

```

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', None]
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[Mark]]) -> Mark:
70     """
71     Checks if there is a winner in a Tic Tac Toe game from a 3x3 matrix. Returns the winning
72     mark if there is one.
73     """
74     # Check rows
75     for row in board:
76         if row[0] is not None and row[0] == row[1] == row[2]:
77             return row[0]
78
79     # Check columns
80     for col in range(3):
81         if board[0][col] is not None and board[0][col] == board[1][col] == board[2][col]:
82             return board[0][col]
83
84     # Check diagonals
85     if board[0][0] is not None and board[0][0] == board[1][1] == board[2][2]:
86         return board[0][0]
87     if board[0][2] is not None and board[0][2] == board[1][1] == board[2][0]:
88         return board[0][2]
89
90     # No winner
91     return None
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[Mark]] = []
104         for _ in range(3):
105             self.board.append([])
106             for _ in range(3):
107                 self.board[-1].append(TickBox(self))
108
109         # This is set to the winner if there is one
110         self.winning_mark: Mark = None
111
112     def draw(self, x: int, y: int) -> None:

```

```

112     """
113     Draws the board
114     """
115
116     # Draws a large black square to fill in with white squares
117     pygame.draw.rect(screen, Color.DARK_GRAY, (x, y, self.BOARD_SIZE -
self.BOARD_LINE_WIDTH, self.BOARD_SIZE - self.BOARD_LINE_WIDTH))
118
119     # Loop through all 9 squares
120     for boxX in range(3):
121         for boxY in range(3):
122             # Draw a TickBox
123             self.board[boxY][boxX].draw(x + ((self.BOARD_SIZE / 3) * boxX), y +
((self.BOARD_SIZE / 3) * boxY))
124
125     # If there is a winner, draw the winning mark over the whole board on top of a
transparent background
126     if self.winning_mark:
127         # Draw a transparent background
128         transparent_bg = pygame.Surface((self.BOARD_SIZE, self.BOARD_SIZE))
129         transparent_bg.set_alpha(200)
130         transparent_bg.fill(Color.WHITE)
131         screen.blit(transparent_bg, (x, y))
132
133     # Draw the winning mark over the board
134     if self.winning_mark == 'X':
135         DrawShape.X(x, y, self.BOARD_SIZE, self.BOARD_LINE_WIDTH, Color.BLUE)
136     elif self.winning_mark == 'O':
137         DrawShape.O(x, y, self.BOARD_SIZE, 15, Color.RED)
138
139     def set_mark(self, x: int, y: int, mark: Mark) -> None:
140         """
141         Sets a box to a mark
142         """
143         self.board[y][x].mark = mark
144
145     def get_box(self, x: int, y: int) -> 'TickBox':
146         """
147         Gets the TickBox at a specific coordinate
148         """
149         return self.board[y][x]
150
151     def reset(self) -> None:
152         """
153         Resets the board
154         """
155         for row in self.board:
156             for box in row:
157                 box.mark = None
158
159     def check_winner(self) -> Mark:
160         """
161         Checks if there is a winner. Returns the winner if there is one and set winning_mark
to the corresponding mark.
162         """
163         # Transform the board into a 3x3 matrix with only the marks
164         mark_matrix: List[List[Mark]] = []
165         for row in self.board:

```

```

166         mark_matrix.append([])
167         for box in row:
168             mark_matrix[-1].append(box.mark)
169
170     # Save the winning mark and return it
171     self.winning_mark = check_winner(mark_matrix)
172     return self.winning_mark
173
174     def is_full(self) -> bool:
175         """
176         Checks if the board is full
177         """
178         for row in self.board:
179             for box in row:
180                 if box.mark is None:
181                     return False
182         return True
183
184     class TickBox:
185         SIZE = (TicTacToeBoard.BOARD_SIZE - (TicTacToeBoard.BOARD_LINE_WIDTH * 2)) // 3
186
187         XO_WIDTH = 7
188         XO_MARGIN = 10
189
190         def __init__(self, board: TicTacToeBoard) -> None:
191             # Rectangle of the box
192             self.rect: Optional[pygame.Rect] = None
193
194             # Mark of the box
195             self.mark: Mark = None
196
197             # The board this box is in
198             self.board: TicTacToeBoard = board
199
200             # Whether or not the box is highlighted
201             self.highlighted: bool = False
202
203         def draw(self, x: int, y: int) -> None:
204             self.rect = pygame.Rect(x, y, self.SIZE, self.SIZE)
205             pygame.draw.rect(screen, Color.WHITE, self.rect)
206             self.draw_mark(x, y)
207
208         def draw_mark(self, x: int, y: int) -> None:
209             # If highlighted, draw a highlight background
210             if self.highlighted:
211                 pygame.draw.rect(screen, Color.HIGHLIGHT, self.rect)
212
213             # Draw the X or O
214             if self.mark == 'X':
215                 DrawShape.X(
216                     x + self.XO_MARGIN,
217                     y + self.XO_MARGIN,
218                     self.SIZE - (self.XO_MARGIN * 2),
219                     self.XO_WIDTH
220                 )
221
222             elif self.mark == 'O':

