



KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

CSE 4110: Artificial Intelligence Laboratory

Report on-

NIM GAME (a game played between AI and human to pick stones from pile)

Date of Submission: 04/09/2024

Submitted by-

Farzana Rahman

Roll: 1907028

Dept.: CSE

Section: A

Year: 4th

Semester: 1st

Submitted to-

Md. Shahidul Salim Lecturer Department of Computer Science and Engineering Khulna University of Engineering & Technology (KUET)	Most. Kaniz Fatema Isha Lecturer Department of Computer Science and Engineering Khulna University of Engineering & Technology (KUET)
---	--

TABLE OF CONTENTS....

<i>Headings</i>	<i>Page number</i>
<i>Objectives</i>	2
<i>Introduction</i>	2
<i>Simple demonstration</i>	2-5
<i>Features</i>	5
<i>Flow Chart</i>	6
<i>How to play</i>	7-9
<i>Implementation</i>	9-11
<i>Pseudocode</i>	11-19
<i>Results</i>	19
<i>Discussion</i>	19
<i>Conclusion</i>	19-20
<i>Reference</i>	20

Objectives:

- To develop a project that will be played between an AI agent and a human.
- To be able to add two different modes – Easy and Hard.
- To use different algorithms to optimize this game.
- To allow the AI agent to play in three different methods.
- To be able to see AI agent has won which games.
- To use Genetic Algorithm's "selection" step to select one method in Hard mode.
- To implement this game using python.

Introduction:


- **Nim** is a mathematical game of strategy in which two players take turns removing (or "nimming") objects (stones) from distinct heaps or piles.
- On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap or pile.
- In this game, maximum 3 stones can be removed at a time.
- Stones should be picked only from a definite pile.
- The goal of the game is to take the last object.
- The player who takes the objects will win this game.

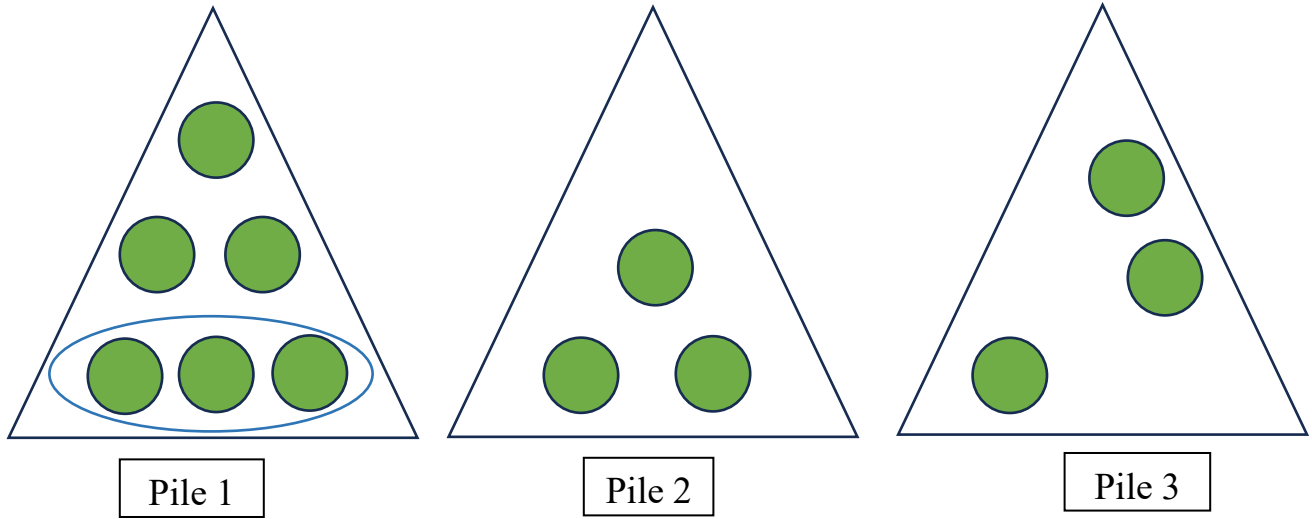
Simple demonstration:

- ⇒ Let's say there are three piles.
- ⇒ Human player will be the first one to pick stones.
- ⇒ Then AI Agent will pick.
- ⇒ The one who will pick the last stones will win this game.
- ⇒ Human's turns are circled in blue line.
- ⇒ AI agent's turns are circled as red lines.

