

```

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 row[0]
78
79     # Check columns
80     for col in range(3):
81         if mark == board[0][col] == board[1][col] == board[2][col]:
82             return board[0][col]
83
84     # Check diagonals
85     if mark == board[0][0] == board[1][1] == board[2][2]:
86         return board[0][0]
87     if mark == 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[Optional[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: Optional[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: Optional[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: Mark) -> bool:
160         """
161         Checks if the given mark is the winner. Returns True if there is one and sets
winning_mark to the corresponding mark.
162         """
163         # Transform the board into a 3x3 matrix with only the marks
164         mark_matrix: List[List[Optional[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     # Check if the mark has won
171     is_winner = check_winner(mark_matrix, mark)
172
173     # If there is a winner, set the winning mark
174     if is_winner:
175         self.winning_mark = mark
176
177     return is_winner
178
179 def is_full(self) -> bool:
180     """
181     Checks if the board is full
182     """
183     for row in self.board:
184         for box in row:
185             if box.mark is None:
186                 return False
187     return True
188
189 class TickBox:
190     SIZE = (TicTacToeBoard.BOARD_SIZE - (TicTacToeBoard.BOARD_LINE_WIDTH * 2)) // 3
191
192     XO_WIDTH = 7
193     XO_MARGIN = 10
194
195     def __init__(self, board: TicTacToeBoard) -> None:
196         # Rectangle of the box
197         self.rect: Optional[pygame.Rect] = None
198
199         # Mark of the box
200         self.mark: Optional[Mark] = None
201
202         # The board this box is in
203         self.board: TicTacToeBoard = board
204
205         # Whether or not the box is highlighted
206         self.highlighted: bool = False
207
208     def draw(self, x: int, y: int) -> None:
209         self.rect = pygame.Rect(x, y, self.SIZE, self.SIZE)
210         pygame.draw.rect(screen, Color.WHITE, self.rect)
211         self.draw_mark(x, y)
212
213     def draw_mark(self, x: int, y: int) -> None:
214         # If highlighted, draw a highlight background
215         if self.highlighted:
216             pygame.draw.rect(screen, Color.HIGHLIGHT, self.rect)
217
218         # Draw the X or O
219         if self.mark == 'X':
220             DrawShape.X(
221                 x + self.XO_MARGIN,
222                 y + self.XO_MARGIN,

```

```

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

```

```

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

```

```

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

```

```

392     pygame.mouse.set_system_cursor(pygame.SYSTEM_CURSOR_HAND)
393     hovering_tickbox.highlighted = True
394
395     # If the mouse has moved off of a tickbox, unhighlight it and set the cursor to default
396     if last_hovering_tickbox and hovering_tickbox is not last_hovering_tickbox:
397         pygame.mouse.set_system_cursor(pygame.SYSTEM_CURSOR_ARROW)
398         last_hovering_tickbox.highlighted = False
399
400     for event in pygame.event.get(): # Gets all the events which have occurred until now
401         # Listens for the the X button at the top right
402         if event.type == pygame.QUIT:
403             running = False
404         # Listens for mouse left-click down
405         elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
406             # If the mouse is hovering over a tickbox, set it as the clicked tickbox and wait
407             for the mouse to be released on it
408                 if hovering_tickbox:
409                     clicked_tickbox = hovering_tickbox
410
411                 if winner and reset_button_rect.collidepoint(pygame.mouse.get_pos()):
412                     board = InceptionBoard()
413                     turn = 'X'
414                     winner = None
415
416             # Listens for mouse left-click release
417             elif event.type == pygame.MOUSEBUTTONUP and event.button == 1:
418                 # If the mouse release is on the same tickbox as the mouse click
419                 if clicked_tickbox and clicked_tickbox is hovering_tickbox:
420
421                     # Get the TicTacToeBoard that the clicked tickbox is in
422                     miniboard = clicked_tickbox.board
423
424                     # If the box is empty and the board isn't over, fill it with the current
425                     player's mark
426                     if not clicked_tickbox.mark and not miniboard.check_winner(turn) and not
427                     winner:
428                         clicked_tickbox.mark = turn
429
430                     # If there is a tie, reset the board
431                     if not miniboard.check_winner(turn) and miniboard.is_full():
432                         miniboard.reset()
433
434                     # Check if there is a winner
435                     winner = board.check_winner(turn)
436                     if not winner and board.is_full():
437                         winner = 'T'
438
439                     turn = 'X' if turn == 'O' else 'O'
440
441                     # Reset the clicked tickbox
442                     clicked_tickbox = None
443
444             # Clear the screen
445             screen.fill(Color.WHITE)
446
447             # Draw information at the top of the board
448             # If there is no winner, draw the current player's turn
449             if not winner:

```



```

447     # Draw the text
448     text_surface: pygame.Surface = Font.CURRENT_TURN.render('Turn:', True, Color.BLACK)
449     screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
450
451     # Draw the current player's mark
452     if turn == 'X':
453         DrawShape.X(*INFO_SHAPE_ARGS)
454     # If it's Player O's turn, draw an O
455     else:
456         DrawShape.O(*INFO_SHAPE_ARGS)
457 # If there is a winner, draw the winner and a button to reset the board
458 else:
459     # If it's not a tie, draw the winner
460     if winner != 'T':
461         # Draw the text
462         text_surface: pygame.Surface = Font.WINNER.render('Winner!', True, Color.BLACK)
463         screen.blit(text_surface, ((WIDTH // 2) - text_surface.get_width(), 20))
464
465         # Draw the winner's mark
466         if winner == 'X':
467             DrawShape.X(*INFO_SHAPE_ARGS)
468         # If it's Player O's turn, draw an O
469         else:
470             DrawShape.O(*INFO_SHAPE_ARGS)
471
472     # If it's a tie, draw the Tie text
473     else:
474         # Draw the text
475         text_surface: pygame.Surface = Font.WINNER.render('Tie!', True, Color.BLACK)
476         screen.blit(text_surface, ((WIDTH // 2) - (text_surface.get_width() // 2), 20))
477
478     # Draw the reset button
479     pygame.draw.rect(screen, Color.LIGHT_GRAY, reset_button_rect)
480     reset_text_surface: pygame.Surface = Font.RESET_BUTTON.render('Reset', True,
Color.BLACK)
481     screen.blit(reset_text_surface, ((WIDTH // 2) - (reset_text_surface.get_width() // 2),
100))
482
483
484     # Draw the board in the horizontal center and bottom of the screen
485     board.draw((WIDTH // 2) - (board.BOARD_SIZE // 2), HEIGHT - board.BOARD_SIZE)
486
487     # Update the screen
488     pygame.display.flip()
489
490 pygame.quit()

```