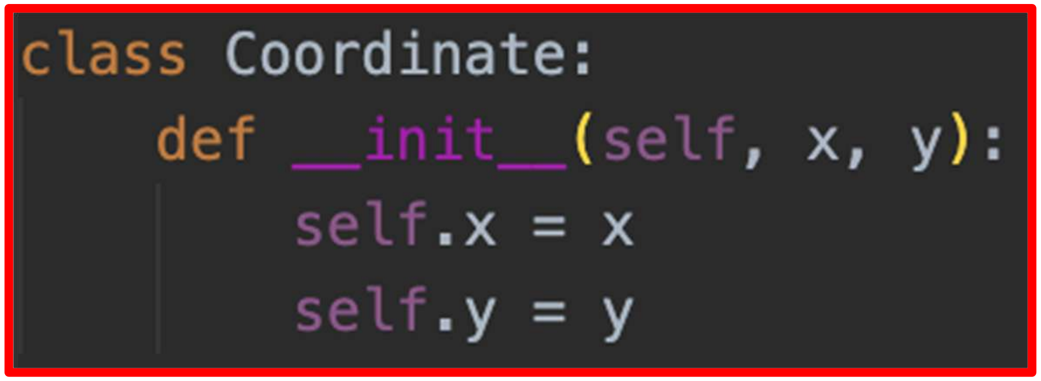
Intro to Coding

Class 9

Review: Initialized Classes

Coordinate Class:

- `self.x` represents the x (horizontal) position
- `self.y` represents the y (vertical) position

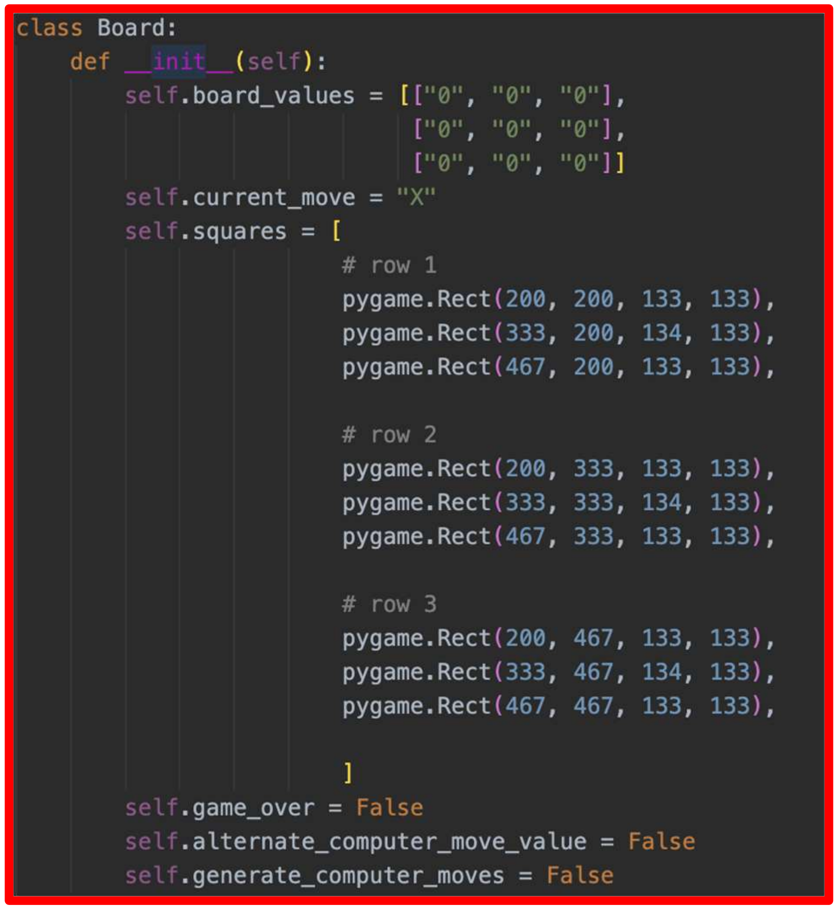


```python
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Review: Initialized Classes (2)

Board Class:

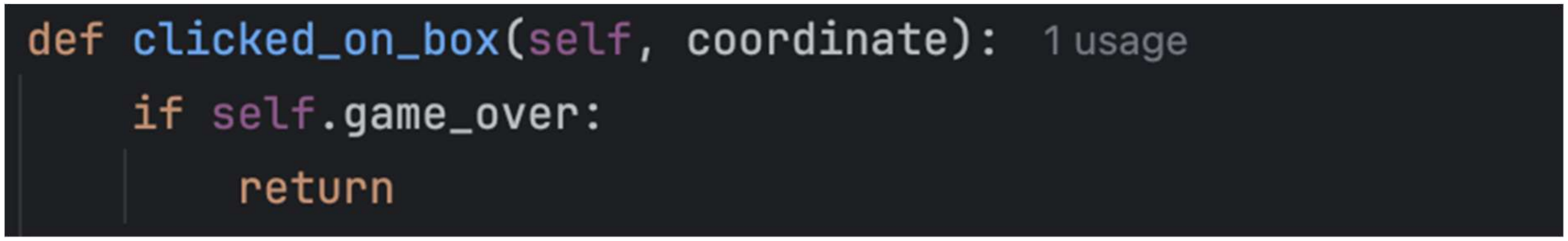
- Represents overall game board
- `self.board_values` - shows the current game board (`0` = blank square, `O` = occupied by O, `X` = occupied by X)
- `self.current_move` - shows whose turn it is to place next (starts with X)
- `self.squares` - list representing the “hitboxes” all 9 squares where you can place an O/X
- `self.game_over` - true or false value, represents if the game is continuing or finished



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

                        # 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),

                        ]
        self.game_over = False
        self.alternate_computer_move_value = False
        self.generate_computer_moves = False
```

