Session 4: Game Playing in AI

Course Title: Computational Intelligence
Course Code: CSC401A

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: *vaishali.c*s.et@msruas.ac.in

AI Applications

- Game Playing
- Natural Language Processing
- Computer Vision
- Speech Recognition
- Robotics
- Expert Systems

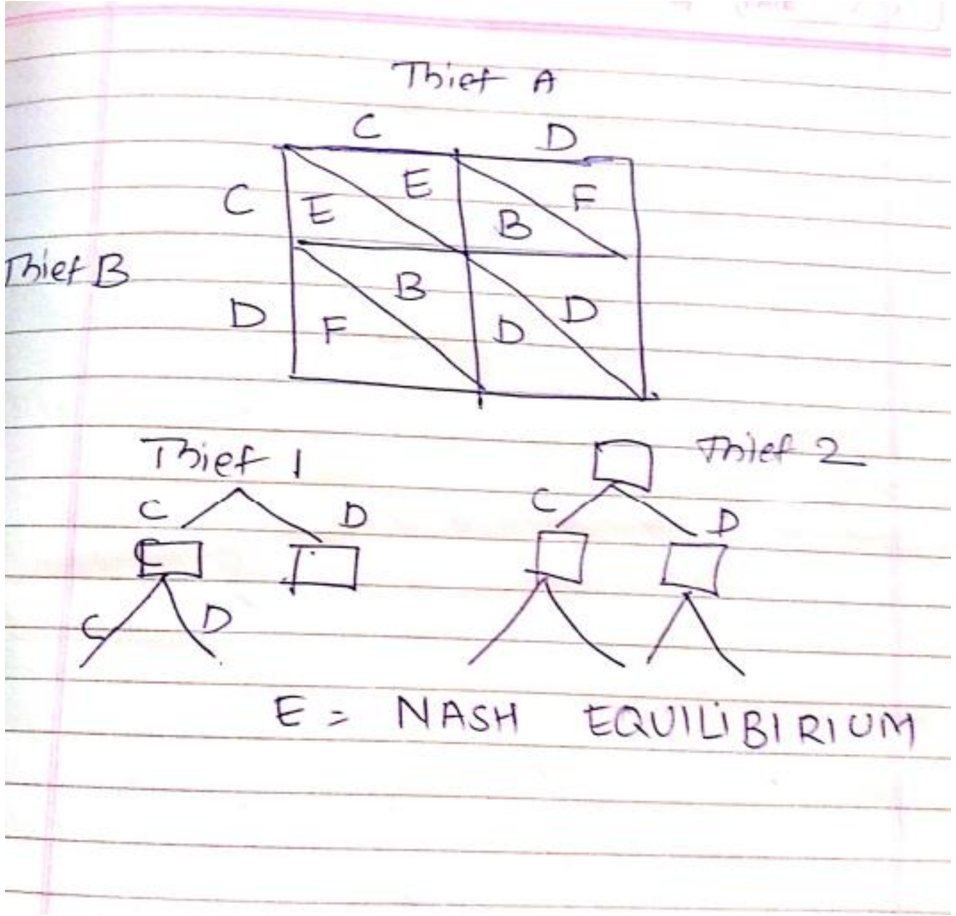
Game Playing

Game playing is a search problem defined by:

- Initial state
- Successor Player
- Goal Test
- Path Cost/Utility/Pay Off Function

- AI helps in solving different game problems by following improved strategies. With the present AI techniques it is not possible to differentiate whether a human or machine is playing.

