TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Đề tài:

Trò chơi cờ vua

Giảng viên hướng dẫn: Từ Lãng Phiêu
Nhóm thực hiện: Nhóm 21
Sinh viên thực hiện: 3120410456 – Hồ Tú Tài
3120410344 – Nguyễn Anh Nghĩa
3120410427 – Phan Minh Quang
3120410306 – Nguyễn Thành Lực

TP. HỒ CHÍ MINH, THÁNG 5/2024

DANH SÁCH THÀNH VIÊN NHÓM 21

MSSV	Họ và tên	% công việc
3120410456	Hồ Tú Tài	25%
3120410306	Nguyễn Thành Lực	25%
3120410344	Nguyễn Anh Nghĩa	25%
3120410427	Phan Minh Quang (Chuyển từ nhóm 5 sáng T2 và T6)	25%

Mục lục

1 TỔNG QUAN GIAO DIỆN **3**
1.1 Các thành phần trên giao diện . . . 3
1.2 Xử lí sự kiện button . . . 8

2 Chế độ Chess Online **10**
2.1 Giới thiệu . . . 10
2.2 Hướng dẫn cách chơi Chess . . . 10
2.3 FlowChart . . . 12
2.4 Thư viện cần dùng . . . 12
2.5 Thiết kế client . . . 12
2.6 Thiết kế bàn cờ . . . 15
2.6.1 Cập nhập di chuyển . . . 16
2.6.2 Vẽ bàn cờ và quân cờ . . . 17
2.6.3 Lấy danh sách nước đi nguy hiểm cho một màu cụ thể . . . 18
2.6.4 Kiểm tra màu bị chiếu . . . 19
2.6.5 Xử lý chọn ô trên bàn cờ . . . 20
2.6.6 Đặt lại trạng thái quân cờ . . . 22
2.6.7 Kiểm tra bị chiếu . . . 22
2.6.8 Di chuyển . . . 23
2.7 Thiết kế quân cờ . . . 24
2.7.1 Hiển thị quân cờ . . . 24
2.7.2 Khởi tạo lớp Piece . . . 25
2.7.3 Lớp Bishop và cách di chuyển . . . 25
2.7.4 Lớp King và cách di chuyển . . . 26
2.7.5 Lớp Knight và cách di chuyển . . . 27
2.7.6 Lớp Pawn và cách di chuyển . . . 27
2.7.7 Lớp Queen và cách di chuyển . . . 28
2.7.8 Lớp Rook và cách di chuyển . . . 29
2.8 Thiết kế server . . . 30
2.8.1 Khởi tạo socket và các biến . . . 30
2.8.2 Đọc danh sách . . . 31
2.8.3 Xử lý kết nối . . . 32
2.8.4 kết nối client . . . 33
2.9 Thiết kế chính . . . 34
2.9.1 Kiểm tra install . . . 34
2.9.2 Tạo biến và hình ảnh . . . 35
2.9.3 Vẽ cửa sổ,bảng,thời gian,thông báo và trạng thái . . . 36
2.9.4 Màn hình kết thúc . . . 36
2.9.5 Xác định vị trí ô cờ . . . 37

3 Web demo và tải game **38**

1 TỔNG QUAN GIAO DIỆN

1.1 Các thành phần trên giao diện

Giao diện được thiết kế bằng python bao gồm một cửa sổ hiển thị hình ảnh giới thiệu chương trình, với bên trái bao gồm tiêu đề Chess game và hình ảnh, và bên phải gồm 3 nút button để bao gồm nút Play, nút Variants và nút Quit

Trong nút Variants sẽ bao gồm thêm 3 nút đó là Variant 1, Variant 2 và nút Back.

'Tạo cửa sổ game và tiêu đề

```
pygame.init()

WIDTH = 800
HEIGHT = 600

win = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Main Menu")
```

‘

`pygame.init()` dùng để khởi tạo thư viện Pygame. Ta tạo cửa sổ với kích thước chiều rộng và chiều dài với width = 800 và height = 600.Đặt tiêu đề cho cửa sổ màn hình với `pygame.display.set_caption()`

‘Tạo màu sắc, Font và nút button

```
# Màu sắc
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# Font
FONT_TITLE = pygame.font.Font(None, 80)
FONT_BUTTON = pygame.font.Font(None, 50)

# Các nút menu
button1 = pygame.Rect(500, 200, 200, 80)
button2 = pygame.Rect(500, 300, 200, 80)
button3 = pygame.Rect(500, 400, 200, 80)
```

WHITE = (255, 255, 255): Định nghĩa màu trắng bằng cách sử dụng các giá trị RGB (Red, Green, Blue) là (255, 255, 255). Mỗi giá trị trong khoảng từ 0 đến 255 đại diện cho mức độ của màu sắc tương ứng.

BLACK = (0, 0, 0): Định nghĩa màu đen bằng cách sử dụng các giá trị RGB là (0, 0, 0). Màu đen không có sự hiện diện của màu sắc. FONT_TITLE = pygame.font.Font(None, 80): Tạo một đối tượng font chữ có kích thước 80. Đối tượng font này sẽ được sử dụng để hiển thị tiêu đề.

FONT_BUTTON = pygame.font.Font(None, 50): Tạo một đối tượng font chữ có kích thước 50. Đối tượng font này sẽ được sử dụng để hiển thị nút menu.

button1 = pygame.Rect(500, 200, 200, 80): Tạo một đối tượng hình chữ nhật (pygame.Rect) đại diện cho nút menu thứ nhất. Đối tượng này có vị trí x là 500, vị trí y là 200, chiều rộng là 200 và chiều cao là 80.

Tương tự, button2 và button3 là các đối tượng hình chữ nhật đại diện cho nút menu thứ hai và thứ ba. Chúng có vị trí và kích thước khác nhau.

'Vẽ giao diện menu

```
def draw_menu():
    win.fill(WHITE)
    win.blit(main_menu_image, (0, 0))
    title_text = FONT_TITLE.render("Chess Game", True, WHITE)

    win.blit(title_text, (WIDTH / 4 - title_text.get_width() / 2, 100))
    if button1_visible:
        pygame.draw.rect(win, BLACK, button1)
        button1_text = FONT_BUTTON.render("Play", True, WHITE)
        win.blit(button1_text, (button1.x + button1.width / 2 - button1_text.get_width() / 2, button1.y + 20))
    if button2_visible:
        pygame.draw.rect(win, BLACK, button2)
        button2_text = FONT_BUTTON.render("Variants", True, WHITE)
        win.blit(button2_text, (button2.x + button2.width / 2 - button2_text.get_width() / 2, button2.y + 20))
    if button3_visible:
        pygame.draw.rect(win, BLACK, button3)
        button3_text = FONT_BUTTON.render("Quit", True, WHITE)
        win.blit(button3_text, (button3.x + button3.width / 2 - button3_text.get_width() / 2, button3.y + 20))

    if variant_button1_visible:
        pygame.draw.rect(win, BLACK, variant_button1)
        variant_button1_text = FONT_BUTTON.render("Variant 1", True, WHITE)
        win.blit(variant_button1_text, (variant_button1.x + variant_button1.width / 2 - variant_button1_text.get_width() / 2, variant_button1.y +
    if variant_button2_visible:
        pygame.draw.rect(win, BLACK, variant_button2)
        variant_button2_text = FONT_BUTTON.render("Variant 2", True, WHITE)
        win.blit(variant_button2_text, (variant_button2.x + variant_button2.width / 2 - variant_button2_text.get_width() / 2, variant_button2.y +
    if variant_button3_visible:
        pygame.draw.rect(win, BLACK, variant_button3)
        variant_button3_text = FONT_BUTTON.render("Back", True, WHITE)
        win.blit(variant_button3_text, (variant_button3.x + variant_button3.width / 2 - variant_button3_text.get_width() / 2, variant_button3.y +
```

win.fill(WHITE): Đổ màu trắng (WHITE) lên cửa sổ (win) để làm nền cho menu.

win.blit(main_menu_image,(0, 0)): Vẽ hình ảnh main_menu_image lên cửa sổ tại vị trí (0, 0). Hình ảnh này có thể là hình nền cho menu.

title_text = FONT_TITLE.render("Chess Game", True, WHITE): Tạo một đối tượng văn bản (title_text) bằng cách sử dụng phương thức render() của đối tượng font chữ FONT_TITLE. Văn bản được hiển thị là "Chess Game", có màu chữ trắng (WHITE).

$$\text{win.blit(title\_text, (}\frac{\text{WIDTH}}{4} - \frac{\text{title\_text.get\_width()}}{2}\text{, 100))}$$

: Vẽ đối tượng văn bản title_text lên cửa sổ tại vị trí tính toán. Vị trí x là WIDTH / 4 - title_text.get_width() / 2 để căn giữa văn bản theo trục x. Vị trí y là 100.

Các điều kiện if kiểm tra biến button1_visible, button2_visible, button3_visible, variant_button1_visible, varian_button2_visible, variant_button3_visible để xác định xem các nút menu và nút biến thể có hiển thị hay không.

pygame.draw.rect(win, BLACK, button1): Vẽ một hình chữ nhật (pygame.Rect) lên cửa sổ win. Hình chữ nhật này có màu đen(BLACK) và được xác định bởi đối tượng button1.

button1_text = FONT_BUTTON.render("Play", True, WHITE): Tạo một đối tượng văn bản (button1_text) để hiển thị văn bản "Play" trên nút button1. Văn bản có màu chữ trắng(WHITE).

win.blit(button1_text, (button1.x + button1.width / 2 - button1_text.get_width() / 2, button1.y + 20)): Vẽ đối tượng văn bản button1_text lên cửa sổ tại vị trí tính toán. Vị trí x là button1.x + button1.width / 2 - button1_text.get_width() / 2 để căn giữa văn bản theo trục x. Vị trí y là button1.y + 20.

'Vẽ màn hình chờ

```
def draw_waiting_screen():
    win.fill(WHITE)
    waiting_text = FONT_TITLE.render("Waiting...", True, BLACK)
    win.blit(waiting_text, (WIDTH / 2 - waiting_text.get_width() / 2, HEIGHT / 2 - waiting_text.get_height() / 2))
```

win.fill(WHITE): Đổ màu trắng (WHITE) lên cửa sổ (win) để làm nền cho màn hình chờ.

waiting_text = FONT_TITLE.render("Waiting...", True, BLACK): Tạo một đối tượng văn bản (waiting_text) bằng cách sử dụng phương thức render() của đối tượng font chữ FONT_TITLE.

Văn bản được hiển thị là "Waiting...", có màu chữ đen (BLACK).

win.blit(waiting_text, (WIDTH / 2 - waiting_text.get_width() / 2, HEIGHT / 2 - waiting_text.get_height() / 2)): Vẽ đối tượng văn bản waiting_text lên cửa sổ tại vị trí tính toán. Vị trí x là WIDTH / 2 - waiting_text.get_width() / 2 để căn giữa văn bản theo trục x. Vị trí y là HEIGHT / 2 - waiting_text.get_height() / 2 để căn giữa văn bản theo trục y.

