

```

1 # pylint: disable=no-member, unused-wildcard-import
2 from typing import List, Literal, Optional, Tuple
3 import pygame
4
5 # Window size
6 WIDTH = 800
7 HEIGHT = 900
8
9 # Coordinates and sizes of the shapes on the top of the board
10 INFO_SHAPE_ARGS = ((WIDTH // 2) + 10, 30, 50, 7)
11
12 # Reset button dimensions
13 RESET_BUTTON_WIDTH = 100
14 RESET_BUTTON_HEIGHT = 50
15
16 # Typing shortcuts
17 Mark = Literal['X', 'O']
18
19 # Initialize pygame
20 pygame.init()
21 pygame.font.init()
22 screen = pygame.display.set_mode((WIDTH, HEIGHT))
23 pygame.display.set_caption("Inception Tic Tac Toe")
24
25 # Colors
26 class Color:
27     WHITE = (255, 255, 255)
28     DARK_GRAY = (50, 50, 50)
29     LIGHT_GRAY = (224, 224, 224)
30     BLACK = (0, 0, 0)
31     RED = (255, 0, 0)
32     BLUE = (0, 0, 255)
33     HIGHLIGHT = (255, 252, 179)
34
35 # Fonts
36 class Font:
37     CURRENT_TURN = pygame.font.SysFont('Book Antiqua', 60)
38     WINNER = pygame.font.SysFont('Britannic', 60)
39     RESET_BUTTON = pygame.font.SysFont('Arial', 40)
40
41
42 # Shapes
43 class DrawShape:
44     @staticmethod
45     def X(x: int, y: int, size: int, line_width: int, color: Tuple[int, int, int] =
Color.BLUE) -> None:
46         # Draw a blue line from the top left to the bottom right
47         pygame.draw.line(screen, color, (x, y), (x + size, y + size), line_width)
48
49         # Draw a blue line from the top right to the bottom left
50         pygame.draw.line(screen, color, (x, y + size), (x + size, y), line_width)
51
52     @staticmethod
53     def O(x: int, y: int, radius: int, line_thickness: int, color: Tuple[int, int, int] =
Color.RED) -> None:
54         # Get the center of the box from the coordinates and size
55         circle_center = (

```

```

56         x + (radius // 2),
57         y + (radius // 2)
58     )
59
60     # Draw outer red circle
61     pygame.draw.circle(
62         screen,
63         color,
64         circle_center,
65         radius // 2,
66         line_thickness
67     )
68
69 def check_winner(board: List[List[Optional[Mark]]], mark: Mark) -> bool:
70     """
71     Checks if the given mark has won in a Tic Tac Toe game from a 3x3 matrix. Returns True if
72     the mark has won.
73     """
74     # Check rows
75     for row in board:
76         if mark == row[0] == row[1] == row[2]:
77             return True
78
79     # Check columns
80     for col in range(3):
81         if mark == board[0][col] == board[1][col] == board[2][col]:
82             return True
83
84     # Check diagonals
85     if mark == board[0][0] == board[1][1] == board[2][2]:
86         return True
87     elif mark == board[0][2] == board[1][1] == board[2][0]:
88         return True
89
90     # No winner
91     return False
92
93 class TicTacToeBoard:
94     """
95     Tic Tac Toe Board
96     """
97
98     BOARD_SIZE = 200
99     BOARD_LINE_WIDTH = 10
100
101     def __init__(self) -> None:
102         # None = empty, X = X, O = O
103         self.board: List[List[Optional[Mark]]] = []
104
105         # Repeat 3 times
106         for _ in range(3):
107             # Append a new row
108             self.board.append([])
109             # Repeat 3 times
110             for _ in range(3):
111                 # Append a new box at the end of the last row
112                 self.board[-1].append(TickBox(self))

