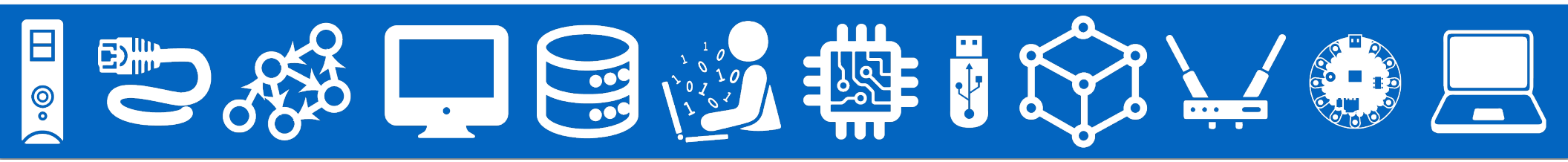
CS134:
Tic Tac Toe (3)
`TTTLetter` & `Game`



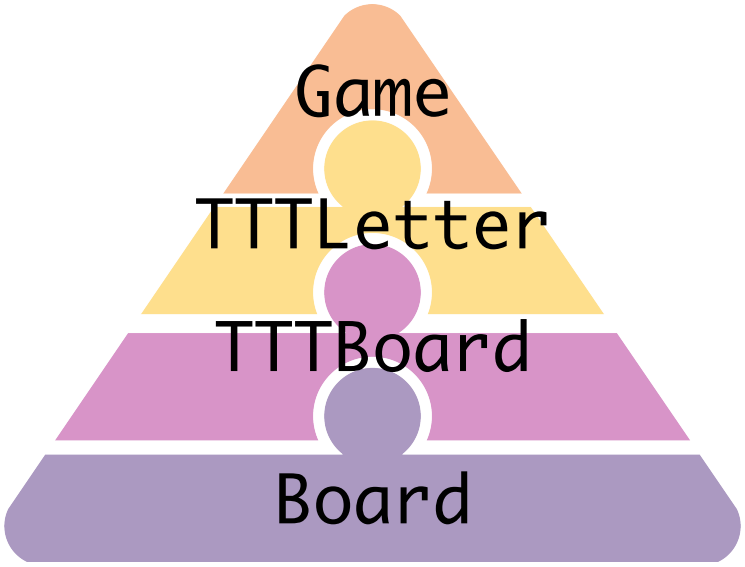
Announcements & Logistics

- **HW 8** due tonight @ 10 pm
- No new HW this week
- **Lab 9 Boggle** starts today/tomorrow: Lab is decomposed into **three** 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 (BoggleGame)** due Nov 30/Dec 1
  - Please spend time planning and thinking about design with your partner before your lab session!

Do You Have Any Questions?

Last Time

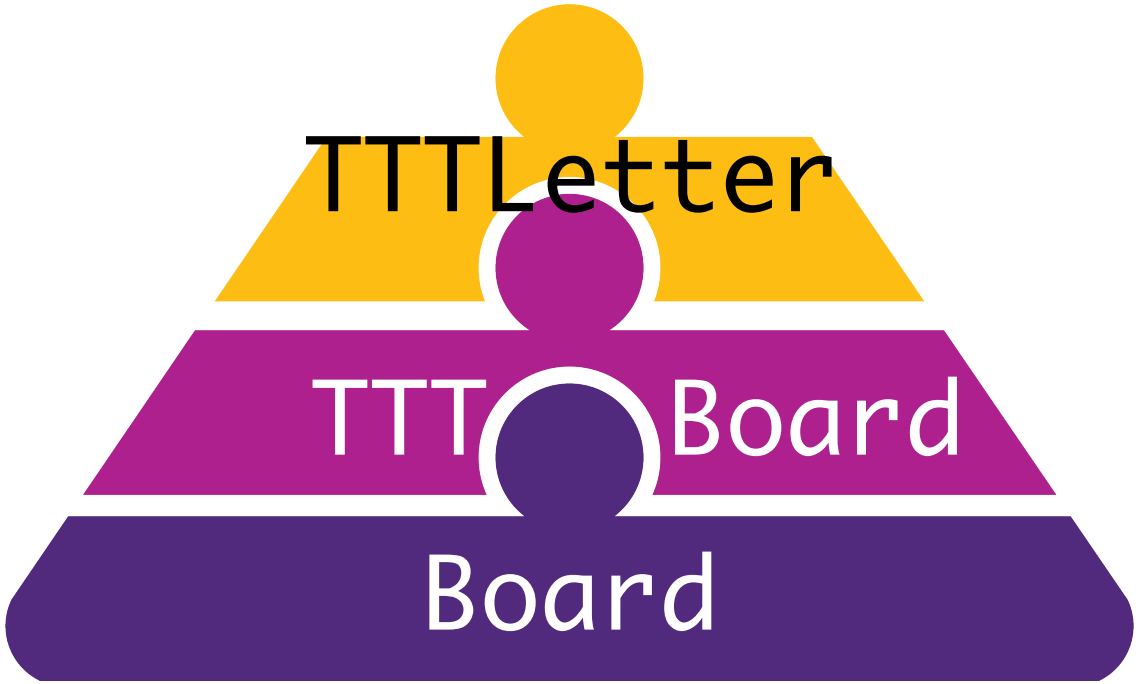
- (Briefly) Looked at important helper methods in the `Board` class
- Discussed how to build the `TTTBoard` class
  - Added a grid of `TTTLetters` to the `Board` class
  - Discussed logic to check for win on `TTTBoard`
  - Any questions?

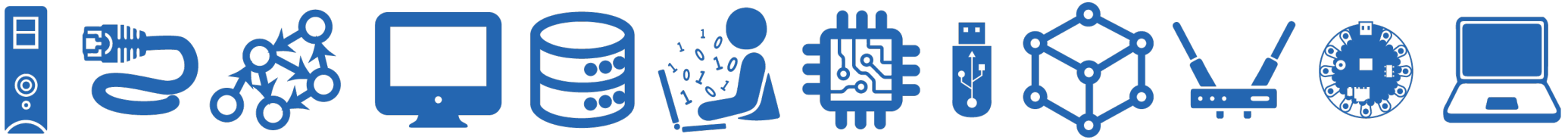


Today's Plan

- Finish our game! Woohoo!
- 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

TTTLetter Class





TTT Letters

- We have already seen a glimpse of what **TTTLetter**s needs to do
- In fact it has to support this functionality for **TTTBoard**!

```
class TTTLetter(builtins.object)
 |  TTTLetter(board, col=-1, row=-1, letter='')
 |
 |  A TTT letter has several attributes that define it:
 |  *  _row, _col coordinates indicate its position in the grid (ints)
 |  *  _textObj denotes the Text object from the graphics module,
 |     which has attributes such as size, style, color, etc
 |     and supports methods such as getText(), setText() etc.
 |  *  _rect denotes the Rectangle object from the graphics module,
 |     which has attributes such as color and supports methods such as
 |     getFillColor(), setFillColor() etc.
 |
 |  Methods defined here:
 |
 |  __init__(self, board, col=-1, row=-1, letter='')
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __str__(self)
 |      Return str(self).
 |
 |  getLetter(self)
 |      Returns letter (text of type str) associated with self._textObj
 |
 |  setLetter(self, char)
```

TTTLetter: __init__

- Let's think about **`__init__`** first
  - Use passed-in parameters (**`col, row, letter`**) to initialize **`__slots__`** attributes

initialize **`__slots__`** attributes

```
from graphics import *
from board import Board

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

    def __init__(self, board, col=-1, row=-1, letter=""):

        # variables needed for graphical testing
        xInset = board.getXInset()
        yInset = board.getYInset()
        size = board.getSize()
        win = board.getWin()

        # set row and column attributes
        self._col = col
        self._row = row

        # make rectangle and add to graphical window
        p1 = Point(xInset + size * col, yInset + size * row)
        p2 = Point(xInset + size * (col + 1), yInset + size * (row + 1))
        self._rect = board._makeRect(p1, p2, "white")

        # update text in center of rectangle
        self._textObj = Text(self._rect.getCenter(), letter)
        self._textObj.draw(win)
```