1.2 Xử lí sự kiện button

```
def handle_menu_click(pos):
    global current_screen, button1_visible, button2_visible, button3_visible, variant_button1_visible, variant_button2_visible, variant_button3_v
    if button1.collidepoint(pos) and button1_visible:
        main_game()
    elif button2.collidepoint(pos) and button2_visible:
        button1_visible = False
        button2_visible = False
        button3_visible = False
        variant_button1_visible = True
        variant_button2_visible = True
        variant_button3_visible = True
    elif button3.collidepoint(pos) and button3_visible:
        pygame.quit()
        sys.exit()
    elif variant_button1.collidepoint(pos) and variant_button1_visible:
        current_screen = "waiting"
        # Xử lý logic cho màn hình biến thể 1
    elif variant_button2.collidepoint(pos) and variant_button2_visible:
        current_screen = "waiting"
        # Xử lý logic cho màn hình biến thể 2
    elif variant_button3.collidepoint(pos) and variant_button3_visible:
        button1_visible = True
        button2_visible = True
        button3_visible = True
        variant_button1_visible = False
        variant_button2_visible = False
        variant_button3_visible = False
        current_screen = "menu"
```

global current_screen, button1_visible, button2_visible, button3_visible, variant_button1_visible, variant_button2_visible, variant_button3_visible: Khai báo các biến toàn cục để được sử dụng và thay đổi giá trị trong hàm.

Dòng if button1.collidepoint(pos) and button1_visible: kiểm tra xem người dùng đã nhấp chuột vào button1 và button1_visible đang được hiển thị. Nếu điều kiện đúng, gọi hàm main_game() để chuyển sang màn hình chơi chính.

Dòng elif button2.collidepoint(pos) and button2_visible: kiểm tra xem người dùng đã nhấp chuột vào button2 và button2_visible đang được hiển thị. Nếu điều kiện đúng, ẩn các nút button1, button2 và button3, và hiển thị các nút variant_button1, variant_button2 và variant_button3.

Dòng elif button3.collidepoint(pos) and button3_visible: kiểm tra xem người dùng đã nhấp chuột

vào button3 và button3_visible đang được hiển thị. Nếu điều kiện đúng, gọi hàm pygame.quit() để thoát khỏi trò chơi và sys.exit() để thoát khỏi chương trình.

Dòng elif variant_button1.collidepoint(pos) and variant_button1_visible: kiểm tra xem người dùng đã nhấp chuột vào variant_button1 và variant_button1_visible đang được hiển thị. Nếu điều kiện đúng, gán giá trị "waiting" cho biến current_screen và xử lý logic cho màn hình biến thể 1.

Dòng elif variant_button2.collidepoint(pos) and variant_button2_visible: kiểm tra xem người dùng đã nhấp chuột vào variant_button2 và variant_button2_visible đang được hiển thị. Nếu điều kiện đúng, gán giá trị "waiting" cho biến current_screen và xử lý logic cho màn hình biến thể 2.

Dòng elif variant_button3.collidepoint(pos) and variant_button3_visible: kiểm tra xem người dùng đã nhấp chuột vào variant_button3 và variant_button3_visible đang được hiển thị. Nếu điều kiện đúng, hiển thị lại các nút button1, button2 và button3, và ẩn các nút variant_button1, variant_button2 và variant_button3. Đồng thời, gán giá trị "menu" cho biến current_screen để quay lại màn hình menu.

2 Chế độ Chess Online

2.1 Giới thiệu

Chess là một trò chơi bàn phương thế giới có nguồn gốc từ Ấn Độ cổ đại và trở thành một trong những trò chơi bàn phương phổ biến nhất trên toàn thế giới. Nó thuộc thể loại trò chơi chiến lược, yêu cầu người chơi có khả năng tư duy chiến thuật, lập kế hoạch và sự sáng tạo.

Trò chơi Chess diễn ra trên một bàn cờ vuông gồm 64 ô, được chia thành 32 ô màu trắng và 32 ô màu đen. Mỗi người chơi điều khiển một bộ quân cờ gồm các quân cờ vua, hậu, xe, tượng, mã và tốt. Mục tiêu của trò chơi là bắt hoặc chiếm đóng quân vua của đối thủ, được gọi là "chiếu hết" (checkmate).

Mỗi quân cờ di chuyển theo các quy tắc cụ thể. Ví dụ, vua có thể di chuyển một ô theo bất kỳ hướng nào, xe di chuyển theo hàng ngang hoặc cột dọc, tượng di chuyển theo đường chéo, mã di chuyển theo đường L và tốt di chuyển về phía trước một ô. Quân hậu là quân mạnh nhất và có thể di chuyển theo bất kỳ hướng nào trên bàn cờ.

Trò chơi Chess yêu cầu người chơi có khả năng định hình chiến thuật, dự đoán và phản ứng nhanh. Nó kết hợp sự phân tích chi tiết và tính toán xa trước để đưa ra những nước đi tốt nhất. Các trận đấu Chess có thể diễn ra giữa hai người chơi hoặc giữa người chơi và máy tính.

Chess được coi là một môn thể thao tinh thần và là một trò chơi có giá trị giáo dục cao. Nó giúp cải thiện tư duy logic, khả năng giải quyết vấn đề, ý thức chiến thuật và kỹ năng quản lý thời gian. Ngoài ra, Chess cũng là một hoạt động giải trí thú vị và thách thức, thu hút hàng triệu người chơi trên toàn thế giới.

Để tối ưu hóa trò chơi và làm cho cách chơi trở nên dễ dàng hơn, phù hợp với mọi độ tuổi và thiết bị, nhóm em đã làm cho trò chơi với cách chơi đơn giản hơn. Điều này giúp người chơi dễ dàng tương tác với trò chơi một cách tự nhiên và thuận tiện, mang lại trải nghiệm chơi game tốt hơn và thú vị hơn.

2.2 Hướng dẫn cách chơi Chess

Trò chơi Chess có nhiều quy tắc và chi tiết, do đó hướng dẫn dưới đây chỉ tập trung vào các quy tắc cơ bản.

Bàn cờ: Chess được chơi trên một bàn cờ vuông gồm 64 ô, được chia thành 8 hàng (được đánh số từ 1 đến 8) và 8 cột (được đánh số từ a đến h). Ô ở góc bên trái dưới của bàn cờ là ô a1 và ô ở góc bên phải trên là ô h8.

Quân cờ: Mỗi người chơi bắt đầu với một bộ quân cờ gồm vua, hậu, xe, tượng, mã, và tốt. Người chơi di chuyển các quân cờ của mình để tấn công và bảo vệ.

Mục tiêu: Mục tiêu của trò chơi là chiếu hết (*checkmate*) quân vua của đối thủ. Điều này xảy ra khi quân vua của đối phương bị chiếu và không có bước đi nào để tránh bị chiếu.

Quy tắc di chuyển cơ bản:

- *Vua*: Di chuyển một ô theo bất kỳ hướng nào.
- *Hậu*: Di chuyển theo hàng ngang, cột dọc hoặc đường chéo.

- *Xe*: Di chuyển theo hàng ngang hoặc cột dọc.
- *Tượng*: Di chuyển theo đường chéo.
- *Mã*: Di chuyển theo đường chéo nhưng có thể nhảy qua quân cờ khác.
- *Tốt*: Di chuyển về phía trước một ô, nhưng có một số quy tắc đặc biệt cho nước đi ban đầu và việc bắt quân.

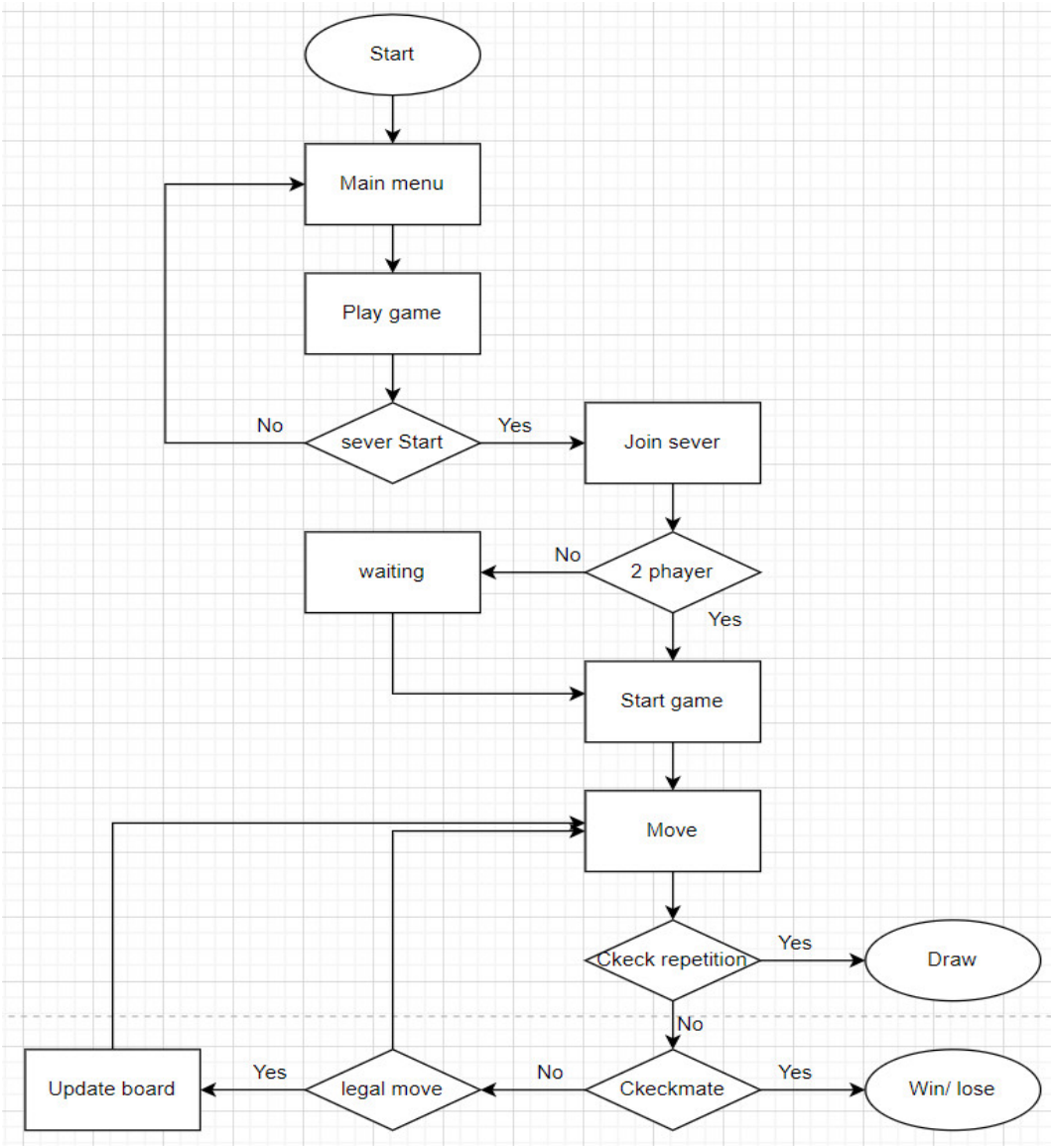
Nước đi và bắt quân:

- Người chơi lần lượt thực hiện nước đi của mình.
- Mỗi lượt, người chơi có thể di chuyển một quân cờ của mình theo quy tắc di chuyển của từng quân.
- Nếu quân cờ của bạn đứng trên ô mà có quân cờ của đối thủ, bạn có thể "bắt" quân cờ đó bằng cách di chuyển quân của bạn vào ô đó.
- Một số quân cờ có các quy tắc di chuyển đặc biệt, chẳng hạn như "nước đi hai ô" cho tốt trong lần di chuyển đầu tiên.

