Intro to Coding

Class 8

Collisions

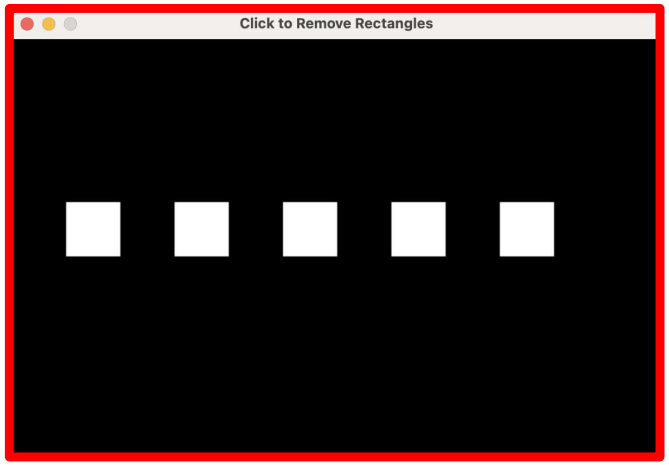
- **Collision detection:** determining if two or more game objects overlap

Here are some more important collision detection methods to know:

- `Rect.collidepoint()`
  - A method in the Pygame library used to check if a point lies within the boundaries of a `Rect` object.
  - This function is commonly used for collision detection, especially for interactions involving **mouse clicks**
- `Rect1.colliderect(Rect2)`
  - Checks if two rectangles overlap (collision detection!)
  - The method is called **on** `Rect1` and `Rect2` is passed as an **argument**
  - The method returns True if the two rectangles intersect, and False otherwise.
- `Rect.collidelist(list_of_rects)`
  - Recall this from last class!
  - Loops through the list in order and checks for overlap with the Rect it is called on
  - Returns the index of the first overlapping rect from the list OR a -1 if no rect overlaps

Collisions activity

You will be given code for a Pygame program that displays a screen with a black background and 5 white rectangles



Modify the code so that every time the user clicks a white rectangle, it disappears

Prewritten code can be found here:
https://gist.github.com/al-1108/f36801167d5b77ab9a0dc720e50035c8

Solution

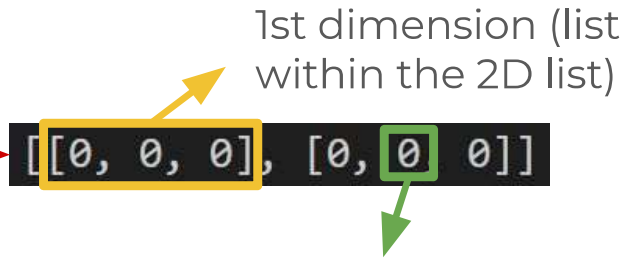
```
# when you click
elif event.type == pygame.MOUSEBUTTONDOWN:
    # get mouse position
    mouse_pos = pygame.mouse.get_pos()
    # for each rectangle
    for r in rects:
        # if the rectangle collides with the mouse (when the mouse is clicked)
        if r.collidepoint(mouse_pos):
            # remove rectangle
            rects.remove(r)
```

Full solution:
https://gist.github.com/al-1108/7373c11a277c8c6fa0c47f1b19726ec8

2D Lists

- 2D lists **store data** in both **rows** and **columns** (*ie. multiplication tables, chess/checker board*). In other words, it's a **list within a list**.
- To generate a 2D list with dimensions `N` by `M`, use two for loops
  - This 2D list contains `N` lists, each list has a size of `M`

```
n = 2
m = 3
list_2d = [[0 for i in range(m)] for j in range(n)]
```



- An element at row `i` and column `j` can be accessed with the following syntax: `list_name[i - 1][j - 1]`. We subtract 1 due to 0-indexing.

```
l = [[1, 2, 3], [4, 5, 6]]
print(l[1][2])
```

2D Lists

- Just like 1D lists, 2D lists can be printed using `print(list_name)`

```
l = [[1, 2, 3], [4, 5, 6]]
print(l)
```

```
[[1, 2, 3], [4, 5, 6]]
```

- To print a 2D list row by row, use an iterator-based `for` loop

```
l = [[1, 2, 3], [4, 5, 6]]
for i in l:
    print(i)
```

```
[1, 2, 3]
[4, 5, 6]
```

Appending with 2D Lists

- To extend a 2D list, more lists can be *appended* to the 1st dimension

```
l = [[1, 2, 3], [4, 5, 6]]
l.append([7, 8, 9])
for i in l: print(i)
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

- Values can also be *appended* to the 2nd dimension, but the **row** must be *specified*

```
l = [[1, 2, 3], [4, 5, 6]]
l[1].append(7)
for i in l: print(i)
```

```
[1, 2, 3]
[4, 5, 6, 7]
```

2D Lists Activity

Write a program that will generate a 5x5 table, which is initially empty. It should then prompt the user to input the number of changes they would like to make.

For each change, the user will enter the row **r** and column **c** of the cell they would like to change, as well as its new value. The program should then print the table after each change.

