מסמך מסכם- פרויקט סיום באלגוריתמים מתקדמים לתכנון ותזמון מערכות נבונות

מגישים:

דניאל כהן, אושרת כהן, בר לוי אטיאס

לינק לסרטון המצגת:

https://youtu.be/aOjH7QSvHgQ

לינק לסרטון המשחק:

https://youtu.be/jvj5KD5Hm6A

לינק ל- GitHub:

https://github.com/barleviatias/smart_tic_tac_toe

מבוא לפרויקט:

הפרויקט שלנו היה לממש משחק איקס עיגול משודרג, המשתמש באלגוריתם Minimax עם גיזום Alpha-Beta. שילוב כפתור כוכב המאפשר לאפס ולחסום תא אחד במהלך המשחק אך ניתן להשתמש בו פעם אחת בלבד. הוספנו רמת קושי, לוח בגודל 5X5 שניצחון במקרה זה נקבע ע"פ 4 סימנים רצופים של אחד מהשחקנים. בלוח משחק 3X3 הניצחון נקבע ע"פ 3 סימנים רצופים.. מטרתנו הייתה ליצור משחק מאתגר יותר, אינטראקטיבי ומעניין, המשלב אלגוריתמים מתקדמים וממשק ידידותי למשתמש.

הבעיה:

איקס עיגול הוא משחק פשוט אך מאתגר, במיוחד כאשר המטרה היא לבנות יריב ממוחשב שיכול לשחק בצורה אופטימלית. הבעיה שלנו הייתה למצוא דרך לממש מחשב כשחקן המסוגל לשחק באופן חכם, כך שיוכל להוות אתגר ראוי לשחקן אנושי.

הפתרון:

פתרנו את הבעיה על ידי שימוש באלגוריתם Minimax, שהוא אלגוריתם לבחירת המהלך האופטימלי במשחקים דו-שחקניים. כדי לשפר את ביצועי האלגוריתם ולהפחית את כמות הצמתים הנבדקים, שילבנו את גיזום Alpha-Beta. בנוסף, הוספנו למשחק אפשרות להשתמש בכוכבית לאיפוס וחסימת תא אחד בלוח, מה שמוסיף אלמנט אסטרטגי נוסף למשחק וגם דרגת קושי נוספת המתארת את השחקנים.

<u>נרחיב על האלגוריתמים בהם בחרנו להשתמש:</u>

<u>אלגוריתם Minimax</u>

אלגוריתם Minimax נועד למציאת המהלך האופטימלי במשחקים דו-שחקניים על ידי חישוב כל האפשרויות להמשך המשחק ודירוגן לפי פוטנציאל לניצחון או לתיקו.

מימוש: האלגוריתם עובר על כל מהלך אפשרי בלוח, מחשב את ערך המהלך ומחזיר את המהלך הטוב ביותר לשחקן המחשב.

<u>גיזום Alpha-Beta</u>

טכניקת אופטימיזציה לאלגוריתם Minimax שמטרתה להפחית את מספר הצמתים הנבדקים בעץ המשחק.

מימוש: הגיזום שומר על גבולות ערכים (alpha ו-beta) המייצגים את הערכים הגבוהים והנמוכים ביותר שנבדקו, ומדלג על צמתים שאינם עומדים בקריטריונים אלו.

<u>מימוש בפועל של הפרויקט כולו:</u>

שפת תכנות: השתמשנו ב-Python ובספריית PyGame לממשק הגרפי של המשחק. השימוש ב-PyGame אפשר לנו ליצור ממשק משתמש אינטואיטיבי ונוח לשימוש.

<u>ממשק גרפי:</u>

הממשק כולל לוח משחק בגודל 3x3 או 5x5, כפתור הוראות משחק, כפתור כוכבית לחסימת תאים וכפתורי איפוס המשחק ואיפוס הניקוד.

בנוסף, לצורך נוחות הוספנו כפתור חזרה לתפריט הראשי, במקרה שיש צורך לקרוא שוב את הוראות המשחק או לבחור רמת קושי שונה.

הוספנו חיווי למשתמש בעת ניצחון או תיקו, זאת ועוד כאשר אחד השחקנים מנצח התאים איתו ניצח מסומנים בקו.

