Project Report

Student Name: Ritik Saini	**UID: 24MCA20138**
Branch: MCA-General	**Section: 24MCA-3A**
Semester: 1st	**Date of Performance: 05-11-2024**
Subject: Python Programming Lab	**Subject Code: 24CAH-606**

1. **Aim of the project:** Make a game **"The Checkers Game".**

2. **Hardware and Software Requirements:**

➢ **Software:**

- **Download:** I have downloaded the latest version (3.12.5) of Python from the official website: python.org.
- **Installation:** During installation, I have checked the option to "Add Python to PATH"on Windows to make running Python from the command line easier.
- **Jupyter Notebook:** An interactive environment that allows us to write code, display outputs, and include markdown notes in a notebook format. Ideal for data science andexperimentation. Install Jupyter Notebook.
- **Anaconda**: A distribution that includes Python and many scientific libraries, along with a package manager (conda) and the Jupyter Notebook. It's especially useful for data science. Downloaded Anaconda version (2.6.2).

➢ **Hardware:**

- I have used my personal laptop here are the following specification:
- **Processor:** Intel Core i5 8th Generation.
- **RAM:** 8GB DDR4.
- **Storage:** 1TB and 128GB SSD.
- **Operating System:** Windows 11 Pro.

3. **Program Logic:**

➢ **Import pygame**

- The program begins by importing pygame.

- **Game Board:** The Checkers game is played on a rectangular board consisting of 64 squares, arranged in an 8x8 grid.

- **Pieces:** The game includes two types of pieces: Red and White checkers.

- **Movement:** Checkers move diagonally, occupying adjacent squares.

- **Jumping:** A checker can jump over an opponent's piece to an empty square immediately after it, capturing the opponent's piece.

- **King:** A checker becomes a king when it reaches the opposite end of the board, acquiring increased mobility and capturing capability.

- **Game Over:** The game concludes when one player has no checkers remaining on the board, declaring the opponent the winner.

- **Define Functions:**

  - **Piece Class**: Represents an individual game piece, encapsulating its properties and behavior.

  - **Board Class**: Represents the game board, managing its layout, pieces, and interactions.

  - **Game Class**: Encapsulates the game logic, governing piece movements, captures, and game flow.

  - **Main Function**: Initializes and runs the game, orchestrating the game loop and event handling.