Chiếu và chiếu hết:

- *Chiếu*: Khi vua của bạn bị đối thủ "chiếu" (có thể bị bắt ở lượt tiếp theo), bạn phải di chuyển vua ra khỏi tình huống chiếu.
- *Chiếu hết*: Nếu vua của bạn bị chiếu và không còn bước đi hợp lệ nào để tránh chiếu, bạn bị "chiếu hết" và thua cuộc.

2.3 FlowChart



2.4 Thư viện cần dùng

import time, pygame,socket,pickle,os,sys

2.5 Thiết kế client

__init__: Phương thức khởi tạo lớp Network. Trong phương thức này, một đối tượng socket được tạo bằng cách sử dụng họ và cổng của máy chủ. Sau đó, phương thức connect() được gọi để thiết lập kết nối với máy chủ. Kết quả nhận được từ máy chủ sau khi kết nối được giải nén bằng module pickle và được lưu vào thuộc tính board.

```python
class Network:
    def __init__(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.host = "192.168.1.15"
        self.port = 5555
        self.addr = (self.host, self.port)
        self.board = self.connect()
        self.board = pickle.loads(self.board)

    def connect(self):
        self.client.connect(self.addr)
        return self.client.recv(4096*8)

    def disconnect(self):
        self.client.close()
```

```python
    def send(self, data, pick=False):
        """
        :param data: str
        :return: str
        """
        start_time = time.time()
        while time.time() - start_time < 5:
            try:
                if pick:
                    self.client.send(pickle.dumps(data))
                else:
                    self.client.send(str.encode(data))
                reply = self.client.recv(4096*8)
                try:
                    reply = pickle.loads(reply)
                    break
                except Exception as e:
                    print(e)

            except socket.error as e:
                print(e)


        return reply
```

connect(self): Phương thức này thực hiện kết nối với máy chủ thông qua đối tượng socket. Địa chỉ IP và cổng của máy chủ được lưu trong thuộc tính addr. Sau khi kết nối thành công, dữ liệu nhận được từ máy chủ được trả về.

disconnect(self): Phương thức này đóng kết nối socket với máy chủ bằng cách gọi phương thức close() trên đối tượng socket.

send(self, data, pick=False): Phương thức này dùng để gửi dữ liệu từ máy khách đến máy chủ thông qua kết nối socket. Tham số data là dữ liệu cần gửi dưới dạng chuỗi. Nếu tham số pick được đặt thành True, dữ liệu sẽ được đóng gói bằng module pickle trước khi gửi. Kết quả trả về từ máy chủ sau khi gửi dữ liệu cũng được giải nén bằng module pickle. Phương thức này sẽ chờ đợi trong vòng 5 giây để nhận phản hồi từ máy chủ.

2.6 Thiết kế bàn cờ

```
class Board:
    rect = (250, 50, 550, 550)
    startX = rect[0]
    startY = rect[1]
    def __init__(self, rows, cols):
        self.rows = rows
        self.cols = cols

        self.ready = False

        self.last = None

        self.copy = True

        self.board = [[0 for x in range(8)] for _ in range(rows)]

        self.board[0][0] = Rook(0, 0, "b")
        self.board[0][1] = Knight(0, 1, "b")
        self.board[0][2] = Bishop(0, 2, "b")
        self.board[0][3] = Queen(0, 3, "b")
        self.board[0][4] = King(0, 4, "b")
        self.board[0][5] = Bishop(0, 5, "b")
        self.board[0][6] = Knight(0, 6, "b")
        self.board[0][7] = Rook(0, 7, "b")

        self.board[1][0] = Pawn(1, 0, "b")
        self.board[1][1] = Pawn(1, 1, "b")
        self.board[1][2] = Pawn(1, 2, "b")
        self.board[1][3] = Pawn(1, 3, "b")
        self.board[1][4] = Pawn(1, 4, "b")
        self.board[1][5] = Pawn(1, 5, "b")
        self.board[1][6] = Pawn(1, 6, "b")
```

```
        self.board[1][7] = Pawn(1, 7, "b")
        self.board[7][0] = Rook(7, 0, "w")
        self.board[7][1] = Knight(7, 1, "w")
        self.board[7][2] = Bishop(7, 2, "w")
        self.board[7][3] = Queen(7, 3, "w")
        self.board[7][4] = King(7, 4, "w")
        self.board[7][5] = Bishop(7, 5, "w")
        self.board[7][6] = Knight(7, 6, "w")
        self.board[7][7] = Rook(7, 7, "w")
        self.board[6][0] = Pawn(6, 0, "w")
        self.board[6][1] = Pawn(6, 1, "w")
        self.board[6][2] = Pawn(6, 2, "w")
        self.board[6][3] = Pawn(6, 3, "w")
        self.board[6][4] = Pawn(6, 4, "w")
        self.board[6][5] = Pawn(6, 5, "w")
        self.board[6][6] = Pawn(6, 6, "w")
        self.board[6][7] = Pawn(6, 7, "w")
        self.p1Name = "Player 1"
        self.p2Name = "Player 2"
        self.turn = "w"
        self.time1 = 900
        self.time2 = 900
        self.storedTime1 = 0
        self.storedTime2 = 0
        self.winner = None
        self.startTime = time.time()
```

Ta định nghĩa một lớp Board. Trong đó **rect = (250, 50, 550, 550)**: Biến rect chứa một tuple đại diện cho hình chữ nhật giới hạn độ rộng và độ cao của bảng. Cụ thể, (250, 50, 550, 550) đại diện cho một hình chữ nhật có tọa độ góc trái trên là (250, 50) và kích thước là 550x550.

startX = rect[0] và startY = rect[1]: Biến startX và startY lưu trữ tọa độ x và y của góc trái trên của hình chữ nhật, được lấy từ rect. **__init__(self, rows, cols)**: Phương thức khởi tạo của lớp Board. Nhận vào hai tham số rows và cols đại diện cho số hàng và số cột của bảng.

self.ready, self.last, self.copy: Các biến instance dùng để lưu trạng thái của bảng.

self.board: Một danh sách hai chiều (rows hàng và cols cột) đại diện cho bảng. Ban đầu, tất cả các ô đều được đặt thành 0.

Các dòng tiếp theo của mã đặt các quân cờ vào vị trí ban đầu trên bảng. Ví dụ: self.board[0][0] = Rook(0, 0, "b") đặt một quân Rook (xe) màu đen vào ô đầu tiên trên cùng bên trái của bảng.

self.p1Name và self.p2Name: Lưu trữ tên của người chơi 1 và người chơi 2.

self.turn: Xác định lượt đi của người chơi. Ban đầu, lượt đi là "w" (white).

self.time1 và self.time2: Lưu trữ thời gian còn lại của người chơi 1 và người chơi 2.

self.storedTime1 và self.storedTime2: Lưu trữ thời gian đã được lưu trữ của người chơi 1 và người chơi 2.

self.winner: Lưu trữ thông tin người chiến thắng (nếu có).

self.startTime: Thời điểm bắt đầu trò chơi, được thiết lập bằng thời gian hiện tại khi đối tượng Board được khởi tạo.

2.6.1 Cập nhập di chuyển

```
def update_moves(self):
 for row in self.board:
    for piece in row:
        if piece:
            piece.update_valid_moves(self.board)
```

Dòng đầu tiên khai báo phương thức update_moves trong lớp Board. Vòng lặp đầu tiên for row in self.board: lặp qua từng hàng trong bảng.

Vòng lặp thứ hai for piece in row: lặp qua từng ô trong hàng. Câu lệnh điều kiện if piece: kiểm tra xem ô hiện tại có chứa một quân cờ hay không. Nếu có, điều kiện sẽ đúng và câu lệnh bên trong sẽ được thực thi.

piece.update_valid_moves(self.board) gọi phương thức update_valid_moves trên đối tượng piece (quân cờ). Phương thức này được sử dụng để cập nhật danh sách các nước đi hợp lệ cho quân cờ dựa trên trạng thái hiện tại của bảng. self.board được truyền vào phương thức để cung cấp thông tin về trạng thái hiện tại của bảng cho quân cờ.

2.6.2 Vẽ bàn cờ và quân cờ

```
def draw(self, win, color):
  if self.last and color == self.turn:
    y, x = self.last[0]
    y1, x1 = self.last[1]
    xx = (4 - x) + round(self.startX + (x * self.rect[2] / 8))
    yy = 3 + round(self.startY + (y * self.rect[3] / 8))
    pygame.draw.circle(win, (0, 0, 255), (xx + 32, yy + 30), 34, 4)
    xx1 = (4 - x) + round(self.startX + (x1 * self.rect[2] / 8))
    yy1 = 3 + round(self.startY + (y1 * self.rect[3] / 8))
    pygame.draw.circle(win, (0, 0, 255), (xx1 + 32, yy1 + 30), 34, 4)

  for row in self.board:
    for piece in row:
        if piece:
            piece.draw(win, color)
            if piece.isSelected:
                s = (row, piece)
```

Dòng đầu tiên khai báo phương thức draw trong lớp Board. Phương thức này nhận hai tham số là win (cửa sổ trò chơi) và color (màu sắc). Câu lệnh điều kiện if self.last and color == self.turn: kiểm tra xem có lượt chơi trước đó (self.last) và màu sắc hiện tại (color) có khớp với lượt chơi hiện tại (self.turn) hay không. Nếu điều kiện này đúng, câu lệnh bên trong sẽ được thực thi.

Dòng y, x = self.last[0] và y1, x1 = self.last[1] giải nén các tọa độ hàng và cột của lượt chơi trước đó từ self.last.

Dòng xx = (4 - x) + round(self.startX + (x * self.rect[2] / 8)) tính toán tọa độ x của quân cờ đã được đánh trước đó dựa trên x (cột) và self.startX (tọa độ x ban đầu của bảng).

Dòng yy = 3 + round(self.startY + (y * self.rect[3] / 8)) tính toán tọa độ y của quân cờ đã được đánh trước đó dựa trên y (hàng) và self.startY (tọa độ y ban đầu của bảng).

Dòng pygame.draw.circle(win, (0, 0, 255), (xx + 32, yy + 30), 34, 4) vẽ một hình tròn tại vị trí (xx + 32, yy + 30) trên cửa sổ trò chơi win. Hình tròn có bán kính 34 pixel, màu sắc là (0, 0, 255) (màu xanh lam) và độ dày của đường viền là 4 pixel.