<u>אתגרים:</u>

- מימוש אלגוריתם Minimax וגיזום Alpha-Beta: דרשנו לחשב כל מהלך אפשרי ולבצע אופטימיזציה של הביצועים כדי להבטיח שהמחשב יוכל לקבל החלטות במהירות.
- מימוש כפתור הכוכבית: היה צורך לטפל במקרים מיוחדים כמו שימוש בכוכבית כאשר הלוח ריק.
- עיצוב ממשק גרפי אינטואיטיבי: הוספנו עיצוב ידידותי ושימושי כדי להבטיח חווית משתמש טובה.

לסיכום:

בפרויקט זה מימשנו משחק איקס עיגול משודרג המשלב אלגוריתמים מתקדמים עם ממשק משתמש אינטואיטיבי. בעזרת אלגוריתם Minimax וגיזום Alpha-Beta הצלחנו ליצור יריב ממוחשב חכם, והוספת כפתור הכוכבית הוסיפה רובד נוסף של אסטרטגיה למשחק.רמת קושי נוספת שבאה לידי ביטוי בגודל לוח המשחק.

ההתמודדות עם האתגרים הטכניים והעיצוביים תרמה רבות להבנה שלנו באלגוריתמים מתקדמים ובפיתוח ממשקי משתמש.