```
How many changes would you like to make to the table? 3
Enter row #: 1
Enter column #: 2
Enter the cell's new value: abc
[' ', 'abc', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
Enter row #: 5
Enter column #: 5
Enter the cell's new value: abcdefg
[' ', 'abc', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', 'abcdefg']
Enter row #: 1
Enter column #: 2
Enter the cell's new value: 1234567890
[' ', '1234567890', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', 'abcdefg']
```

Solution

Use the `for _ in range` loop to generate the empty grid. When updating the grid, ensure that `1` is subtracted from both the row and column. The `print_grid` function ensures the grid is printed row by row.

```
def print_grid(g):
    for i in g:
        print(i)


g = [[" " for _ in range(5)] for _ in range(5)]
n = int(input("How many changes would you like to make to the table? "))
for i in range(n):
    r = int(input("Enter row #: "))
    c = int(input("Enter column #: "))
    s = input("Enter the cell's new value: ")
    g[r - 1][c - 1] = s
    print_grid(g)
```

Tic Tac Toe

Part 1: Setup and Grid

Setup: Initialization

Importing the pygame and random libraries

Initializing pygame so its features can be used such as screen

Declaring fonts to be used

Note: `Coordinate` is just a pair of points, but creating a class specifically for it just makes the code easier to read

```
import pygame
import random

pygame.init()
screen = pygame.display.set_mode((800, 800))

font = pygame.font.Font("freesansbold.ttf", 50)
small_font = pygame.font.Font("freesansbold.ttf", 20)
text_x = font.render("X", True, (255, 255, 255))
text_o = font.render("O", True, (255, 255, 255))

class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Setup: Initialization

What important buttons immediately come to our attention?

We want to be able to reset the game after each round, so we will add a `clear` button.

Additionally, the game should have both singleplayer (vs. computer) and multiplayer, so we will add a button to switch modes

```
import pygame
import random

pygame.init()
screen = pygame.display.set_mode((800, 800))

font = pygame.font.Font( name: "freesansbold.ttf",  size: 50)
small_font = pygame.font.Font( name: "freesansbold.ttf",  size: 20)
text_x = font.render( text: "X",  antialias: True,  color: (255, 255, 255))
text_o = font.render( text: "O",  antialias: True,  color: (255, 255, 255))

class Coordinate:  1 usage
    def __init__(self, x, y):
        self.x = x
        self.y = y

clear_button_rect = pygame.Rect(300, 50, 200, 75)
mode_button_rect = pygame.Rect(575, 50, 175, 75)
```

Setup: Initialization

So far, we will start by adding their respective ‘hitboxes’ that mark the region where the button can be clicked

Keep this out of the main loop, just like with the other initialization steps - we only need to define this once.

```
import pygame
import random

pygame.init()
screen = pygame.display.set_mode((800, 800))

font = pygame.font.Font( name: "freesansbold.ttf",  size: 50)
small_font = pygame.font.Font( name: "freesansbold.ttf",  size: 20)
text_x = font.render( text: "X",  antialias: True,  color: (255, 255, 255))
text_o = font.render( text: "O",  antialias: True,  color: (255, 255, 255))

class Coordinate:  1 usage
    def __init__(self, x, y):
        self.x = x
        self.y = y

clear_button_rect = pygame.Rect(300, 50, 200, 75)
mode_button_rect = pygame.Rect(575, 50, 175, 75)
```

Setup: Grid and UI

Let's move on to setting up the grid. In the main loop, we can draw four lines to form the grid - try modifying these coordinates to your liking.

Note: we simplify the code a bit by putting `white` instead of `(255,255,255)`. Don't forget to initialize this at the top!