Các dòng tương tự tiếp theo cũng vẽ một hình tròn tại vị trí (xx1 + 32, yy1 + 30) dựa trên tọa độ của lượt chơi thứ hai trong self.last. Tiếp theo, vòng lặp for row in self.board: lặp qua từng hàng trong bảng.

Trong vòng lặp, vòng lặp for piece in row: lặp qua từng ô trong hàng. Câu lệnh điều kiện if piece: kiểm tra xem ô hiện tại có chứa một quân cờ hay không.

piece.draw(win, color) gọi phương thức draw trên đối tượng piece (quân cờ) để vẽ quân cờ đó lên cửa sổ trò chơi win với màu sắc color. Nếu piece được chọn (piece.isSelected), dòng s = (row, piece) gán giá trị (row, piece) vào biến s.

2.6.3 Lấy danh sách nước đi nguy hiểm cho một màu cụ thể

```
def get_danger_moves(self, color):
    danger_moves = []
    for row in self.board:
        for piece in row:
            if piece and piece.color != color:
                danger_moves.extend(piece.move_list)
    return danger_moves
```

Dòng đầu tiên khai báo phương thức get_danger_moves trong lớp Board. Phương thức này nhận hai tham số là self và color.

Dòng thứ hai khởi tạo một danh sách rỗng danger_moves để lưu trữ các nước đi nguy hiểm.

Vòng lặp đầu tiên for row in self.board: lặp qua từng hàng trong bảng.

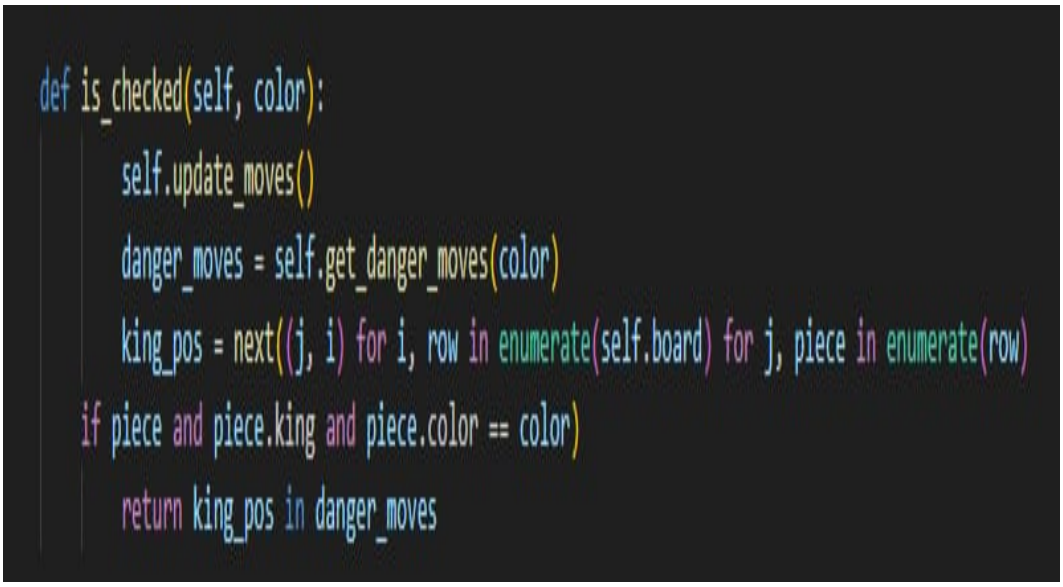
Vòng lặp thứ hai for piece in row: lặp qua từng ô trong hàng. Câu lệnh điều kiện if piece and piece.color != color: kiểm tra xem ô hiện tại có chứa một quân cờ (piece) và màu sắc của quân cờ khác với color hay không. Nếu điều kiện này đúng, câu lệnh bên trong sẽ được thực thi.

Dòng danger_moves.extend(piece.move_list) mở rộng danh sách danger_moves bằng cách thêm tất cả các nước đi trong piece.move_list (danh sách các nước đi hợp lệ của quân cờ) vào danh sách danger_moves.

Sau khi lặp qua tất cả các ô trên bảng, danh sách danger_moves chứa tất cả các nước đi nguy hiểm cho màu cụ thể (color).

Dòng cuối cùng return danger_moves trả về danh sách danger_moves như là kết quả của phương thức.

2.6.4 Kiểm tra màu bị chiếu



Dòng đầu tiên khai báo phương thức is_checked trong lớp Board. Phương thức này nhận hai tham số là self và color.

Dòng thứ hai gọi phương thức update_moves để cập nhật danh sách các nước đi của các quân cờ trên bảng.

Dòng thứ ba gọi phương thức get_danger_moves để lấy danh sách các nước đi nguy hiểm cho màu cụ thể (color). Kết quả được lưu trong biến danger_moves.

Dòng thứ tư sử dụng biểu thức generator để tìm vị trí của vua của màu cụ thể (color) trên bảng. Biểu thức generator này lặp qua từng hàng và từng ô trên bảng, và trả về tọa độ (j, i) (cột, hàng) đầu tiên mà tìm thấy một quân cờ là vua (piece.king) và có màu sắc là color.

Dòng thứ năm sử dụng câu lệnh return king_pos in danger_moves để kiểm tra xem vị trí của vua (king_pos) có nằm trong danh sách các nước đi nguy hiểm (danger_moves) hay không. Nếu vị trí của vua nằm trong danh sách nước đi nguy hiểm, tức là vua đó bị chiếu, phương thức sẽ trả về True. Ngược lại, phương thức sẽ trả về False.

2.6.5 Xử lý chọn ô trên bàn cờ

```
def select(self, col, row, color):
    changed = False
    prev = (-1, -1)

    for i in range(self.rows):
        for j in range(self.cols):
            if self.board[i][j] != 0 and self.board[i][j].selected:
                prev = (i, j)

    if self.board[row][col] == 0 and prev != (-1, -1):
        moves = self.board[prev[0]][prev[1]].move_list
        if (col, row) in moves:
            changed = self.move(prev, (row, col), color)
    else:
        if prev == (-1, -1):
            self.reset_selected()
            if self.board[row][col] != 0:
                self.board[row][col].selected = True
        else:
            if (
                self.board[prev[0]][prev[1]].color != self.board[row][col].color
                and (col, row) in self.board[prev[0]][prev[1]].move_list
            ):
                changed = self.move(prev, (row, col), color)
                if self.board[row][col].color == color:
                    self.board[row][col].selected = True
            else:
```

```
                if (
                    self.board[row][col].color == color
                    and not self.board[prev[0]][prev[1]].moved
                    and self.board[prev[0]][prev[1]].rook
                    and self.board[row][col].king
                    and col != prev[1]
                    and prev != (-1, -1)
                ):
                    castle = True
                    if prev[1] < col:
                        for j in range(prev[1] + 1, col):
                            if self.board[row][j] != 0:
                                castle = False
                        if castle:
                            changed = self.move(prev, (row, 3), color)
                            changed = self.move((row, col), (row, 2), color)
                        if not changed:
                            self.board[row][col].selected = True
                    else:
                        for j in range(col + 1, prev[1]):
                            if self.board[row][j] != 0:
                                castle = False
                        if castle:
                            changed = self.move(prev, (row, 6), color)
                            changed = self.move((row, col), (row, 5), color)
                        if not changed:
                            self.board[row][col].selected = True
                else:
                    self.board[row][col].selected = True
```

```
if changed:
    if self.turn == "w":
        self.turn = "b"
        self.reset_selected()
    else:
        self.turn = "w"
        self.reset_selected()
```

Dòng đầu tiên khai báo phương thức select trong lớp Board. Phương thức này nhận ba tham số là self, col, row và color.

Dòng thứ hai khởi tạo một biến changed với giá trị ban đầu là False.

Dòng thứ tư và thứ năm lặp qua tất cả các ô trên bảng để tìm ô trước đó đã được chọn (nếu có). Ô trước đó được lưu trữ trong biến prev dưới dạng một cặp giá trị (i, j) tương ứng với hàng

và cột của ô trước đó.

Câu lệnh điều kiện tiếp theo kiểm tra xem ô hiện tại có được chọn không (self.board[row][col] == 0) và có tồn tại một ô trước đó đã được chọn (prev != (-1, -1)) hay không. Nếu điều kiện này đúng, câu lệnh bên trong sẽ được thực thi.

Dòng thứ tám lấy danh sách các nước đi của quân cờ được chọn từ ô trước đó

(self.board[prev[0]][prev[1]].move_list) và kiểm tra xem (col, row) có nằm trong danh sách nước đi hay không ((col, row) in moves). Nếu điều kiện này đúng, phương thức thực hiện di chuyển từ ô trước đó đến ô hiện tại bằng cách gọi phương thức move và gán giá trị True cho biến changed.

Nếu điều kiện ở dòng thứ tám không đúng, phương thức tiếp tục xử lý các trường hợp khác.

Câu lệnh điều kiện tiếp theo kiểm tra xem có tồn tại ô trước đó đã được chọn hay không (prev == (-1, -1)). Nếu không có ô trước đó được chọn, phương thức đặt lại trạng thái chọn cho tất cả các ô trên bảng (self.reset_selected()) và kiểm tra xem ô hiện tại có chứa một quân cờ không (self.board[row][col] != 0). Nếu ô hiện tại chứa một quân cờ, quân cờ đó được đánh dấu là đã được chọn (self.board[row][col].selected = True).

Nếu điều kiện ở dòng thứ mười không đúng, phương thức tiếp tục xử lý các trường hợp khác.

Câu lệnh điều kiện tiếp theo kiểm tra xem màu sắc của quân cờ ở ô trước đó khác với màu sắc của quân cờ ở ô hiện tại (self.board[prev[0]][prev[1]].color != self.board[row][col].color) và (col, row) nằm trong danh sách các nước đi của quân cờ ở ô trước đó ((col, row)

in self.board[prev[0]][prev[1]].move_list). Nếu điều kiện này đúng, phương thức thực hiện di chuyển từ ô trước đó đến ô hiện tại bằng cách gọi phương thức move và gán giá trị True cho biến changed. Nếu quân cờ ở ô hiện tại có cùng mMàu (self.board[row][col].color == color), quân cờ ở ô hiện tại được đánh dấu là đã được chọn (self.board[row][col].selected = True).

Nếu điều kiện ở dòng thứ mười không đúng, phương thức tiếp tục xử lý các trường hợp khác.

