" Unveiling game of nim's mystery"

Author: ElMehdi Bouktib

University: University of Doha for science and technology

Department: College of computing and information technology

Class: DSAI3203(Fundamentals of AI)

Instructor: MahmoudZadeh, Somaiyeh

Date: March 30, 2024

**Report’s Outline**

1. Abstract
2. Introduction
3. Background
4. AI Game
5. Game of Nim
6. Dynamic Programming
7. Analysis
8. Winning strategy
9. Algorithm Description
10. Implementation Details
11. Experimental phase
12. Summary and Conclusion
13. References

**Abstract**

The game of Nim represents a well-known puzzle where Game artificial intelligence can be applied to create an AI agent to not only play the game but also win it in most cases. In this project dynamic programming has been applied to create a bot that can simulate a player 2 entity that aims to win the game against a human opponent. This paper clarifies the fundamental principles employed in the development of the final model, encompassing the Game of Nim, Game AI, and Dynamic Programming. Through a detailed investigation we delved deep into the details of the winning strategy by an exploration that encompasses a comprehensive analysis of various gameplay scenarios aiming to gain a refined understanding of the dynamics at play. Our ultimate objective is to develop an accurate algorithm that applies our findings to effectively tackle the complexities of the environment and consistently achieve success.

**Intr****oduction**

In the domain of artificial intelligence AI, researchers have always focused on designing artificially intelligent agents competent to perform human-centric activities. Including playing games where they seek applying strategic thinking and decision-making approaches to devise effective plans and outperform opponents. One such environment that illustrates these challenges is the Game of Nim. Originating from China and posing a complex strategic problem in spite of it rules simplicity, it provided an ideal case study for mathematicians, game theorists, and AI researchers. By the end of the exploration, we aspire to instantiate an AI system that is skilled enough to outpace human opponents in this contest. Aligning with the goal mentioned above into building smart entities and through analyzing the problem we seek to contribute to the growth in the field of game AI and to enhance general understanding of AI-driven decision-making in advanced systems. In the subsequent sections, we will discuss the “game of nimming” ’s structure in depth as well as the Game AI field that we will utilize the dynamic programming technique from to solve the dilemma by, then we will dive into the analysis where we examine winning strategies, describe the algorithm before the implementation stage, and finally present experimental findings.

**Background**

i.AI Game

**The field of Artificial intelligence had great leaps since the beginning of the current millennium as it progressed quickly with the advancements made is the field of computational power and hardware efficiency resulting in an applicability in multiple filed such as health, finance, robotics, and games where we witness many applications such as Non playable characters (NPCs), Procedural Content Generation (PCG) and AI driven computer opponent which usually uses reinforcement learning and a range of other techniques and theories like state space search, Bayesian AI and game theory.**

ii.Game of Nim

Game of Nim is a two-player mathematical game originated from ancient China. It starts by having a number of piles (2 to 5 in our case) where each has a variable number of elements (in our case sticks). The players take turns removing object according to the following rules:

* A player can take sticks from a single pile at his turn.
* A player can take 1 to as many sticks as he wants from the chosen pile.

The winner of the contest is the one to take the last stick in the variation studied in this report. Despite having a simple set of rules, the game of nimming poses a complex strategic enigma.

iii.Dynamic Programming

Representing a mathematical and algorithmic framework, Dynamic Programming stands out to be a technique designed in the 1950s by Richard Bellman that is applicable in multiple areas. According to dynamic programming’s Wikipedia page it was defined as:

“It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.” (*Wikipedia*, 2020)

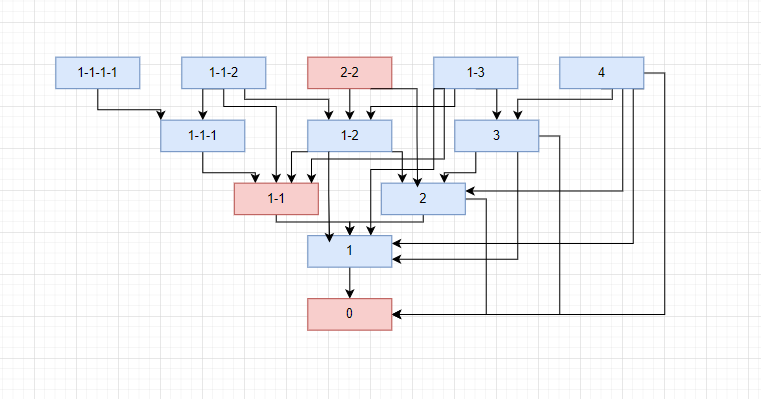
**Analysis**

i. Winning strategy

To figure out the winning strategy, we will start by taking the example of a game of nim where we have 4 sticks in the pile(s) and draw the tree of all possible situations we can get to, we will then label them as a W or L situations where W stand for winning situation and L stands for losing situation. So, the computer should aim to always be in the winning (blue) positions and simultaneously push the opponent to end up in the losing (red) positions. This ensures that all outcomes from these positions favor the computer's victory in each contest.

