CODE FOR PYTHON RED-BLUE-NIM GAME

import argparse

# Constants

RED\_SCORE = 2

BLUE\_SCORE = 3

MAX\_REMOVE = 2

# GameState class

class GameState:

def \_\_init\_\_(self, red, blue, is\_misere):

self.red = red

self.blue = blue

self.is\_misere = is\_misere

def is\_game\_over(self):

return self.red == 0 or self.blue == 0

def get\_score(self):

return self.red \* RED\_SCORE + self.blue \* BLUE\_SCORE

def clone(self):

return GameState(self.red, self.blue, self.is\_misere)

# AI functions

def minmax(state, depth, alpha, beta, maximizing\_player):

if depth == 0 or state.is\_game\_over():

return state.get\_score(), None

best\_move = None

if maximizing\_player:

max\_eval = float('-inf')

for move in get\_possible\_moves(state):

new\_state = apply\_move(state.clone(), move)

eval, \_ = minmax(new\_state, depth - 1, alpha, beta, False)

if eval > max\_eval:

max\_eval = eval

best\_move = move

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval, best\_move

else:

min\_eval = float('inf')

for move in get\_possible\_moves(state):

new\_state = apply\_move(state.clone(), move)

eval, \_ = minmax(new\_state, depth - 1, alpha, beta, True)

if eval < min\_eval:

min\_eval = eval

best\_move = move

beta = min(beta, eval)

if beta <= alpha:

break

return min\_eval, best\_move

# Utility functions

def get\_possible\_moves(state):

moves = []

for color in ['red', 'blue']:

for count in range(1, min(MAX\_REMOVE, getattr(state, color)) + 1):

moves.append((color, count))

return moves if not state.is\_misere else reversed(moves)

def apply\_move(state, move):

color, count = move

setattr(state, color, getattr(state, color) - count)

return state

# Game functions

def play\_game(red, blue, is\_misere, first\_player, depth):

state = GameState(red, blue, is\_misere)

current\_player = first\_player

while not state.is\_game\_over():

print(f"\nCurrent state: Red: {state.red}, Blue: {state.blue}")

if current\_player == 'human':

move = get\_human\_move(state)

else:

\_, move = minmax(state, depth, float('-inf'), float('inf'), True)

print(f"Computer chooses: {move}")

state = apply\_move(state, move)

current\_player = 'computer' if current\_player == 'human' else 'human'

print(f"\nGame over! Final state: Red: {state.red}, Blue: {state.blue}")

score = state.get\_score()

winner = 'Human' if current\_player == 'computer' else 'Computer'

if state.is\_misere:

winner = 'Human' if current\_player == 'human' else 'Computer'

print(f"{winner} wins! Score: {score}")

def get\_human\_move(state):

while True:

try:

color = input("Choose color (red/blue): ").lower()

count = int(input("Choose number of marbles to remove (1-2): "))

if color in ['red', 'blue'] and 1 <= count <= min(MAX\_REMOVE, getattr(state, color)):

return color, count

except ValueError:

pass

print("Invalid move. Try again.")

def parse\_arguments():

parser = argparse.ArgumentParser(description="Red-Blue Nim Game")

parser.add\_argument("red", type=int, help="Number of red marbles")

parser.add\_argument("blue", type=int, help="Number of blue marbles")

parser.add\_argument("--version", choices=['standard', 'misere'], default='standard', help="Game version")

parser.add\_argument("--first", choices=['computer', 'human'], default='computer', help="First player")

parser.add\_argument("--depth", type=int, default=10, help="Search depth for AI")

return parser.parse\_args()

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

args = parse\_arguments()

play\_game(args.red, args.blue, args.version == 'misere', args.first, args.depth)

**OUTPUT DISPLAY**