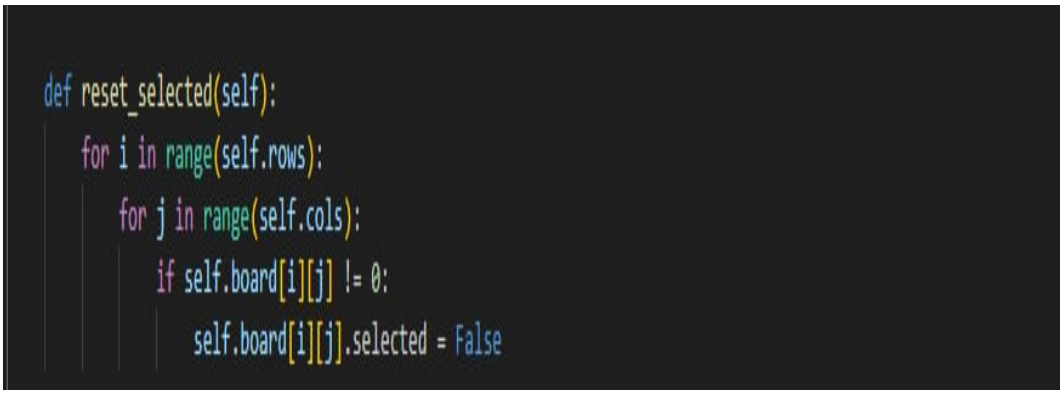
Câu lệnh điều kiện tiếp theo kiểm tra xem màu sắc của quân cờ ở ô hiện tại là màu cụ thể (self.board[row][col].color == color), quân cờ ở ô trước đó chưa được di chuyển

(not self.board[prev[0]][prev[1]].moved), quân cờ ở ô trước đó là một xe

(self.board[prev[0]][prev[1]].rook) và quân cờ ở ô hiện tại là một vua (self.board[row][col].king). Nếu các điều kiện này đúng, phương thức kiểm tra các điều kiện khác để xác định xem di chuyển là một nước đi đặc biệt như "castle" hay không. Nếu là "castle", phương thức thực hiện di chuyển tương ứng và gán giá trị True cho biến changed. Nếu không phải, quân cờ ở ô hiện tại được đánh dấu là đã được chọn (self.board[row][col].selected = True).

Cuối cùng, phương thức kiểm tra xem nếu changed được đặt thành True, tức là di chuyển đã được thực hiện thành công, thì nó thực hiện các bước tiếp theo như thay đổi lượt chơi (self.turn) và đặt lại trạng thái chọn (self.reset_selected()).

2.6.6 Đặt lại trạng thái quân cờ



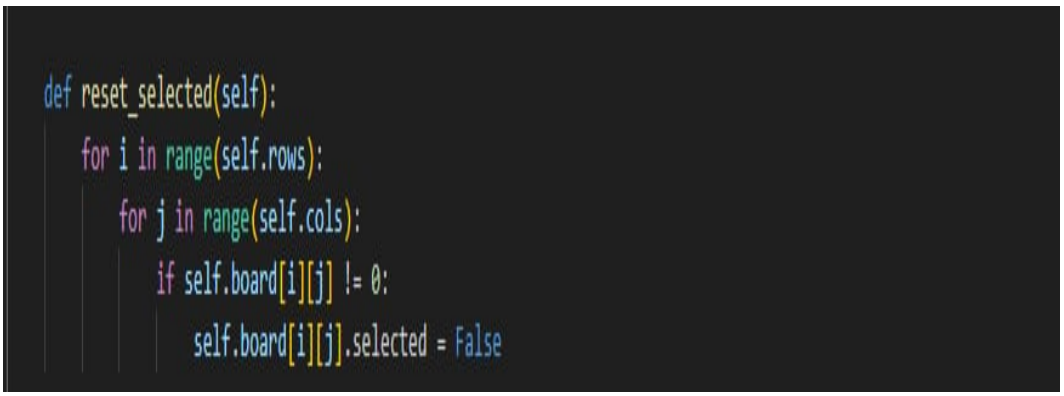
Dòng đầu tiên khai báo phương thức reset_selected trong lớp Board. Phương thức này không nhận bất kỳ tham số nào ngoài self.

Dòng thứ hai và thứ ba lặp qua tất cả các ô trên bảng bằng hai vòng lặp for lồng nhau.

Câu lệnh điều kiện trong vòng lặp kiểm tra xem ô hiện tại (self.board[i][j]) có chứa một quân cờ hay không (self.board[i][j] != 0). Nếu ô hiện tại chứa một quân cờ, câu lệnh bên trong sẽ được thực thi.

Trong câu lệnh bên trong, giá trị selected của quân cờ tại ô hiện tại được đặt thành False để đánh dấu rằng quân cờ không được chọn.

2.6.7 Kiểm tra bị chiếu



Dòng đầu tiên khai báo phương thức check_mate trong lớp Board. Phương thức này nhận hai tham số là self và color, trong đó color đại diện cho màu cờ cần kiểm tra.

Dòng thứ hai kiểm tra xem màu cờ được kiểm tra có đang bị chiếu (self.is_checked(color)) hay không. Nếu màu cờ đang bị chiếu, câu lệnh bên trong sẽ được thực thi.

Trong câu lệnh bên trong, biến king được gán giá trị là quân cờ vua có màu cần kiểm tra (piece and piece.king and piece.color == color). Đây là một cách để tìm quân cờ vua của màu cờ được kiểm tra trong bảng cờ. Nếu không có quân cờ vua thỏa mãn điều kiện, king sẽ có giá trị None.

Câu lệnh tiếp theo valid_moves = king.valid_moves(self.board) lấy danh sách các nước đi hợp lệ của quân cờ vua (king) từ phương thức valid_moves của quân cờ. Các nước đi này là những nước mà quân cờ vua có thể di chuyển đến trên bảng cờ hiện tại (self.board).

Câu lệnh danger_moves = self.get_danger_moves(color) lấy danh sách các nước đi tiềm năng của đối thủ (color) từ phương thức get_danger_moves. Đây là các nước đi mà đối thủ có thể chiếu quân cờ vua.

Cuối cùng, câu lệnh return len(valid_moves) $>$ 0 and all(move in danger_moves for move in valid_moves) kiểm tra xem có tồn tại ít nhất một nước đi hợp lệ cho quân cờ vua (len(valid_moves) $>$ 0) và tất cả các nước đi hợp lệ đó đều nằm trong danh sách các nước đi tiềm năng của đối thủ (all(move in danger_moves for move in valid_moves)). Nếu cả hai điều kiện này đều đúng, tức là quân cờ vua không thể tránh được bị chiếu và không có nước đi hợp lệ để thoát khỏi chiếu, phương thức trả về giá trị True, ngược lại sẽ trả về False.

2.6.8 Di chuyển

```
def move(self, start, end, color):
    checkedBefore = self.is_checked(color)
    changed = True
    nBoard = self.board[:]
    if nBoard[start[0]][start[1]].pawn:
        nBoard[start[0]][start[1]].first = False

    nBoard[start[0]][start[1]].change_pos((end[0], end[1]))
    nBoard[end[0]][end[1]] = nBoard[start[0]][start[1]]
    nBoard[start[0]][start[1]] = 0
    self.board = nBoard

    if self.is_checked(color) or (checkedBefore and self.is_checked(color)):
        changed = False
        nBoard = self.board[:]
        if nBoard[end[0]][end[1]].pawn:
            nBoard[end[0]][end[1]].first = True

        nBoard[end[0]][end[1]].change_pos((start[0], start[1]))
        nBoard[start[0]][start[1]] = nBoard[end[0]][end[1]]
        nBoard[end[0]][end[1]] = 0
        self.board = nBoard
    else:
        self.reset_selected()
```

Dòng đầu tiên khai báo phương thức move trong lớp Board. Phương thức này nhận ba tham số là self, start, end và color. start và end là tọa độ (dòng, cột) của ô ban đầu và ô đích của nước đi, color đại diện cho màu cờ của người chơi thực hiện nước đi.

Dòng thứ hai khởi tạo biến checkedBefore để lưu trữ trạng thái "bị chiếu" trước khi thực hiện nước đi. Biến này sẽ được sử dụng để kiểm tra xem sau khi thực hiện nước đi có gây chiếu không.

Dòng thứ ba khởi tạo biến changed với giá trị ban đầu là True. Biến này sẽ được sử dụng để đánh dấu xem nước đi đã thay đổi trạng thái của bảng cờ hay không.

Dòng thứ tư tạo một bản sao của self.board và gán cho nBoard. Bản sao này sẽ được sử dụng để thực hiện các thay đổi trong quá trình di chuyển quân cờ.

Dòng thứ năm kiểm tra xem quân cờ ở ô ban đầu có phải là quân cờ tốt (pawn) hay không. Nếu là quân cờ tốt, thuộc tính first của nó được đặt thành False. Thuộc tính first được sử dụng để đánh dấu xem quân cờ tốt đã di chuyển lần đầu tiên hay chưa.

Dòng thứ sáu và thứ bảy di chuyển quân cờ từ ô ban đầu (start) đến ô đích (end). Đầu tiên, phương thức change_pos của quân cờ tại ô ban đầu được gọi để cập nhật vị trí mới. Sau đó, quân cờ tại ô đích được gán bằng quân cờ tại ô ban đầu và ô ban đầu được đặt thành giá trị 0 để chỉ trống.

Dòng thứ tám kiểm tra xem sau khi thực hiện nước đi có gây chiếu (self.is_checked(color)) hoặc đã bị chiếu trước đó và vẫn bị chiếu (checkedBefore and self.is_checked(color)) hay không. Nếu điều kiện này đúng, tức là nước đi không hợp lệ vì vua của màu cờ đang bị chiếu hoặc vua đã bị chiếu trước đó và vẫn bị chiếu sau nước đi, biến changed được đặt thành False.

Trong câu lệnh bên trong, bản sao của self.board được lưu vào nBoard. Nếu quân cờ tại ô đích là quân cờ tốt (nBoard[end[0]][end[1]].pawn), thuộc tính first của nó được đặt thành True để đánh dấu là quân cờ tốt đã di chuyển lần đầu tiên.

Dòng thứ 15 và 16 đảo ngược lại các thay đổi trên bảng cờ nếu nước đi không hợp lệ. Tức là quân cờ được di chuyển từ ô đích (end) trở lại ô ban đầu (start), và nếu quân cờ tại ô đích là quân cờ tốt, thuộc tính first của nó sẽ được đặt lại thành True.

Dòng thứ 18 được thực hiện nếu nước đi hợp lệ. Trong trường hợp này, phương thức reset_selected được gọi để đặt lại trạng thái chọn trên bảng cờ.

Dòng thứ 20 và 21 cập nhật danh sách các nước đi cho các quân cờ trên bảng cờ thông qua phương thức update_moves. Dòng thứ 23 đánh dấu nước đi cuối cùng (self.last) bằng cặp tọa độ (start, end) của nước đi.

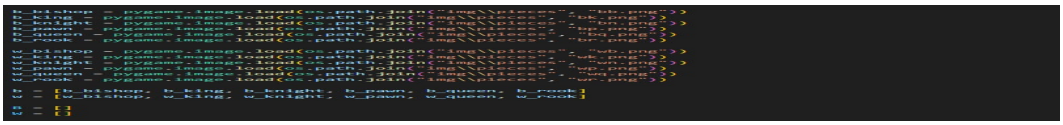
Dòng thứ 24 và 25 cập nhật thời gian đã trôi qua (storedTime1 hoặc storedTime2) dựa trên màu cờ của người chơi thực hiện nước đi và thời gian trôi qua từ lần chơi trước đến hiện tại.