```

```

223         DrawShape.O(
224             x + (self.XO_MARGIN // 2),
225             y + (self.XO_MARGIN // 2),
226             self.SIZE - self.XO_MARGIN,
227             self.XO_WIDTH
228         )
229
230     def is_hovered_over(self, mouse_pos: Tuple[int, int]) -> bool:
231         if not self.rect:
232             return False
233
234         # Return true if the mouse is over the box
235         if self.rect.collidepoint(mouse_pos):
236             return True
237         return False
238
239 class InceptionBoard:
240     """
241     Inception Board
242     """
243
244     BOARD_SIZE = 750
245     BOARD_LINE_WIDTH = 10
246     BOARD_BOX_SIZE = (BOARD_SIZE - (BOARD_LINE_WIDTH * 2)) // 3
247
248
249     def __init__(self) -> None:
250         # None = empty, X = X, O = O
251         self.board: List[List[TicTacToeBoard]] = []
252
253         # Repeat 3 times
254         for _ in range(3):
255             # Add a list to the board
256             self.board.append([])
257             # Repeat 3 times
258             for _ in range(3):
259                 # Add a TicTacToeBoard to the last list in the board
260                 self.board[-1].append(TicTacToeBoard())
261
262     def get_board(self, x: int, y: int) -> TicTacToeBoard:
263         """
264         Gets the TicTacToeBoard at a specific coordinate
265         """
266         return self.board[y][x]
267
268     def draw(self, x: int, y: int) -> None:
269         """
270         Draws the board
271         """
272
273         # Draws a large black square to fill in with tic tac toe boards
274         pygame.draw.rect(screen, Color.BLACK, (x, y, self.BOARD_SIZE - self.BOARD_LINE_WIDTH,
self.BOARD_SIZE - self.BOARD_LINE_WIDTH))
275
276         # Loop through all 9 squares
277         for boxX in range(3):
278             for boxY in range(3):

```

```

279
280         # Draw a smaller white square in the correct position for the background
281         pygame.draw.rect(
282             screen,
283             Color.WHITE,
284             (
285                 x + ((self.BOARD_SIZE / 3) * boxX),
286                 y + ((self.BOARD_SIZE / 3) * boxY),
287                 self.BOARD_BOX_SIZE,
288                 self.BOARD_BOX_SIZE
289             )
290         )
291
292         # Draw a tic tac toe board in the center of the box
293         self.board[boxY][boxX].draw(
294             x + ((self.BOARD_SIZE / 3) * boxX) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2,
295             y + ((self.BOARD_SIZE / 3) * boxY) + self.BOARD_BOX_SIZE // 2 -
TicTacToeBoard.BOARD_SIZE // 2
296         )
297
298     def get_hovering_tickbox(self) -> Optional[TickBox]:
299         """
300         Gets the TickBox that the mouse is hovering over
301         """
302         mouse_pos = pygame.mouse.get_pos()
303
304         # Loop through all 9 boards
305         for miniboardX in range(3):
306             for miniBoardY in range(3):
307
308                 # Get the board
309                 miniboard = self.get_board(miniboardX, miniBoardY)
310
311                 # Loop through all 9 boxes
312                 for tickboxX in range(3):
313                     for tickboxY in range(3):
314
315                         # Get the tickbox
316                         tickbox = miniboard.get_box(tickboxX, tickboxY)
317
318                         # If the mouse is over the tickbox, return it
319                         if tickbox.is_hovered_over(mouse_pos):
320                             return tickbox
321
322     def check_winner(self) -> Mark:
323         """
324         Checks if there is a winner. Returns the winner if there is one.
325         """
326         # Transform the board into a 3x3 matrix with only the marks
327         mark_matrix: List[List[Mark]] = []
328         for row in self.board:
329             mark_matrix.append([])
330             for board in row:
331                 mark_matrix[-1].append(board.winning_mark)
332
333         return check_winner(mark_matrix)
334

