```python
# pylint: disable=no-member, unused-wildcard-import
from typing import List, Literal, Optional, Tuple
import pygame

# Window size
WIDTH = 800
HEIGHT = 900

# Coordinates and sizes of the shapes on the top of the board
INFO_SHAPE_ARGS = ((WIDTH // 2) + 10, 30, 50, 7)

# Reset button dimensions
RESET_BUTTON_WIDTH = 100
RESET_BUTTON_HEIGHT = 50

# Typing shortcuts
Mark = Literal['X', 'O']

# Initialize pygame
pygame.init()
pygame.font.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Inception Tic Tac Toe")

# Colors
class Color:
    WHITE = (255, 255, 255)
    DARK_GRAY = (50, 50, 50)
    LIGHT_GRAY = (224, 224, 224)
    BLACK = (0, 0, 0)
    RED = (255, 0, 0)
    BLUE = (0, 0, 255)
    HIGHLIGHT = (255, 252, 179)

# Fonts
class Font:
    CURRENT_TURN = pygame.font.SysFont('Book Antiqua', 60)
    WINNER = pygame.font.SysFont('Britannic', 60)
    RESET_BUTTON = pygame.font.SysFont('Arial', 40)


# Shapes
class DrawShape:
    @staticmethod
    def X(x: int, y: int, size: int, line_width: int, color: Tuple[int, int, int] =
Color.BLUE) -> None:
        # Draw a blue line from the top left to the bottom right
        pygame.draw.line(screen, color, (x, y), (x + size, y + size), line_width)

        # Draw a blue line from the top right to the bottom left
        pygame.draw.line(screen, color, (x, y + size), (x + size, y), line_width)

    @staticmethod
    def O(x: int, y: int, radius: int, line_thickness: int, color: Tuple[int, int, int] =
Color.RED) -> None:
        # Get the center of the box from the coordinates and size
        circle_center = (
            x + (radius // 2),
            y + (radius // 2)
        )

        # Draw outer red circle
        pygame.draw.circle(
            screen,
            color,
            circle_center,
            radius // 2,
            line_thickness
        )

def check_winner(board: List[List[Optional[Mark]]], mark: Mark) -> bool:
    """
    Checks if the given mark has won in a Tic Tac Toe game from a 3x3 matrix. Returns True if the mark has won.
    """
    # Check rows
    for row in board:
        if mark == row[0] == row[1] == row[2]:
            return True

    # Check columns
    for col in range(3):
        if mark == board[0][col] == board[1][col] == board[2][col]:
            return True

    # Check diagonals
    if mark == board[0][0] == board[1][1] == board[2][2]:
        return True
    elif mark == board[0][2] == board[1][1] == board[2][0]:
        return True

    # No winner
    return False

class TicTacToeBoard:
    """
    Tic Tac Toe Board
    """

    BOARD_SIZE = 200
    BOARD_LINE_WIDTH = 10

    def __init__(self) -> None:
        # None = empty, X = X, O = O
        self.board: List[List[TickBox]] = []

        # Repeat 3 times
        for _ in range(3):
            # Append a new row
            self.board.append([])
            # Repeat 3 times
            for _ in range(3):
                # Append a new box at the end of the last row
                self.board[-1].append(TickBox(self))

        # This is set to the winner if there is one
        self.winning_mark: Optional[Mark] = None

    def draw(self, x: int, y: int) -> None:
        """
        Draws the board
        """

        # Draws a large black square to fill in with white squares
        pygame.draw.rect(screen, Color.DARK_GRAY, (x, y, self.BOARD_SIZE - self.BOARD_LINE_WIDTH, self.BOARD_SIZE - self.BOARD_LINE_WIDTH))

        # Loop through all 9 squares
        for boxX in range(3):
            for boxY in range(3):
                # Draw a TickBox
                self.board[boxY][boxX].draw(x + ((self.BOARD_SIZE / 3) * boxX), y + ((self.BOARD_SIZE / 3) * boxY))

        # If there is a winner, draw the winning mark over the whole board on top of a transparent background
        if self.winning_mark:
            # Draw a transparent background
            transparent_bg = pygame.Surface((self.BOARD_SIZE, self.BOARD_SIZE))
            transparent_bg.set_alpha(200)
            transparent_bg.fill(Color.WHITE)
            screen.blit(transparent_bg, (x, y))

            # Draw the winning mark over the board
