Implementing Boggle

- Your task is to work with a partner to implement a one-player version of Boggle
  - Working with a partner: commit, push, pull often!
- More specifically, you will implement the following classes/scripts:
  - BoggleLetter
  - BoggleBoard (subclass of Board)
  - BoggleWords
  - Game
- BoggleLetter and BoggleBoard due Nov 17/18
- BoggleWords and Game due Dec 1/2
Merge conflicts

• Don’t panic! This just means you and your partner had edits to the same file

```bash
Merge branch 'main' of https://evolene.cs.williams.edu/cs134-labs/
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

• Just press “Ctrl+x” and “y” when prompted to exit
Review: Board Class

- Grid for the game
- Upper text area
- Right text area
- Lower text area
- Reset/Exit buttons
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.
BoggleLetter

• Think about how we implemented the TTTLetter class
• Capture all of the functionality required for a *single* BoggleLetter
• Start by reviewing TTTLetter
• Think about what extra functionality is required in Boggle
• Testing code should produce this pic

```
jeannie@rugger9-2021 jeannie-solution
A at Board position (1, 1)
BoggleLetter('1', '1', 'A', 'black')
B at Board position (1, 2)
C at Board position (3, 1)
A <-> B: True
B <-> A: True
C <-> C: False
C <-> A: False
B <-> C: False
```
BoggleBoard

• Inherit all methods and attributes from Board class
• Think about what additional features are needed
• Review TTTBoard as necessary
• Make sure you properly “shake the Boggle cubes”
• Tester code produces this:
Shaking Boggle Cubes

- We have row * col total cubes (16)
- For each spot in our grid, we need to:
  - First pick a random cube (pick a number from 0 to 15)
  - Then pick a random side on that cube
- After using a cube, we don’t want to use it again
- How to accomplish that?
- Move chosen cube to bottom of our list of lists (CUBES) by swapping with cube at the bottom
- Then when we pick the next cube, pick a random cube from the remaining cubes (by selecting a random number from 0 to 14)
  - Swap with second to last cube
- Rinse and repeat!
BoggleWords

• Think of this as a container for currently used BoggleLetters and your words
• Pay attention to the attributes and method definitions
• Prevent duplicates
• For extra credit, can implement a recursive word list instead (more details next week)
Game

- Implement all game logic here
- CAREFULLY think through game states (on pencil and paper first!)
- Use variables when necessary to capture relevant state
- Also handle reset and exit buttons
Good luck!