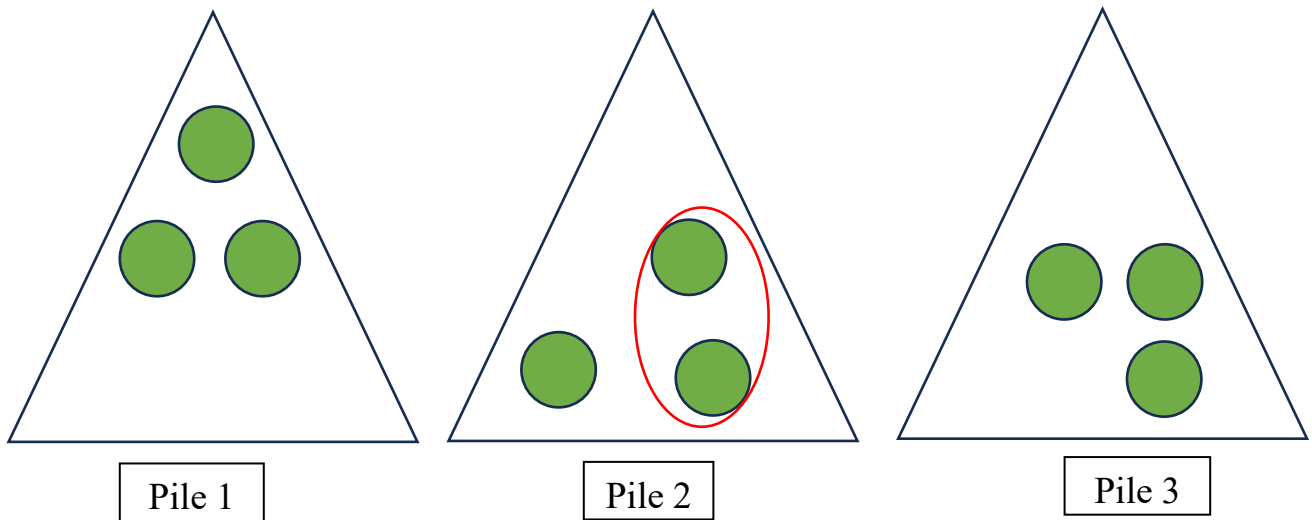
⇒ Now, let's see the demonstration---

 → Human's turn

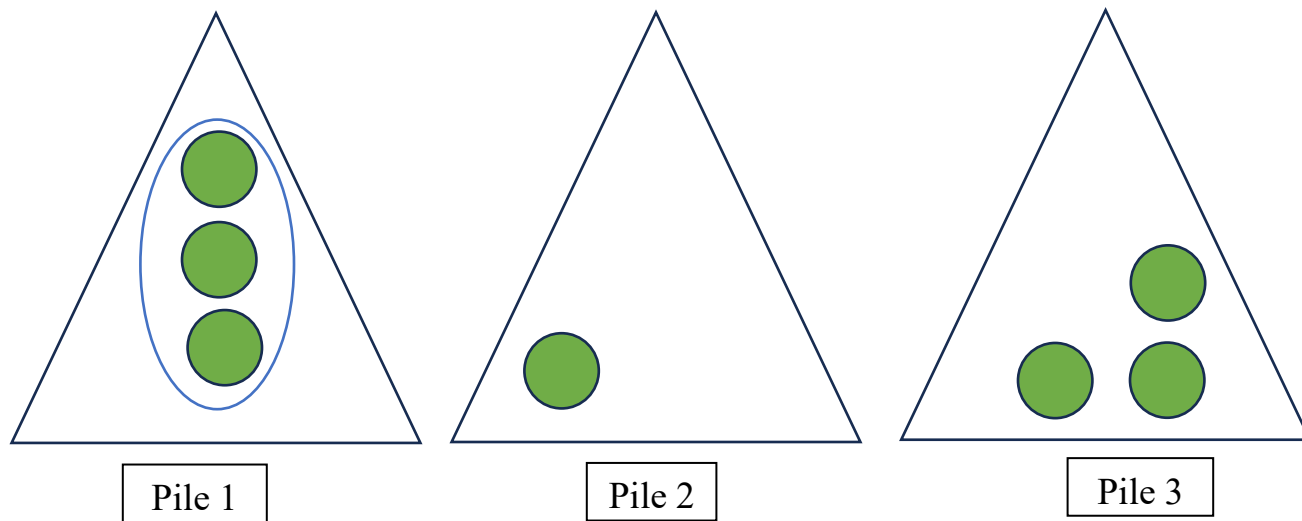