`TTTLetter`: Getters, Setters, `__str__`

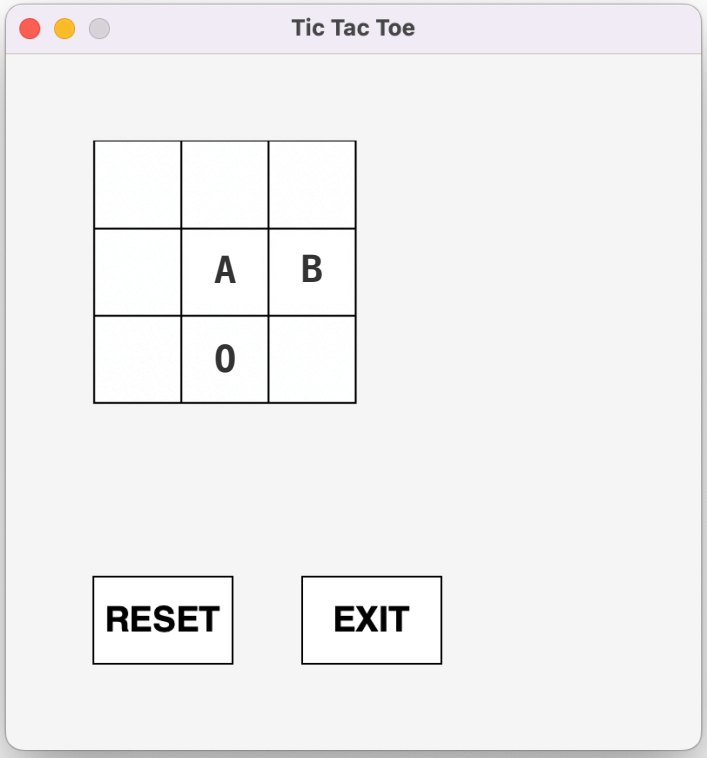
- Now let's implement the necessary getter/setter methods
  - We don't need/want to expose the Text object
  - We don't want to allow the row, col to be changed
  - We only expose the string (letter) of the Text object, so they are the only getter/setter methods we need
  - `__str__` useful for debugging and testing

```
def getLetter(self):
   return self._textObj.getText()

def setLetter(self, char):
    self._textObj.setText(char)
    if char == 'X':
        self._rect.setFillColor("light blue")
    elif char == 'O':
        self._rect.setFillColor("pink")
    else:
        self._rect.setFillColor("white")

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

Testing TTTLetter

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

```
win = GraphWin("Tic Tac Toe", 400, 400)
board = Board(win, rows=3, cols=3)

letter = TTTLetter(board, 1, 1, "A")
letter2 = TTTLetter(board, 1, 2, "O")
letter3 = TTTLetter(board, 2, 1, "B")
```

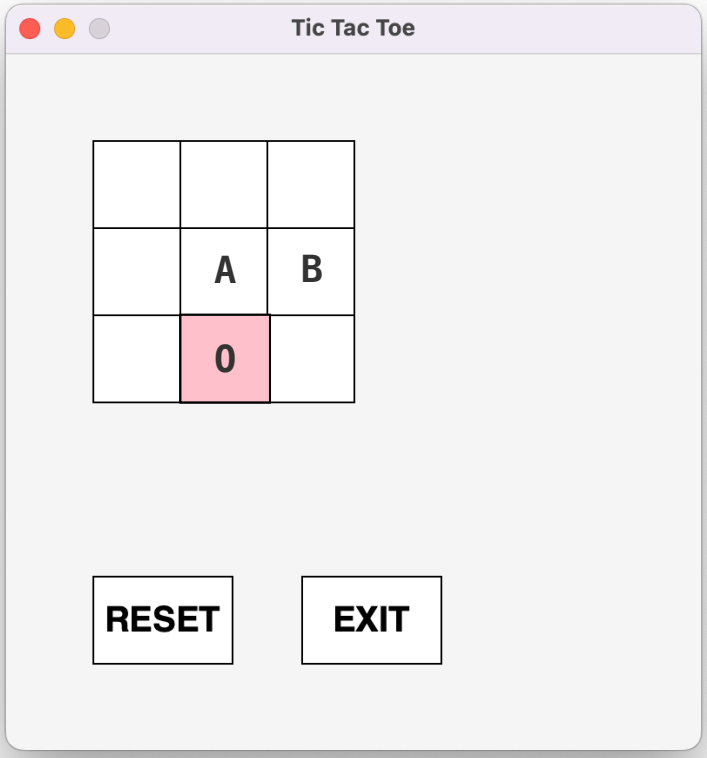


Testing TTTLetter

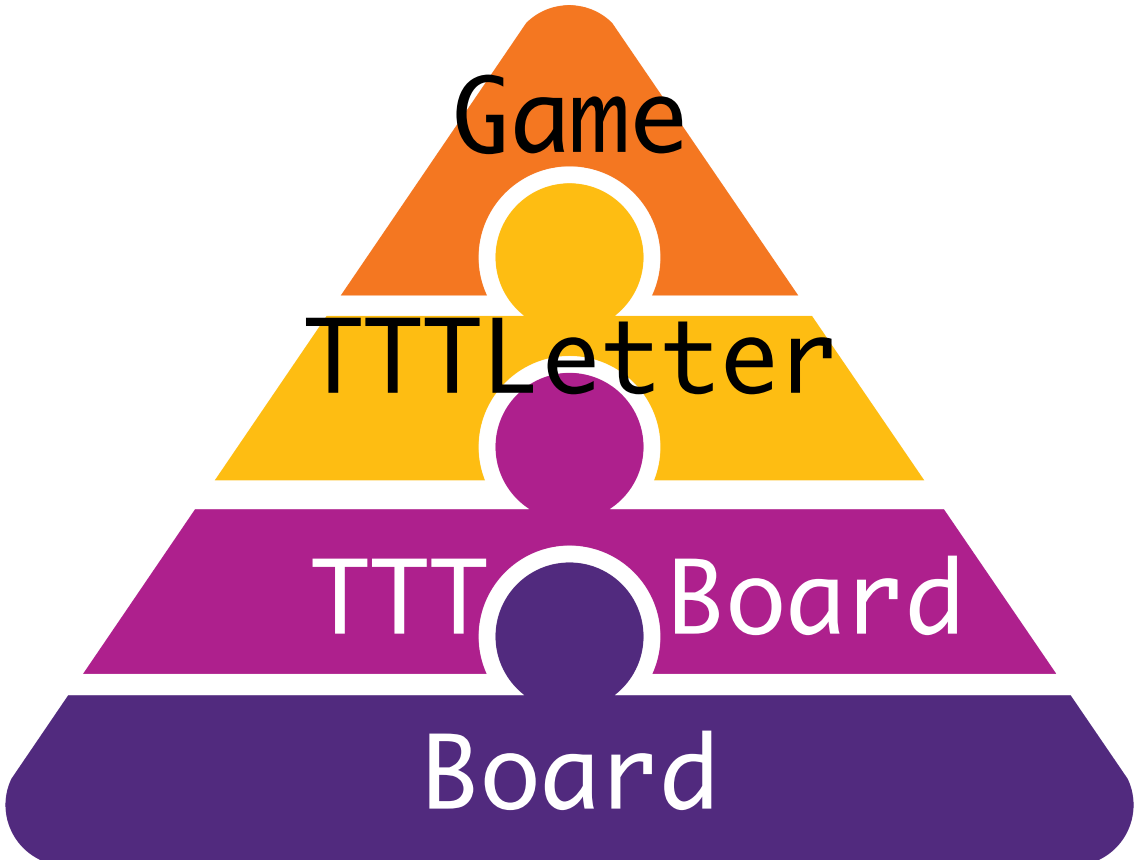
- It's always a good idea to test our class and methods in isolation

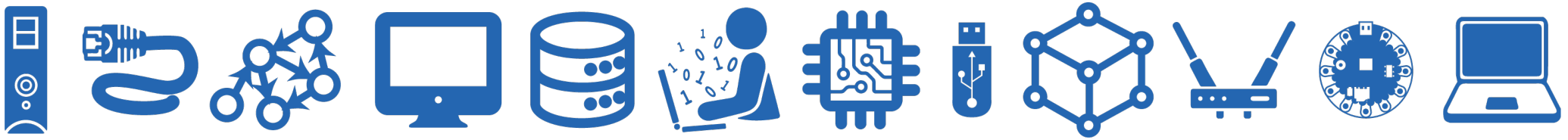
```
win = GraphWin("Tic Tac Toe", 400, 400)
board = Board(win, rows=3, cols=3)

letter = TTTLetter(board, 1, 1, "A")
letter2 = TTTLetter(board, 1, 2, "O")
letter3 = TTTLetter(board, 2, 1, "B")

