K. J. Somaiya College of Engineering, Mumbai-77

Batch: A-4 **Roll No.:** 16010122151
Experiment No. 06
Group No :- 5
Signature of the Staff In-charge with date

Title: Implementation of Adversarial Search Algorithm

Objective: Implementation of Alpha-Beta Pruning algorithm

Expected Outcome of Experiment:

Course Outcome	After successful completion of the course students should be able to
CO 2	Analyse and solve problems for goal based agent architecture (searching and planning algorithms).

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russel and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata Mcgraw Hill Publications**
3. **www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf**
4. **http://cs.lmu.edu/~ray/notes/asearch/**
5. **www.cs.cornell.edu/courses/cs4700/2011fa/.../06_adversarialsearch.pdf**

Pre Lab/ Prior Concepts: Two/Multi player Games and rules, state-space tree, searching algorithms and their analysis properties

Historical Profile: - The game playing has been integral part of human life. The multiplayer games are competitive environment in which everyone tries to gain more points for himself and wishes the opponent to gain minimum.

The game can be represented in form of a state space tree and one can follow the path from root to some goal node, for either of the player.

New Concepts to be learned: Adversarial search, min-max algorithm, Alpha-Beta pruning,

---

Adversarial Search Concept:-

We use Adversarial Search to make optimal decisions in a two-player competitive game. The MinMax algorithm is employed to simulate all possible moves and counter-moves to determine the best outcome, and Alpha-Beta Pruning is applied to reduce the number of nodes evaluated, improving efficiency.

Alpha-beta pruning algorithm: MinMax algorithm

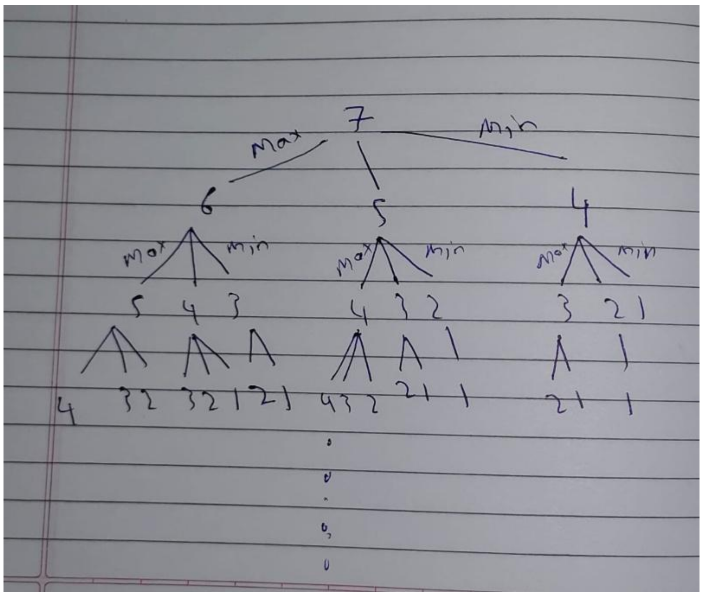
Chosen Problem:

- **Starting Number: 7**
- **Players: Two (Max and Min), taking alternate turns**
- **Allowed Moves: Subtract 1, 2, or 3 from the current number**
- **Goal: The player who reaches exactly 1 wins the game**

Problem Description:

This problem is a simplified turn-based, zero-sum game where two players alternately subtract 1, 2, or 3 from a starting number (7). Using the **MinMax algorithm**, we model all possible future moves and outcomes, aiming to select the best move at each turn. The **Alpha-Beta Pruning** enhancement significantly reduces the number of evaluated game states, making the search efficient while preserving the optimal strategy.

Solution tree for chosen Problem:



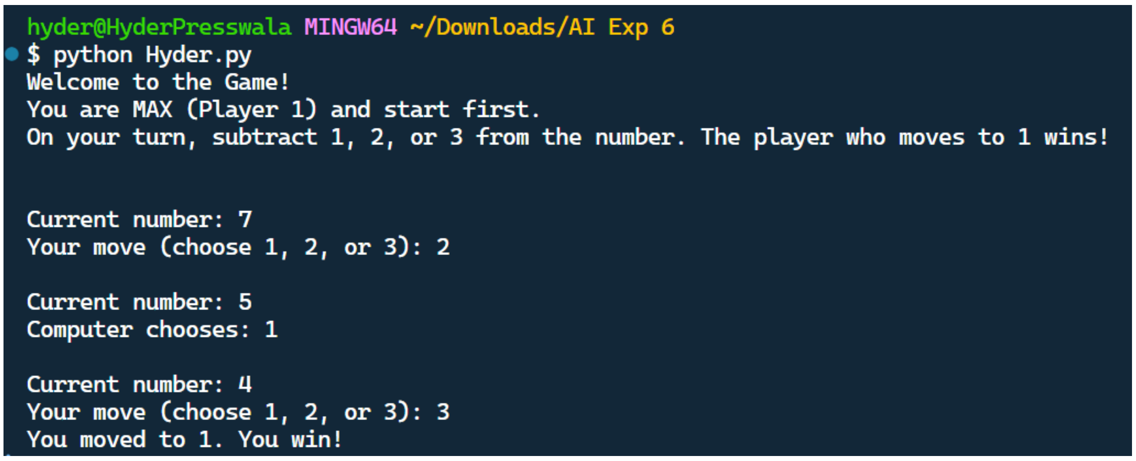
Source Code:-

```python
def alpha_beta(n, is_maximizing, alpha, beta):
    if n == 1:
        return 1 if not is_maximizing else -1

    if is_maximizing:
        max_eval = float('-inf')
        for move in [1, 2, 3]:
            if n - move >= 1:
                eval = alpha_beta(n - move, False, alpha, beta)
                max_eval = max(max_eval, eval)
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break
        return max_eval
    else:
        min_eval = float('inf')
        for move in [1, 2, 3]:
            if n - move >= 1:
                eval = alpha_beta(n - move, True, alpha, beta)
                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break
        return min_eval
```

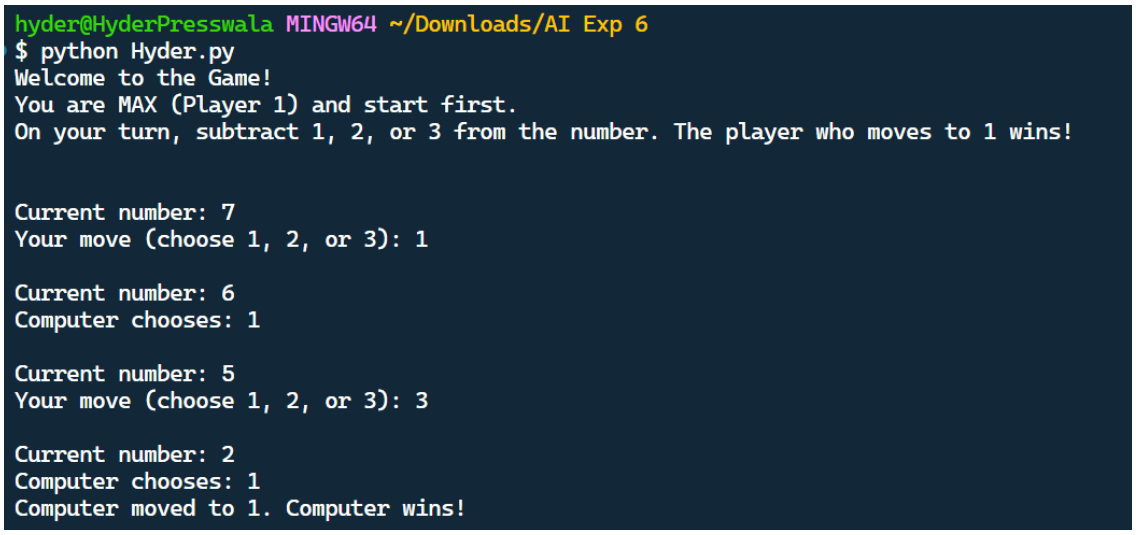
```python
def computer_move(n):
    best_score = float('inf')
    best_move = None
    for move in [1, 2, 3]:
        if n - move >= 1:
            score = alpha_beta(n - move, True, float('-inf'),
float('inf'))
            if score < best_score:
                best_score = score
                best_move = move
    return best_move


def play_game():
    n = 7
    print("Welcome to the Game!")
    print("You are MAX (Player 1) and start first.")
    print("On your turn, subtract 1, 2, or 3 from the number. The
player who moves to 1 wins!\n")

    is_user_turn = True

    while n > 1:
        print(f"\nCurrent number: {n}")

        if is_user_turn:
            move = int(input("Your move (choose 1, 2, or 3): "))
            while move not in [1, 2, 3] or n - move < 1:
                print("Invalid move. Try again.")
                move = int(input("Your move (choose 1, 2, or 3): "))
            n -= move
            if n == 1:
                print("You moved to 1. You win! ")
                return
        else:
            move = computer_move(n)
            print(f"Computer chooses: {move}")
            n -= move
            if n == 1:
                print("Computer moved to 1. Computer wins! ")
                return

        is_user_turn = not is_user_turn
```

```python
# Start the game
play_game()
```