Dòng thứ 26 đặt thời gian bắt đầu (startTime) cho lượt chơi tiếp theo.

Cuối cùng, phương thức trả về giá trị của biến changed, đại diện cho thành công hay thất bại của nước đi.

2.7 Thiết kế quân cờ

2.7.1 Hiển thị quân cờ



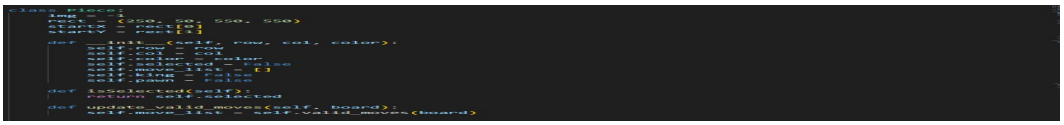
Các biến b_bishop, b_king, b_knight, b_pawn, b_queen, b_rook chứa hình ảnh của các quân cờ màu đen.

Các biến w_bishop, w_king, w_knight, w_pawn, w_queen, w_rook chứa hình ảnh của các quân cờ màu trắng.

Các biến b và w là danh sách chứa các hình ảnh của các quân cờ màu đen và màu trắng tương ứng.

Các danh sách B và W được khởi tạo rỗng và có thể được sử dụng để lưu trữ các hình ảnh của các quân cờ màu đen và màu trắng trong bàn cờ.

2.7.2 Khởi tạo lớp Piece



img, rect, startX, startY là các thuộc tính của lớp Piece được khởi tạo với một số giá trị mặc định.

Phương thức __init__ được sử dụng để khởi tạo một đối tượng Piece. Nó nhận ba tham số là row, col, color để chỉ định hàng, cột và màu sắc của quân cờ.

selected là một thuộc tính đánh dấu xem quân cờ có được chọn hay không.

move_list là một danh sách chứa các nước đi hợp lệ của quân cờ. king và pawn là các thuộc tính đánh dấu xem quân cờ là quân vua hay quân tốt.

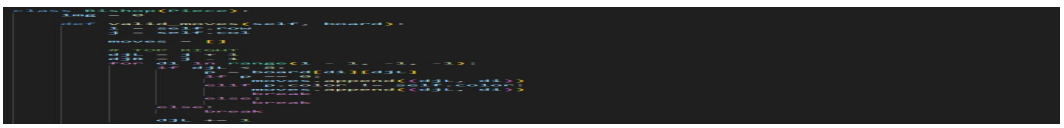
isSelected là một phương thức trả về giá trị của thuộc tính selected. update_valid_moves là một phương thức để cập nhật danh sách các nước đi hợp lệ của quân cờ dựa trên trạng thái hiện tại của bàn cờ.

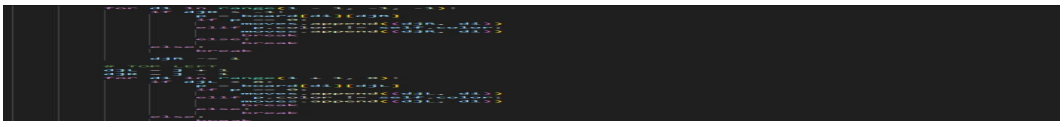
draw là một phương thức để vẽ quân cờ lên màn hình. Nó nhận tham số win là cửa sổ Pygame và color là màu sắc của người chơi. Phương thức này sẽ vẽ hình ảnh của quân cờ tại vị trí phù hợp trên màn hình và vẽ một hình chữ nhật đỏ xung quanh quân cờ nếu nó được chọn.

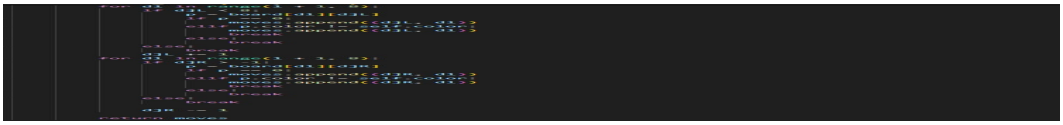
change_pos là một phương thức để thay đổi vị trí của quân cờ. Nó nhận tham số pos là một cặp tọa độ (hàng, cột) và cập nhật thuộc tính row và col của quân cờ.

__str__ là một phương thức trả về một chuỗi biểu diễn của quân cờ, bao gồm cột và hàng của nó.

2.7.3 Lớp Bishop và cách di chuyển







img = 0 xác định chỉ số hình ảnh của quân tượng trong danh sách B hoặc W.

valid_moves là một phương thức để tìm các nước đi hợp lệ của quân tượng dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân tượng có thể di chuyển đến.

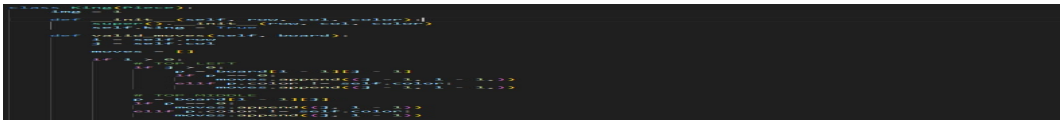
Phương thức valid_moves sử dụng một vòng lặp và kiểm tra các ô trên đường chéo của quân tượng theo các hướng: top right, top left.

Đối với mỗi ô trên đường chéo, nếu ô đó trống, quân tượng có thể di chuyển đến đó. Nếu ô đó chứa một quân cờ của màu khác, quân tượng có thể di chuyển đến đó và sau đó dừng lại.

Nếu ô đó chứa một quân cờ cùng màu, quân tượng không thể di chuyển đến đó và vòng lặp dừng lại.

Cuối cùng, danh sách các nước đi hợp lệ của quân tượng được trả về.

2.7.4 Lớp King và cách di chuyển



img = 1 xác định chỉ số hình ảnh của quân vua trong danh sách B hoặc W.

Phương thức __init__ được gọi khi một đối tượng King được khởi tạo. Nó gọi phương thức __init__ của lớp cha Piece và đặt thuộc tính king thành True.

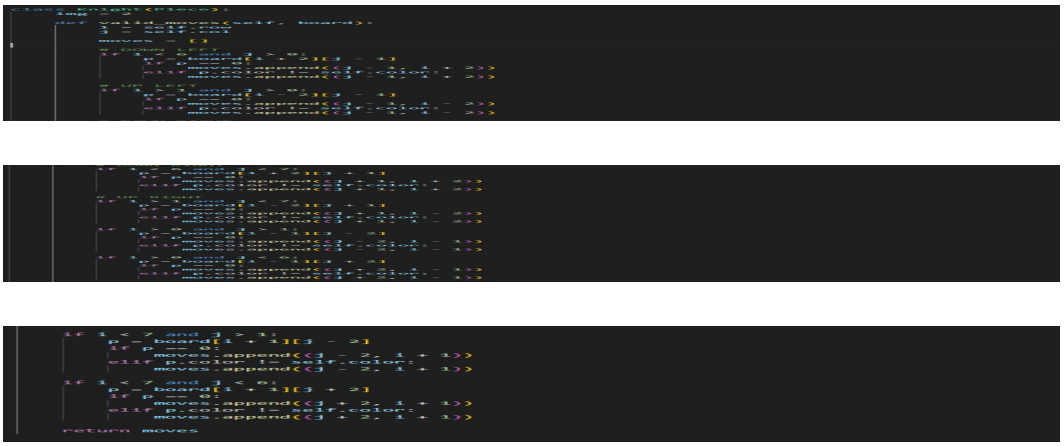
valid_moves là một phương thức để tìm các nước đi hợp lệ của quân vua dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân vua có thể di chuyển đến.

Phương thức valid_moves kiểm tra các ô xung quanh quân vua, bao gồm các ô đường chéo và các ô hàng ngang, dọc.

Đối với mỗi ô, nếu ô đó trống, quân vua có thể di chuyển đến đó. Nếu ô đó chứa một quân cờ của màu khác, quân vua có thể di chuyển đến đó.

Cuối cùng, danh sách các nước đi hợp lệ của quân vua được trả về.

2.7.5 Lớp Knight và cách di chuyển



img = 2 xác định chỉ số hình ảnh của quân mã trong danh sách B hoặc W.

Phương thức valid_moves là một phương thức để tìm các nước đi hợp lệ của quân mã dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân mã có thể di chuyển đến.

Phương thức valid_moves xác định các ô mà quân mã có thể di chuyển đến bằng cách kiểm tra các ô theo các hướng: up left, up right, left up, left down, right up, right down, down left, down right.

Đối với mỗi ô, nếu ô đó trống, quân mã có thể di chuyển đến đó. Nếu ô đó chứa một quân cờ của màu khác, quân mã có thể di chuyển đến đó. Cuối cùng, danh sách các nước đi hợp lệ của quân mã được trả về.

2.7.6 Lớp Pawn và cách di chuyển

img = 3 xác định chỉ số hình ảnh của quân tốt trong danh sách B hoặc W. Phương thức __init__ được gọi khi một đối tượng Pawn được khởi tạo. Nó gọi phương thức __init__ của lớp cha Piece và đặt các thuộc tính first thành True, queen thành False, và pawn thành True. Các thuộc tính này sẽ được sử dụng để theo dõi trạng thái của quân tốt.

Phương thức valid_moves là một phương thức để tìm các nước đi hợp lệ của quân tốt dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân tốt có thể di chuyển đến.

Phương thức valid_moves xác định các ô mà quân tốt có thể di chuyển đến bằng cách kiểm tra các ô theo các hướng khác nhau, tùy thuộc vào màu của quân tốt.

Đối với quân tốt màu đen (self.color == "b"), các ô kiểm tra bao gồm ô phía trước, các ô chéo để ăn quân đối phương, và ô phía trước 2 ô trong lần di chuyển đầu tiên.

Đối với quân tốt màu trắng (self.color != "b"), các ô kiểm tra bao gồm ô phía trên, các ô chéo để ăn quân đối phương, và ô phía trên 2 ô trong lần di chuyển đầu tiên.

Cuối cùng, danh sách các nước đi hợp lệ của quân tốt được trả về.

2.7.7 Lớp Queen và cách di chuyển

img = 4 xác định chỉ số hình ảnh của quân hậu trong danh sách B hoặc W.

Phương thức valid_moves là một phương thức để tìm các nước đi hợp lệ của quân hậu dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân hậu có thể di chuyển đến.

Phương thức valid_moves kiểm tra các ô theo các hướng khác nhau, tùy thuộc vào vị trí hiện tại của quân hậu.

Các hướng di chuyển kiểm tra bao gồm: TOP RIGHT, TOP LEFT, UP, DOWN, LEFT, và RIGHT.

Đối với mỗi hướng, quân hậu di chuyển từ ô hiện tại theo hướng đó cho đến khi gặp biên bàn cờ hoặc gặp quân đối phương.

Nếu quân hậu gặp ô trống, nó được thêm vào danh sách các nước đi hợp lệ.

Nếu quân hậu gặp quân đối phương, nó được thêm vào danh sách các nước đi hợp lệ, và vòng lặp dừng.

Cuối cùng, danh sách các nước đi hợp lệ của quân hậu được trả về.