```

```

112
113     # This is set to the winner if there is one
114     self.winning_mark: Optional[Mark] = None
115
116     def draw(self, x: int, y: int) -> None:
117         """
118         Draws the board
119         """
120
121         # Draws a large black square to fill in with white squares
122         pygame.draw.rect(screen, Color.DARK_GRAY, (x, y, self.BOARD_SIZE -
self.BOARD_LINE_WIDTH, self.BOARD_SIZE - self.BOARD_LINE_WIDTH))
123
124         # Loop through all 9 squares
125         for boxX in range(3):
126             for boxY in range(3):
127                 # Draw a TickBox
128                 self.board[boxY][boxX].draw(x + ((self.BOARD_SIZE / 3) * boxX), y +
((self.BOARD_SIZE / 3) * boxY))
129
130         # If there is a winner, draw the winning mark over the whole board on top of a
transparent background
131         if self.winning_mark:
132             # Draw a transparent background
133             transparent_bg = pygame.Surface((self.BOARD_SIZE, self.BOARD_SIZE))
134             transparent_bg.set_alpha(200)
135             transparent_bg.fill(Color.WHITE)
136             screen.blit(transparent_bg, (x, y))
137
138             # Draw the winning mark over the board
139             if self.winning_mark == 'X':
140                 DrawShape.X(x, y, self.BOARD_SIZE, self.BOARD_LINE_WIDTH, Color.BLUE)
141             elif self.winning_mark == 'O':
142                 DrawShape.O(x, y, self.BOARD_SIZE, 15, Color.RED)
143
144     def set_mark(self, x: int, y: int, mark: Optional[Mark]) -> None:
145         """
146         Sets a box to a mark
147         """
148         self.board[y][x].mark = mark
149
150     def get_box(self, x: int, y: int) -> 'TickBox':
151         """
152         Gets the TickBox at a specific coordinate
153         """
154         return self.board[y][x]
155
156     def reset(self) -> None:
157         """
158         Resets the board
159         """
160         for row in self.board:
161             for box in row:
162                 box.mark = None
163
164     def check_winner(self, mark: Mark) -> bool:
165         """
166         Checks if the given mark is the winner. Returns True if there is one and sets

```

```

    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)

```

```

223     # Draw the X or O
224     if self.mark == 'X':
225         DrawShape.X(
226             x + self.XO_MARGIN,
227             y + self.XO_MARGIN,
228             self.SIZE - (self.XO_MARGIN * 2),
229             self.XO_WIDTH
230         )
231
232     elif self.mark == 'O':
233         DrawShape.O(
234             x + (self.XO_MARGIN // 2),
235             y + (self.XO_MARGIN // 2),
236             self.SIZE - self.XO_MARGIN,
237             self.XO_WIDTH
238         )
239
240     def is_hovered_over(self, mouse_pos: Tuple[int, int]) -> bool:
241         if not self.rect:
242             return False
243
244         # Return true if the mouse is over the box
245         if self.rect.collidepoint(mouse_pos):
246             return True
247         return False
248
249     class InceptionBoard:
250         """
251         Inception Board
252         """
253
254         BOARD_SIZE = 750
255         BOARD_LINE_WIDTH = 10
256         BOARD_BOX_SIZE = (BOARD_SIZE - (BOARD_LINE_WIDTH * 2)) // 3
257
258     def __init__(self) -> None:
259         # None = empty, X = X, O = O
260         self.board: List[List[TicTacToeBoard]] = []
261
262         # Repeat 3 times
263         for _ in range(3):
264             # Add a list to the board
265             self.board.append([])
266             # Repeat 3 times
267             for _ in range(3):
268                 # Add a TicTacToeBoard to the last list in the board
269                 self.board[-1].append(TicTacToeBoard())
270
271     def get_board(self, x: int, y: int) -> TicTacToeBoard:
272         """
273         Gets the TicTacToeBoard at a specific coordinate
274         """
275         return self.board[y][x]
276
277     def draw(self, x: int, y: int) -> None:
278         """
279