letter2.setLetter("O")
print(letter2)
```

```
O at Board position (1, 2)
```



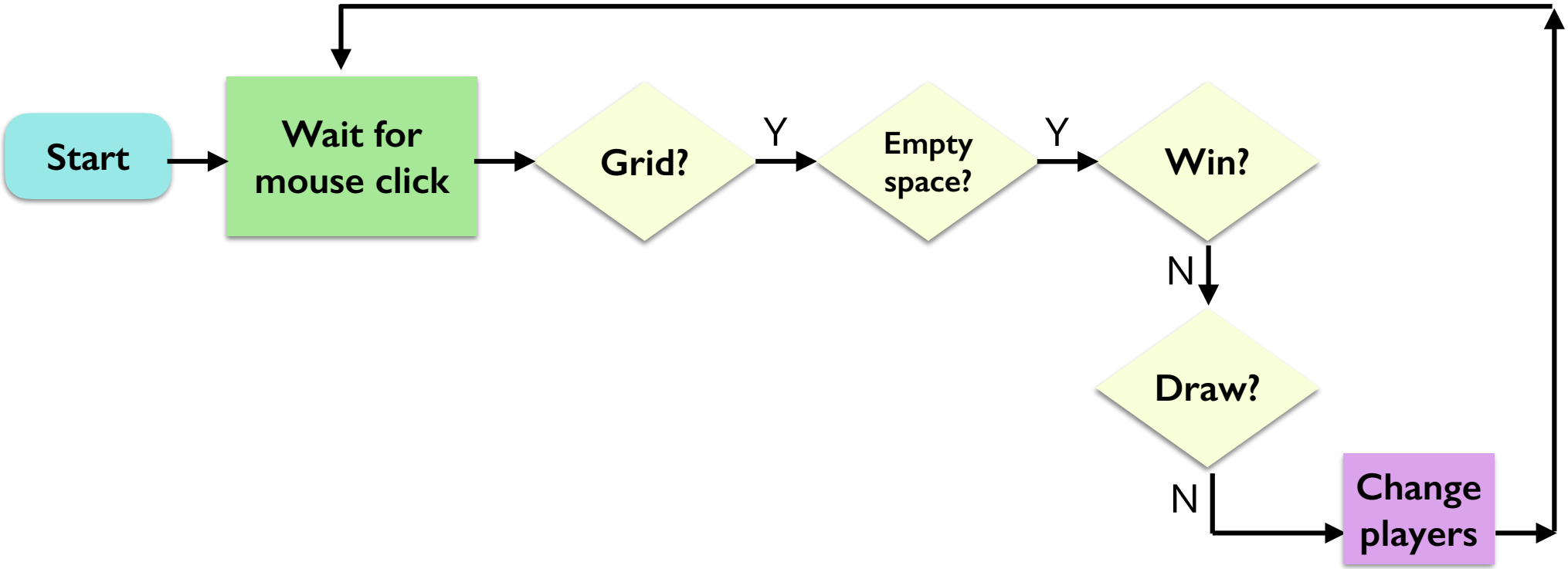
TTT Game Logic





Finally…TTT Game Logic

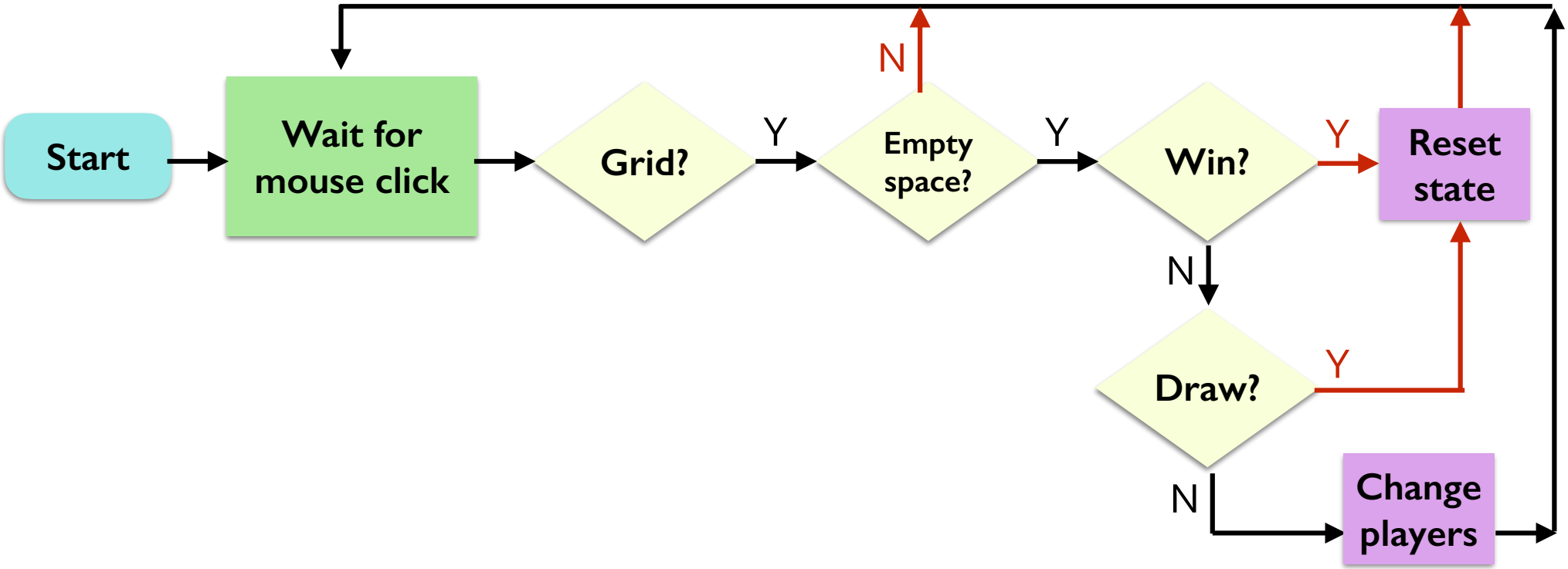
- Let's create a TTT flowchart to help us think through the state of the game at various stages



Let's think about the "common" case: a valid move in the middle of the game

Finally…TTT Game Logic

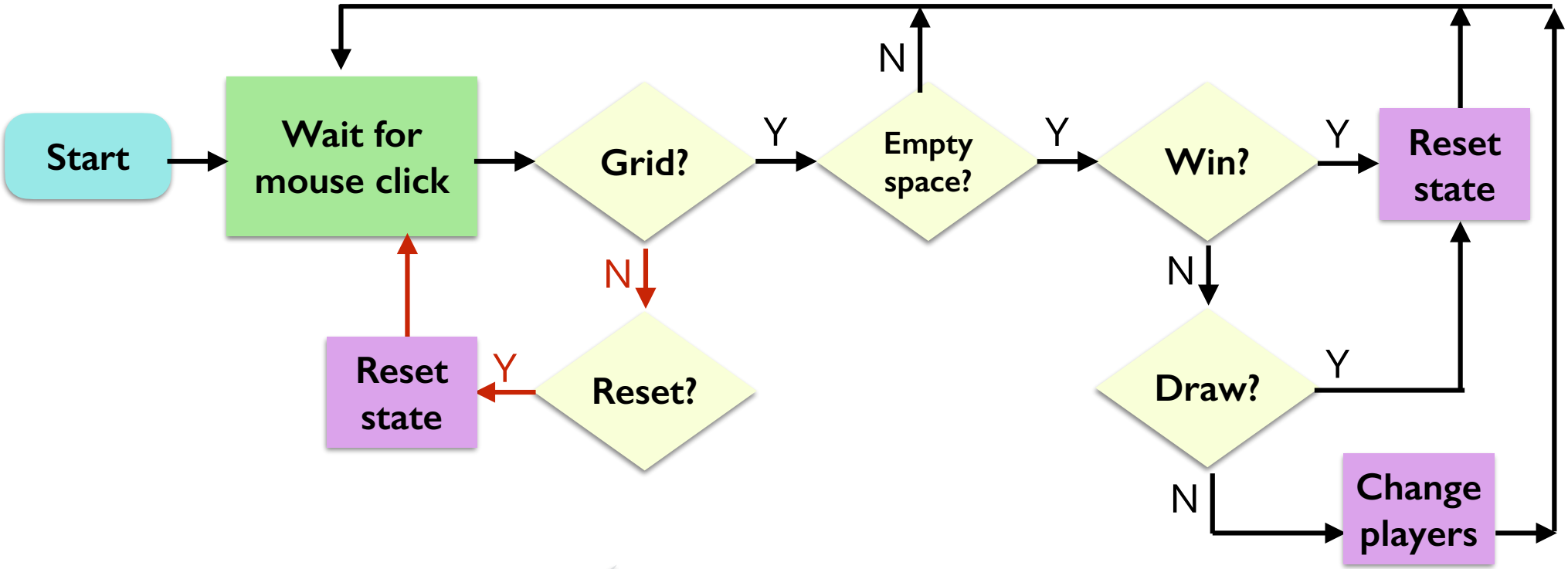
- Let's create a TTT flowchart to help us think through the state of the game at various stages



Now let's consider the case of a win, draw, or invalid move

Finally…TTT Game Logic

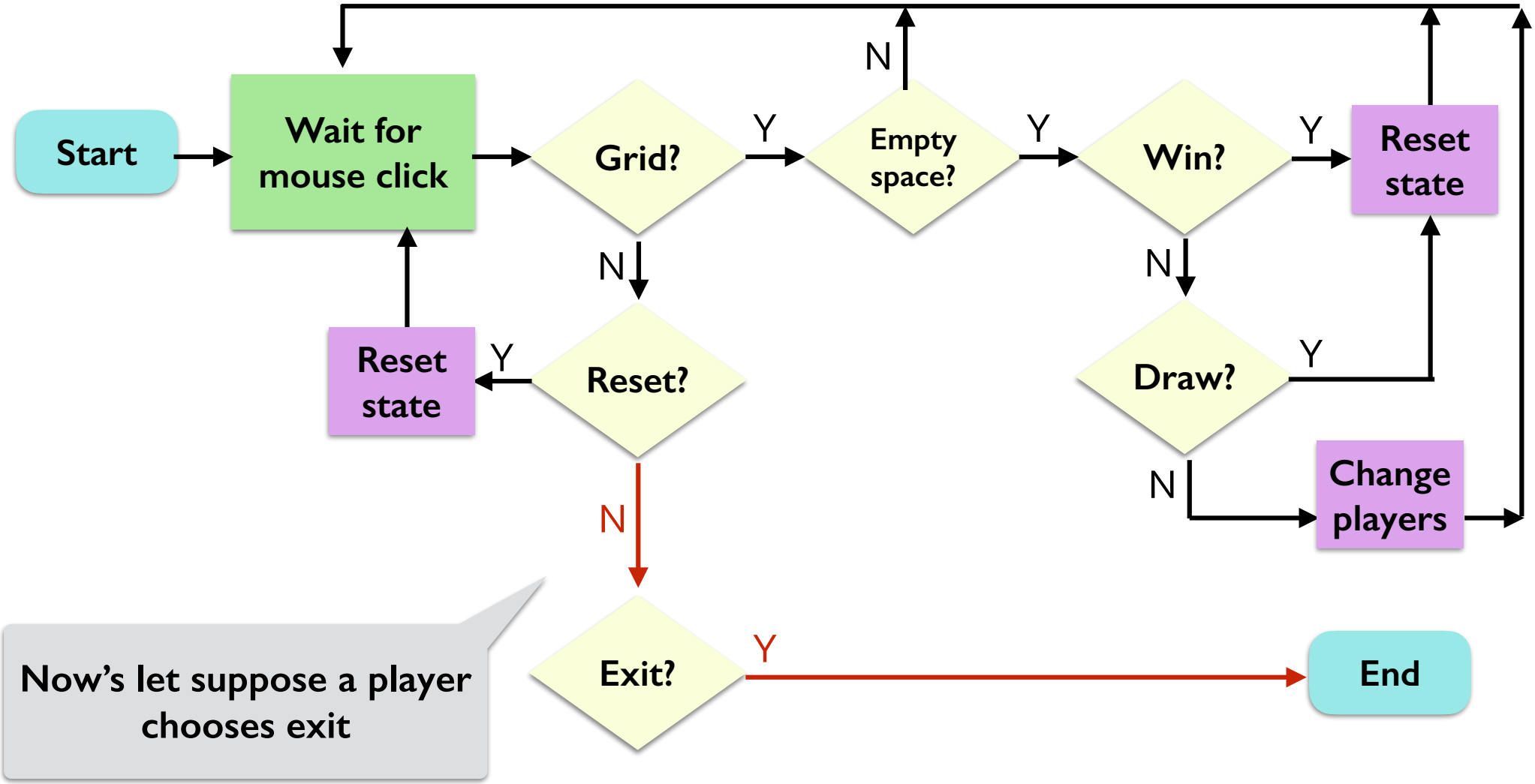
- Let's create a TTT flowchart to help us think through the state of the game at various stages