צירוף הקוד:

```
import pygameimport pygame
import sys
import random
import time
import math

# אתחול Pygame
pygame.init()

# קבועים
WIDTH, HEIGHT = 600, 700
BOARD_SIZE = 500
BUTTON_WIDTH, BUTTON_HEIGHT = 150, 50

# צבעים
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
BEIGE = (245, 245, 220)
BROWN = (139, 69, 19)
YELLOW = (255, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
```

```
# גופנים
FONT = pygame.font.Font(None, 24)
LARGE_FONT = pygame.font.Font(None, 68)
XO_FONT = pygame.font.Font(None, 120)  # גופן גדול יותר ל-X ו-O

class TicTacToe:
    def __init__(self):
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Tic Tac Toe")
        self.clock = pygame.time.Clock()
        self.star_color = YELLOW
        self.board_size = None
        self.cell_size = None
        self.board = None
        self.current_player = 'X'
        self.player_x_wins = 0
        self.player_o_wins = 0
        self.x_star_used = False
        self.o_star_used = False
        self.star_mode = False
        self.winning_line = None
        self.winner = None
        self.disabled_cells = []
        self.computer_thinking = False
        self.computer_move_time = None
        self.message = ""
        self.message_time = 0

        self.state = "menu"

    # מציירת את התפריט הראשי
    def draw_menu(self):
        self.screen.fill(BEIGE)
        title = LARGE_FONT.render("Tic Tac Toe", True, BROWN)
        self.screen.blit(title, (WIDTH // 2 - title.get_width() // 2, 50))

        button_3x3 = pygame.Rect(WIDTH // 2 - BUTTON_WIDTH // 2, 200,
BUTTON_WIDTH, BUTTON_HEIGHT)
        button_5x5 = pygame.Rect(WIDTH // 2 - BUTTON_WIDTH // 2, 300,
BUTTON_WIDTH, BUTTON_HEIGHT)
```

```
        button_instructions = pygame.Rect(WIDTH // 2 - BUTTON_WIDTH // 2, 400,
BUTTON_WIDTH, BUTTON_HEIGHT)

        pygame.draw.rect(self.screen, BROWN, button_3x3)
        pygame.draw.rect(self.screen, BROWN, button_5x5)
        pygame.draw.rect(self.screen, BROWN, button_instructions)

        text_3x3 = FONT.render("3x3", True, WHITE)
        text_5x5 = FONT.render("5x5", True, WHITE)
        text_instructions = FONT.render("Instructions", True, WHITE)

        self.screen.blit(text_3x3, (button_3x3.x + button_3x3.width // 2 -
text_3x3.get_width() // 2,
                                    button_3x3.y + button_3x3.height // 2 -
text_3x3.get_height() // 2))
        self.screen.blit(text_5x5, (button_5x5.x + button_5x5.width // 2 -
text_5x5.get_width() // 2,
                                    button_5x5.y + button_5x5.height // 2 -
text_5x5.get_height() // 2))
        self.screen.blit(text_instructions,
                         (button_instructions.x + button_instructions.width //
2 - text_instructions.get_width() // 2,
                          button_instructions.y + button_instructions.height
// 2 - text_instructions.get_height() // 2))

        return button_3x3, button_5x5, button_instructions

    # מציירת כוכב על המסך
    def draw_star(self, surface, color, x, y, size):
        points = []
        for i in range(10):
            angle = math.pi * 2 * i / 10 - math.pi / 2
            radius = size / 2 if i % 2 == 0 else size / 4
            point_x = x + radius * math.cos(angle)
            point_y = y + radius * math.sin(angle)
            points.append((point_x, point_y))
        pygame.draw.polygon(surface, color, points)

    # מציירת את לוח המשחק
    def draw_board(self):
        self.screen.fill(BEIGE)
```

```
        # Draw header
        header = FONT.render(f"Player X: {self.player_x_wins} | Player O:
{self.player_o_wins}", True, BROWN)
        self.screen.blit(header, (WIDTH // 2 - header.get_width() // 2, 20))

        # Draw board
        for i in range(self.board_size):
            for j in range(self.board_size):
                rect = pygame.Rect(j * self.cell_size + (WIDTH - BOARD_SIZE)
// 2,
                                   i * self.cell_size + (HEIGHT - BOARD_SIZE)
// 2,
                                   self.cell_size, self.cell_size)
                pygame.draw.rect(self.screen, BROWN, rect, 2)

                if self.board[i][j] == 'X':
                    text = XO_FONT.render('X', True, RED)
                    self.screen.blit(text, (rect.centerx - text.get_width() //
2,
                                            rect.centery - text.get_height()
// 2))
                elif self.board[i][j] == 'O':
                    text = XO_FONT.render('O', True, BLUE)
                    self.screen.blit(text, (rect.centerx - text.get_width() //
2,
                                            rect.centery - text.get_height()
// 2))
                elif (i, j) in self.disabled_cells:
                    star_size = min(self.cell_size, self.cell_size) * 0.6
                    self.draw_star(self.screen, self.star_color, rect.centerx,
rect.centery, star_size)

        # Draw message
        if self.message and time.time() < self.message_time:
            message_surface = FONT.render(self.message, True, RED)
            self.screen.blit(message_surface, (WIDTH // 2 -
message_surface.get_width() // 2, HEIGHT - 100))

        # Draw winning line if exists and there's a winner
        if self.winning_line and self.winner:
```

```
            start, end = self.winning_line
            start_pos = ((start[1] + 0.5) * self.cell_size + (WIDTH -
BOARD_SIZE) // 2,
                         (start[0] + 0.5) * self.cell_size + (HEIGHT -