 → Agent's turn



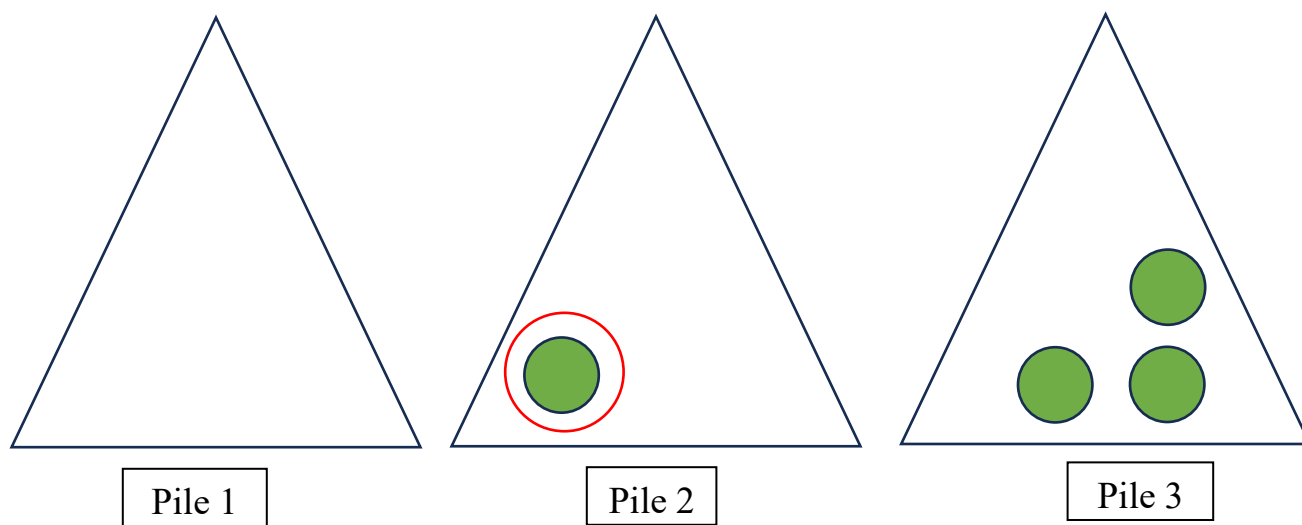
⇒ After human's turn:



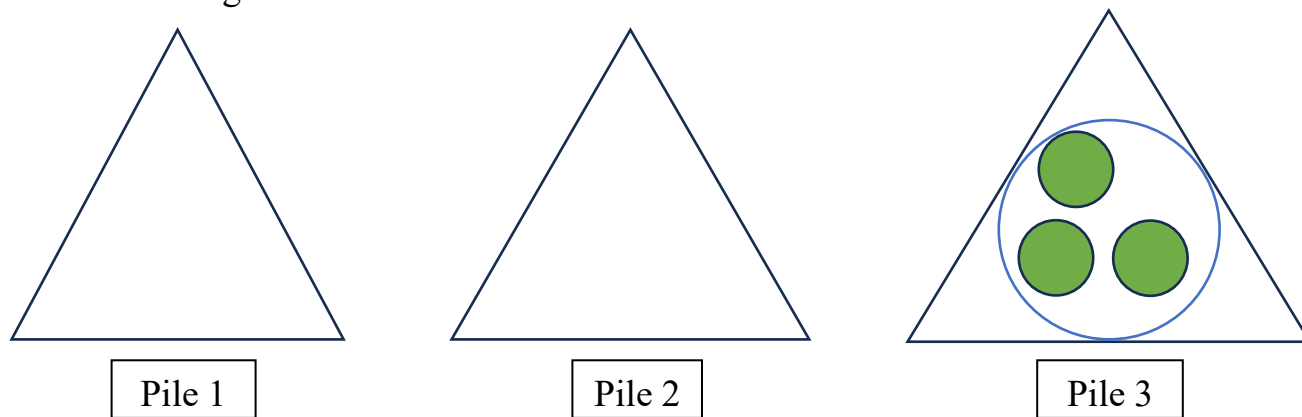
⇒ After agent's turn:



⇒ After human's turn:



⇒ After agent's turn:

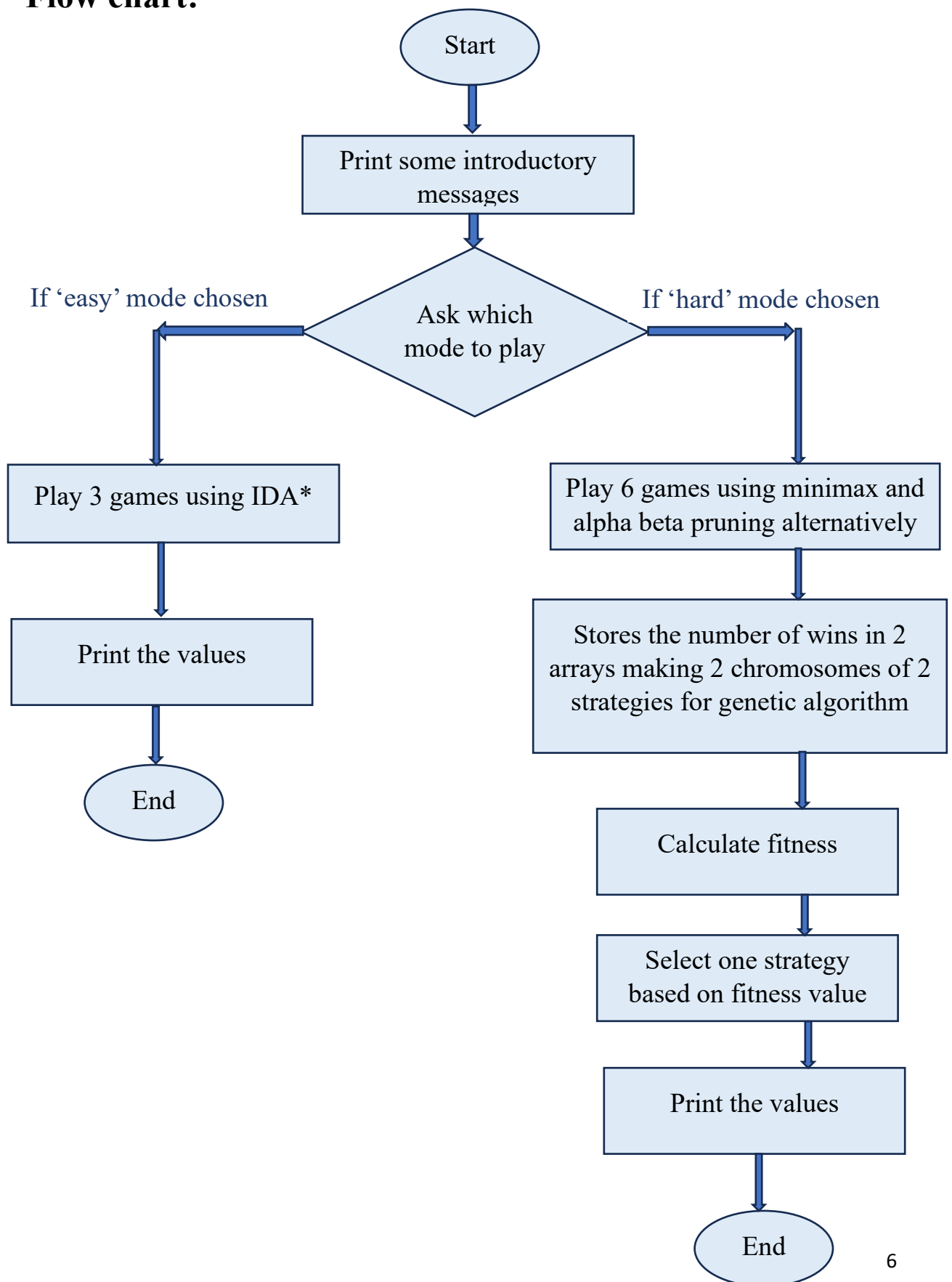


- ⇒ So, from the above demonstration, we can see that, human has picked the last 3 items.
- ⇒ So, human will win this game.