Now's let suppose a player chooses reset

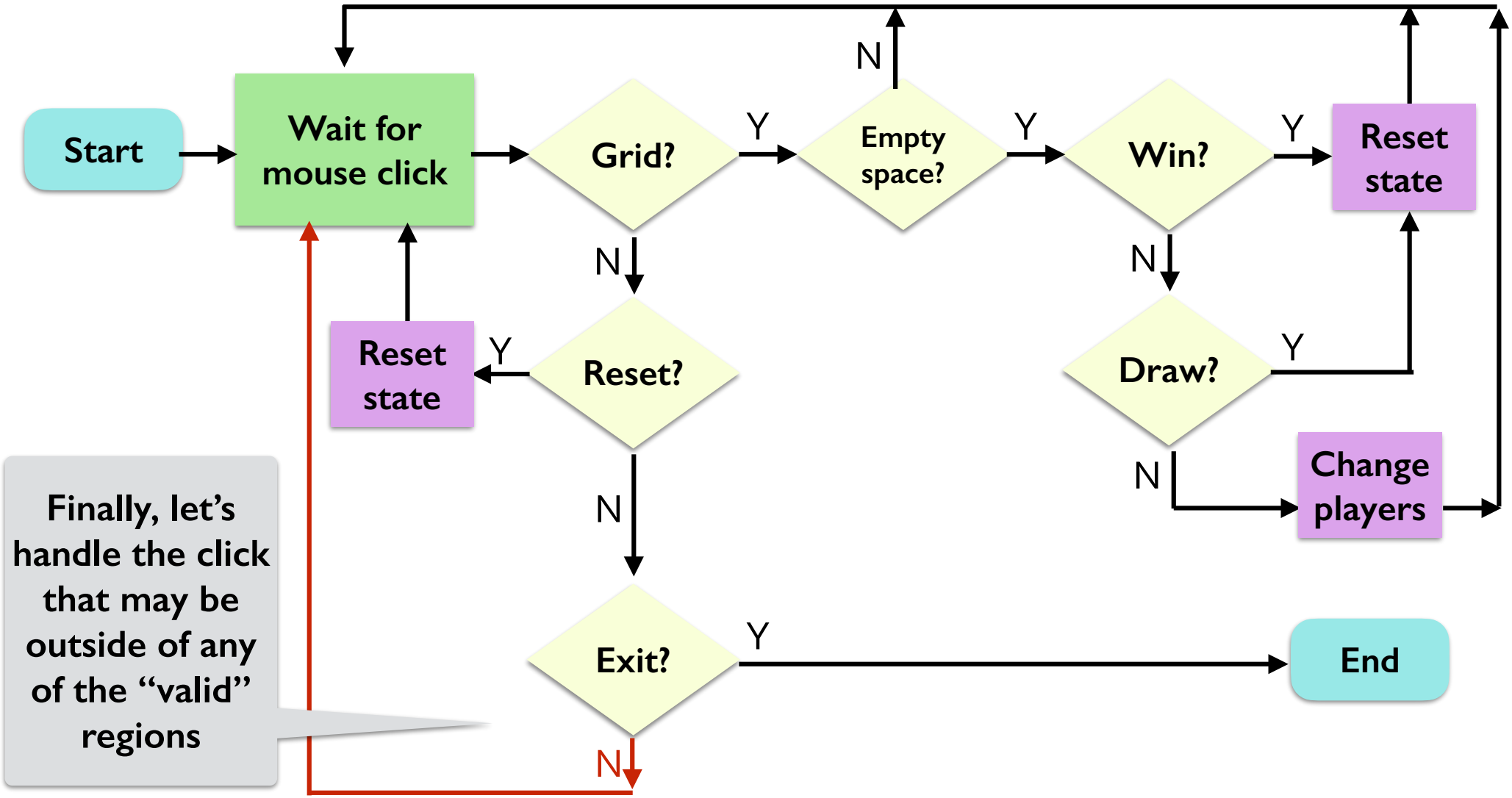
Finally…TTT Game Logic

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



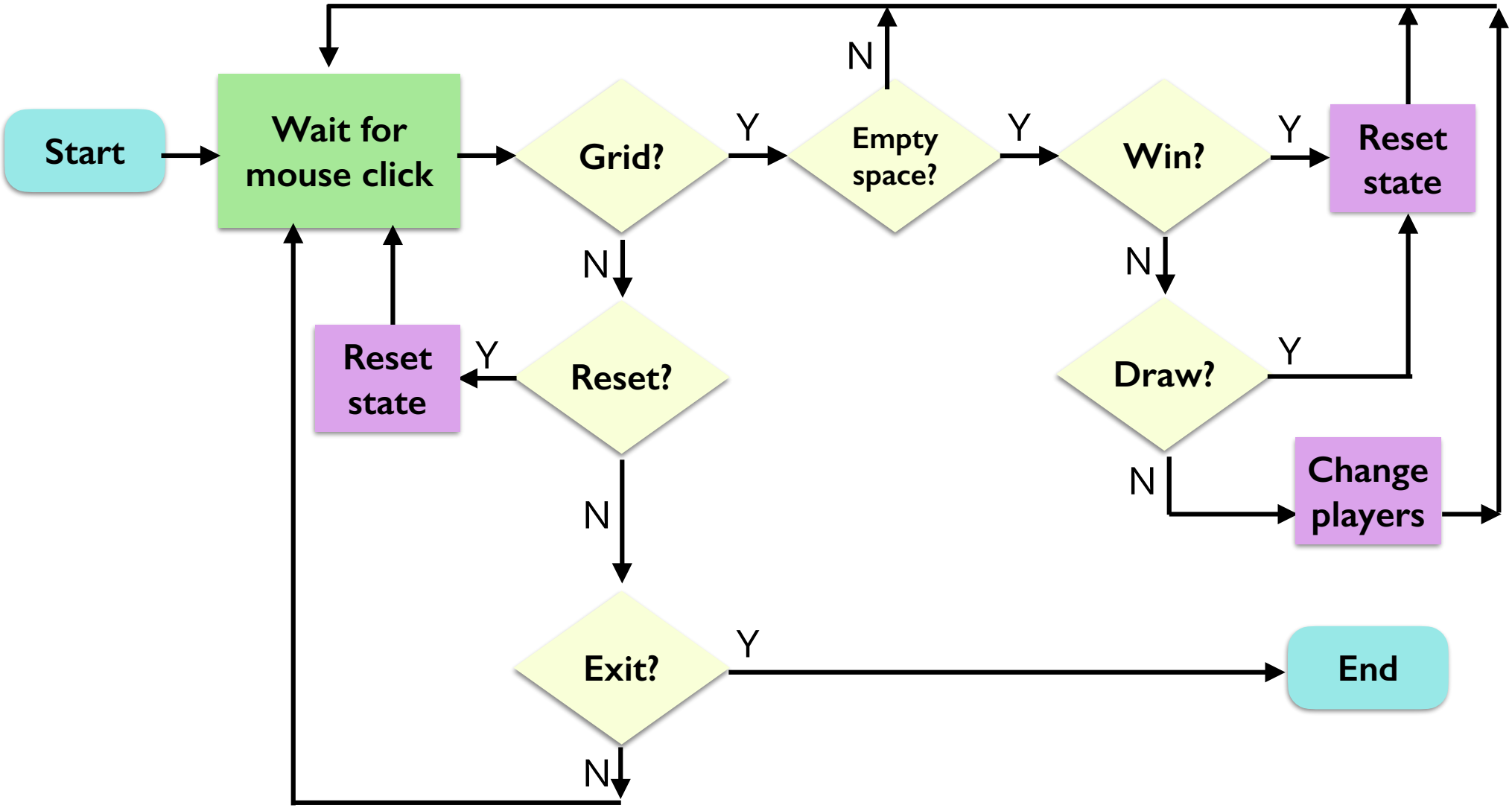
Finally…TTT Game Logic

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



Finally…TTT Game Logic

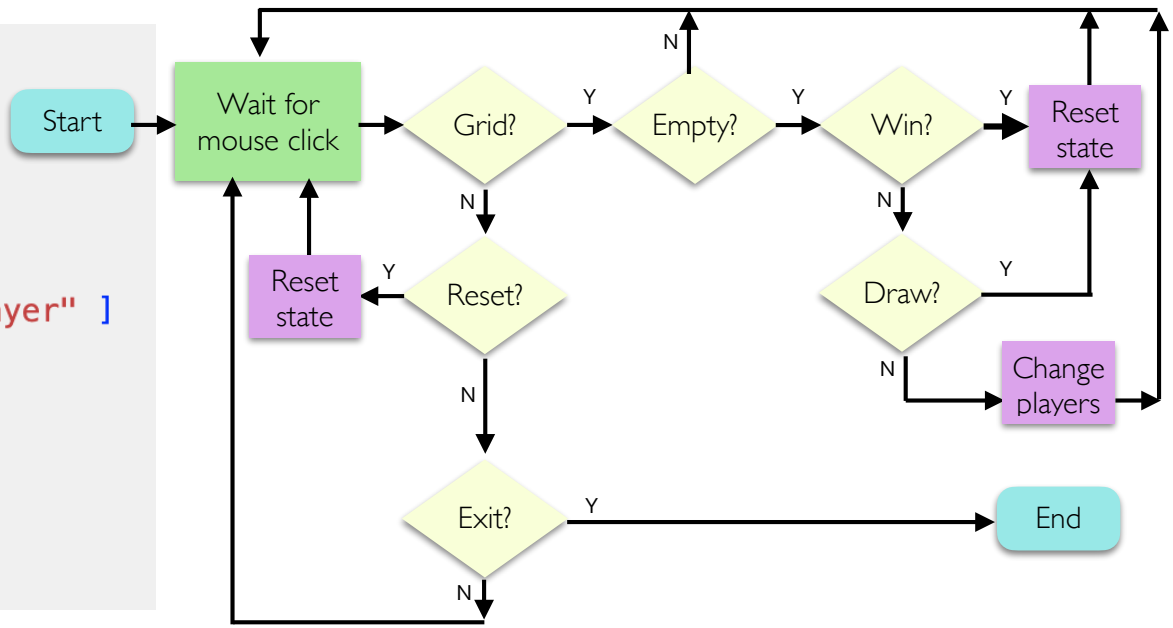
- Let's create a TTT flowchart to help us think through the state of the game at various stages



Translating our Logic to Code

- Let's think about **`__init__`**:
  - What do we need?
    - a board, player, and maybe numMoves (to detect draws easily)

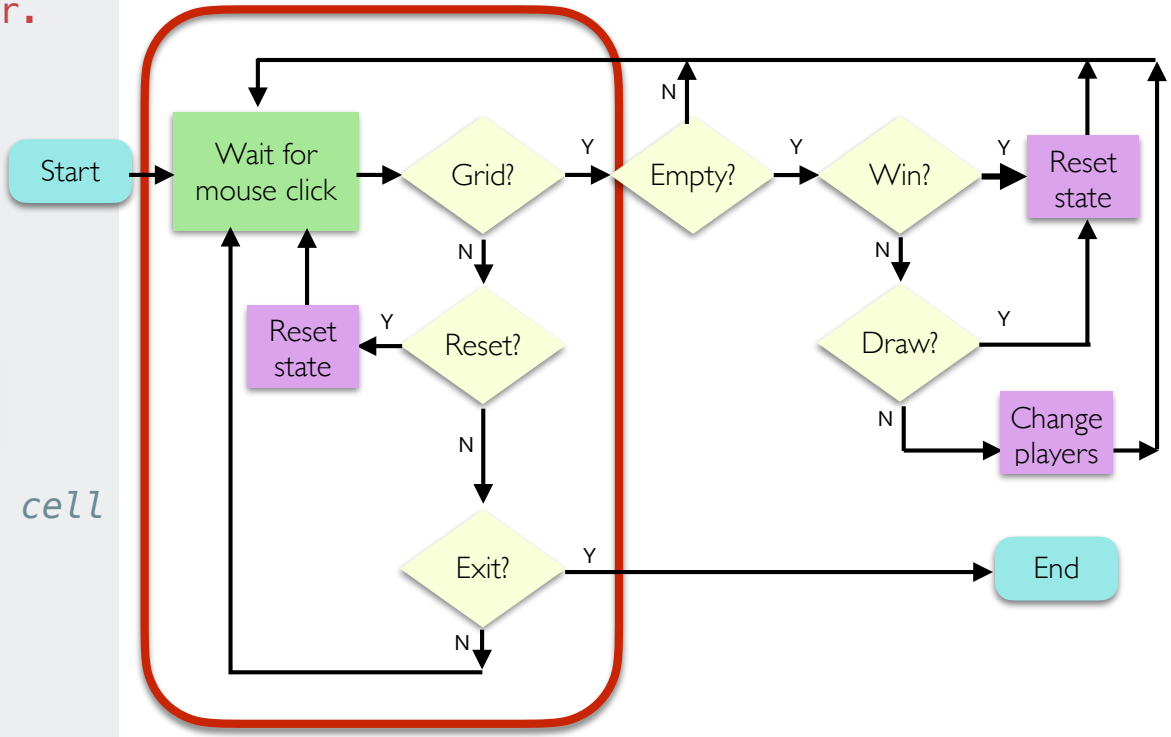
```
from graphics import GraphWin
from tttboard import TTTBoard
from tttletter import TTTLetter