Checking for Clicks

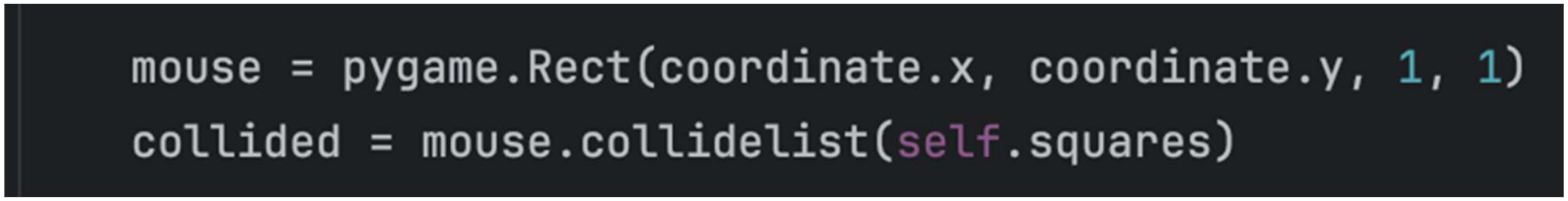
We must create a function that handles what happens when the user clicks on the game board.



```
def clicked_on_box(self, coordinate):  1 usage
    if self.game_over:
        return
```

- First we must define the function using an appropriate name and all necessary parameters
  - The `coordinate` parameter is the `(x, y)` position of the mouse click
- If the game is already over (`self.game_over == True`), we simply ignore any clicks

Checking for Clicks

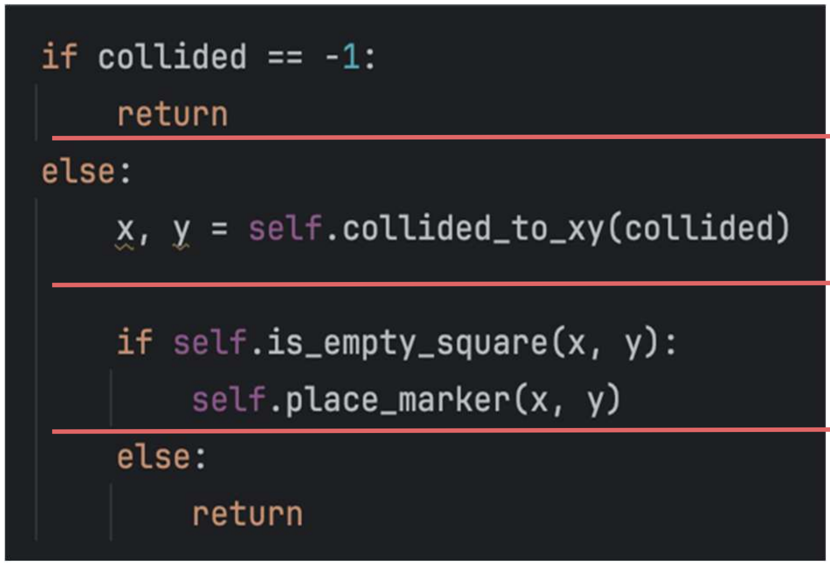


```
mouse = pygame.Rect(coordinate.x, coordinate.y, 1, 1)
collided = mouse.collidelist(self.squares)
```

- We then create a tiny 1x1 rectangle at the mouse click position. This allows us to check if the click intersects with any of our squares.
- `collidelist()` returns the index of the square clicked, or -1 if no square was clicked

- Recall detecting collisions with a mouse click...
- We **cannot** use the `collidepoint()` method because that checks overlap for ONE rectangle, while we want to check a list of many!

Section 1: Checking for Clicks

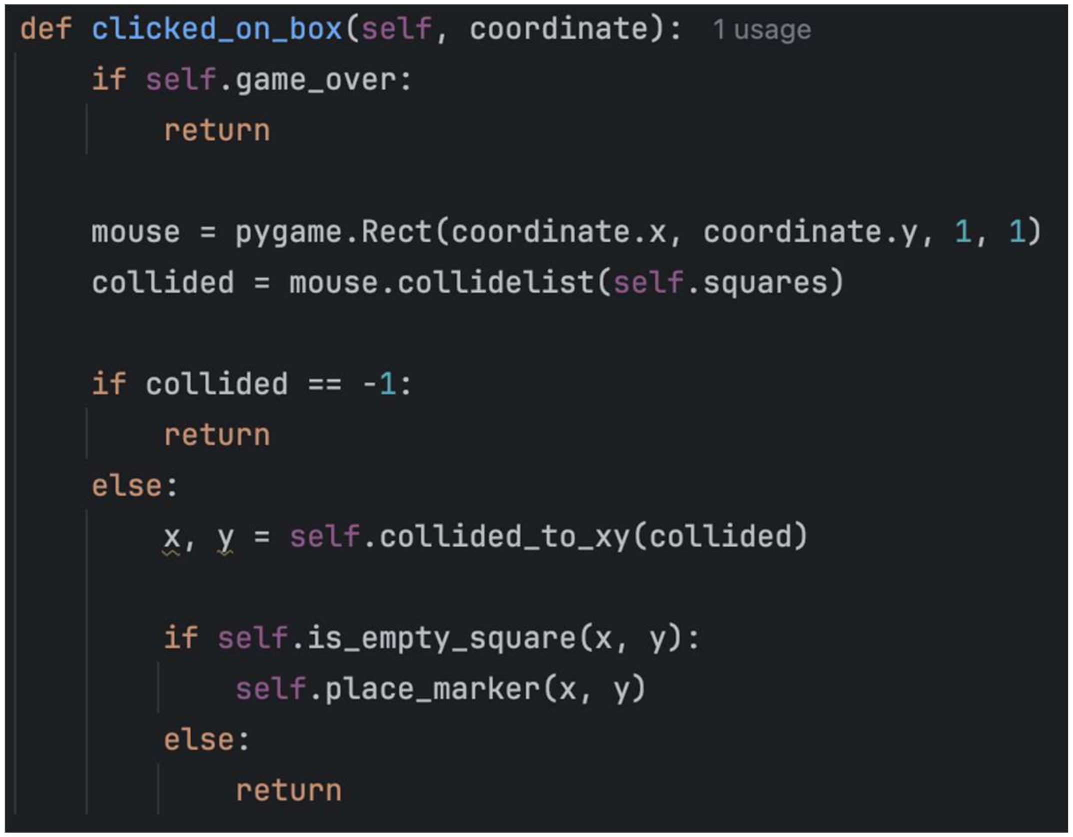


```
if collided == -1:
    return
else:
    x, y = self.collided_to_xy(collided)

    if self.is_empty_square(x, y):
        self.place_marker(x, y)
    else:
        return
```