Features:

- ❑ There are two game modes:
 - Easy
 - Hard
- ❑ In **Easy** mode, AI agent plays using IDA* (Iterative Deepening A star) algorithm.
- ❑ In **Hard** mode, AI agent plays using minimax algorithm and alpha-beta pruning.
- ❑ In Easy mode, 3 games will be played and it will show AI agent has won which games.
 - An array is used to store this.
 - When AI agent wins, it will be stored as '1' and when loses, it will be stored as '0'.
- ❑ In Hard mode, total 6 games will be played.
 - 3 games will be played using Minimax and the other 3 games will be played using Alpha-beta pruning.
 - There will be two arrays to store the number of wins in each strategy.
 - Then **Selection** step of **Genetic Algorithm** is applied.
 - Here, two chromosomes will be the two arrays that stored the wins and loses in each game for two different strategies.
 - Then using fitness function, one of the strategies will be suggested.
- ❑ **NB:** Easy mode will not play optimally but Hard mode will be play optimally.

Flow chart:



How to play:

- ✓ The game will be played in console.
- ✓ In this game, the name of the AI agent is **NimX**.
- ✓ When the python file is run, then the AI agent will start interacting with human.
- ✓ First, it will tell its name and then asks the human's name-

```
Hello!!! I am NimX...
```

```
What is your name???
```

- ✓ After human enters name, then it will ask if the human wants to play or not.

```
Do you want to play with me???
```

```
If you want then press 'y' otherwise press 'n'
```

- ✓ If the human presses 'y' then it will give the options of the modes-

```
Choose difficulty level:
```

```
1. Easy (NimX uses IDA*)
```

```
2. Hard (NimX uses Minimax and Alpha-Beta)
```

```
Enter 1 or 2:
```

- ✓ If Easy mode is selected then the game will be started.
 - At first, it will show the initial board.

```
Let's see how the board is looking right now...
```

```
-----
```

```
Pile_no 1: /*\/*\*\
```

```
Pile_no 2: /*\/*\/*\/*\/*\
```

```
Pile_no 3: /*\/*\*\
```

```
-----
```

- Then it will ask the human that from which pile it wants to remove stones and how many-


```
NimX is playing using IDA*.....  
->->->->->->->->->->->->->->->->->->->  
From which pile you want to remove stones, Fiza??: 1  
How many stones do you want to remove (maximum 3)??: 3
```

- After that NimX will make its turn and the status of the board after NimX's turns will be shown-

```
NimX removes 1 stones from pile 2.

After NimX's turn,,,
Let's see how the board is looking right now...
-----
Pile_no 1:
Pile_no 2: /*\/*\/*\/*\/*\
Pile_no 3: /*\/*\/*\/*\
-----
```

- Now it will again ask the human to choose pile number and the number of stones to be removed.
- The game will be continued according to the previous steps.
- The final game will terminate when three games are played and each of the game will terminate when the board becomes empty.
- This is the final result of the game-

IDA* wins: [0, 0, 0]

Genetic algorithm is not applicable in Easy mode.

Genetic algorithm is not applicable in Easy mode.

- Here, '0' means Nimx (AI agent) has lost the game.

- ✓ If Hard mode is selected then the game will be started.
 - 1st, 3rd and 5th games will be played using Minimax and the 2nd, 4th and 5th games will be played using alpha-beta pruning.
 - The games will be played as previously mentioned way.
 - After 6 games are played then the output will be like-

```
Minimax chromosome (wins): [1, 1, 1]
Alpha-Beta chromosome (wins): [1, 1, 1]

1 represents wins and 0 represents lose

Algorithm: Minimax, Fitness: 1.0

Algorithm: Alpha_Beta, Fitness: 1.0

Selected Chromosome: [1, 1, 1] with algorithm: Minimax
```

- Here, 1's means all games are won by NimX (AI agent).
- Then Selection method is applied to select a specific algorithm based on the fitness value.