class TTTGame:
  __slots__ = [ "_board", "_numMoves", "_player" ]

  def __init__(self, win):
    self._board = TTTBoard(win)
    self._numMoves = 0
    self._player = "X"
```



Translating our Logic to Code

- Now let's write a method for handling a single mouse click (point)
- 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

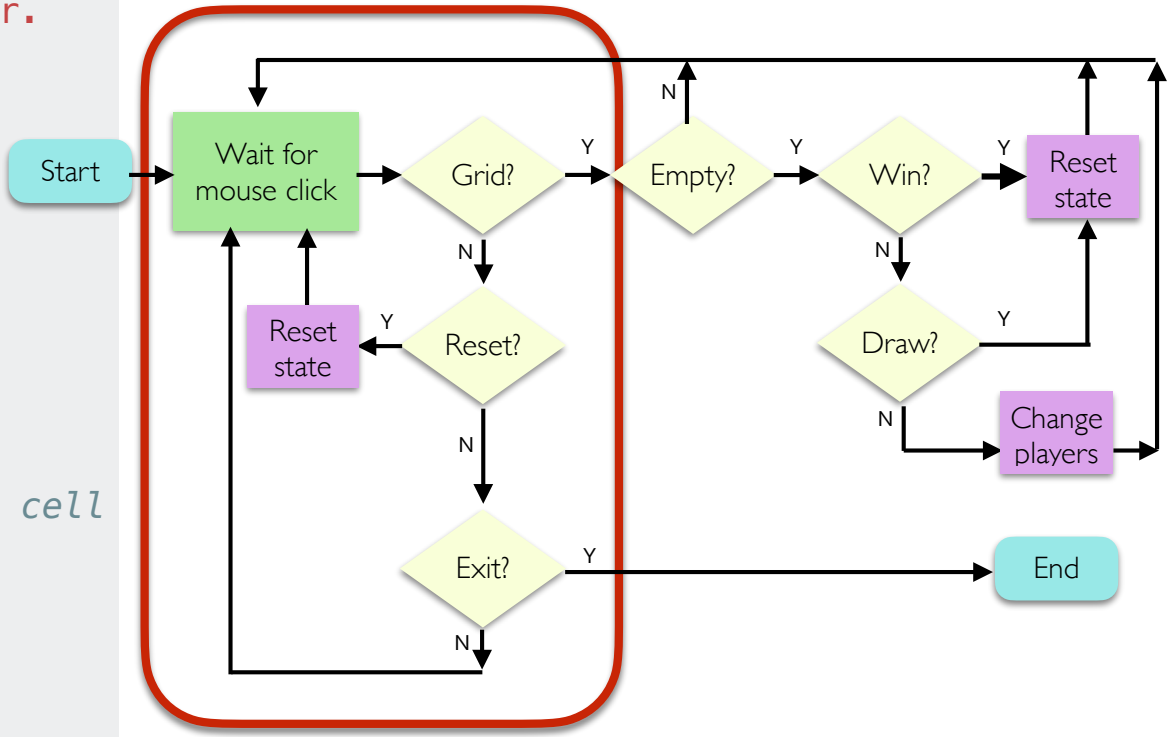
```
def doOneClick(self, point):
        """
        Implements logic for processing one
        click. Returns True if play should
        continue, & False if game is over.
        """
        # step 1: check for exit button
        # and exit (return False)
        if ??????????????