- If the click didn't hit any square, do nothing
- Otherwise, convert the square index to board coordinates (row x, column y)
- If the square is empty, place the player's marker (X or O) in that square
- If the square is already occupied, do nothing

Section 1: Checking for Clicks

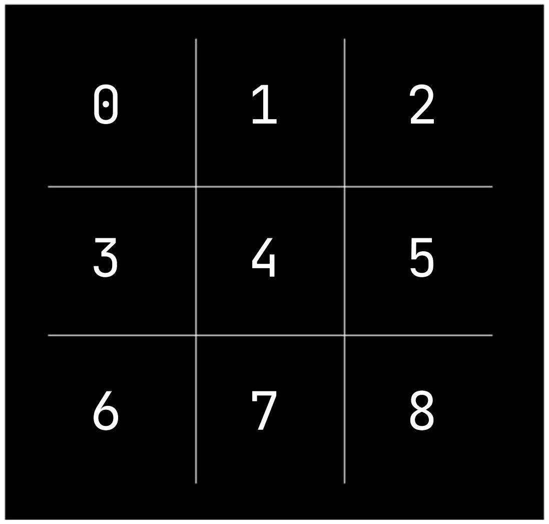
Now we put it all together!



```
def clicked_on_box(self, coordinate):  1 usage
    if self.game_over:
        return

    mouse = pygame.Rect(coordinate.x, coordinate.y, 1, 1)
    collided = mouse.collidelist(self.squares)

    if collided == -1:
        return
    else:
        x, y = self.collided_to_xy(collided)

        if self.is_empty_square(x, y):
            self.place_marker(x, y)
        else:
            return
```

Finding the Row and Column of a Clicked Box

This function finds the `X` and `Y` indices of a collided box. Here is a table showing the index of the rectangle in `self.squares`, and its corresponding `X` and `Y` indices.



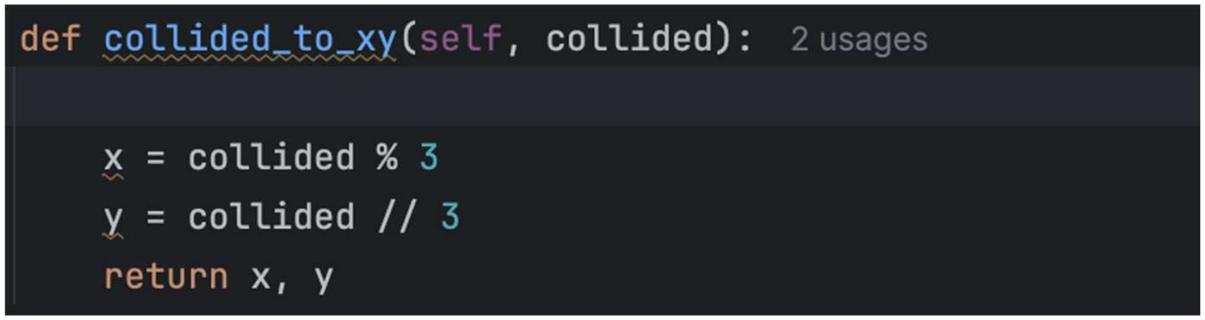
0 (X:0, Y:0)	1 (X:1, Y:0)	$\boxed{2}$ (X:2, Y:0)
3 (X:0, Y:1)	4 (X:1, Y:1)	5 (X:2, Y:1)
6 (X:0, Y:2)	7 (X:1, Y:2)	8 (X:2, Y:2)

Corresponding index in `self.squares`

Finding the Row and Column of a Clicked Box

Integer division by 3 yields the X-index, and taking the remainder when divided by 3 yields the Y-index.