4. Code:

```
 import pygame
import sys
from pygame.locals import QUIT, MOUSEBUTTONDOWN

WIDTH, HEIGHT = 680, 680
ROWS, COLS = 8, 8
SQUARE_SIZE = WIDTH // COLS

RED = (255, 0, 0)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
```

```
TAN = (210, 180, 140)
BROWN = (139, 69, 19)
GREY = (128, 128, 128)
GREEN = (0, 255, 0)
HIGHLIGHT = (0, 255, 255)

pygame.init()
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Checkers')



class Piece:
    PADDING = 15
    OUTLINE = 2

    def __init__(self, row, col, color):
        self.row = row
        self.col = col
        self.color = color
        self.king = False
        self.calc_pos()

    def calc_pos(self):
        self.x = SQUARE_SIZE * self.col + SQUARE_SIZE // 2
        self.y = SQUARE_SIZE * self.row + SQUARE_SIZE // 2

    def make_king(self):
        self.king = True

    def draw(self, win):
        radius = SQUARE_SIZE // 2 - self.PADDING
        pygame.draw.circle(win, GREY, (self.x, self.y), radius + self.OUTLINE)
        pygame.draw.circle(win, self.color, (self.x, self.y), radius)


class Board:
    def __init__(self):
        self.board = []
        self.create_board()

    def create_board(self):
        for row in range(ROWS):
            self.board.append([])
            for col in range(COLS):
                if (col + row) % 2 == 1:
                    if row < 3:
                        self.board[row].append(Piece(row, col, WHITE))
                    elif row > 4:
                        self.board[row].append(Piece(row, col, RED))
                    else:
                        self.board[row].append(0)
                else:
                    self.board[row].append(0)

    def draw_squares(self, win):
        win.fill(BLACK)
        for row in range(ROWS):
            for col in range(COLS):
                color = BROWN if (col + row) % 2 == 0 else TAN
                pygame.draw.rect(win, color, (col * SQUARE_SIZE, row * SQUARE_SIZE,
     SQUARE_SIZE, SQUARE_SIZE))

    def draw(self, win):
        self.draw_squares(win)
        for row in range(ROWS):
            for col in range(COLS):
                piece = self.board[row][col]
                if piece != 0:
                    piece.draw(win)

    def move(self, piece, row, col):
        self.board[piece.row][piece.col] = 0
        self.board[row][col] = piece
        piece.row, piece.col = row, col
        piece.calc_pos()
        # King the piece if it reaches the opposite end
        if (piece.color == WHITE and row == 0) or (piece.color == RED and row == ROWS - 1):
            piece.make_king()

    def get_piece(self, row, col):
        return self.board[row][col]

    def remove(self, pieces):
        for piece in pieces:
            self.board[piece.row][piece.col] = 0

    def valid_moves(self, piece):
        moves = {}
        directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
        for d in directions:
            row, col = piece.row + d[0], piece.col + d[1]
            if 0 <= row < ROWS and 0 <= col < COLS:
                if self.board[row][col] == 0:
                    moves[(row, col)] = []
                # Jumping logic
                row_jump, col_jump = piece.row + 2 * d[0], piece.col + 2 * d[1]
                if 0 <= row_jump < ROWS and 0 <= col_jump < COLS:
                    if (self.board[row][col] != 0 and
                            self.board[row][col].color != piece.color and
                            self.board[row_jump][col_jump] == 0):  # Valid jump
                        moves[(row_jump, col_jump)] = [self.board[row][col]]
        return moves


class Game:
    def __init__(self, win):
        self.win = win
        self.turn = RED  # Player starts
        self.selected = None
        self.valid_moves = {}
    self.board = Board()

def select(self, row, col):
    piece = self.board.get_piece(row, col)
    if self.selected:
        result = self._move(row, col)
        if not result:
            self.selected = None
            self.select(row, col)
    if piece != 0 and piece.color == self.turn:
        self.selected = piece
        self.valid_moves = self.board.valid_moves(piece)
        return True
    return False

def _move(self, row, col):
    piece = self.board.get_piece(row, col)
    if self.selected and (row, col) in self.valid_moves:
        self.board.move(self.selected, row, col)
        skipped = self.valid_moves[(row, col)]
        if skipped:
            self.board.remove(skipped)
        self.change_turn()
    else:
        return False
    return True

def change_turn(self):
    self.valid_moves = {}
    self.turn = WHITE if self.turn == RED else RED

def draw_valid_moves(self, moves):
    for move in moves:
        row, col = move
        pygame.draw.circle(self.win, HIGHLIGHT, (col * SQUARE_SIZE + SQUARE_SIZE
 // 2, row * SQUARE_SIZE + SQUARE_SIZE // 2), 15)
```

```
def update(self):
    self.board.draw(self.win)
    if self.selected:
        self.draw_valid_moves(self.valid_moves)
    self.draw_turn()
    pygame.display.update()

def draw_turn(self):
    font = pygame.font.Font(None, 36)
    text = f"{'Red' if self.turn == RED else 'White'}'s Turn"
    text_surface = font.render(text, True, BLACK)
    self.win.blit(text_surface, (10, 10))

def check_game_over(self):
    red_pieces = [piece for row in self.board.board for piece in row if piece != 0 and
 piece.color == RED]
    white_pieces = [piece for row in self.board.board for piece in row if piece != 0 and
 piece.color == WHITE]

    if not red_pieces:
        self.game_over_alert("White wins!")
        return True
    elif not white_pieces:
        self.game_over_alert("Red wins!")
        return True

    return False

def game_over_alert(self, message):
    font = pygame.font.Font(None, 74)
    text_surface = font.render(message, True, GREEN)
    text_rect = text_surface.get_rect(center=(WIDTH / 2, HEIGHT / 2))
    self.win.blit(text_surface, text_rect)
    pygame.display.update()
    pygame.time.delay(3000)
    pygame.quit()
    sys.exit()
```