        # step 2: check for reset button
        # and reset game
        elif ??????????????

        # step 3: check if click is on a cell
        #in the grid
        elif ??????????????:
        # keep going!
        return True
```



Translating our Logic to Code

- Now let's write a method for handling a single mouse click (point)
- 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

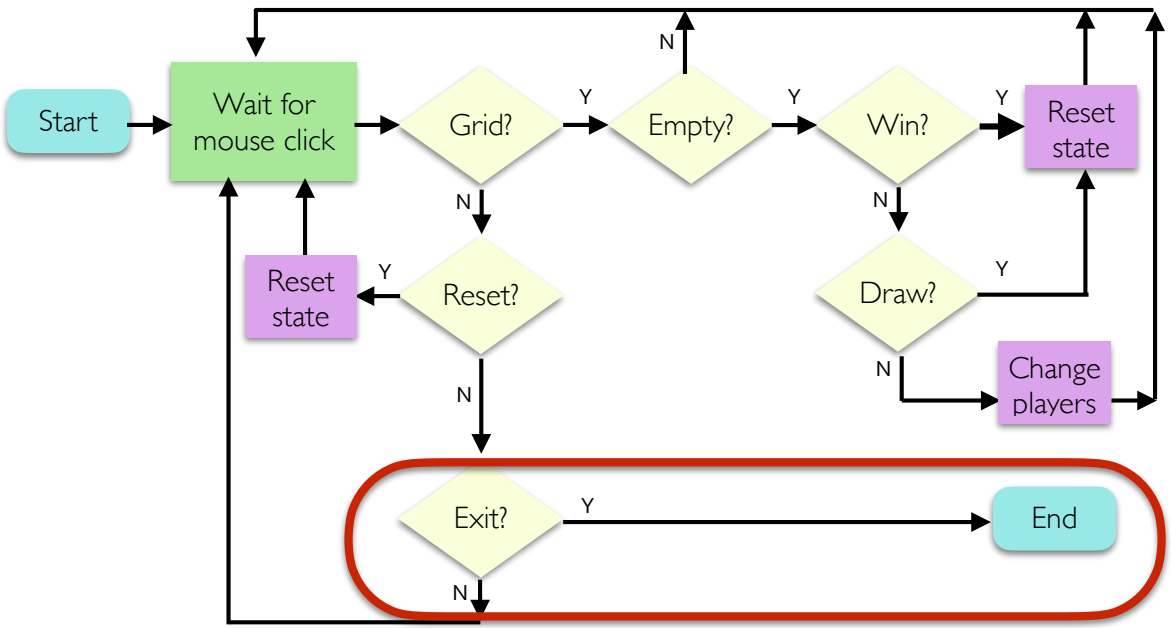
```
def doOneClick(self, point):
        """
        Implements logic for processing one
        click. Returns True if play should
        continue, & False if game is over.
        """
        # step 1: check for exit button
        # and exit (return False)
        if self._board.inExit(point):


        # step 2: check for reset button
        # and reset game
        elif self._board.inReset(point):


        # step 3: check if click is on a cell
        #in the grid
        elif self._board.inGrid(point):

        # keep going!
        return True
```



Translating our Logic to Code

- Let's handle the "exit" button first (since it's the easiest)

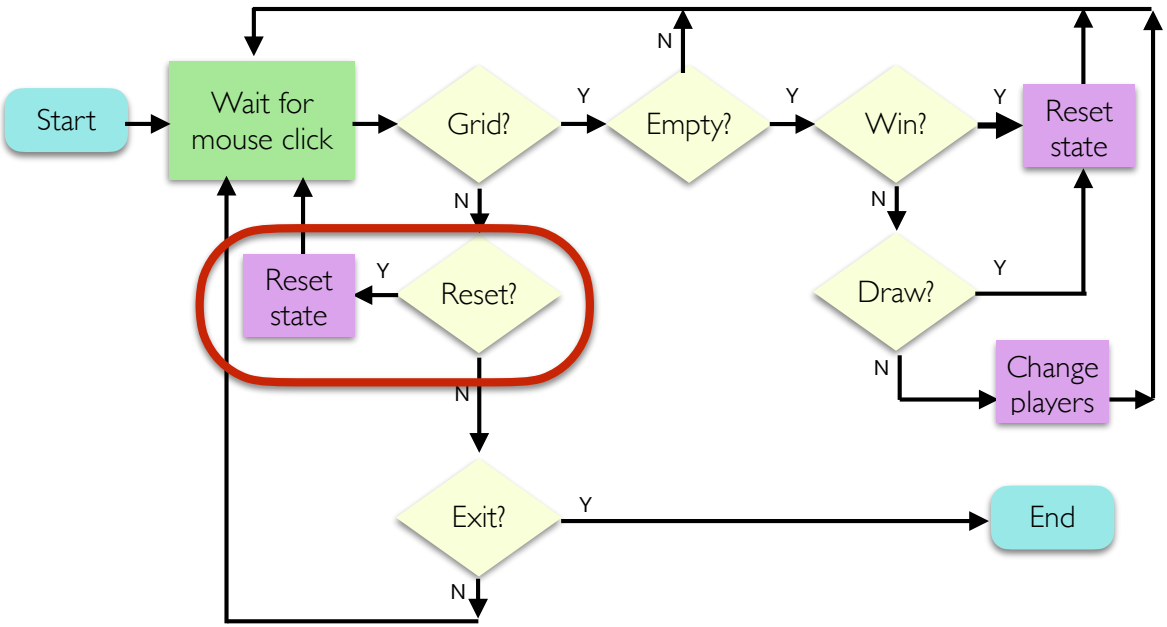
```
# step 1: check for exit button and exit (return False)
if self._board.inExit(point):
    # game over
    return False
```



Translating our Logic to Code

- Now let's handle reset

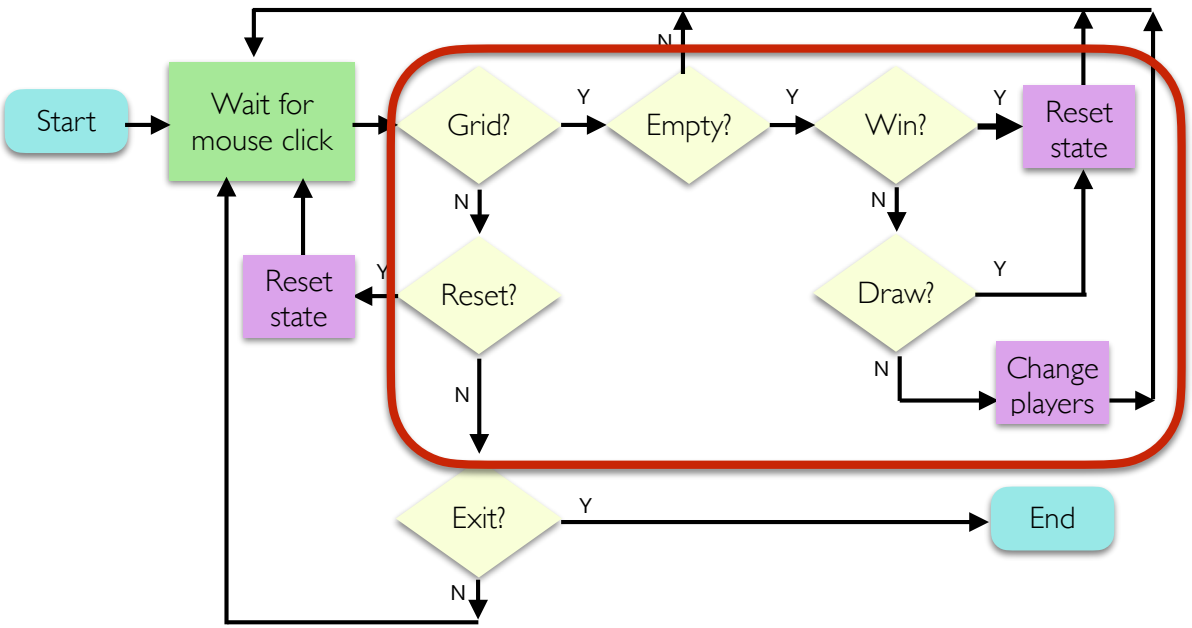
```
# step 2: check for reset button and reset game
elif self._board.inReset(point):
      self._board.reset()
      self._board.setStringToUpperText("")
      self._numMoves = 0
      self._player = "X"
```



Translating our Logic to Code

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