0 (X:0, Y:0)	1 (X:1, Y:0)	2 (X:2, Y:0)
3 (X:0, Y:1)	4 (X:1, Y:1)	5 (X:2, Y:1)
6 (X:0, Y:2)	7 (X:1, Y:2)	8 (X:2, Y:2)

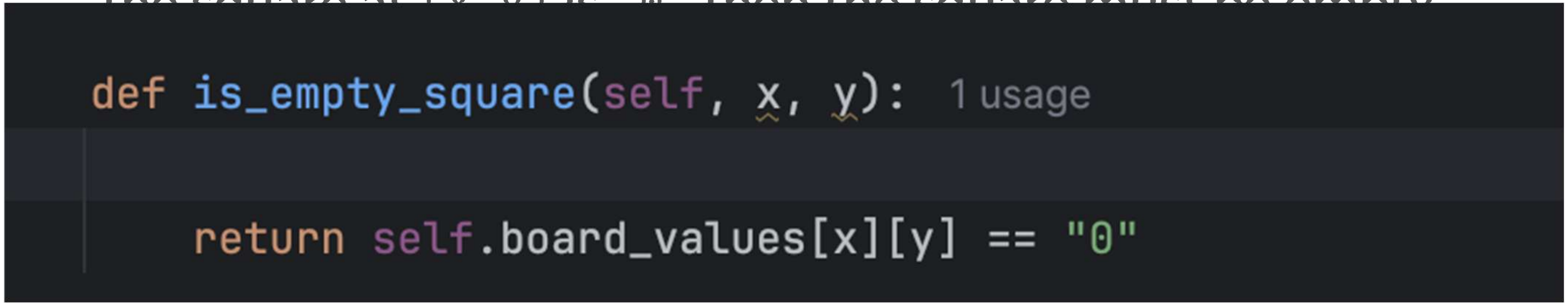


```
def collided_to_xy(self, collided):  2 usages

    x = collided % 3
    y = collided // 3
    return x, y
```

Checking if a Square is Empty

The following function checks if a square is empty at row `x` and column `y`

- The program will return a boolean value indicating whether or not the value of the square at `(x,y)` is “`0`”
- Recall: empty squares are denoted by “`0`”, therefore if the value of the square at `(x, y)` is “`0`” then the square must be empty

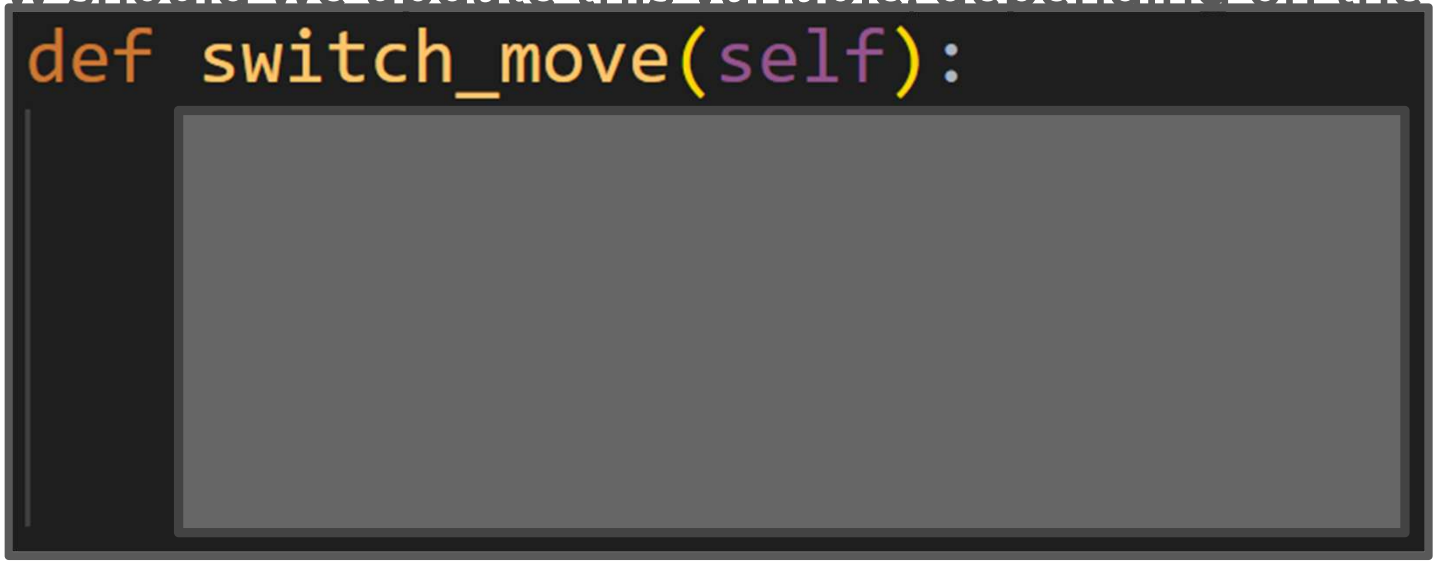


```python
def is_empty_square(self, x, y):  1 usage

    return self.board_values[x][y] == "0"
```

Switching Moves

The following function should swap the current marker (ie. X to O, O to X)

- What variable must be *updated*?
- How should we update this variable, depending on the marker?