2.7.8 Lớp Rook và cách di chuyển

```
class Queen(Piece):
    img = 4

    def valid_moves(self, board):
        i = self.row
        j = self.col

        moves = []

        # TOP RIGHT
        djL = j + 1
        djR = j - 1
        for di in range(i - 1, -1, -1):
            if djL < 8:
                p = board[di][djL]
                if p == 0:
                    moves.append((djL, di))
                elif p.color != self.color:
                    moves.append((djL, di))
                    break
                else:
                    djL = 9

                djL += 1
```

```
            for di in range(i - 1, -1, -1):
                if djR > -1:
                    p = board[di][djR]
                    if p == 0:
                        moves.append((djR, di))
                    elif p.color != self.color:
                        moves.append((djR, di))
                        break
                    else:
                        djR = -1

                djR -= 1

            # TOP LEFT
            djL = j + 1
            djR = j - 1
            for di in range(i + 1, 8):
                if djL < 8:
                    p = board[di][djL]
                    if p == 0:
                        moves.append((djL, di))
                    elif p.color != self.color:
                        moves.append((djL, di))
                        break
                    else:
                        djL = 9
                djL += 1
```

img = 5 xác định chỉ số hình ảnh của quân xe trong danh sách B hoặc W. Phương thức valid_moves là một phương thức để tìm các nước đi hợp lệ của quân xe dựa trên trạng thái hiện tại của bàn cờ. Phương thức này trả về một danh sách các cặp tọa độ (cột, hàng) biểu thị các ô mà quân xe có thể di chuyển đến.

Phương thức valid_moves kiểm tra các ô theo các hướng khác nhau, tùy thuộc vào vị trí hiện tại của quân xe.

Các hướng di chuyển kiểm tra bao gồm: UP, DOWN, LEFT và RIGHT. Đối với mỗi hướng, quân xe di chuyển từ ô hiện tại theo hướng đó cho đến khi gặp biên bàn cờ hoặc gặp quân đối phương.

Nếu quân xe gặp ô trống, nó được thêm vào danh sách các nước đi hợp lệ.

Nếu quân xe gặp quân đối phương, nó được thêm vào danh sách các nước đi hợp lệ, và vòng lặp dừng.

Cuối cùng, danh sách các nước đi hợp lệ của quân xe được trả về.

2.8 Thiết kế server

2.8.1 Khởi tạo socket và các biến

```
import socket
from _thread import *
from board import Board
import pickle
import time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server = "192.168.1.15"
port = 5555

server_ip = socket.gethostbyname(server)

try:
    s.bind((server, port))

except socket.error as e:
    print(str(e))

s.listen()
print("[START] Waiting for a connection")

connections = 0

games = {0:Board(8, 8)}

spectartor_ids = []
specs = 0
```

Dòng s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) tạo một socket mới và gán cho biến s.

server = "192.168.1.15" xác định địa chỉ IP của máy chủ. port = 5555 xác định số cổng để lắng nghe kết nối.

server_ip = socket.gethostbyname(server) lấy địa chỉ IP tương ứng với tên miền hoặc địa chỉ IP của máy chủ.

Tiếp theo, trong khối try, s.bind((server, port)) gắn kết socket tới địa chỉ IP và cổng đã xác định.

Nếu gặp lỗi trong quá trình gắn kết, thông báo lỗi sẽ được in ra màn hình.

Dòng s.listen() cho phép socket lắng nghe kết nối từ các client.

Dòng print("[START] Waiting for a connection") in ra thông báo chờ kết nối từ client.

connections = 0 khởi tạo biến connections để đếm số lượng kết nối. games = 0:Board(8, 8) tạo một từ điển games với khóa là 0 và giá trị là một đối tượng Board có kích thước 8x8.

spectartor_ids = [] khởi tạo một danh sách rỗng spectator_ids để lưu trữ danh sách các khán giả.

specs = 0 khởi tạo biến specs để đếm số lượng khán giả.

2.8.2 Đọc danh sách

```
def read_specs():
    global spectartor_ids

    spectartor_ids = []
    try:
        with open("specs.txt", "r") as f:
            for line in f:
                spectartor_ids.append(line.strip())
    except:
        print("[ERROR] No specs.txt file found, creating one...")
        open("specs.txt", "w")
```

global spectartor_ids cho phép hàm truy cập và thay đổi biến toàn cục spectator_ids.

spectartor_ids = [] gán danh sách rỗng cho biến spectator_ids để làm mới danh sách khán giả.

Trong khối try, hàm mở tệp tin "specs.txt" với chế độ chỉ đọc ("r") và gán cho biến f.

Dùng vòng lặp for để lặp qua từng dòng trong tệp tin f. Trong mỗi vòng lặp, line.strip() được sử dụng để loại bỏ các ký tự trống (khoảng trắng, dấu xuống dòng) từ đầu và cuối dòng và sau đó, dòng được thêm vào danh sách spectator_ids.

Nếu có lỗi xảy ra trong quá trình đọc tệp tin (ví dụ: tệp tin không tồn tại), một thông báo lỗi sẽ được in ra màn hình.

Nếu không tìm thấy tệp tin "specs.txt", thông báo lỗi sẽ được in ra màn hình và một tệp tin mới có tên "specs.txt" sẽ được tạo.

2.8.3 Xử lý kết nối

```python
def threaded_client(conn, game, spec=False):
    global pos, games, currentId, connections, specs

    if not spec:
        name = None
        bo = games[game]

        if connections % 2 == 0:
            currentId = "w"
        else:
            currentId = "b"

        bo.start_user = currentId

        # Pickle the object and send it to the server
        data_string = pickle.dumps(bo)

        if currentId == "b":
            bo.ready = True
            bo.startTime = time.time()

        conn.send(data_string)
        connections += 1
```

global pos, games, currentId, connections, specs cho phép hàm truy cập và thay đổi các biến toàn cục pos, games, currentId, connections, và specs.

Nếu spec là False, đồng nghĩa với việc kết nối đến từ một client tham gia trò chơi.

Biến name được khởi tạo với giá trị None.

Biến bo được gán bằng đối tượng Board tương ứng với trò chơi hiện tại trong từ điển games.

Dựa vào số lượng kết nối hiện tại (connections), biến currentId được gán giá trị "w" hoặc "b" để xác định bên chơi hiện tại.

Trạng thái khởi đầu của đối tượng bo được cập nhật thông qua các thuộc tính start_user, ready, và startTime.

Đối tượng bo được chuyển đổi thành một chuỗi dữ liệu bằng cách sử dụng pickle và gửi đến client qua kết nối conn.

Trong vòng lặp vô hạn, hàm lắng nghe và xử lý dữ liệu từ client. Nếu không nhận được dữ liệu (not d), vòng lặp sẽ kết thúc.

Nếu nhận được dữ liệu, dữ liệu được giải mã và kiểm tra để xác định hành động cần thực hiện.

Các hành động bao gồm: lựa chọn ô trên bàn cờ (select), thông báo người chiến thắng (winner), cập nhật các nước đi (update moves), và gửi tên người chơi (name).

Đối tượng bo được cập nhật sau mỗi hành động và được gửi lại cho client.

Khi vòng lặp kết thúc, số lượng kết nối giảm đi 1 và trò chơi được xóa khỏi từ điển games. Thông báo về việc người chơi rời khỏi trò chơi và kết nối được đóng.

Nếu spec là True, đồng nghĩa với việc kết nối đến từ một khán giả. Danh sách các trò chơi khả dụng được lấy từ danh sách khóa trong từ điển games.

Biến game_ind được khởi tạo với giá trị 0 để chỉ định trò chơi đầu tiên trong danh sách.

Đối tượng bo được gán bằng đối tượng Board tương ứng với trò chơi hiện tại trong danh sách trò chơi khả dụng.

Đối tượng bo được chuyển đổi thành một chuỗi dữ liệu bằng cách sử dụng pickle và gửi đến khán giả qua kết nối conn.

Trong vòng lặp vô hạn, hàm lắng nghe và xử lý dữ liệu từ khán giả. Nếu không nhận được dữ liệu (not d), vòng lặp sẽ kết thúc.

Nếu nhận được dữ liệu, dữ liệu được giải mã và kiểm tra để xác định hành động cần thực hiện.

Hành động bao gồm di chuyển qua các trò chơi tiếp theo (forward)và quay lại các trò chơi trước đó (back).

Đối tượng bo được cập nhật sau mỗi hành động và được gửi lại cho khán giả.

Khi vòng lặp kết thúc, thông báo về việc khán giả rời khỏi trò chơi và kết nối được đóng. Số lượng khán giả giảm đi 1.

2.8.4 kết nối client

```
while True:
    read_specs()
    if connections < 6:
        conn, addr = s.accept()
        spec = False
        g = -1
        print("[CONNECT] New connection")

        for game in games.keys():
            if games[game].ready == False:
                g=game

        if g == -1:
            try:
                g = list(games.keys())[-1]+1
                games[g] = Board(8,8)
            except:
                g = 0
                games[g] = Board(8,8)

        '''if addr[0] in spectartor_ids and specs == 0:
            spec = True
            print("[SPECTATOR DATA] Games to view: ")
            print("[SPECTATOR DATA]", games.keys())
            g = 0
            specs += 1'''

        print("[DATA] Number of Connections:", connections+1)
        print("[DATA] Number of Games:", len(games))

        start_new_thread(threaded_client, (conn,g,spec))
```

Trong vòng lặp vô hạn while True:, có các bước sau:

read_specs() là một hàm được gọi để đọc thông tin về khán giả. Nếu số lượng kết nối (connections) nhỏ hơn 6, tiến hành chấp nhận một kết nối mới từ client.

Hàm s.accept() được gọi để chấp nhận kết nối từ client và trả về một đối tượng conn và địa chỉ addr.

Biến spec được gán giá trị False để xác định rằng kết nối đến từ một client chứ không phải khán giả.

Biến g được khởi tạo với giá trị -1 để xác định trò chơi mà client sẽ tham gia.

Thông báo về việc có một kết nối mới được in ra màn hình.

Vòng lặp for duyệt qua các trò chơi trong từ điển games để tìm trò chơi chưa sẵn sàng (ready = False).

Nếu tìm thấy trò chơi chưa sẵn sàng, biến g được gán giá trị của trò chơi đó.

Nếu không tìm thấy trò chơi chưa sẵn sàng, một trò chơi mới được tạo ra.

Nếu danh sách khóa của từ điển games không rỗng, g được gán giá trị là khóa cuối cùng trong danh sách tăng thêm 1. Ví dụ: Nếu danh sách khóa là [0, 1, 2], thì g sẽ được gán giá trị là 3.

Nếu danh sách khóa của từ điển games là rỗng, g được gán giá trị là 0.

In ra màn hình thông tin về số lượng kết nối hiện tại (connections+1) và số lượng trò chơi (len(games)).

Sử dụng hàm start_new_thread() để tạo một luồng mới và chạy hàm threaded_client() để xử lý kết nối từ client hoặc khán giả mới. Đối số conn là đối tượng kết nối, g là trò chơi được gán cho client hoặc khán giả, và spec chỉ định xem kết nối đến từ client hay khán giả.