```
# step 3: check if click is on a cell in the grid
elif self._board.inGrid(point):

    # get the letter at the point the user clicked
    tlet = self._board.getTTTLetterAtPoint(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

```
# make sure this square is vacant
if


   # valid move, so increment numMoves


   # check for win or draw




   # not a win or draw, swap players

       # set player to X or O
```

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

```
# make sure this square is vacant
if tlet.getLetter() == "":
   tlet.setLetter(self._player)

   # valid move, so increment numMoves
   self._numMoves += 1

   # check for win or draw
   winFlag = self._board.checkForWin(self._player)
   if winFlag:
        self._board.setStringToUpperText(self._player+" WINS!")
   elif self._numMoves == 9:
        self._board.setStringToUpperText("DRAW!")
   # not a win or draw, swap players
   else:
       # set player to X or O
       if self._player == "X":
          self._player = "O"
       else:
          self._player = "X"
```

TTT Summary

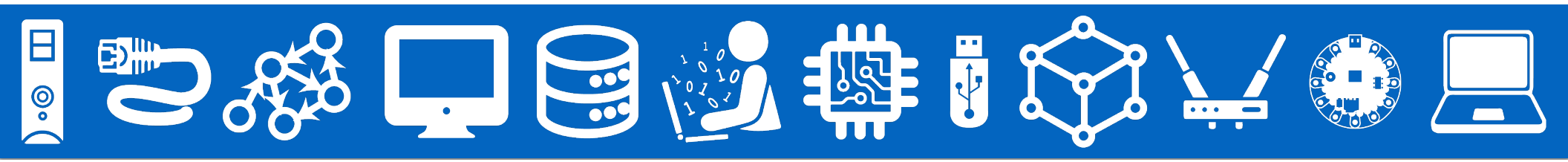
- Basic strategy
  - `Board`: start general, don't think about game specific details
  - `TTTBoard`: extend generic board with TTT specific features
    - Inherit everything, update attributes/methods as needed
  - `TTTLetter`: isolate functionality of a single `TTTLetter` on board
    - Think about 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

Boggle Strategies

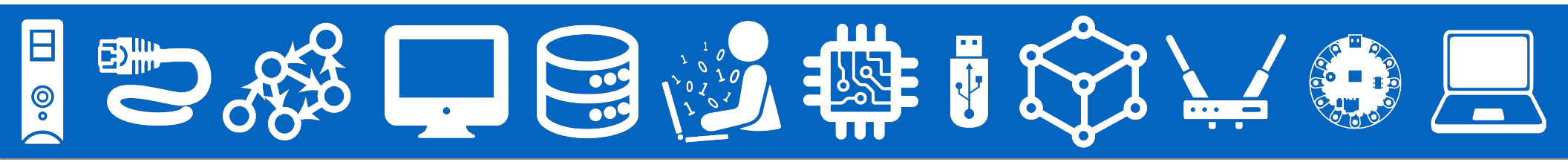
- At a high level, Tic Tac Toe and Boggle have a lot in common, but the game state of Boggle is more complicated
- In Lab 9 you should follow a similar strategy to what we did with TTT
- ***Don't forget the bigger picture as you implement individual methods***
- Think holistically about how the objects/classes work together
- Isolate functionality and test often (use `__str__` to print values as needed)
- **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 three logical parts)
  - **Parts 1 & 2 (BoggleLetter & BoggleBoard)** due Nov 16/17
  - We will run our tests on these and return automated feedback (similar to Lab 4 part 1); you are allowed/encouraged to revise it afterwards
  - **Parts 3 (BoggleGame)** (and revised Parts 1 and 2) due Nov 30/ Dec 1

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
- 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)
- **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

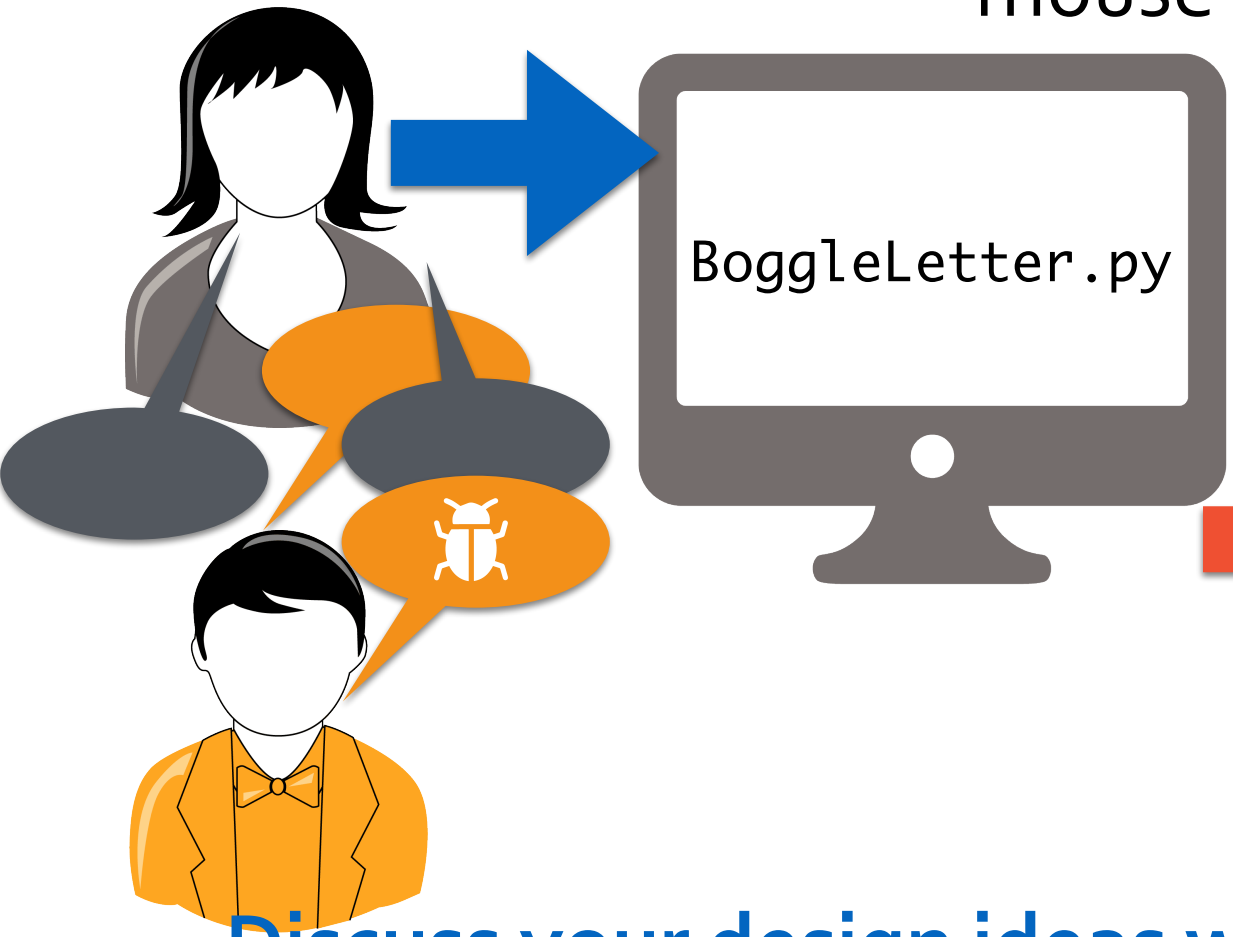
GOOD LUCK and HAVE FUN!

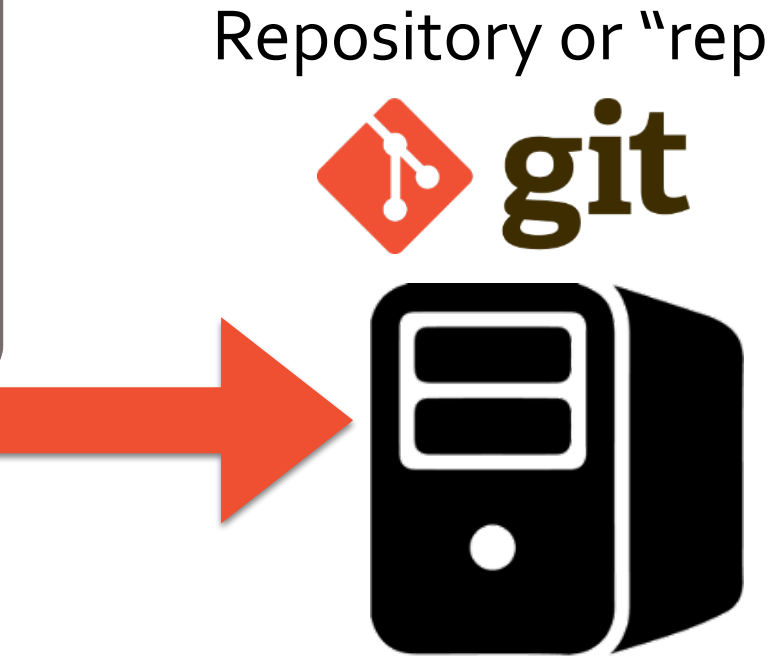
Working with a Partner

- "Pair Programming" (or programming with a partner) is an *Agile* software development technique from Extreme Programming
  - It's used in the real world!
  - Produces better solutions than produced individually!
  - Spreads knowledge!
- It's good to be able to talk through complex ideas with someone else before diving into implementation details
- Benefit from *both* partners' knowledge of problem-solving & debugging