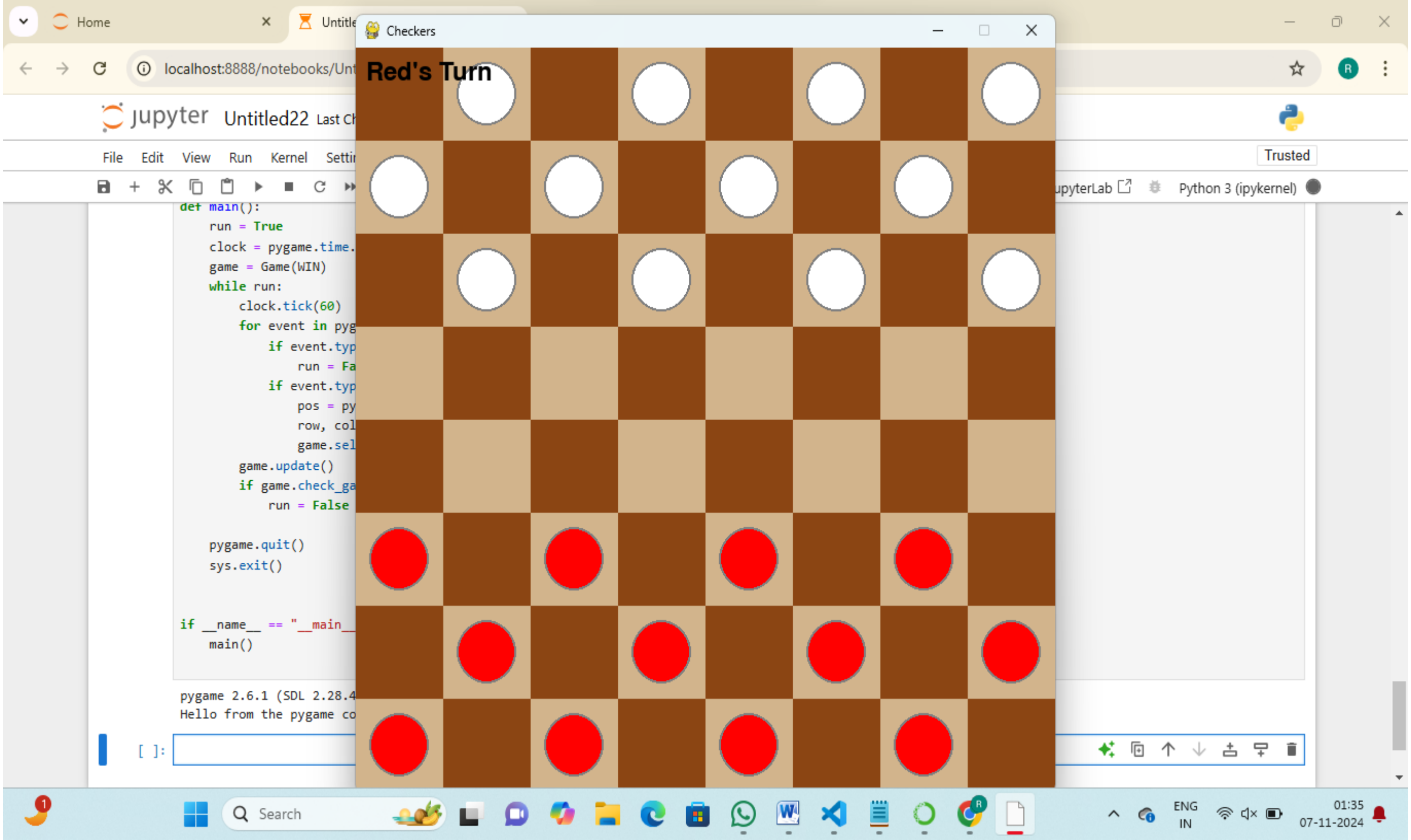
```
def main():
        run = True
        clock = pygame.time.Clock()
        game = Game(WIN)
        while run:
            clock.tick(60)
            for event in pygame.event.get():
                if event.type == QUIT:
                    run = False
                if event.type == MOUSEBUTTONDOWN:
                    pos = pygame.mouse.get_pos()
                    row, col = pos[1] // SQUARE_SIZE, pos[0] // SQUARE_SIZE
                    game.select(row, col)
            game.update()
            if game.check_game_over():
                run = False  # Exit loop if game is over

          pygame.quit()
        sys.exit()



      if __name__ == "__main__":
            main()
```

