RAJALAKSHMI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR 2025 - 2026

SEMESTER III

ARTIFICIAL INTELLIGENCE LABORATORY

MINI PROJECT REPORT

REGISTER NUMBER	**2117240070022**
NAME	**ANBUSELVAN M**
PROJECT TITLE	**NIM GAME USING AI**
DATE OF SUBMISSION	**29/10/2025**
FACULTY IN-CHARGE	**Mrs. M. Divya**

Signature of Faculty In-charge

INTRODUCTION

The Nim Game is a classical mathematical strategy game that demonstrates how artificial intelligence can make optimal decisions using logic and computation. It involves players taking turns to remove objects from different piles, and the player who removes the last object wins. The project applies game theory and bitwise operations to enable an AI opponent that always plays optimally, showcasing intelligent decision-making principles.

PROBLEM STATEMENT

To design and implement an AI-based Nim Game in Python where a human can play against an intelligent computer opponent. The system should:

- Represent multiple piles of objects dynamically.
- Allow turns between player and AI.
- Use an algorithm that ensures the AI makes optimal moves.
- Display results clearly after each move and declare the winner accurately.

GOAL

- The main goal of this project is to demonstrate AI decision-making using game theory and logical computation. It aims to create a fair and interactive game where the AI applies mathematical reasoning to always play the best possible move.

THEORETICAL BACKGROUND

• The Nim Game is based on combinatorial game theory, which studies strategies in games of perfect information.
• The core mathematical concept is the Nim-sum, calculated using the bitwise XOR of all pile sizes.
• A position is losing if the Nim-sum equals zero and winning otherwise.
• The AI uses this principle to determine how many objects to remove to force the opponent into a losing state.

ALGORITHM EXPLANATION

• Initialize the number of piles and their object counts.
• Calculate the Nim-sum (XOR of all pile values).
• If the Nim-sum equals 0 → the player is in a losing position.
• Otherwise, find a pile where (pile_value XOR nim_sum) < pile_value and remove enough objects to make the new Nim-sum = 0.
• Alternate turns between the user and AI until all piles are empty.
• The player making the last move wins.

EXAMPLE

Piles = [3, 4, 5]
Nim-sum = 3 ⊕ 4 ⊕ 5 = 2
AI finds a pile to reduce Nim-sum to 0.
→ From pile 2 (5 objects), remove 3 → New pile = [3, 4, 2]
→ New Nim-sum = 3 ⊕ 4 ⊕ 2 = 5 → Player continues.
The process repeats until all piles are empty, and the winner is announced.