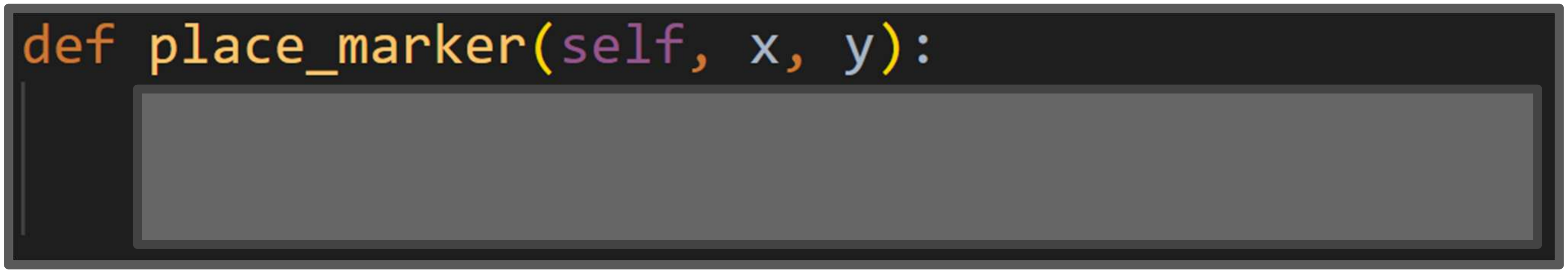


```
def switch_move(self):
```

Placing Markers

The following function should process a marker being placed at a square on row `x` and column `y`, and then preparing for the next move. `(x,y)` is 0-indexed, and `(x,y)` is guaranteed to be unoccupied.

- What object must be *updated*? How can we update it at `(x,y)`?
- What variable must be *switched*? What method can be called to do this?

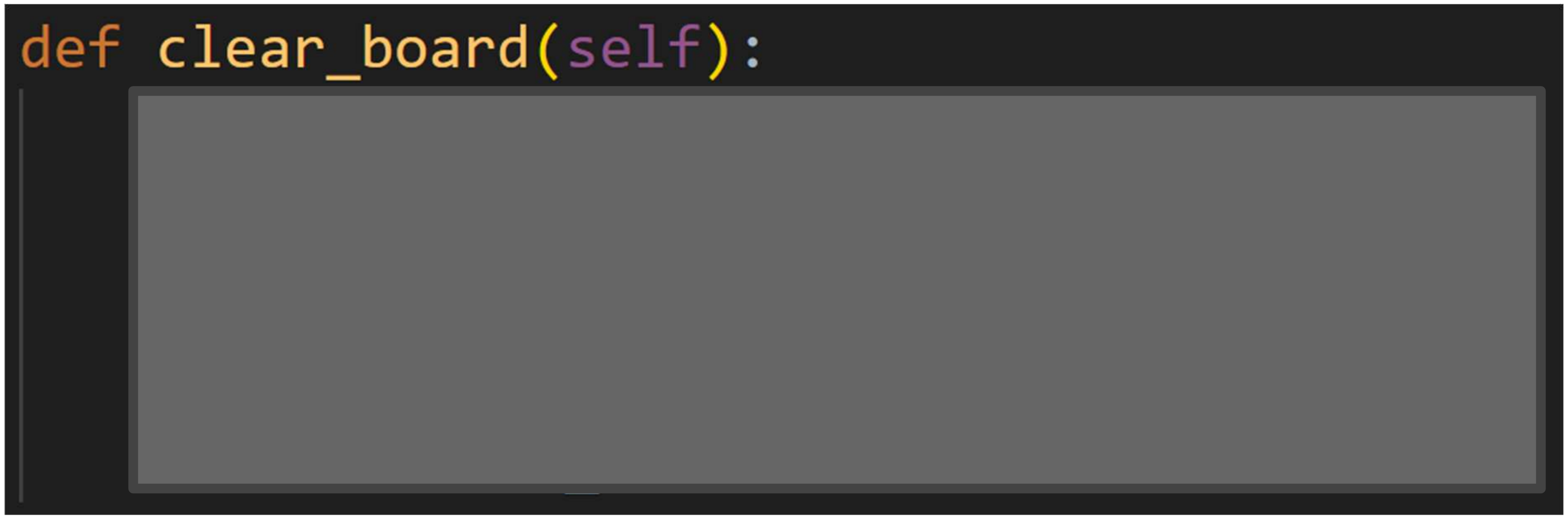


```
def place_marker(self, x, y):
```

Resetting the Game

The following function should reset the board. In other words, it needs to set up the board as brand new.

- What variables and objects must be **reset**? Where in the code have we set up these variables and objects before?



```
def clear_board(self):
```

Displaying Text + Buttons

Now, we need to render some more information on the heads-up display (HUD). This should contain:

- Whose turn it is (O or X)
- What gamemode was selected (Multiplayer or Singleplayer)
- A button to change this gamemode

Displaying Text + Buttons

Recall: we instantiate a new text element with the **render method** of a font. In this case, we can choose to use `small_font` or `font`. To save space, we will pick the former.

[redacted]

Similar logic can be applied for the other text-based elements!

