|  |  |  |
| --- | --- | --- |
|  | **K. J. Somaiya Institute of Engineering and Information Technology, Sion, Mumbai**  *Accredited ‘A’ Grade by NAAC with 3.21 CGPA*  *3 Programs Accredited by National Board of Accreditation*  *Permanently Affiliated to University of Mumbai,*  *Best College Award by University of Mumbai (Urban Region), ISTE (MH), and CSI (Mumbai)*  *UGC Recognized Institute under Section 2(f) and 12(B) of the UGC Act, 1956* |  |

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Academic Year (2020-2021) Odd Semester VII**

**Course: Artificial Intelligence**

**Experiment No. 06**

**Aim: Implementation of Adversarial Search Techniques.**

6.1 Implement Min Max Algorithm.

**Objectives:**

To make students understand various AI methods like searching and game playing and how to apply them to solve real applications

**Outcomes:**

Develop intelligent algorithms for constraint satisfaction problems and also design intelligent systems for Game Playing

**Theory:**

**6.1 Implement Min-Max Algorithm.**

1. Introduction to Adversarial Search Techniques.

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

1. Game Playing, Min-Max Search, Alpha Beta Pruning:

Game Playing is an important domain of artificial intelligence. Games don’t require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

* Generate procedure so that only good moves are generated.
* Test procedure so that the best move can be explored first.

The most common search technique in game playing is [Minimax search procedure](https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/). It is a depth-first depth-limited search procedure.

Alpha Beta Pruning:

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameters Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called the Alpha-Beta Algorithm.Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.

The two-parameter can be defined as:

1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
3. Algorithm/Pseudocode(Min-Max).

function minimax(node, depth, maximizingPlayer) is

if depth ==0 or node is a terminal node then

return static evaluation of node

if MaximizingPlayer then // for Maximizer Player

maxEva= -infinity

for each child of node do

eva= minimax(child, depth-1, false)

maxEva= max(maxEva,eva) //gives Maximum of the values

return maxEva

else // for Minimizer player

minEva= +infinity

for each child of node do

eva= minimax(child, depth-1, true)

minEva= min(minEva, eva) //gives minimum of the values

return minEva

Problem statement :

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there are n stones/coin in a pile. On each player's turn, that player makes a move

consisting of removing any non-zero square number of stones in the pile.

Also, if a player cannot make a move, he/she loses the game.

Given a positive integer n. Return True if and only if Alice wins the game otherwise return False,

assuming both players play optimally.

Example 1:

Input: n = 1

Output: true

Explanation: Alice can remove 1 stone winning the game because Bob doesn't have any moves.

Example 2:

Input: n = 2

Output: false

Explanation: Alice can only remove 1 stone, after that Bob removes the last one winning the

game (2 -> 1 -> 0).

Example 3:

Input: n = 4

Output: true

Explanation: n is already a perfect square, Alice can win with one move, removing 4 stones (4

-> 0).

Example 4:

Input: n = 7

Output: false

Explanation: Alice can't win the game if Bob plays optimally.

If Alice starts removing 4 stones, Bob will remove 1 stone then Alice should remove only 1 stone

and finally Bob removes the last one (7 -> 3 -> 2 -> 1 -> 0).

If Alice starts removing 1 stone, Bob will remove 4 stones then Alice only can remove 1 stone

and finally Bob removes the last one (7 -> 6 -> 2 -> 1 -> 0)

CODE:

n=int(input())

import math

def winnerSquareGame(n):

if n == 0:

return False

for x in range(1, math.floor(math.sqrt(n)) + 1):

if not winnerSquareGame(n - (x \* x)):

return True

return False

k=winnerSquareGame(n)

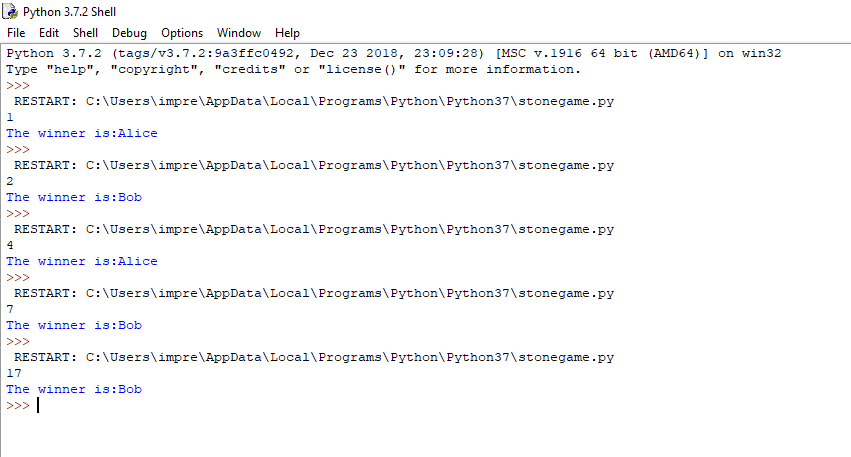
if k ==0:

print("The winner is:Bob")

else:

print("The winner is:Alice")

**OUTPUT:**



**Conclusion:**

From this experiment, we understood adversarial search techniques, min-max algorithm and game playing. Also, we implemented a min-max algorithm for the game of stones successfully.