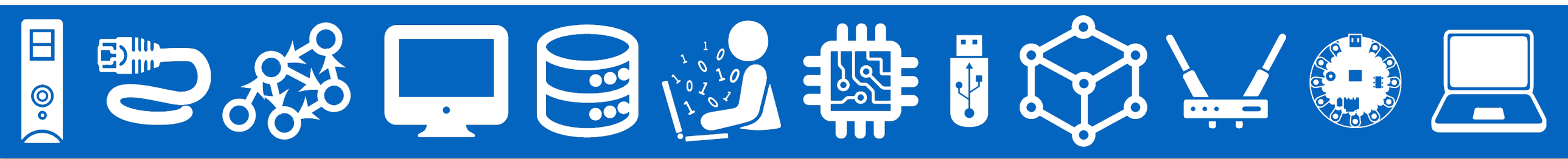
GOOD LUCK and HAVE FUN!

The end!



CS134:

Lab 9



Lab 9 Overview

- **User-defined Types with Inheritance!**
  - Using the `Board` class from...class

- Multi-week partners lab (counts as two labs in terms of grade; Lab is decomposed into two logical parts)
  - **Part 1 (Boggle Board & Cube)** due Nov 13/14
  - We will run our tests on these and return automated feedback (similar to Lab 4 part 1); will lose credit on Part II if you don't submit Part I
  - You are allowed/encouraged to revise it afterwards
  - **Part 2 (Boggle Game)** (and revised Part 1) due Nov 20/21

Boggle Strategies

- Tic Tac Toe and Boggle are similar, but Boggle's game state is more complicated
- In Lab 9 you should follow a similar strategy to what we did with 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__` to print values as needed
  - `if __name__=='__main__'` also useful for isolated testing!
- **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!

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?

Playing Boggle

- **Word Game**: objective is to have the most points at the end of each 3 minute game.
- **Pieces:** 16 six-sided letter cubes that are held in a small plastic grid with a covered lid

1. One player shakes the grid to jumble the letters around
2. Then each player has 3 minutes to find as many words (3+ characters) as they can from the jumbled letters
3. Each player records as many words as they can create from the letters that are adjoining **horizontally, vertically, or diagonally**
4. A cube cannot be used more than once within a single word.
5. The more words a player can find, the more points the player earns.