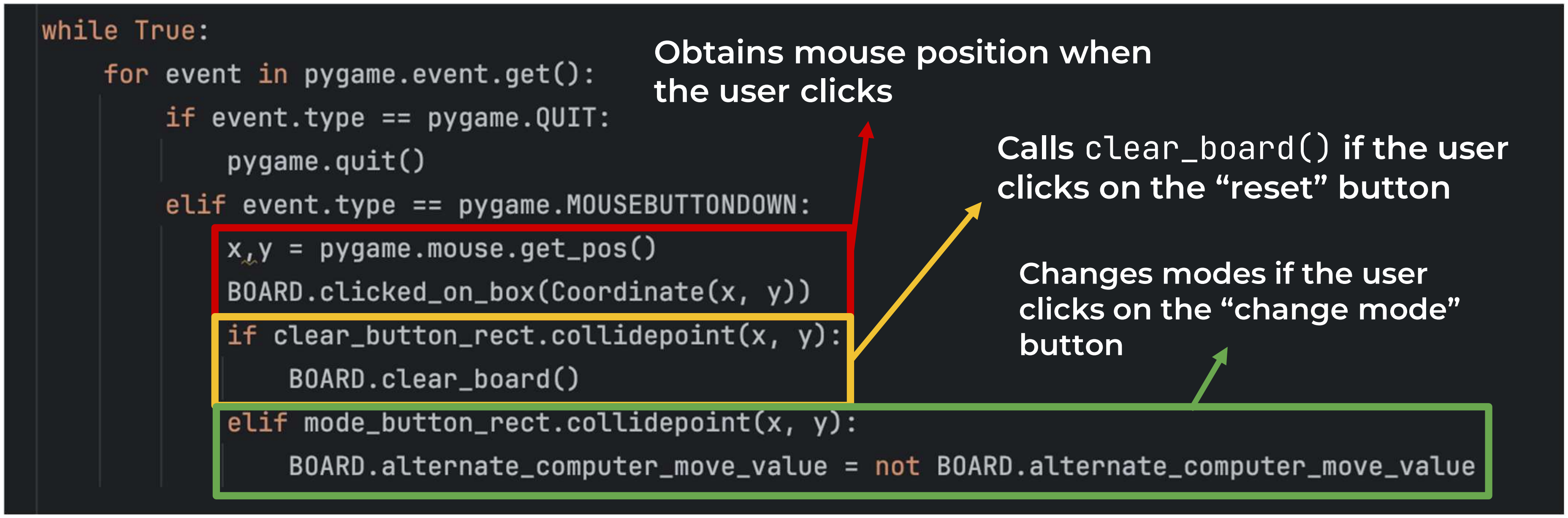
Displaying Text + Buttons

Let’s create our button. What’s the easiest way to represent a button with the Pygame elements that we have learned? How about rendering text that says “Change Mode”?

[redacted]

Handling Button Clicks

Currently, the buttons don’t do anything when clicked. We can use event handling to process mouse clicks.

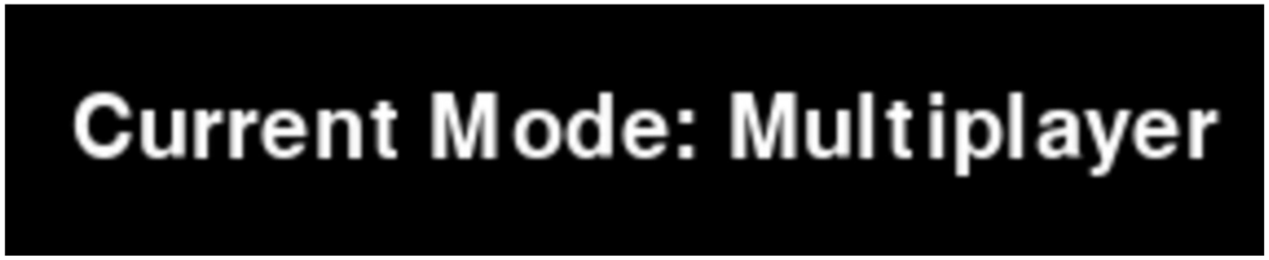


```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            x,y = pygame.mouse.get_pos()
            BOARD.clicked_on_box(Coordinate(x, y))
            if clear_button_rect.collidepoint(x, y):
                BOARD.clear_board()
            elif mode_button_rect.collidepoint(x, y):
                BOARD.alternate_computer_move_value = not BOARD.alternate_computer_move_value
```

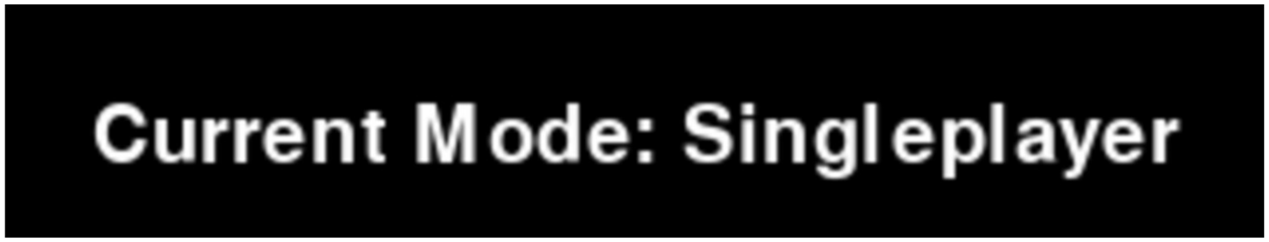
Obtains mouse position when the user clicks
Calls clear_board() if the user clicks on the “reset” button
Changes modes if the user clicks on the “change mode” button

Displaying Text + Buttons

Now, we want to display the “Current Mode”. Hint: we can follow similar logic to before, except now with **dynamic text** with the help of f-strings!