5. Result:



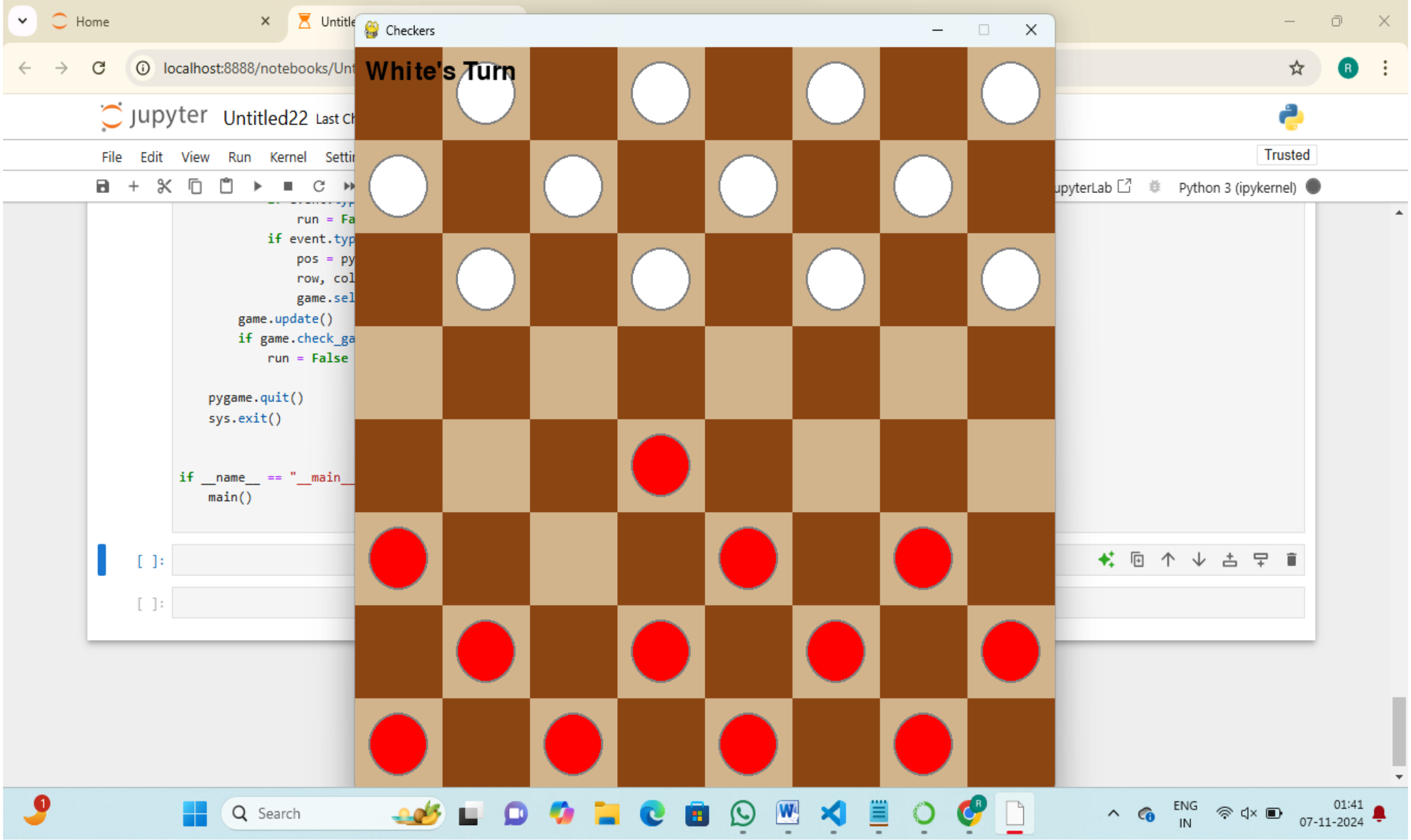
```
def main():
    run = True
    clock = pygame.time.
    game = Game(WIN)
    while run:
        clock.tick(60)
        for event in pyg
            if event.typ
                run = Fa
            if event.typ
                pos = py
                row, col
                game.sel
        game.update()
        if game.check_ga
            run = False

    pygame.quit()
    sys.exit()


if __name__ == "__main_
    main()
```

```
pygame 2.6.1 (SDL 2.28.4
Hello from the pygame co
```

```
[ ]:
```



```
                run = Fa
            if event.typ
                pos = py
                row, col
                game.sel
        game.update()
        if game.check_ga
            run = False

    pygame.quit()
    sys.exit()


if __name__ == "__main_
    main()
```

```
[ ]:
[ ]:
```

6. Learning outcomes (What I have learnt):

- Utilizing the Pygame library.
- Implementing game logic and features in game development.
- Applying Object-Oriented Programming (OOP) concepts.
- Handling graphics and user input in game development.
- Implementing game logic and rules in game development.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Worksheet		08
2.	Viva		10
3.	Simulation		12
4.	Total Marks		30

Teacher's Signature