Implementation:

- I have set the AI agent's name "**NimX**".
- **Constraints:**
 - Pile numbers can be between 2 to 3.
 - Stones in each pile can be between 1 to 6.
 - One can pick maximum 3 stones at each turn from a pile.
- **Main Function (main):**
 - It asks for user input to start the game (yes/no).
 - Depending on the user's choice (var == 'y'),
 - initializes the game parameters (emptyList, randPile, etc.) and proceeds to play games based on the chosen difficulty level (1 for easy with IDA* and 2 for hard with Minimax and Alpha-Beta).
 - Collects wins for each algorithm (ida_wins, minimax_wins, alphabeta_wins).

- After playing the specified number of games, if in hard mode (difficulty == '2'), stores the wins in minimax_chromosome and alphabeta_chromosome.
- **Game Initialization (get_board):**
 - It sets up the initial game board by randomly assigning a number of piles (randPile) with a random number of stones in each pile (randRock).
- **Input Validation (get_valid_input):**
 - It ensures that user input for selecting piles and stones to remove is valid according to the constraints mentioned before proceeding with the game.
- **Game Execution Functions (play_with_MINIMAX, play_with_alpha_beta, play_with_IDA):**
 - It executes the game logic for each strategy:
 - play_with_MINIMAX and play_with_alpha_beta manage turns between the human player (name1) and the AI (NimX) using Minimax and Alpha-Beta pruning strategies respectively.
 - play_with_IDA handles the game when the AI uses the IDA* algorithm.
- **AI Move Strategies (ai_move_minimax, ai_move_alpha_beta, ai_move_ida):**
- It is the implementations for the AI agent's moves based on the strategies:
 - ai_move_minimax and ai_move_alpha_beta determine the best move using Minimax and Alpha-Beta pruning respectively.
 - ai_move_ida employs the IDA* algorithm to find the best move.
- **Genetic Algorithm (genetic_algorithm):**

- If in the hard mode (difficulty == '2'), it initializes a population of strategies (Minimax and Alpha_Beta) with their respective minimax_chromosome and alphabeta_chromosome.
- Then it evaluates the fitness of each strategy based on the wins accumulated (calculate_fitness).
- It selects the best-performing strategy (selection) to potentially evolve and improve in subsequent generations.

So, that's how I have implemented Minimax algorithm, alpha-beta pruning, IDA* algorithm and genetic algorithm in this game.

As in the Easy mode, AI agent plays using IDA* algorithm, so it doesn't give optimal moves.

But in the Hard mode, AI gives optimal moves as it plays using minimax and alpha-beta. In this mode, it is hard to win.

Pseudocode:

import random is added to use randint() function

Define variables and data structures

- minimax_chromosome = [] #Stores wins for Minimax algorithm
- alphabeta_chromosome = [] #Stores wins for Alpha-Beta algorithm

Function definitions

Main function for the Nim game

function main ():

- Initialize game and get player's name and difficulty level
- Print introductory messages and get user input
- ...
- Initialize game board
- Generate random integers for pile number and stones in the pile
- Initialize the number of play rounds

- Initialize the arrays for storing the number of wins – ida wins, minimax wins, alpha-beta wins
- While total games less than games per algorithm:
- Generate initial game board
- **function get_board ():**
 - ...
- if difficulty equals '1':
 - **function play_with_ida ():**
- elif difficulty equals '2':
 - **function play_with_minimax_and_alphabeta ():**
 - Alternate between Minimax and Alpha-Beta for each game
 - ...
 - Store game outcomes in respective chromosomes which will be used in genetic algorithm
 - ...
- Add 1 to the total games to keep track of the number of games played
- #After playing, if in hard mode, apply genetic algorithm
- Initialize population with Minimax and Alpha-Beta chromosomes with minimax wins and alpha-beta wins
- **function apply_genetic_algorithm ():**
 - Initialize the population with two chromosomes minimax and alphabeta
 - Define fitness calculation and selection process

function get_board ():