`git` with a Partner

Pair Programming: One person "drives", take turns who uses keyboard/ mouse





Repository or "repo"

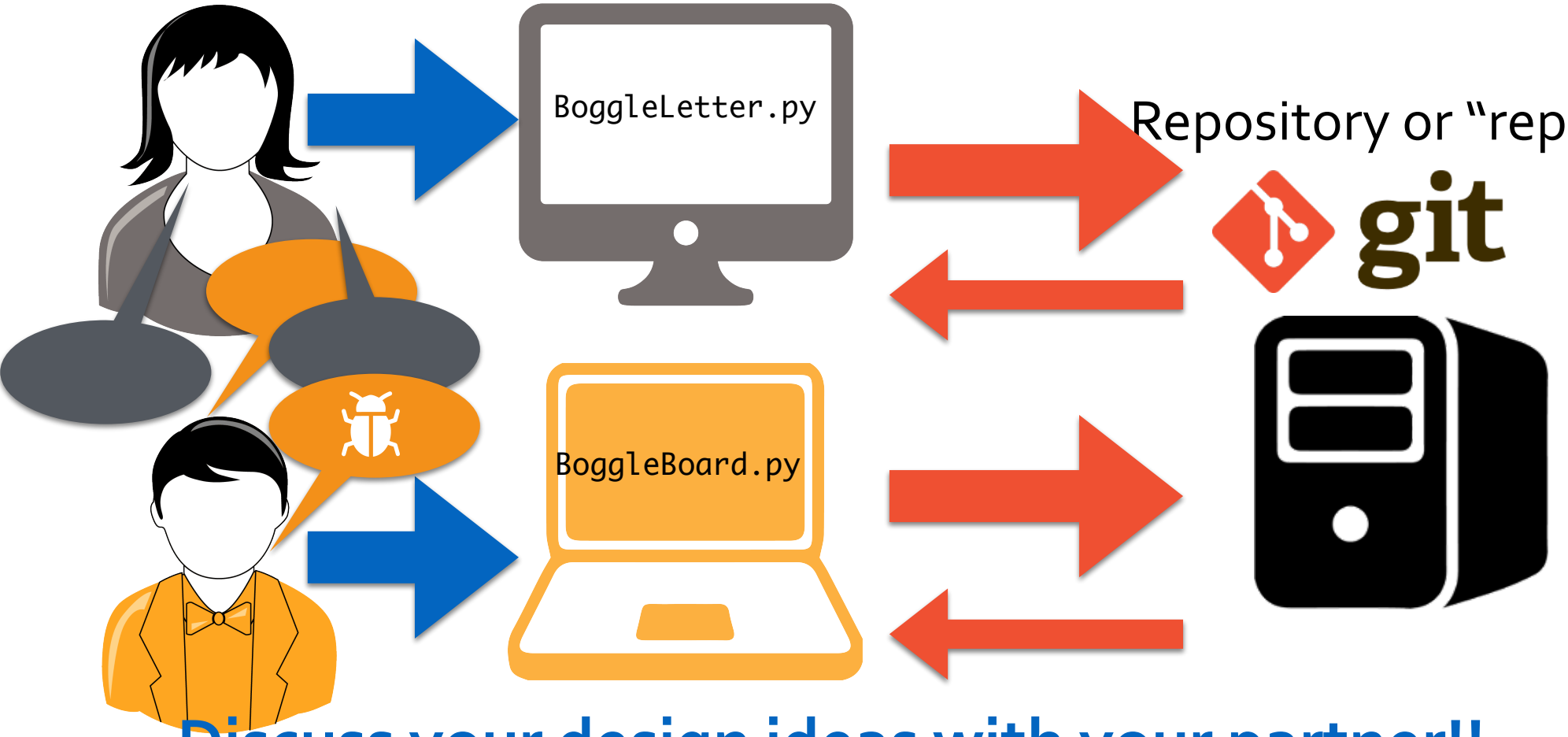
Discuss your design ideas with your partner!!

Identify bugs & bug fixes together!

`git` with a Partner

Jigsaw Programming:

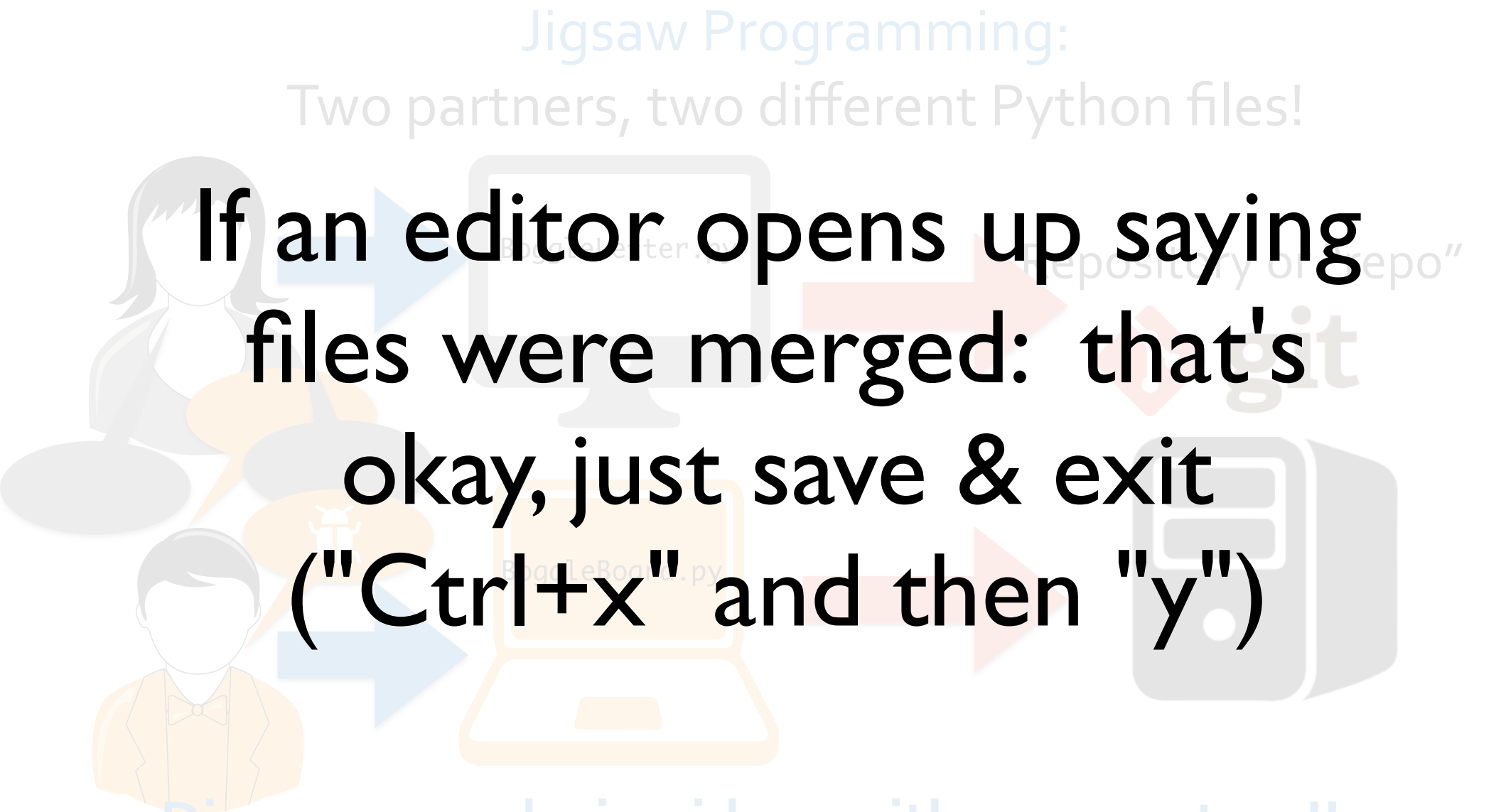
Two partners, two different Python files!



Discuss your design ideas with your partner!!

Identify bugs & bug fixes together!

`git` with a Partner



If an editor opens up saying files were merged: that's okay, just save & exit ("Ctrl+x" and then "y")

`git` with a Partner

DO: Talk to your partner *a lot*!

DO NOT WORK ON THE SAME FILE AT THE SAME TIME!

There will be frustration!

And suffering!

And Lida will probably have to save you!

Git Reminders

- If machine doesn't have the repo, **`git clone`** the repo
  - Grab URL from https://evolene.cs.williams.edu/ (or Lida's email)
  - `git clone <URL HERE>`
- **`git add/commit/push`** frequently, as you get work done

- To grab your partner's edits, **`git pull`**
  - (if you've already **`git clone`**d the repo)
  - If you have not **`git clone`**d the repo, then **`git clone`**

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?