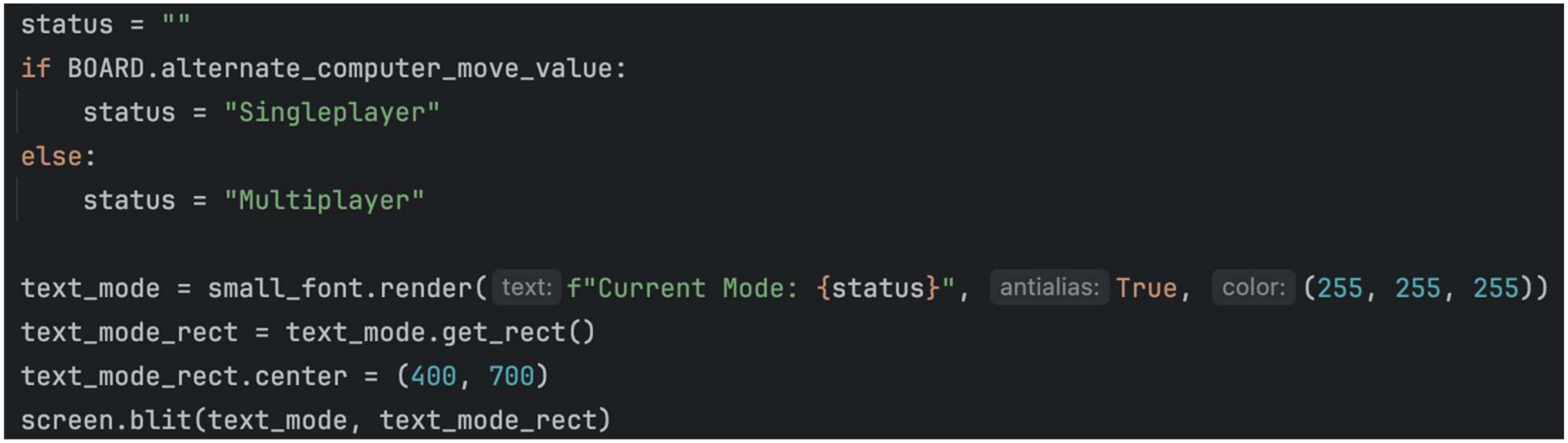
```
Current Mode: Multiplayer
```



```
Current Mode: Singleplayer
```

Displaying Text + Buttons

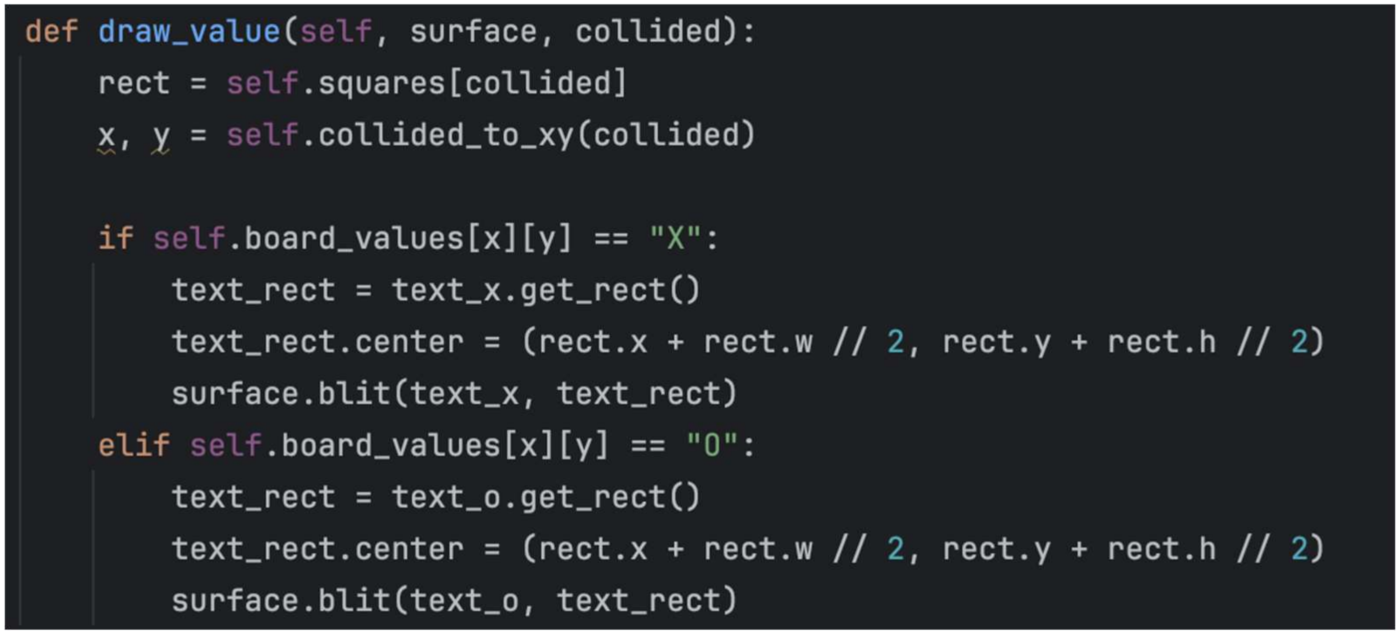
Once again, it follows the same logic, except this time with f-strings and a different position.



```
status = ""
if BOARD.alternate_computer_move_value:
    status = "Singleplayer"
else:
    status = "Multiplayer"

text_mode = small_font.render( text: f"Current Mode: {status}",  antialias: True,  color: (255, 255, 255))
text_mode_rect = text_mode.get_rect()
text_mode_rect.center = (400, 700)
screen.blit(text_mode, text_mode_rect)
```