- Clear the board
- Randomly determine number of piles and stones in each pile
- Append this pile number and stones in the board

function get_valid_input():

- Start a loop that will continue until valid input is received
- Prompt the player to enter the pile number from which they want to remove stones
 - Store the input in a variable called piles

- Remove any leading or trailing spaces from the input
- Prompt the player to enter the number of stones they want to remove (maximum 3)
 - Store the input in a variable called stones
 - Remove any leading or trailing spaces from the input
- Check if both piles and stones have valid inputs:
 - Ensure that piles and stones are not empty
 - Ensure that both inputs are numeric by checking if they consist only of digits
 - Convert piles and stones to integers.
 - Validate the player's input:
 - Ensure that the number of stones is greater than 0 and less than or equal to 3.
 - Ensure that the pile number is within the valid range (between 1 and the number of piles in emptyList).
 - Ensure that the number of stones to be removed is not greater than the number of stones in the selected pile.
 - If the input is valid:
 - Exit the loop.
 - If the input is invalid:
 - Display an error message to the player.
 - Continue the loop to prompt the player again.
- After exiting the loop (meaning valid input has been received):
 - Subtract the number of stones from the selected pile in the emptyList.

Function to perform Minimax algorithm for game strategy

function minimax (emptyList, depth, isMaximizing):

- # Base case: Check if all piles are empty
- if all elements in emptyList are 0:
 - return -1 if isMaximizing else 1
- # If maximizing player's turn
- if isMaximizing:
 - maxVal = negative infinity
 - # Iterate through each pile
 - for each pile in range (length of emptyList):
 - if emptyList[pile] > 0:
 - # Try removing stones from 1 to 3 from current pile
 - for stones in range (1, minimum of 4 and (current pile's stones + 1)):
 - newEmptyList = create a copy of emptyList
 - decrease stones from newEmptyList [pile]
 - eval = minimax (newEmptyList, depth + 1, False)
 - maxVal = maximum of maxVal and eval
 - return maxVal
 - else:
 - # If minimizing player's turn
 - minVal = positive infinity

Iterate through each pile

- for each pile in range (length of emptyList):
 - if emptyList[pile] > 0:
 - # Try removing stones from 1 to 3 from current pile
 - for stones in range (1, minimum of 4 and (current pile's stones + 1)):
 - newEmptyList = create a copy of emptyList
 - decrease stones from newEmptyList [pile]
 - eval = minimax (newEmptyList, depth + 1, True)
 - minVal = minimum of minVal and eval
 - return minVal

Function to determine the best move using Minimax algorithm

function best_move_minimax(emptyList):

- bestMove = None
- bestValue = negative infinity
- # Iterate through each pile
- for each pile in range (length of emptyList):
 - if emptyList[pile] > 0:
 - # Try removing stones from 1 to 3 from current pile
 - for stones in range (1, minimum of 4 and (current pile's stones + 1)):
 - newEmptyList = create a copy of emptyList
 - decrease stones from newEmptyList [pile]
 - moveValue = minimax (newEmptyList, 0, False)
 - # Update best move if current move value is better
 - if moveValue > bestValue:
 - set bestValue to moveValue
 - set bestMove to (current pile + 1, stones)
 - return bestMove

Function to perform alpha-beta pruning algorithm for game strategy

function alpha_beta_algorithm():

- Implement Alpha-Beta pruning algorithm for optimal move selection
- Similar to minimax algorithm but here two new variables alpha and beta are introduced. Their value will also be updated
- If beta becomes less or equal than alpha, then the loop will break
- Alpha is the maximum value and beta is the minimum value

Function to perform IDA* algorithm for game strategy

function ida_star ():

- Set threshold to the value of the heuristic function applied to emptyList
- Set path to an empty list
- While True:
 - Call search with emptyList, g = 0, threshold, and path
 - If search returns 'FOUND':

- Return the first move in the path
- If search returns infinity:
 - Return None (no solution found)
- Update threshold to the value returned by search.

function search ():