2.9 Thiết kế chính

2.9.1 Kiểm tra install

```
def install(package):
    subprocess.call([sys.executable, "-m", "pip", "install", package])

try:
    print("[GAME] Trying to import pygame")
    import pygame
except:
    print("[EXCEPTION] Pygame not installed")

    try:
        print("[GAME] Trying to install pygame via pip")
        import pip
        install("pygame")
        print("[GAME] Pygame has been installed")
    except:
        print("[EXCEPTION] Pip not installed on system")
        print("[GAME] Trying to install pip")
        get_pip.main()
        print("[GAME] Pip has been installed")
        try:
            print("[GAME] Trying to install pygame")
            import pip
            install("pygame")
            print("[GAME] Pygame has been installed")
        except:
            print("[ERROR 1] Pygame could not be installed")
```

Hàm install(package) sử dụng subprocess.call() để gọi lệnh pip để cài đặt gói Python. Đối số package là tên của gói cần cài đặt.

Trong khối mã chính: Dòng import pygame được sử dụng để kiểm tra xem gói pygame đã được cài đặt hay chưa.

Nếu lệnh import pygame gây ra ngoại lệ, tức là gói pygame chưa được cài đặt, các bước tiếp theo được thực hiện.

Dòng import pip được sử dụng để kiểm tra xem gói pip đã được cài đặt hay chưa.

Nếu lệnh import pip gây ra ngoại lệ, tức là gói pip chưa được cài đặt, các bước tiếp theo được thực hiện.

Dòng get_pip.main() được sử dụng để cài đặt gói pip trên hệ thống. Sau đó, lệnh import pip được sử dụng một lần nữa để đảm bảo gói pip đã được cài đặt.

Cuối cùng, hàm install("pygame") được gọi để cài đặt gói pygame. Nếu lệnh import pygame gây ra ngoại lệ sau khi cài đặt, thông báo lỗi "Pygame could not be installed" được in ra màn hình.

2.9.2 Tạo biến và hình ảnh

```
pygame.font.init()

board = pygame.transform.scale(pygame.image.load(os.path.join("img","board_alt.png")), (800, 600))
chessbg = pygame.image.load(os.path.join("img", "chessbg.png"))
rect = (250, 50, 550, 550)

turn = "w"
```

pygame.font.init() được sử dụng để khởi tạo module font trong pygame, cho phép sử dụng các font chữ trong trò chơi.

board =pygame.transform.scale(pygame.image.load(os.path.join("img","board_alt.png")), (800, 600)) tải hình ảnh "board_alt.png" từ thư mục "img" và thay đổi kích thước của hình ảnh thành (800, 600) bằng phương thức pygame.transform.scale(). Kết quả được gán cho biến board, đại diện cho hình ảnh bảng trò chơi.

chessbg = pygame.image.load(os.path.join("img", "chessbg.png")) tải hình ảnh "chessbg.png" từ thư mục "img" bằng phương thức pygame.image.load(). Kết quả được gán cho biến chessbg, đại diện cho hình ảnh nền của trò chơi cờ vua.

rect = (250, 50, 550, 550) định nghĩa một hình chữ nhật (rect) với tọa độ và kích thước. Hình chữ nhật này được sử dụng để vẽ bảng cờ trên cửa sổ trò chơi.

turn = "w" gán giá trị "w" cho biến turn, đại diện cho lượt đi của người chơi màu trắng (white).

2.9.3 Vẽ cửa sổ,bảng,thời gian,thông báo và trạng thái

```
def redraw_gameWindow(win, bo, p1, p2, color, ready):
    win.blit(board, (0, 0))
    bo.draw(win, color)

    formatTime1 = str(int(p1//60)) + ":" + str(int(p1%60))
    formatTime2 = str(int(p2 // 60)) + ":" + str(int(p2 % 60))
    if int(p1%60) < 10:
        formatTime1 = formatTime1[:-1] + "0" + formatTime1[-1]
    if int(p2%60) < 10:
        formatTime2 = formatTime2[:-1] + "0" + formatTime2[-1]

    font = pygame.font.SysFont("comicsans", 30)
    try:
        txt = font.render(bo.p1Name + "\'s Time: " + str(formatTime2), 1, (255, 255, 255))
        txt2 = font.render(bo.p2Name + "\'s Time: " + str(formatTime1), 1, (255,255,255))
    except Exception as e:
        print(e)
    win.blit(txt, (520,10))
    win.blit(txt2, (520, 700))

    txt = font.render("Press q to Quit", 1, (255, 255, 255))
    win.blit(txt, (10, 20))
```

```
    if color == "s":
        txt3 = font.render("SPECTATOR MODE", 1, (255, 0, 0))
        win.blit(txt3, (width/2-txt3.get_width()/2, 10))

    if not ready:
        show = "Waiting for Player"
        if color == "s":
            show = "Waiting for Players"
        font = pygame.font.SysFont("comicsans", 80)
        txt = font.render(show, 1, (255, 0, 0))
        win.blit(txt, (width/2 - txt.get_width()/2, 300))

    if not color == "s":
        font = pygame.font.SysFont("comicsans", 30)
        if color == "w":
            txt3 = font.render("YOU ARE WHITE", 1, (255, 0, 0))
            win.blit(txt3, (width / 2 - txt3.get_width() / 2, 10))
        else:
            txt3 = font.render("YOU ARE BLACK", 1, (255, 0, 0))
            win.blit(txt3, (width / 2 - txt3.get_width() / 2, 10))

        if bo.turn == color:
            txt3 = font.render("YOUR TURN", 1, (255, 0, 0))
            win.blit(txt3, (width / 2 - txt3.get_width() / 2, 700))
        else:
            txt3 = font.render("THEIR TURN", 1, (255, 0, 0))
            win.blit(txt3, (width / 2 - txt3.get_width() / 2, 700))
    pygame.display.update()
```

win.blit(board, (0, 0)) được sử dụng để vẽ hình ảnh bảng trò chơi lên cửa sổ tại vị trí (0, 0).

bo.draw(win, color) được sử dụng để vẽ các quân cờ trên bảng trò chơi với màu sắc tương ứng.

Thời gian của người chơi 1 và người chơi 2 được định dạng thành chuỗi "phút:giây" và được lưu trong biến formatTime1 và formatTime2. Nếu giây của thời gian người chơi 1 hoặc người chơi 2 nhỏ hơn 10, một số 0 được thêm vào phía trước giây để hiển thị đúng định dạng. Một đối tượng font được tạo bằng pygame.font.SysFont("comicsans", 30) để sử dụng font chữ "comicsans" với kích thước 30.

Với các thông tin về thời gian của người chơi 1 và người chơi 2, sử dụng font.render() để tạo các đối tượng văn bản và vẽ chúng lên cửa sổ với các tọa độ tương ứng.

Vẽ thông tin "Press q to Quit" lên cửa sổ.

Nếu màu sắc là "s" (khán giả), vẽ thông tin "SPECTATOR MODE" lên cửa sổ.

Nếu trò chơi chưa sẵn sàng, hiển thị thông báo "Waiting for Player" hoặc "Waiting for Players" (nếu là khán giả) trên cửa sổ.

Nếu không phải là khán giả, hiển thị thông tin về màu sắc và lượt đi của người chơi hiện tại trên cửa sổ.

Cập nhật cửa sổ hiển thị với các thay đổi được vẽ lên bằng pygame.display.update().

2.9.4 Màn hình kết thúc

```
def end_screen(win, text):
    pygame.font.init()
    font = pygame.font.SysFont("comicsans", 80)
    txt = font.render(text,1, (255,0,0))
    win.blit(txt, (width / 2 - txt.get_width() / 2, 300))
    pygame.display.update()

    pygame.time.set_timer(pygame.USEREVENT+1, 3000)

    run = True
    while run:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
                run = False
            elif event.type == pygame.KEYDOWN:
                run = False
            elif event.type == pygame.USEREVENT+1:
                run = False
```

pygame.font.init() được sử dụng để khởi tạo module font trong pygame, cho phép sử dụng các font chữ trong trò chơi.

Một đối tượng font được tạo bằng pygame.font.SysFont("comicsans", 80) để sử dụng font chữ "comicsans" với kích thước 80.

Sử dụng font.render() để tạo một đối tượng văn bản với nội dung và màu sắc tương ứng.

Vẽ đối tượng văn bản lên cửa sổ tại vị trí trung tâm ngang và 300 pixel từ phía trên.

Cập nhật cửa sổ hiển thị với các thay đổi được vẽ lên bằng pygame.display.update().

Đặt một bộ đếm thời gian với pygame.time.set_timer(pygame.USEREVENT+1, 3000), đảm bảo rằng sự kiện với mã pygame.USEREVENT+1 sẽ được kích hoạt sau 3 giây (3000 mili giây).

Một vòng lặp while được sử dụng để theo dõi các sự kiện và xử lý chúng.

Trong vòng lặp, kiểm tra các sự kiện được lấy từ pygame.event.get(). Nếu sự kiện là pygame.QUIT (người dùng đóng cửa sổ), hoặc sự kiện phím được nhấn (pygame.KEYDOWN), hoặc sự kiện với mã pygame.USEREVENT+1 (hết thời gian đợi), thì thoát khỏi vòng lặp và kết thúc hàm.

2.9.5 Xác định vị trí ô cờ

```
def click(pos):
    """
    :return: pos (x, y) in range 0-7 0-7
    """
    x = pos[0]
    y = pos[1]
    if rect[0] < x < rect[0] + rect[2]:
        if rect[1] < y < rect[1] + rect[3]:
            divX = x - rect[0]
            divY = y - rect[1]
            i = int(divX / (rect[2]/8))
            j = int(divY / (rect[3]/8))
            return i, j

    return -1, -1
```

Hàm nhận đầu vào là pos, đại diện cho tọa độ (x, y) của vị trí chuột được nhấp.

Biến x và y được gán giá trị tương ứng từ pos. Kiểm tra xem x có nằm trong khoảng giới hạn ngang của ô cờ (rect[0] tới rect[0] + rect[2]) và y có nằm trong khoảng giới hạn dọc của ô cờ (rect[1] tới rect[1] + rect[3]).

Nếu x và y đều nằm trong khoảng giới hạn của ô cờ, tính toán giá trị divX và divY là hiệu của x và y so với giới hạn trái và trên của ô cờ.

Tính toán giá trị i và j bằng cách chia nguyên divX cho rect[2]/8 và divY cho rect[3]/8, tương ứng với vị trí ô cờ trong bảng 8x8. Trả về giá trị (i, j) cho biết vị trí ô cờ được nhấp chuột trong bảng. Các giá trị i và j nằm trong khoảng từ 0 đến 7.

Nếu x hoặc y không nằm trong khoảng giới hạn của ô cờ, hoặc không thỏa mãn điều kiện kiểm tra, trả về (-1, -1) để chỉ ra rằng không có ô cờ nào được nhấp chuột.

3 Web demo và tải game

Link : *https* : *//github.com/quang*90111*/chess*