            if self.winning_mark == 'X':
                DrawShape.X(x, y, self.BOARD_SIZE, self.BOARD_LINE_WIDTH, Color.BLUE)
            elif self.winning_mark == 'O':
                DrawShape.O(x, y, self.BOARD_SIZE, 15, Color.RED)

    def set_mark(self, x: int, y: int, mark: Optional[Mark]) -> None:
        """
        Sets a box to a mark
        """
        self.board[y][x].mark = mark

    def get_box(self, x: int, y: int) -> 'TickBox':
        """
        Gets the TickBox at a specific coordinate
        """
        return self.board[y][x]

    def reset(self) -> None:
        """
        Resets the board
        """
        for row in self.board:
            for box in row:
                box.mark = None

    def check_winner(self, mark: Mark) -> bool:
        """
        Checks if the given mark is the winner. Returns True if there is one and sets
```

```python
winning_mark to the corresponding mark.
        """
        # Transform the board into a 3x3 matrix with only the marks
        mark_matrix: List[List[Optional[Mark]]] = []
        for row in self.board:
            mark_matrix.append([])
            for box in row:
                mark_matrix[-1].append(box.mark)

        # Check if the mark has won
        is_winner = check_winner(mark_matrix, mark)

        # If there is a winner, set the winning mark
        if is_winner:
            self.winning_mark = mark

        return is_winner

    def is_full(self) -> bool:
        """
        Checks if the board is full
        """
        for row in self.board:
            for box in row:
                if box.mark is None:
                    return False
        return True

class TickBox:
    SIZE = (TicTacToeBoard.BOARD_SIZE - (TicTacToeBoard.BOARD_LINE_WIDTH * 2)) // 3

    XO_WIDTH = 7
    XO_MARGIN = 10

    def __init__(self, board: TicTacToeBoard) -> None:
        # Rectangle of the box
        self.rect: Optional[pygame.Rect] = None

        # Mark of the box
        self.mark: Optional[Mark] = None

        # The board this box is in
        self.board: TicTacToeBoard = board

        # Whether or not the box is highlighted
        self.highlighted: bool = False

    def draw(self, x: int, y: int) -> None:
        self.rect = pygame.Rect(x, y, self.SIZE, self.SIZE)
        pygame.draw.rect(screen, Color.WHITE, self.rect)
        self.draw_mark(x, y)

    def draw_mark(self, x: int, y: int) -> None:
        # If highlighted, draw a highlight background
        if self.highlighted:
            pygame.draw.rect(screen, Color.HIGHLIGHT, self.rect)

        # Draw the X or O
        if self.mark == 'X':
            DrawShape.X(
                x + self.XO_MARGIN,
                y + self.XO_MARGIN,
                self.SIZE - (self.XO_MARGIN * 2),
                self.XO_WIDTH
            )

        elif self.mark == 'O':
            DrawShape.O(
                x + (self.XO_MARGIN // 2),
                y + (self.XO_MARGIN // 2),
                self.SIZE - self.XO_MARGIN,
                self.XO_WIDTH
            )

    def is_hovered_over(self, mouse_pos: Tuple[int, int]) -> bool:
        if not self.rect:
            return False

        # Return true if the mouse is over the box
        if self.rect.collidepoint(mouse_pos):
            return True
        return False

class InceptionBoard:
    """
    Inception Board
    """

    BOARD_SIZE = 750
    BOARD_LINE_WIDTH = 10
    BOARD_BOX_SIZE = (BOARD_SIZE - (BOARD_LINE_WIDTH * 2)) // 3


    def __init__(self) -> None:
        # None = empty, X = X, O = O
        self.board: List[List[TicTacToeBoard]] = []

        # Repeat 3 times
        for _ in range(3):
            # Add a list to the board
            self.board.append([])
            # Repeat 3 times
            for _ in range(3):
                # Add a TicTacToeBoard to the last list in the board
                self.board[-1].append(TicTacToeBoard())

    def get_board(self, x: int, y: int) -> TicTacToeBoard:
        """
        Gets the TicTacToeBoard at a specific coordinate
        """
        return self.board[y][x]

    def draw(self, x: int, y: int) -> None:
        """
        Draws the board
        """

        # Draws a large black square to fill in with tic tac toe boards
        pygame.draw.rect(screen, Color.BLACK, (x, y, self.BOARD_SIZE - self.BOARD_LINE_WIDTH,
self.BOARD_SIZE - self.BOARD_LINE_WIDTH))

        # Loop through all 9 squares
        for boxX in range(3):
            for boxY in range(3):

                # Draw a smaller white square in the correct position for the background
                pygame.draw.rect(
                    screen,
                    Color.WHITE,
                    (
                        x + ((self.BOARD_SIZE / 3) * boxX),
                        y + ((self.BOARD_SIZE / 3) * boxY),
                        self.BOARD_BOX_SIZE,
                        self.BOARD_BOX_SIZE
                    )
                )

                # Draw a tic tac toe board in the center of the box
                self.board[boxY][boxX].draw(
                    x + ((self.BOARD_SIZE / 3) * boxX) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2,
                    y + ((self.BOARD_SIZE / 3) * boxY) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2
                )

    def get_hovering_tickbox(self) -> Optional[TickBox]:
        """
        Gets the TickBox that the mouse is hovering over
        """
        mouse_pos = pygame.mouse.get_pos()

        # Loop through all 9 boards
        for miniboardX in range(3):
            for miniBoardY in range(3):

                # Get the board
                miniboard = self.get_board(miniboardX, miniBoardY)

                # Loop through all 9 boxes
                for tickboxX in range(3):
                    for tickboxY in range(3):

                        # Get the tickbox
                        tickbox = miniboard.get_box(tickboxX, tickboxY)

                        # If the mouse is over the tickbox, return it
                        if tickbox.is_hovered_over(mouse_pos):
                            return tickbox

    def check_winner(self, mark: Mark) -> bool:
        """
        Checks if the given mark is the winner. Returns True if there is one.
        """
        # Transform the board into a 3x3 matrix with only the marks
        mark_matrix: List[List[Optional[Mark]]] = []
        for row in self.board:
            mark_matrix.append([])
            for board in row:
                mark_matrix[-1].append(board.winning_mark)

        # Check if the mark has won
        is_winner = check_winner(mark_matrix, mark)

        # If there is a winner, set the winning mark
        if is_winner:
            self.winning_mark = mark

        return is_winner

    def is_full(self) -> bool:
        """
        Checks if every mini board has a winner
        """
        for row in self.board:
            for miniboard in row:
                if not miniboard.winning_mark:
                    return False
        return True

# True while the game is running
running: bool = True

# The main board
board: InceptionBoard = InceptionBoard()

# Which player's turn it is
turn: Mark = 'X'

# The tickbox that the mouse is currently holding left click on
clicked_tickbox: Optional[TickBox] = None

# The tickbox that the mouse was hovering over last frame
last_hovering_tickbox: Optional[TickBox] = None

# The tickbox that the mouse is currently hovering over
hovering_tickbox: Optional[TickBox] = None

# None = No winner yet, X = X wins, O = O wins, T = Tie
winner: Literal['X', 'O', 'T', None] = False

# Reset button rectangle
reset_button_rect: pygame.Rect = pygame.Rect(
    (WIDTH // 2) - (RESET_BUTTON_WIDTH // 2),
    100,
    RESET_BUTTON_WIDTH,
    RESET_BUTTON_HEIGHT
)

while running:
    last_hovering_tickbox = hovering_tickbox
    hovering_tickbox = board.get_hovering_tickbox()

    # If the mouse is hovering over an active tickbox, highlight it and set cursor to hand
    if hovering_tickbox and not hovering_tickbox.board.winning_mark and not winner:
        pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_HAND)
        hovering_tickbox.highlighted = True

    # If the mouse has moved off of a tickbox, unhighlight it and set the cursor to default