BOARD_SIZE) // 2)
            end_pos = ((end[1] + 0.5) * self.cell_size + (WIDTH - BOARD_SIZE)
// 2,
                       (end[0] + 0.5) * self.cell_size + (HEIGHT - BOARD_SIZE)
// 2)
            pygame.draw.line(self.screen, GREEN, start_pos, end_pos, 5)

        # Draw "Computer thinking..." message
        if self.computer_thinking:
            thinking_text = FONT.render("Computer thinking...", True, BROWN)
            self.screen.blit(thinking_text, (WIDTH // 2 -
thinking_text.get_width() // 2, HEIGHT - 100))

        # Draw buttons
        button_star = pygame.Rect(50, HEIGHT - 70, BUTTON_WIDTH,
BUTTON_HEIGHT)
        button_reset = pygame.Rect(WIDTH // 2 - BUTTON_WIDTH // 2, HEIGHT -
70, BUTTON_WIDTH, BUTTON_HEIGHT)
        button_reset_scores = pygame.Rect(WIDTH - BUTTON_WIDTH - 50, HEIGHT -
70, BUTTON_WIDTH, BUTTON_HEIGHT)
        button_main_menu = pygame.Rect(WIDTH - BUTTON_WIDTH - 50, 20,
BUTTON_WIDTH, BUTTON_HEIGHT)

        pygame.draw.rect(self.screen, YELLOW if not (self.x_star_used and
self.current_player == 'X' or
                                                     self.o_star_used and
self.current_player == 'O') else BROWN,
                         button_star)
        pygame.draw.rect(self.screen, BROWN, button_reset)
        pygame.draw.rect(self.screen, BROWN, button_reset_scores)
        pygame.draw.rect(self.screen, BROWN, button_main_menu)

        text_star = FONT.render("Use Star", True, BLACK)
        text_reset = FONT.render("Reset Game", True, WHITE)
        text_reset_scores = FONT.render("Reset Scores", True, WHITE)
        text_main_menu = FONT.render("Main Menu", True, WHITE)
```

```
        self.screen.blit(text_star, (button_star.x + button_star.width // 2 -
text_star.get_width() // 2,
                                     button_star.y + button_star.height // 2 -
text_star.get_height() // 2))
        self.screen.blit(text_reset, (button_reset.x + button_reset.width // 2
- text_reset.get_width() // 2,
                                      button_reset.y + button_reset.height //
2 - text_reset.get_height() // 2))
        self.screen.blit(text_reset_scores,
                         (button_reset_scores.x + button_reset_scores.width //
2 - text_reset_scores.get_width() // 2,
                          button_reset_scores.y + button_reset_scores.height
// 2 - text_reset_scores.get_height() // 2))
        self.screen.blit(text_main_menu,
                         (button_main_menu.x + button_main_menu.width // 2 -
text_main_menu.get_width() // 2,
                          button_main_menu.y + button_main_menu.height // 2 -
text_main_menu.get_height() // 2))

        return button_star, button_reset, button_reset_scores,
button_main_menu

    # מציירת את מסך ההוראות
    def draw_instructions(self):
        self.screen.fill(BEIGE)
        title = LARGE_FONT.render("Instructions", True, BROWN)
        self.screen.blit(title, (WIDTH // 2 - title.get_width() // 2, 50))

        instructions = [
            "Game Objective:",
            "Be the first player to arrange four consecutive marks",
            "(vertical, horizontal, or diagonal) on the game board.",
            "",
            "Difficulty Selection:",
            "3x3 board: Victory is determined by 3 consecutive marks.",
            "5x5 board: Victory is determined by 4 consecutive marks.",
            "",
            "Star Mark:",
            "A star mark can be used once per game.",
            "The star mark resets and blocks a cell on the board,",
            "preventing any player from using it.",
```

```
            "",
            "Game Process:",
            "- Click on a cell to place your mark (X or O).",
            "- Arrange consecutive marks to win.",
            "- Game ends in a draw if all cells are filled with no winner.",
            "",
            "Score Management:",
            "- Victories are counted for players X and O.",
            "- Reset scores using the 'Reset Scores' button."
        ]

        y = 150
        for line in instructions:
            text = FONT.render(line, True, BROWN)
            self.screen.blit(text, (50, y))
            y += 30

        button_back = pygame.Rect(WIDTH // 2 - BUTTON_WIDTH // 2, HEIGHT - 70,
BUTTON_WIDTH, BUTTON_HEIGHT)
        pygame.draw.rect(self.screen, BROWN, button_back)
        text_back = FONT.render("Back", True, WHITE)
        self.screen.blit(text_back, (button_back.x + button_back.width // 2 -
text_back.get_width() // 2,
                                     button_back.y + button_back.height // 2 -
text_back.get_height() // 2))

        return button_back

    # מתחילה משחק חדש
    def start_game(self, size):
        self.board_size = size
        self.cell_size = BOARD_SIZE // size
        self.board = [['' for _ in range(self.board_size)] for _ in
range(self.board_size)]
        self.current_player = 'X'
        self.x_star_used = False
        self.o_star_used = False
        self.star_mode = False
        self.disabled_cells = []
        self.winning_line = None
        self.winner = None
```

```
        self.state = "game"

    # מטפלת בלחיצה על תא בלוח
    def click(self, i, j):
        if self.star_mode:
            self.star_click(i, j)
        elif self.board[i][j] == '' and (i, j) not in self.disabled_cells:
            self.board[i][j] = self.current_player
            if self.check_game_over():