Visually Updating the Board

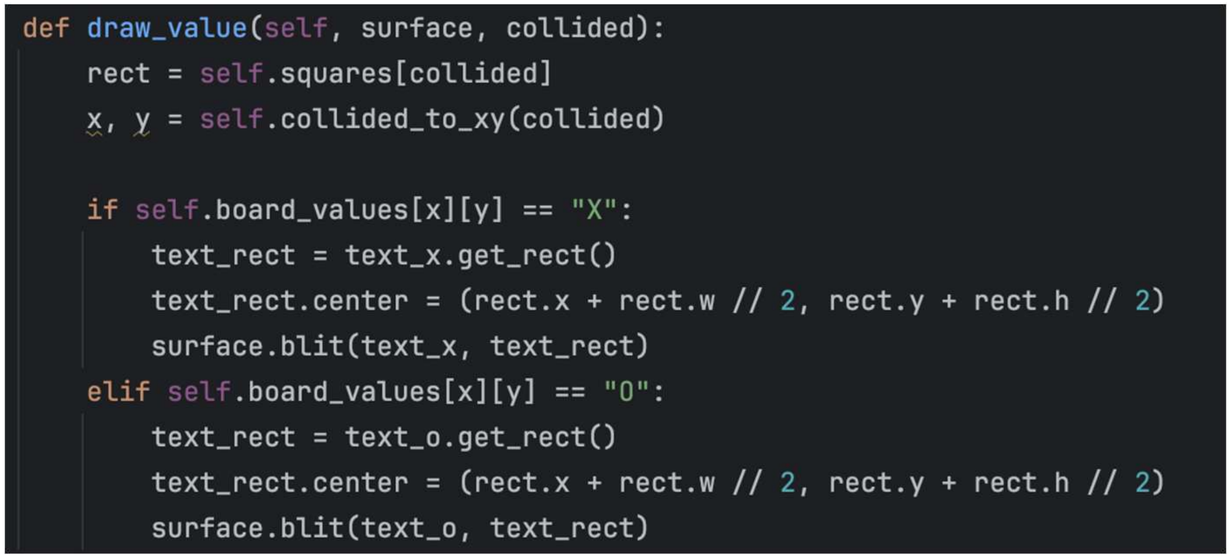
So far, the code has only updated the internal board data after a move has been made. We can write a function to draw the marker at position `(x,y)` on the board.



```python
def draw_value(self, surface, collided):
    rect = self.squares[collided]
    x, y = self.collided_to_xy(collided)

    if self.board_values[x][y] == "X":
        text_rect = text_x.get_rect()
        text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
        surface.blit(text_x, text_rect)
    elif self.board_values[x][y] == "O":
        text_rect = text_o.get_rect()
        text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
        surface.blit(text_o, text_rect)
```

Visually Updating the Board

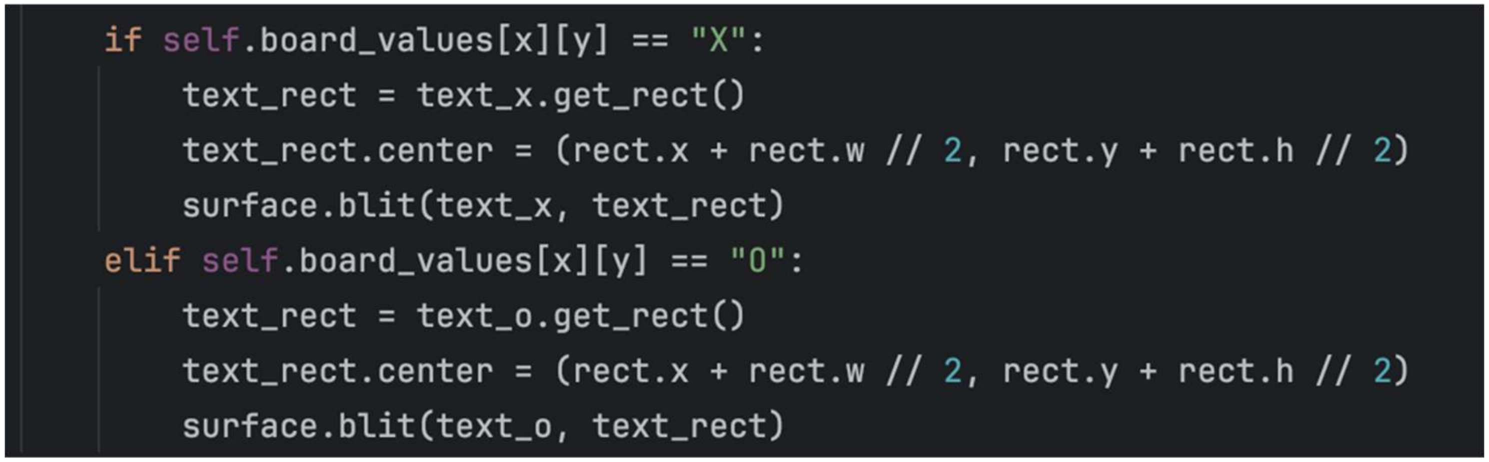
- `collided` - refers to the index of the square that is being drawn
- `rect` - the rect object representing the square being drawn
- `text_x` and `text_o` - initialized at the very top of your code; font objects representing X or O



```python
def draw_value(self, surface, collided):
    rect = self.squares[collided]
    x, y = self.collided_to_xy(collided)

    if self.board_values[x][y] == "X":
        text_rect = text_x.get_rect()
        text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
        surface.blit(text_x, text_rect)
    elif self.board_values[x][y] == "O":
        text_rect = text_o.get_rect()
        text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
        surface.blit(text_o, text_rect)
```

Visually Updating the Board

- `rect.w` and `rect.h` represent the width and the height of the rectangle, respectively
- We add `rect.w // 2` and `rect.h // 2` to the `x` and `y` coordinates of the rectangle (ie. coordinates at the top-left corner) to find the rectangle's center, and we center our text there



```
if self.board_values[x][y] == "X":
    text_rect = text_x.get_rect()
    text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
    surface.blit(text_x, text_rect)
elif self.board_values[x][y] == "O":
    text_rect = text_o.get_rect()
    text_rect.center = (rect.x + rect.w // 2, rect.y + rect.h // 2)
    surface.blit(text_o, text_rect)
```