***Code:***

import argparse

# Define the game rules

class GameRules:

def \_\_init\_\_(self, version):

self.version = version

def is\_game\_over(self, red\_marbles, blue\_marbles):

if self.version == 'standard':

return red\_marbles == 0 or blue\_marbles == 0

elif self.version == 'misere':

return red\_marbles == 0 or blue\_marbles == 0

def calculate\_score(self, red\_marbles, blue\_marbles):

return 2 \* red\_marbles + 3 \* blue\_marbles

# Define the MinMax algorithm with Alpha Beta Pruning

class MinMax:

def \_\_init\_\_(self, game\_rules, depth):

self.game\_rules = game\_rules

self.depth = depth

def minimax(self, red\_marbles, blue\_marbles, depth, alpha, beta, maximizing\_player):

if depth == 0 or self.game\_rules.is\_game\_over(red\_marbles, blue\_marbles):

return self.game\_rules.calculate\_score(red\_marbles, blue\_marbles)

if maximizing\_player:

value = -float('inf')

for move in self.get\_moves(red\_marbles, blue\_marbles):

new\_red\_marbles, new\_blue\_marbles = self.apply\_move(red\_marbles, blue\_marbles, move)

value = max(value, self.minimax(new\_red\_marbles, new\_blue\_marbles, depth - 1, alpha, beta, False))

alpha = max(alpha, value)

if alpha >= beta:

break

return value

else:

value = float('inf')

for move in self.get\_moves(red\_marbles, blue\_marbles):

new\_red\_marbles, new\_blue\_marbles = self.apply\_move(red\_marbles, blue\_marbles, move)

value = min(value, self.minimax(new\_red\_marbles, new\_blue\_marbles, depth - 1, alpha, beta, True))

beta = min(beta, value)

if beta <= alpha:

break

return value

def get\_moves(self, red\_marbles, blue\_marbles):

moves = []

if self.game\_rules.version == 'standard':

moves.append((2, 0)) # Pick 2 red marbles

moves.append((0, 2)) # Pick 2 blue marbles

moves.append((1, 0)) # Pick 1 red marble

moves.append((0, 1)) # Pick 1 blue marble

elif self.game\_rules.version == 'misere':

moves.append((0, 1)) # Pick 1 blue marble

moves.append((1, 0)) # Pick 1 red marble

moves.append((0, 2)) # Pick 2 blue marbles

moves.append((2, 0)) # Pick 2 red marbles

return moves

def apply\_move(self, red\_marbles, blue\_marbles, move):

new\_red\_marbles = red\_marbles - move[0]

new\_blue\_marbles = blue\_marbles - move[1]

return new\_red\_marbles, new\_blue\_marbles

# Define the game flow

class Game:

def \_\_init\_\_(self, game\_rules, num\_red, num\_blue, first\_player, depth):

self.game\_rules = game\_rules

self.num\_red = num\_red

self.num\_blue = num\_blue

self.first\_player = first\_player

self.depth = depth

self.minmax = MinMax(game\_rules, depth)

def play(self):

current\_player = self.first\_player

while not self.game\_rules.is\_game\_over(self.num\_red, self.num\_blue):

if current\_player == 'human':

move = self.get\_human\_move()

self.num\_red, self.num\_blue = self.apply\_move(self.num\_red, self.num\_blue, move)

elif current\_player == 'computer':

move = self.minmax.get\_best\_move(self.num\_red, self.num\_blue)

self.num\_red, self.num\_blue = self.apply\_move(self.num\_red, self.num\_blue, move)

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

print("Game over!")

print("Final score:", self.game\_rules.calculate\_score(self.num\_red, self.num\_blue))

def get\_human\_move(self):

while True:

move = input("Enter your move (e.g., '2 0' to pick 2 red marbles): ")

try:

move = tuple(map(int, move.split()))

if move in self.minmax.get\_moves(self.num\_red, self.num\_blue):

return move

else:

print("Invalid move. Please try again.")

except ValueError:

print("Invalid input. Please try again.")

def apply\_move(self, red\_marbles, blue\_marbles, move):

new\_red\_marbles = red\_marbles - move[0]

new\_blue\_marbles = blue\_marbles - move[1]

return new\_red\_marbles, new\_blue\_marbles

def get\_best\_move(self, red\_marbles, blue\_marbles):

best\_move = None

best\_score = -float('inf')

for move in self.minmax.get\_moves(red\_marbles, blue\_marbles):

new\_red\_marbles, new\_blue\_marbles = self.apply\_move(red\_marbles, blue\_marbles, move)

score = self.minmax.minimax(new\_red\_marbles, new\_blue\_marbles, self.depth, -float('inf'), float('inf'), False)

if score > best\_score:

best\_score = score

best\_move = move

return best\_move

def main():

parser = argparse.ArgumentParser(description='Play the marble game')

parser.add\_argument('--num-red', type=int, default=10, help='Number of red marbles')

parser.add\_argument('--num-blue', type=int, default=10, help='Number of blue marbles')

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

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

parser.add\_argument('--depth', type=int, default=5, help='Search depth for AI')

args = parser.parse\_args()

game\_rules = GameRules(args.version)

game = Game(game\_rules, args.num\_red, args.num\_blue, args.first\_player, args.depth)

game.play()

if \_\_name\_\_ == '\_\_main\_\_':

main()

To run the game, save this code to a file (e.g., **marble\_game.py**) and run it from the command line:

python marble\_game.py --num-red 10 --num-blue 10 --version standard --first-player human --depth 5

You can modify the command-line arguments to change the game settings. For example, to play the misère version with 20 red marbles and 20 blue marbles, you can run:

python marble\_game.py --num-red 20 --num-blue 20 --version misere --first-player human --depth 5