**Figure 1**

*Possible states (red and bule rectangle) and action (arrows) in a 4 sticks game of Nim*



ii. Algorithm Description

Observing the possible states and actions from previous figure we can notice that all red positions will always yield a 0 when we “xor” (⊕) the integers in it for example 2 ⊕ 2 = 0 and we get the same outcome for 1 ⊕ 1 while all of the other possibilities result in a positive value different that zero (≠ 0). When playing the game of nim, our goal would be to have a position in which applying the ⊕ operand on the integers result in a number different than 0 and most importantly we should push our opponent to a position where the sum of his values will be equivalent to 0. Therefore, we will define a function that will take the current game state as input and follow the discovered pattern before returning a pile number and the number of sticks to remove from it. This method would execute the following algorithm described in pseudocode:

nimSum <- 0

for each pile in piles:

nimSum <- nimSum XOR length of pile

if nimSum = 0:

pileIndex <- random integer between 0 and length of piles - 1

sticksToRemove <- random integer between 1 and length of piles[pileIndex]

return pileIndex, sticksToRemove

for i <- 0 to length of piles - 1:

xorValue <- nimSum XOR length of piles[i]

if xorValue < length of piles[i]:

return i, length of piles[i] - xorValue

nonEmptyPiles <- []

for i <- 0 to length of piles - 1:

if length of piles[i] > 0:

nonEmptyPiles.append(i)

pileIndex <- random choice from nonEmptyPiles

sticksToRemove <- random integer between 1 and length of piles[pileIndex]

return pileIndex, sticksToRemove

iii. Implementation Details

a. findBestMove(piles):

Input: The environment which represents a 2 dimensions matrix where the outer list represents the piles and the inner list the sticks in the piles.

Role: Calculates the best move for the computer player in a game of Nim based on the current state of the piles employing the nim-sum concept to determine the optimal move.

Output: return the chosen pile index and the number of sticks to remove from it.

b. gameWon(piles):

Input: The environment which represents a 2 dimensions matrix where the outer list represents the piles and the inner list the sticks in the piles.

Role: checks if the game has been won (all piles are empty).

Output: return a Boolean value indicating whether the game has been won or not.

c.vsComputer():

Input: ∅ no explicit input

Role: Execute the loop sequence of turns for the human player and computer while handling wrong inputs, updating the environment, and calling the function to show the game progress to player.

Output: the contest outcome (player win or computer win)

d.vsPlayer():

Input: ∅ no explicit input

Role: Execute the loop sequence of turns for the two human player handling wrong inputs, updating the environment, and calling the function to show the game progress to player.

Output: the contest outcome (player 1 win or player 2 win)

e. createGame(container):

Input: container (list) where the piles of sticks will be stored.

Role: Initializes the game by randomly generating the number of piles and the number of sticks in each pile.

Output: doesn’t have an explicit output but it fills the container with the initial state of the game.

f. printGame(container):

Input: The environment which represents a 2 dimensions matrix where the outer list represents the piles and the inner list the sticks in the piles.

Role: Visualize the current state of the game, i.e., the number of sticks in each pile.

Output: The current state of the game.

g. The code as a whole:

* The code implements a game of Nim, a mathematical strategy game, where players take turns removing sticks from piles. The goal is removing the last stick.
* It provides options for playing against the computer or another human player.
* The computer's movement is determined using the findBestMove() function, which employs nim-sum strategy to make optimal moves.
* The game loop continues until one player wins.
* Overall, the code provides a simple yet functional implementation of the game of Nim, offering both single-player and two-player modes.

iv. Experimental phase

After implementing the game in python, a collection of experiences was conducted to assure quality. Starting by testing the wrong input handling for both vsPlayer() and vsComputer() functions as well as the main program where the code passed the test. Then with find best move function, that exceled in winning in most cases as it becomes impossible to defeat as soon as you make a mistake by giving it the opportunity to reach a winning state as it only redirects you to losing steps from there on. However, the player still has some a minimal chance to win, i.e., when the program generates a game with a small number of piles and sticks (like 2-3) that a human can control by sending the computer to losing states but note that any minor mistake will reverse the roles. The state where computer systems start removing sticks wasn’t implemented however minor changes in the code will make it true, in this new case losing will become impossible except if the computer starts in a losing position, then you may have a chance as long as the game is programmed to start in such a case.

**Summary and Concl****usion**

In this research, we explored the application of dynamic programming in developing an AI agent capable of playing and winning the Game of Nim, a classic mathematical game with complex rules despite its simple rules. We began by providing background information on AI in gaming, the Game of Nim itself, and dynamic programming as a problem-solving technique. Using dynamic programming principles, we devised a winning strategy for the AI agent, which involved identifying winning and losing positions and guiding the agent to make optimal moves to secure victory. We implemented the algorithm in Python and conducted experiments to validate its performance. Our investigation into the Game of Nim and the application of dynamic programming yielded promising results opening the door for further experiments with a leverage in environment parameters such as dimensions, agents and collaborations that can be applied in other problems with numerous states and actions scales, multi-agent systems or even human-AI collaboration.

**References**

Nim - Ahead of the Game. (Nov 10, 2021). from <https://www.youtube.com/watch?v=wP-VpBbgF1g>

The Game of Nim - a math game of strategy using matchsticks! (Oct 2, 2020). [Www.youtube.com](http://Www.youtube.com). from <https://www.youtube.com/watch?v=SUh8C387BVU>

*Wikipedia Contributors*. (2019, February 15). Dynamic programming. Wikipedia; Wikimedia Foundation. <https://en.wikipedia.org/wiki/Dynamic_programming>

*Wikipedia Contributors*. (2020, March 21). Nim. Wikipedia. <https://en.wikipedia.org/wiki/Nim>

Combinatorial Game Theory | Set 2 (Game of Nim). (2016, May 21). GeeksforGeeks. <https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/>