15-112, Week 7, 2D Lists and Animations Practice: Numerical Tic-Tac-Toe

See how far you and a partner can get during today's lab.

This week's lab is one larger problem rather than a set of short ones: you'll build a complete animated game with `cmu_graphics`, using a 2D list to hold the state of the board. Work in small steps, run your code after each one, and check what you see on the screen.

Rules for this lab: no recursion. You may use everything we've covered so far: strings, loops, lists, 2D lists, and `cmu_graphics` animation (`onAppStart`, `redrawAll`, `onStep`, `onMousePress`).

Setup

Start from the animations base file in the demo code repository: `week06/animations_base.py`. Make a `lab06` folder, make a copy of that file inside it, and rename the copy `numericalTicTacToe.py`. As in Lab 4, the `cmu_graphics` folder has to live in the same directory as the `.py` file you're running.

For reference, the skeleton looks like this:

```
from cmu_graphics import *

def onAppStart(app):
    pass

def onKeyPress(app, key):
    pass

def onMousePress(app, x, y):
    pass

def redrawAll(app):
    pass

def onStep(app):
    pass

runApp(width=800, height=800)
```

Note that the window starts at 800x800, but it can still be resized, so don't hardcode that size.

The Game

Numerical Tic-Tac-Toe is a variation of the classic game, invented by the mathematician Ronald Graham. Two players take turns marking the spaces in a three-by-three grid with a number. The player who first makes one horizontal, vertical, or diagonal line sum to **15** is the winner.

In the full game, each player chooses which number to play. In **our simplified version**, the number played on each turn is predetermined, and players choose only *where* to play.

Game Setup

- The window shows a 3x3 grid of numbers, **white on a black background**.
- Initially, every cell contains `0`, meaning empty.
- A **countdown timer** is displayed at the bottom of the screen.
- **Player One** goes first, and the timer starts at **20 seconds**.

Gameplay Rules

- Player One plays the **odd** numbers (1, 3, 5, 7, 9). Player Two plays the **even** numbers (2, 4, 6, 8).
- The sequence of plays is always fixed: Player One places `1`, Player Two places `2`, Player One places `3`, Player Two places `4`, and so on. (So the number placed on each turn is just the number of the turn.)
- On each turn, the current player clicks an empty cell.
- If the clicked cell contains `0`, the move is legal: the cell takes on the player's number and the countdown timer resets to 20 seconds. If the cell is already filled, nothing happens, and it is still the same player's turn.

Winning Condition

- The first player to make a row, column, or diagonal that **sums to 15** wins. Note that a winning line may contain `0`s (for example, `9 + 6 + 0`).
- If the countdown timer reaches `0` before the current player makes a move, **the other player wins**.
- When a player wins, a message appears at the **top** of the window: `"Player One Won!"` or `"Player Two Won!"`.
- Once the game is over, clicks no longer change the board and the timer stops.

Draw Condition

- If every cell is filled and nobody has a line that sums to 15, the message `"It's a Draw"` appears at the top of the window.

Make reasonable assumptions for anything not specified here.

Suggested Plan

Don't try to write everything at once. Get each step working before you go on to the next.

1. **Draw the empty board.** You should see a 3x3 grid of white zeros.
2. **Add the game state.** Decide what else you need to store in `app`. Think about: whose turn it is (or which number is next), how much time is left, and whether the game is over (and who won, if anyone).
3. **Handle clicks.** In `onMousePress`, convert `(mouseX, mouseY)` into a `(row, col)`. Be careful with clicks on the very edge of the window. Then place the next number if the cell is empty.
4. **Detect a win.** Write a helper that returns `True` if any row, column, or diagonal of `app.board` sums to 15. Test it on its own with hand-made boards before you hook it up to the game.
5. **Detect a draw.** Write a helper that returns `True` if no cell contains `0`. Decide which check should happen first if the last move both fills the board *and* makes 15.
6. **Add the countdown.** Use `onStep` to count down. Remember that `onStep` is called `app.stepsPerSecond` times per second (10 by default), so don't subtract a whole second on every step. Display the time at the bottom of the window.
7. **Show the status message** at the top of the window when the game ends, and stop the timer and ignore clicks.

Extensions (If You Finish Early)

- Add a **restart** key: pressing `r` calls `resetBoard` and resets all the other game state.
- **Highlight the winning line** (for example, with a colored line or by coloring the numbers in it).
- Show **whose turn it is** and **which number they are about to play**, in addition to the timer.
- Make the timer text **turn red** when 5 or fewer seconds remain.
- Let the player choose **any unused number** on their turn (the real version of Graham's game), rather than a fixed sequence. What new information do you need to keep track of?