- Set f to the sum of g and the heuristic function applied to emptyList
- If f is greater than threshold:
 - Return f
- If emptyList is entirely zeroes (all piles are empty):
 - Return 'FOUND'
- Set minVal to positive infinity
- For each pile in emptyList:
 - If the current pile has stones (emptyList[pile] > 0):
 - For each possible number of stones to remove (from 1 to the minimum of 3 or the number of stones in the pile):
 - Create a copy of emptyList and store it in newEmptyList
 - Remove the selected number of stones from the current pile in newEmptyList
 - Add the move (pile index + 1, number of stones) to the path
 - Recursively call search with the updated newEmptyList, g + 1, threshold, and path
 - If search returns 'FOUND':
 - Return 'FOUND'

- If the returned value is less than minVal, update minVal
- Remove the last move from the path (backtracking)
- Return minVal as the next threshold

function heuristic ():

- Calculate the heuristic value:
 - Return the sum of the values in emptyList

Functions for population initialization and selection

function init_population():

- Create a list of dictionaries where each dictionary contains:
 - "Algorithm" key with values "Minimax" and "Alpha_Beta"
 - "Chromosome" key with corresponding values minimax_chromosome and alphabeta_chromosome
- Return the initialized population list

function calculate_fitness() :

- Calculate wins:
 - Set wins to the sum of the values in chromosome
- Calculate fitness:
 - If totalGames is greater than 0, set fitness to wins divided by totalGames
 - Otherwise, set fitness to 0
- Return the calculated fitness

function fitness_function ():

- Initialize populationFitness:

- Create an empty list populationFitness
- Evaluate each algorithm in population:
 - For each item in population:
 - Extract the Algorithm and Chromosome from item
 - Calculate the fitness value using calculate_fitness function
 - Print the algorithm name and its fitness value
 - Append to populationFitness a dictionary with:
 - "Algorithm": the name of the algorithm
 - "Chromosome": the corresponding chromosome
 - "Fitness_value": the calculated fitness value
- Return the populationFitness list

function selection ():

- Sort populationFitness by fitness:
 - Sort population_fitness in descending order based on "Fitness_value"
- Select the best chromosome:
 - Set selected to the first item in the sorted list (the one with the highest fitness)
- Print the selected chromosome and its corresponding algorithm
- Return the selected item

function genetic_algorithm():

- Call init_population to create the initial population
- Call fitness_function with the population to calculate fitness values
- Select the best chromosome:
 - Call selection with the calculated fitness values to select the best chromosome

- Return the selected item (algorithm, chromosome, and fitness value)

Results:

As this game has 2 modes, Easy and Hard. In easy mode, AI will be playing using IDA* and 3 games will be played. At the end, it will print an array consisting which games are won by the AI.

Similarly in hard mode, total 6 games will be played, 3 using minimax and 3 using alpha-beta pruning. Here also the wins will be stored in two different arrays. These two arrays will be used as chromosomes in genetic algorithm. Then selection will be performed to select a specific method. At the end, it will show the strategies along with its fitness value.

Discussion:

In this project, I have developed a game called Nim game. In this game, both human and the artificial agent can play. I have added two modes: Easy and Hard. In Easy mode, AI agent plays using IDA* algorithm. In Hard mode, AI plays 3 games using minimax algorithms and 3 games using alpha beta pruning algorithm. Basically, in this game, there are some piles available and, in those piles, there are stones available. Each player has to pick the stones. The one who pick the last stone will win the game. I have also added genetic algorithm. As the number of wins are stored in two different arrays for 2 strategies (minimax and alpha-beta), I have treated these arrays as two chromosomes and using a fitness function, I have calculated the fitness value. Then according to the value, one strategy is selected. The values are then printed. Thus, this game is played.

Conclusion:

The Nim game successfully shows the application of IDA* algorithm, Minimax algorithm, alpha-beta pruning and genetic algorithms. The artificial intelligent agent can make proper and correct decision using minimax and alpha-beta pruning algorithm. This helps the human to play the game more attentively. This project

shows how different techniques can be combined and integrated to solve a problem. In future, more techniques can be added to optimize the performance and make it more efficient.

Reference:

- <https://en.wikipedia.org/wiki/Nim#:~:text=Nim%20is%20a%20mathematical%20game,the%20same%20heap%20or%20pile.>
- Lab materials