IMPLEMENTATION AND CODE

```
from typing import List, Tuple

import sys


def nim_sum(piles: List[int]) -> int:
    x = 0
    for p in piles:
        x ^= p
    return x


def ai_move_optimal(piles: List[int]) -> Tuple[int, int]:
    x = nim_sum(piles)
    if not any(p > 0 for p in piles):
        raise ValueError("No legal moves: all piles are empty")
    if x == 0:
        idx = max(range(len(piles)), key=lambda i: piles[i])
        return idx, 1
```

```
    for i, pile in enumerate(piles):
        target = pile ^ x
        if target < pile:
            remove = pile - target
            return i, remove
    idx = max(range(len(piles)), key=lambda i: piles[i])
    return idx, 1


def apply_move(piles: List[int], pile_idx: int, remove: int) -> None:
    piles[pile_idx] -= remove


def is_game_over(piles: List[int]) -> bool:
    return all(p == 0 for p in piles)


def print_piles(piles: List[int]) -> None:
    parts = [f"[{i}]:{p}" for i, p in enumerate(piles)]
    print("Piles:", " ".join(parts))


def parse_piles(input_str: str) -> List[int]:
    parts = input_str.strip().split()
    if not parts:
        raise ValueError("Empty pile list")
    piles: List[int] = []
```

```
    for x in parts:
        try:
            n = int(x)
        except ValueError:
            raise ValueError(f"Invalid integer in piles: '{x}'")
        if n < 0:
            raise ValueError("Pile sizes must be non-negative")
        piles.append(n)
    return piles


def human_move(piles: List[int]) -> Tuple[int, int]:
    while True:
        try:
            print_piles(piles)
            raw = input("Enter 'pile_index remove_count' (e.g. '1 3'): ")
            if not raw.strip():
                print("Empty input — try again.")
                continue
            parts = raw.strip().split()
            if len(parts) != 2:
                print("Enter exactly two integers.")
                continue
            a, b = parts
```

```
            i = int(a)
            r = int(b)
            if not (0 <= i < len(piles)):
                print("Invalid pile index. Try again.")
                continue
            if r <= 0:
                print("You must remove at least 1.")
                continue
            if r > piles[i]:
                print(f"Pile {i} only has {piles[i]} stones.")
                continue
            return i, r
        except ValueError:
            print("Invalid input. Use two integers like: 1 2")


def game_loop(piles: List[int], players: Tuple[str, str]) -> str:
    turn = 0
    while True:
        if is_game_over(piles):
            winner = f"Player {1 - turn}"
            return winner
        current = players[turn]
        if current == 'human':
```

```
            pile_idx, remove = human_move(piles)
            apply_move(piles, pile_idx, remove)
        elif current == 'ai':
            pile_idx, remove = ai_move_optimal(piles)
            apply_move(piles, pile_idx, remove)
        else:
            raise ValueError(f"Unknown player type: {current}")
        turn = 1 - turn


def choose_mode() -> Tuple[Tuple[str, str], List[int]]:
    raw_mode = input("Mode (1=H vs AI, 2=H vs H, 3=AI vs AI) [default 1]: ")
    mode = raw_mode.strip() or '1'
    if mode not in ('1', '2', '3'):
        mode = '1'
    if mode == '1':
        players = ('human', 'ai')
    elif mode == '2':
        players = ('human', 'human')
    else:
        players = ('ai', 'ai')
    pile_in = input("Enter piles as space-separated integers (press Enter for default 3 4 5): ")
    if pile_in.strip() == '':
        piles = [3, 4, 5]
```

```
    else:
        try:
            piles = parse_piles(pile_in)
        except ValueError as e:
            print(f"Invalid piles input: {e}. Using default [3,4,5].")
            piles = [3, 4, 5]
    return players, piles


def main():
    players, piles = choose_mode()
    print("Starting piles:")
    print_piles(piles)
    winner = game_loop(piles, players)
    print(f"Game over. {winner} wins!")


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        sys.exit(0)
```

OUTPUT

```
Mode (1=H vs AI, 2=H vs H, 3=AI vs AI) [default 1]: 1
Enter piles as space-separated integers (press Enter for default 3 4 5): 3 4 5
Starting piles:
Piles: [0]:3 [1]:4 [2]:5
Piles: [0]:3 [1]:4 [2]:5
Enter 'pile_index remove_count' (e.g. '1 3'): 2 5
Piles: [0]:3 [1]:3 [2]:0
Enter 'pile_index remove_count' (e.g. '1 3'): 1 4
Pile 1 only has 3 stones.
Piles: [0]:3 [1]:3 [2]:0
Enter 'pile_index remove_count' (e.g. '1 3'): 0 3
Game over. Player 1 wins!
PS C:\Users\Lenovo\OneDrive\Desktop\AI MINI PROJECT>
```

RESULTS

- The Nim Game was successfully implemented using Python.
- It allows play between Human vs AI, Human vs Human, and AI vs AI.
- The AI uses the Nim-sum algorithm for optimal decision-making.
- The system clearly displays game status and results after every move.

FUTURE ENHANCEMENT

- Develop a Graphical User Interface (GUI) using Tkinter or Pygame.
- Add Misère Nim mode (where the last move loses).
- Include player name input, score tracking, and difficulty levels.
- Extend the project to a web-based or mobile version for online play.

Git Hub Link of the project and report	https://github.com/Anbuselvan-dev/AI-FINAL-MINI-PROJECT

REFERENCES

- Wikipedia – Article on Nim Game, used to understand the mathematical and game theory foundation of the project.
- YouTube – Video titled “Nim Game Explained | Game Theory | XOR Trick”, referred for visual explanation of game logic and winning strategy.
- GeeksforGeeks – Article on Nim Game in Python, consulted for implementation guidance and algorithm structure.
- ChatGPT (OpenAI) – Utilized to understand theoretical concepts, generate Python code, and refine project documentation into a professional format.