                return
            self.current_player = 'O' if self.current_player == 'X' else 'X'
            if self.current_player == 'O':
                self.computer_thinking = True
                self.computer_move_time = time.time() + 1  # Set computer move
time to 1 second from now

    # מטפלת בלחיצה על תא במצב כוכב
    def star_click(self, i, j):
        if self.board[i][j] != '':
            self.board[i][j] = ''
            self.star_mode = False
            self.disabled_cells.append((i, j))
            self.current_player = 'O' if self.current_player == 'X' else 'X'
            self.check_game_over()
            if self.current_player == 'O':
                self.computer_thinking = True
                self.computer_move_time = time.time() + 1  # Set computer move
time to 1 second from now
        else:
            self.show_message("Cannot use star on empty cell!")
            self.star_mode = False
            if self.current_player == 'X':
                self.x_star_used = False
            else:
                self.o_star_used = False

    # מציגה הודעה על המסך
    def show_message(self, text):
        self.message = text
        self.message_time = time.time() + 2  # Display message for 2 seconds
```

```
    # מבצעת מהלך של המחשב
    def make_computer_move(self):
        start_time = time.time()
        move = self.best_move()
        end_time = time.time()
        think_time = end_time - start_time
        print(f"Computer move calculation took {think_time:.2f} seconds")

        # Ensure minimum thinking time of 1 second
        if time.time() < self.computer_move_time:
            return  # Wait until it's time to make the move

        if move:
            i, j = move
            self.board[i][j] = 'O'
            if self.check_game_over():
                return
            self.current_player = 'X'
        else:
            print("No valid moves available for computer")
        self.computer_thinking = False
        self.computer_move_time = None

    # מוצאת את המהלך הטוב ביותר עבור המחשב
    def best_move(self):
        best_score = float('-inf')
        move = None
        alpha = float('-inf')
        beta = float('inf')
        depth_limit = 3 if self.board_size == 5 else 5  # Adjust depth limit
based on board size

        for i in range(self.board_size):
            for j in range(self.board_size):
                if self.board[i][j] == '' and (i, j) not in
self.disabled_cells:
                    self.board[i][j] = 'O'
                    score = self.minimax(0, False, alpha, beta, depth_limit)
                    self.board[i][j] = ''
                    if score > best_score:
                        best_score = score
```

```
                        move = (i, j)
                    alpha = max(alpha, best_score)
                    if beta <= alpha:
                        break
        return move

    # אלגוריתם מינימקס עם גיזום אלפא-בטא
    def minimax(self, depth, is_maximizing, alpha, beta, depth_limit):
        winner = self.check_winner()
        if winner == 'X':
            return -10 + depth
        elif winner == 'O':
            return 10 - depth
        elif self.is_full() or depth == depth_limit:
            return 0

        if is_maximizing:
            best_score = float('-inf')
            for i in range(self.board_size):
                for j in range(self.board_size):
                    if self.board[i][j] == '' and (i, j) not in
self.disabled_cells:
                        self.board[i][j] = 'O'
                        score = self.minimax(depth + 1, False, alpha, beta,
depth_limit)
                        self.board[i][j] = ''
                        best_score = max(score, best_score)
                        alpha = max(alpha, best_score)
                        if beta <= alpha:
                            break
            return best_score
        else:
            best_score = float('inf')
            for i in range(self.board_size):
                for j in range(self.board_size):
                    if self.board[i][j] == '' and (i, j) not in
self.disabled_cells:
                        self.board[i][j] = 'X'
                        score = self.minimax(depth + 1, True, alpha, beta,
depth_limit)
                        self.board[i][j] = ''
```

```
                        best_score = min(score, best_score)
                        beta = min(beta, best_score)
                        if beta <= alpha:
                            break
            return best_score

    # בודקת אם יש מנצח
    def check_winner(self):
        win_length = 4 if self.board_size == 5 else 3

        # Check rows and columns
        for i in range(self.board_size):
            for j in range(self.board_size - win_length + 1):
                if self.board[i][j] and all(self.board[i][j + k] ==
self.board[i][j] for k in range(win_length)):
                    self.winning_line = ((i, j), (i, j + win_length - 1))
                    return self.board[i][j]
                if self.board[j][i] and all(self.board[j + k][i] ==
self.board[j][i] for k in range(win_length)):
                    self.winning_line = ((j, i), (j + win_length - 1, i))
                    return self.board[j][i]

        # Check diagonals
        for i in range(self.board_size - win_length + 1):
            for j in range(self.board_size - win_length + 1):
