**#PART 1: Mortal Kombat**

#Part 1: Mortal Kombat

def alpha\_beta\_pruning(depth, node\_index, maximizing\_player, values, alpha, beta):

if depth == 3:

return values[node\_index]

if maximizing\_player:

max\_eval = float('-inf')

for i in range(2):

eval = alpha\_beta\_pruning(depth + 1, node\_index \* 2 + i, False, values, alpha, beta)

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = float('inf')

for i in range(2):

eval = alpha\_beta\_pruning(depth + 1, node\_index \* 2 + i, True, values, alpha, beta)

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval

def simulate\_game(start\_player):

utility\_values = [-1, 1, -1, 1, -1, 1, -1, 1]

round\_winners = []

rounds = 3

current\_player = start\_player

for round\_num in range(rounds):

round\_winner\_value = alpha\_beta\_pruning(0, 0, current\_player == 1, utility\_values, float('-inf'), float('inf'))

round\_winner = "Sub-Zero" if round\_winner\_value == 1 else "Scorpion"

round\_winners.append(round\_winner)

current\_player = 1 if current\_player == 0 else 0

return round\_winners

start\_player = int(input("Enter 0 for Scorpion, 1 for Sub-Zero: "))

round\_winners = simulate\_game(start\_player)

game\_winner = round\_winners[-1]

print(f"Game Winner: {game\_winner}")

print(f"Total Rounds Played: {len(round\_winners)}")

for i, winner in enumerate(round\_winners, 1):

print(f"Winner of Round {i}: {winner}")

**#PART 2: Games with Magic**

**#Part 2: Games with Magic**

**def minimax(node, depth, is\_pacman\_turn, alpha, beta, values):**

**if depth == 3:**

**return values[node]**

**if is\_pacman\_turn:**

**max\_eval = float('-inf')**

**for i in range(2):**

**eval = minimax(2 \* node + i, depth + 1, False, alpha, beta, values)**

**max\_eval = max(max\_eval, eval)**

**alpha = max(alpha, eval)**

**if beta <= alpha:**

**break**

**return max\_eval**

**else:**

**min\_eval = float('inf')**

**for i in range(2):**

**eval = minimax(2 \* node + i, depth + 1, True, alpha, beta, values)**

**min\_eval = min(min\_eval, eval)**

**beta = min(beta, eval)**

**if beta <= alpha:**

**break**

**return min\_eval**

**def pacman\_game(c):**

**values = [3, 6, 2, 3, 7, 1, 2, 0]**

**minimax\_value = minimax(0, 0, True, float('-inf'), float('inf'), values)**

**left\_subtree\_value = max(values[0], values[1]) - c**

**right\_subtree\_value = max(values[4], values[5]) - c**

**if left\_subtree\_value > right\_subtree\_value:**

**best\_value\_with\_magic = left\_subtree\_value**

**direction = "left"**

**else:**

**best\_value\_with\_magic = right\_subtree\_value**

**direction = "right"**

**if best\_value\_with\_magic > minimax\_value:**

**print(f"The new minimax value is {best\_value\_with\_magic}. Pacman goes {direction} and uses dark magic.")**

**else:**

**print(f"The minimax value is {minimax\_value}. Pacman does not use dark magic.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**try:**

**c = int(input("Enter the value for dark magic (c): "))**

**pacman\_game(c)**

**except ValueError:**

**print("Invalid input. Please enter an integer.")**