    if last_hovering_tickbox and hovering_tickbox is not last_hovering_tickbox:
        pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_ARROW)
        last_hovering_tickbox.highlighted = False

    for event in pygame.event.get(): # Gets all the events which have occured until now
        # Listens for the the X button at the top right
        if event.type == pygame.QUIT:
            running = False
        # Listens for mouse left-click down
        elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            # If the mouse is hovering over a tickbox, set it as the clicked tickbox and wait for the mouse to be released on it
            if hovering_tickbox:
                clicked_tickbox = hovering_tickbox

            if winner and reset_button_rect.collidepoint(pygame.mouse.get_pos()):
                board = InceptionBoard()
                turn = 'X'
                winner = None

        # Listens for mouse left-click release
        elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:
            # If the mouse release is on the same tickbox as the mouse click
            if clicked_tickbox and clicked_tickbox is hovering_tickbox:

                # Get the TicTacToeBoard that the clicked tickbox is in
                miniboard = clicked_tickbox.board

                # If the box is empty and the board isn't over, fill it with the current player's mark
                if not clicked_tickbox.mark and not miniboard.check_winner(turn) and not winner:
                    clicked_tickbox.mark = turn

                    # If there is a tie, reset the board
                    if not miniboard.check_winner(turn) and miniboard.is_full():
                        miniboard.reset()

                    # Check if there is a winner
                    is_winner = board.check_winner(turn)
                    if not is_winner and board.is_full():
                        winner = 'T'
                    elif is_winner:
                        winner = turn

                    turn = 'X' if turn == 'O' else 'O'

                # Reset the clicked tickbox
                clicked_tickbox = None

    # Clear the screen
    screen.fill(Color.WHITE)

    # Draw information at the top of the board
    # If there is no winner, draw the current player's turn
    if not winner:
        # Draw the text
        text_surface: pygame.Surface = Font.CURRENT_TURN.render('Turn:', True, Color.BLACK)
        screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))

        # Draw the current player's mark
        if turn == 'X':
            DrawShape.X(*INFO_SHAPE_ARGS)
        # If it's Player O's turn, draw an O
        else:
            DrawShape.O(*INFO_SHAPE_ARGS)
    # If there is a winner, draw the winner and a button to reset the board
    else:
        # If it's not a tie, draw the winner
        if winner != 'T':
            # Draw the text
            text_surface: pygame.Surface = Font.WINNER.render('Winner!', True, Color.BLACK)
            screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))

            # Draw the winner's mark
            if winner == 'X':
                DrawShape.X(*INFO_SHAPE_ARGS)
            # If it's Player O is the winner, draw an O
            else:
                DrawShape.O(*INFO_SHAPE_ARGS)

        # If it's a tie, draw the Tie text
        else:
            # Draw the text
            text_surface: pygame.Surface = Font.WINNER.render('Tie!', True, Color.BLACK)
            screen.blit(text_surface, ((WIDTH // 2) - (text_surface.get_width() // 2), 20))

        # Draw the reset button
        pygame.draw.rect(screen, Color.LIGHT_GRAY, reset_button_rect)
        reset_text_surface: pygame.Surface = Font.RESET_BUTTON.render('Reset', True,
Color.BLACK)
        screen.blit(reset_text_surface, ((WIDTH // 2) - (reset_text_surface.get_width() // 2),
100))


    # Draw the board in the horizontal center and bottom of the screen
    board.draw((WIDTH // 2) - (board.BOARD_SIZE // 2), HEIGHT - board.BOARD_SIZE)

    # Update the screen
    pygame.display.flip()

pygame.quit()
```