                if self.board[i][j] and all(
                        self.board[i + k][j + k] == self.board[i][j] for k in
range(win_length)):
                    self.winning_line = ((i, j), (i + win_length - 1, j +
win_length - 1))
                    return self.board[i][j]
                if self.board[i][j + win_length - 1] and all(
                        self.board[i + k][j + win_length - 1 - k] ==
self.board[i][j + win_length - 1] for k in
                        range(win_length)):
                    self.winning_line = ((i, j + win_length - 1), (i +
win_length - 1, j))
                    return self.board[i][j + win_length - 1]

        return None
```

```
    # בודקת אם הלוח מלא
    def is_full(self):
        return all(self.board[i][j] != '' or (i, j) in self.disabled_cells
                   for i in range(self.board_size) for j in
range(self.board_size))

    # בודקת אם המשחק הסתיים
    def check_game_over(self):
        self.winner = self.check_winner()
        if self.winner:
            if self.winner == 'X':
                self.player_x_wins += 1
            else:
                self.player_o_wins += 1
            self.show_game_over(f"{self.winner} WINS!")
            return True
        elif self.is_full():
            self.show_game_over("It's a draw!")
            return True
        return False

    # מציגה מסך סיום משחק
    def show_game_over(self, message):
        self.draw_board()  # Draw the final board state
        overlay = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)
        overlay.fill((255, 255, 255, 128))  # Semi-transparent white
        self.screen.blit(overlay, (0, 0))

        text = LARGE_FONT.render(message, True, BROWN)
        self.screen.blit(text, (WIDTH // 2 - text.get_width() // 2, HEIGHT //
2 - text.get_height() // 2))
        pygame.display.flip()
        pygame.time.wait(2000)
        self.reset_board()

    # מאפסת את הלוח למשחק חדש
    def reset_board(self):
        self.board = [['' for _ in range(self.board_size)] for _ in
range(self.board_size)]
        self.current_player = 'X'
        self.x_star_used = False
```

```
        self.o_star_used = False
        self.star_mode = False
        self.disabled_cells = []
        self.winning_line = None
        self.winner = None

    # מאפסת את הניקוד
    def reset_scores(self):
        self.player_x_wins = 0
        self.player_o_wins = 0

    # מפעילה את המשחק
    def run(self):
        running = True
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False

                if event.type == pygame.MOUSEBUTTONDOWN and not
self.computer_thinking:
                    if self.state == "menu":
                        button_3x3, button_5x5, button_instructions =
self.draw_menu()
                        if button_3x3.collidepoint(event.pos):
                            self.start_game(3)
                        elif button_5x5.collidepoint(event.pos):
                            self.start_game(5)
                        elif button_instructions.collidepoint(event.pos):
                            self.state = "instructions"

                    elif self.state == "instructions":
                        button_back = self.draw_instructions()
                        if button_back.collidepoint(event.pos):
                            self.state = "menu"

                    elif self.state == "game":
                        button_star, button_reset, button_reset_scores,
button_main_menu = self.draw_board()
                        if button_star.collidepoint(event.pos):
```

```
                            if (self.current_player == 'X' and not
self.x_star_used) or \
                                    (self.current_player == 'O' and not
self.o_star_used):
                                self.star_mode = True
                                if self.current_player == 'X':
                                    self.x_star_used = True
                                else:
                                    self.o_star_used = True
                        elif button_reset.collidepoint(event.pos):
                            self.reset_board()
                        elif button_reset_scores.collidepoint(event.pos):
                            self.reset_scores()
                        elif button_main_menu.collidepoint(event.pos):
                            self.state = "menu"
                            self.reset_board()
                        else:
                            x, y = event.pos
                            board_x = (x - (WIDTH - BOARD_SIZE) // 2) //
self.cell_size
                            board_y = (y - (HEIGHT - BOARD_SIZE) // 2) //
self.cell_size
                            if 0 <= board_x < self.board_size and 0 <= board_y
< self.board_size:
                                self.click(board_y, board_x)

            if self.state == "menu":
                self.draw_menu()
            elif self.state == "instructions":
                self.draw_instructions()
            elif self.state == "game":
                self.draw_board()
                if self.computer_thinking and self.computer_move_time and
time.time() >= self.computer_move_time:
                    self.make_computer_move()

            pygame.display.flip()
            self.clock.tick(60)

        pygame.quit()
        sys.exit()
```

```python
if __name__ == "__main__":
    game = TicTacToe()
    game.run()
```