Output Screenshots:



```
hyder@HyderPresswala MINGW64 ~/Downloads/AI Exp 6
$ python Hyder.py
Welcome to the Game!
You are MAX (Player 1) and start first.
On your turn, subtract 1, 2, or 3 from the number. The player who moves to 1 wins!


Current number: 7
Your move (choose 1, 2, or 3): 2

Current number: 5
Computer chooses: 1

Current number: 4
Your move (choose 1, 2, or 3): 3
You moved to 1. You win!
```



```
hyder@HyderPresswala MINGW64 ~/Downloads/AI Exp 6
$ python Hyder.py
Welcome to the Game!
You are MAX (Player 1) and start first.
On your turn, subtract 1, 2, or 3 from the number. The player who moves to 1 wins!


Current number: 7
Your move (choose 1, 2, or 3): 1

Current number: 6
Computer chooses: 1

Current number: 5
Your move (choose 1, 2, or 3): 3

Current number: 2
Computer chooses: 1
Computer moved to 1. Computer wins!
```

Conclusion:- This experiment demonstrates how optimal moves in a turn-based subtraction game can be determined using the state space tree and alpha-beta pruning to ensure a winning strategy.

Post Lab objective Questions:

1. **Which search is equal to minmax search but eliminates the branches that can't influence the final decision?**
   a. Breadth-first search
   b. Depth first search
   c. Alpha-beta pruning
   d. None of the above

Answer: Alpha-beta pruning

2. **Which values are independent in minmax search alogirthm?**
   a. Pruned leaves x and y
   b. Every states are dependant
   c. Root is independent
   d. None of the above

Answer: Pruned leaves x and y

Post Lab Subjective Questions:

1. What is the main purpose of the Alpha-Beta pruning algorithm in game trees?

   The main purpose of the Alpha-Beta pruning algorithm is to optimize the Minimax algorithm by eliminating branches in the game tree that cannot possibly influence the outcome. This reduces the number of nodes evaluated, thereby improving the algorithm's efficiency and making it more suitable for deeper game tree searches.

2. How does Alpha-Beta pruning improve the efficiency of the Minimax algorithm?

Alpha-Beta pruning improves efficiency by avoiding the evaluation of moves that won't affect the final decision. It does this by keeping track of the best values that the maximizer (α) and minimizer (β) can guarantee. If it finds that one side's best move is worse than a previously examined move, it prunes that branch. This can reduce the time complexity from O(b^d) to O(b^(d/2)) in the best case, where b is the branching factor and d is the depth.

3. Explain the terms **alpha** and **beta** in the context of the Alpha-Beta pruning algorithm.

   Alpha (α): The best value that the **maximizer** currently can guarantee at that level or above. It starts at negative infinity and increases during the traversal.

   Beta (β): The best value that the **minimizer** currently can guarantee at that level or above. It starts at positive infinity and decreases during the traversal. When $\alpha \geq \beta$, further evaluation at that node is stopped (pruned).

4. What condition must be met for a node to be pruned during Alpha-Beta pruning?

A node is pruned when the current α value (maximizer's best so far) is greater than or equal to the current β value (minimizer's best so far):
If α ≥ β, then prune the branch.
This means that further exploration won't influence the final decision, so it's safe to ignore that subtree.

5. Compare and contrast Alpha-Beta pruning with the standard Minimax algorithm.

Feature	Minimax	Alpha-Beta Pruning
Efficiency	Evaluates all nodes in the tree	Skips branches that won't influence the outcome
Time Complexity	O(b^d)	O(b^(d/2)) in best case
Final Decision	Same as Alpha-Beta	Same result as Minimax
Speed	Slower due to full tree traversal	Faster due to pruning
Implementation	Simpler	Slightly more complex, requires tracking α and β values