```

```

335     def is_full(self) -> bool:
336         """
337         Checks if every mini board has a winner
338         """
339         for row in self.board:
340             for miniboard in row:
341                 if not miniboard.winning_mark:
342                     return False
343         return True
344
345 # True while the game is running
346 running: bool = True
347
348 # The main board
349 board: InceptionBoard = InceptionBoard()
350
351 # Which player's turn it is
352 turn: Literal['X', 'O'] = 'X'
353
354 # The tickbox that the mouse is currently holding left click on
355 clicked_tickbox: Optional[TickBox] = None
356
357 # The tickbox that the mouse was hovering over last frame
358 last_hovering_tickbox: Optional[TickBox] = None
359
360 # The tickbox that the mouse is currently hovering over
361 hovering_tickbox: Optional[TickBox] = None
362
363 # None = No winner yet, X = X wins, O = O wins, T = Tie
364 winner: Literal['X', 'O', 'T', None] = False
365
366 # Reset button rectangle
367 reset_button_rect: pygame.Rect = pygame.Rect(
368     (WIDTH // 2) - (RESET_BUTTON_WIDTH // 2),
369     100,
370     RESET_BUTTON_WIDTH,
371     RESET_BUTTON_HEIGHT
372 )
373
374 while running:
375     last_hovering_tickbox = hovering_tickbox
376     hovering_tickbox = board.get_hovering_tickbox()
377
378     # If the mouse is hovering over an active tickbox, highlight it and set cursor to hand
379     if hovering_tickbox and not hovering_tickbox.board.winning_mark and not winner:
380         pygame.mouse.set_system_cursor(pygame.SYSTEM_CURSOR_HAND)
381         hovering_tickbox.highlighted = True
382
383     # If the mouse has moved off of a tickbox, unhighlight it and set the cursor to default
384     if last_hovering_tickbox and hovering_tickbox is not last_hovering_tickbox:
385         pygame.mouse.set_system_cursor(pygame.SYSTEM_CURSOR_ARROW)
386         last_hovering_tickbox.highlighted = False
387
388     for event in pygame.event.get(): # Gets all the events which have occurred until now
389         # Listens for the the X button at the top right
390         if event.type == pygame.QUIT:
391             running = False

```

```

392     # Listens for mouse left-click down
393     elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
394         # If the mouse is hovering over a tickbox, set it as the clicked tickbox and wait
for the mouse to be released on it
395         if hovering_tickbox:
396             clicked_tickbox = hovering_tickbox
397
398         if winner and reset_button_rect.collidepoint(pygame.mouse.get_pos()):
399             board = InceptionBoard()
400             turn = 'X'
401             winner = None
402
403     # Listens for mouse left-click release
404     elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:
405         # If the mouse release is on the same tickbox as the mouse click
406         if clicked_tickbox and clicked_tickbox is hovering_tickbox:
407
408             # Get the TicTacToeBoard that the clicked tickbox is in
409             miniboard = clicked_tickbox.board
410
411             # If the box is empty and the board isn't over, fill it with the current
player's mark
412             if not clicked_tickbox.mark and not miniboard.check_winner() and not winner:
413                 clicked_tickbox.mark = turn
414                 turn = 'X' if turn == 'O' else 'O'
415
416             # If there is a tie, reset the board
417             if not miniboard.check_winner() and miniboard.is_full():
418                 miniboard.reset()
419
420             # Check if there is a winner
421             winner = board.check_winner()
422             if not winner and board.is_full():
423                 winner = 'T'
424
425             # Reset the clicked tickbox
426             clicked_tickbox = None
427
428     # Clear the screen
429     screen.fill(Color.WHITE)
430
431     # Draw information at the top of the board
432     # If there is no winner, draw the current player's turn
433     if not winner:
434         # Draw the text
435         text_surface: pygame.Surface = Font.CURRENT_TURN.render('Turn:', True, Color.BLACK)
436         screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
437
438         # Draw the current player's mark
439         if turn == 'X':
440             DrawShape.X(*INFO_SHAPE_ARGS)
441         # If it's Player 0's turn, draw an 0
442         else:
443             DrawShape.O(*INFO_SHAPE_ARGS)
444     # If there is a winner, draw the winner and a button to reset the board
445     else:
446         # If it's not a tie, draw the winner
447         if winner != 'T':

```



```

448         # Draw the text
449         text_surface: pygame.Surface = Font.WINNER.render('Winner!', True, Color.BLACK)
450         screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
451
452         # Draw the winner's mark
453         if winner == 'X':
454             DrawShape.X(*INFO_SHAPE_ARGS)
455         # If it's Player O's turn, draw an O
456         else:
457             DrawShape.O(*INFO_SHAPE_ARGS)
458
459         # If it's a tie, draw the Tie text
460         else:
461             # Draw the text
462             text_surface: pygame.Surface = Font.WINNER.render('Tie!', True, Color.BLACK)
463             screen.blit(text_surface, ((WIDTH // 2) - (text_surface.get_width() // 2), 20))
464
465         # Draw the reset button
466         pygame.draw.rect(screen, Color.LIGHT_GRAY, reset_button_rect)
467         reset_text_surface: pygame.Surface = Font.RESET_BUTTON.render('Reset', True,
Color.BLACK)
468         screen.blit(reset_text_surface, ((WIDTH // 2) - (reset_text_surface.get_width() // 2),
100))
469
470
471         # Draw the board in the horizontal center and bottom of the screen
472         board.draw((WIDTH // 2) - (board.BOARD_SIZE // 2), HEIGHT - board.BOARD_SIZE)
473
474         # Update the screen
475         pygame.display.flip()
476
477     pygame.quit()

```