```

```

280     Draws the board
281     """
282
283     # Draws a large black square to fill in with tic tac toe boards
284     pygame.draw.rect(screen, Color.BLACK, (x, y, self.BOARD_SIZE - self.BOARD_LINE_WIDTH,
self.BOARD_SIZE - self.BOARD_LINE_WIDTH))
285
286     # Loop through all 9 squares
287     for boxX in range(3):
288         for boxY in range(3):
289
290             # Draw a smaller white square in the correct position for the background
291             pygame.draw.rect(
292                 screen,
293                 Color.WHITE,
294                 (
295                     x + ((self.BOARD_SIZE / 3) * boxX),
296                     y + ((self.BOARD_SIZE / 3) * boxY),
297                     self.BOARD_BOX_SIZE,
298                     self.BOARD_BOX_SIZE
299                 )
300             )
301
302             # Draw a tic tac toe board in the center of the box
303             self.board[boxY][boxX].draw(
304                 x + ((self.BOARD_SIZE / 3) * boxX) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2,
305                 y + ((self.BOARD_SIZE / 3) * boxY) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2
306             )
307
308     def get_hovering_tickbox(self) -> Optional[TickBox]:
309         """
310         Gets the TickBox that the mouse is hovering over
311         """
312         mouse_pos = pygame.mouse.get_pos()
313
314         # Loop through all 9 boards
315         for miniboardX in range(3):
316             for miniBoardY in range(3):
317
318                 # Get the board
319                 miniboard = self.get_board(miniboardX, miniBoardY)
320
321                 # Loop through all 9 boxes
322                 for tickboxX in range(3):
323                     for tickboxY in range(3):
324
325                         # Get the tickbox
326                         tickbox = miniboard.get_box(tickboxX, tickboxY)
327
328                         # If the mouse is over the tickbox, return it
329                         if tickbox.is_hovered_over(mouse_pos):
330                             return tickbox
331
332     def check_winner(self, mark: Mark) -> bool:
333         """
334         Checks if the given mark is the winner. Returns True if there is one.

```

```

335     """
336     # Transform the board into a 3x3 matrix with only the marks
337     mark_matrix: List[List[Optional[Mark]]] = []
338     for row in self.board:
339         mark_matrix.append([])
340         for board in row:
341             mark_matrix[-1].append(board.winning_mark)
342
343     # Check if the mark has won
344     is_winner = check_winner(mark_matrix, mark)
345
346     # If there is a winner, set the winning mark
347     if is_winner:
348         self.winning_mark = mark
349
350     return is_winner
351
352 def is_full(self) -> bool:
353     """
354     Checks if every mini board has a winner
355     """
356     for row in self.board:
357         for miniboard in row:
358             if not miniboard.winning_mark:
359                 return False
360     return True
361
362 # True while the game is running
363 running: bool = True
364
365 # The main board
366 board: InceptionBoard = InceptionBoard()
367
368 # Which player's turn it is
369 turn: Mark = 'X'
370
371 # The tickbox that the mouse is currently holding left click on
372 clicked_tickbox: Optional[TickBox] = None
373
374 # The tickbox that the mouse was hovering over last frame
375 last_hovering_tickbox: Optional[TickBox] = None
376
377 # The tickbox that the mouse is currently hovering over
378 hovering_tickbox: Optional[TickBox] = None
379
380 # None = No winner yet, X = X wins, O = O wins, T = Tie
381 winner: Literal['X', 'O', 'T', None] = False
382
383 # Reset button rectangle
384 reset_button_rect: pygame.Rect = pygame.Rect(
385     (WIDTH // 2) - (RESET_BUTTON_WIDTH // 2),
386     100,
387     RESET_BUTTON_WIDTH,
388     RESET_BUTTON_HEIGHT
389 )
390
391 while running:

```

```

392 last_hovering_tickbox = hovering_tickbox
393 hovering_tickbox = board.get_hovering_tickbox()
394
395 # If the mouse is hovering over an active tickbox, highlight it and set cursor to hand
396 if hovering_tickbox and not hovering_tickbox.board.winning_mark and not winner:
397     pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_HAND)
398     hovering_tickbox.highlighted = True
399
400 # If the mouse has moved off of a tickbox, unhighlight it and set the cursor to default
401 if last_hovering_tickbox and hovering_tickbox is not last_hovering_tickbox:
402     pygame.mouse.set_cursor(pygame.SYSTEM_CURSOR_ARROW)
403     last_hovering_tickbox.highlighted = False
404
405 for event in pygame.event.get(): # Gets all the events which have occurred until now
406     # Listens for the the X button at the top right
407     if event.type == pygame.QUIT:
408         running = False
409     # Listens for mouse left-click down
410     elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
411         # If the mouse is hovering over a tickbox, set it as the clicked tickbox and wait
412         for the mouse to be released on it
413             if hovering_tickbox:
414                 clicked_tickbox = hovering_tickbox
415
416             if winner and reset_button_rect.collidepoint(pygame.mouse.get_pos()):
417                 board = InceptionBoard()
418                 turn = 'X'
419                 winner = None
420
421     # Listens for mouse left-click release
422     elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:
423         # If the mouse release is on the same tickbox as the mouse click
424         if clicked_tickbox and clicked_tickbox is hovering_tickbox:
425
426             # Get the TicTacToeBoard that the clicked tickbox is in
427             miniboard = clicked_tickbox.board
428
429             # If the box is empty and the board isn't over, fill it with the current
430             player's mark
431             if not clicked_tickbox.mark and not miniboard.check_winner(turn) and not
432             winner:
433                 clicked_tickbox.mark = turn
434
435                 # If there is a tie, reset the board
436                 if not miniboard.check_winner(turn) and miniboard.is_full():
437                     miniboard.reset()
438
439                 # Check if there is a winner
440                 is_winner = board.check_winner(turn)
441                 if not is_winner and board.is_full():
442                     winner = 'T'
443                 elif is_winner:
444                     winner = turn
445
446                 turn = 'X' if turn == 'O' else 'O'
447
448             # Reset the clicked tickbox
449             clicked_tickbox = None

```



```

447
448 # Clear the screen
449 screen.fill(Color.WHITE)
450
451 # Draw information at the top of the board
452 # If there is no winner, draw the current player's turn
453 if not winner:
454     # Draw the text
455     text_surface: pygame.Surface = Font.CURRENT_TURN.render('Turn:', True, Color.BLACK)
456     screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
457
458     # Draw the current player's mark
459     if turn == 'X':
460         DrawShape.X(*INFO_SHAPE_ARGS)
461     # If it's Player 0's turn, draw an O
462     else:
463         DrawShape.O(*INFO_SHAPE_ARGS)
464 # If there is a winner, draw the winner and a button to reset the board
465 else:
466     # If it's not a tie, draw the winner
467     if winner != 'T':
468         # Draw the text
469         text_surface: pygame.Surface = Font.WINNER.render('Winner!', True, Color.BLACK)
470         screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
471
472         # Draw the winner's mark
473         if winner == 'X':
474             DrawShape.X(*INFO_SHAPE_ARGS)
475         # If it's Player 0 is the winner, draw an O
476         else:
477             DrawShape.O(*INFO_SHAPE_ARGS)
478
479     # If it's a tie, draw the Tie text
480     else:
481         # Draw the text
482         text_surface: pygame.Surface = Font.WINNER.render('Tie!', True, Color.BLACK)
483         screen.blit(text_surface, ((WIDTH // 2) - (text_surface.get_width() // 2), 20))
484
485     # Draw the reset button
486     pygame.draw.rect(screen, Color.LIGHT_GRAY, reset_button_rect)
487     reset_text_surface: pygame.Surface = Font.RESET_BUTTON.render('Reset', True,
Color.BLACK)
488     screen.blit(reset_text_surface, ((WIDTH // 2) - (reset_text_surface.get_width() // 2),
100))
489
490
491 # Draw the board in the horizontal center and bottom of the screen
492 board.draw((WIDTH // 2) - (board.BOARD_SIZE // 2), HEIGHT - board.BOARD_SIZE)
493
494 # Update the screen
495 pygame.display.flip()
496
497 pygame.quit()

```