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
    screen.fill((0, 0, 0))

    white = (255, 255, 255)
    pygame.draw.line(screen, white, (200 + 133, 200), (200 + 133, 600))
    pygame.draw.line(screen, white, (200 + 133 + 134, 200), (200 + 133 + 134, 600))
    pygame.draw.line(screen, white, (200, 200 + 133), (600, 200 + 133))
    pygame.draw.line(screen, white, (200, 200 + 133 + 134), (600, 200 + 133 + 134))
```

Setup: Grid and UI

Let’s move on to the reset button. Recall from last class what we did to generate text. This is pretty much the same thing:

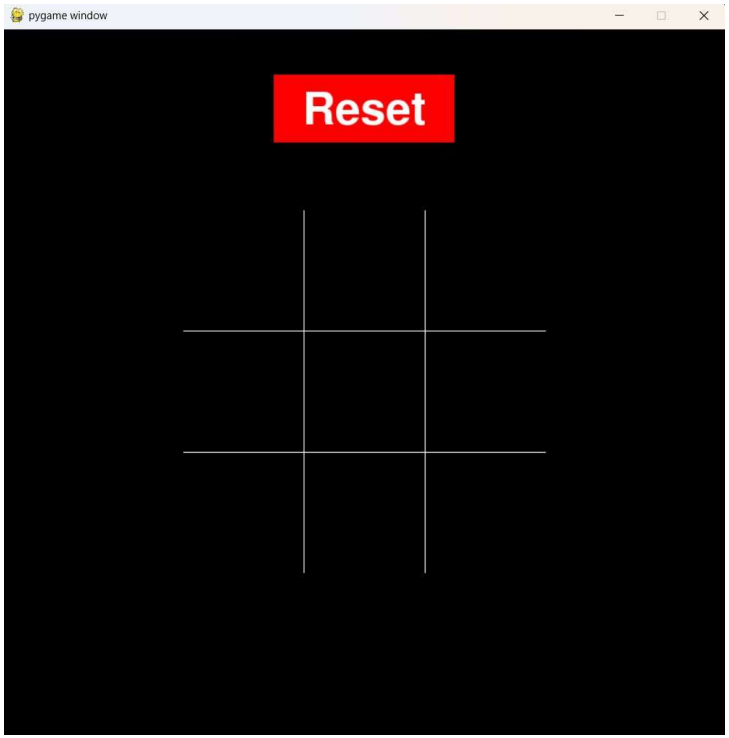
```
clear_board_button = pygame.draw.rect(screen, color: (255, 0, 0), clear_button_rect)
text_reset = font.render( text: "Reset", antialias: True, color: (255, 255, 255))
text_reset_rect = text_reset.get_rect()
text_reset_rect.center = (400, 90)
screen.blit(text_reset, text_reset_rect)

pygame.display.update()
```

Add this after the part where we drew the grid, in the main loop.

Progress Check

Run your code. You should see something like this:



Board Class __init__ Function

We first create a board class to play the game on, initializing a 2D array for our board values to see which box has what value. These lines should be placed above your previous code, but right below

```
class Board:  1 usage
    def __init__(self):
        self.board_values = [["0", "0", "0"],
                             ["0", "0", "0"],
                             ["0", "0", "0"]]
```

Board Class __init__ Function

Then we have to draw the invisible boxes on the screen. Values such as current move and game over are also set here since it is in relevance to the game itself which will be run on the board.

1

```
        self.current_move = "X"
        self.squares = [
                        # row 1
                        pygame.Rect(200, 200, 133, 133),
                        pygame.Rect(333, 200, 134, 133),
                        pygame.Rect(467, 200, 133, 133),
```

2

```
# row 2
pygame.Rect(200, 333, 133, 133),
pygame.Rect(333, 333, 134, 133),
pygame.Rect(467, 333, 133, 133),

# row 3
pygame.Rect(200, 467, 133, 133),
pygame.Rect(333, 467, 134, 133),
pygame.Rect(467, 467, 133, 133),

]
```

3

```
                        ]
        self.game_over = False

        self.alternate_computer_move_value = False
        self.generate_computer_moves = False
```

draw_board function

The `draw_board` function is used to draw the Tic Tac Toe board. This is done using the `pygame.draw.line` function. It takes in the following parameters: surface, colors, start_pos, end_pos, width. We are simply taking the lines we have already written under the while loop, and moving into a function.

```
45      def draw_board(self, surface):
46          white = (255, 255, 255)
47          pygame.draw.line(surface, white,
48                           (200 + 133, 200), (200 + 133, 600))
49          pygame.draw.line(surface, white,
50                           (200 + 133 + 134, 200), (200 + 133 + 134, 600))
51          pygame.draw.line(surface, white,
52                           (200, 200 + 133), (600, 200 + 133))
53          pygame.draw.line(surface, white,
54                           (200, 200 + 133 + 134), (600, 200 + 133 + 134))
```

draw_board function

Now that we have initialized our object and method, we can now simplify the code in the main method

Initializing a new `Board` object

```
BOARD = Board()
clear_button_rect = pygame.Rect(300, 50, 200, 75)
mode_button_rect = pygame.Rect(575, 50, 175, 75)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()

    screen.fill((0, 0, 0))
    BOARD.draw_board(screen)
```

All instances of `pygame.draw.line()` can be now be replaced with the `.draw_board()` function