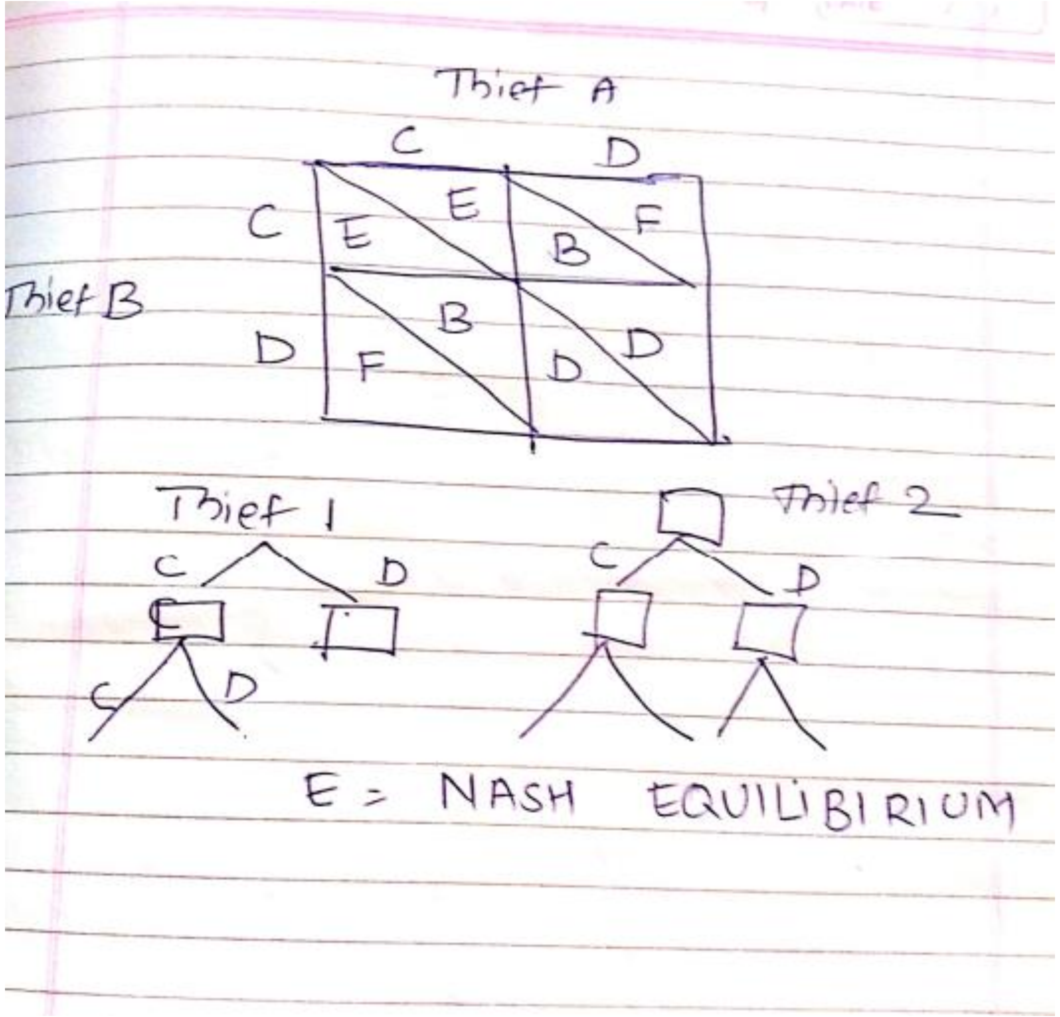
Games



Thief A
C
D
C
E
E
B
F
Thief B
D
F
B
D
D
Thief 1
C
D
C
D
Thief 2
C
D
E = NASH EQUILIBIRIUM

Features of Games

- Zero sum, alternate moves deterministic board games are of AI interest. Zero sum or price war games are those where one person's gain is another person's loss.
- We will discuss games that are deterministic, zero sum, two player, alternate moves, and with complete information.
- These games can be represented by layered tree.



Game Playing

- A game must feel like natural
- Obey laws of the game
- Characters aware of the environment
- Path Finding
- Decision making
- Planning

The AI gaming is about illusion of human behaviour. It is repeating and smart to certain extent

AI games get integrated in the environment. AI gives illusion of a human body languages.

AI Games

Core AI needs various Computer Science disciplines such as:

- Knowledge-based systems
- Machine Learning
- Multi-agent systems
- Computer Graphics and Animation
- Data Structures

AI Games

1. Strategy Games: Real time strategy (RTS), Time-based strategy (TBS), Helicopter View
2. Role Playing Games: Single player, multi-player
3. Action Games: First person shooter, first person sneaker
4. Sport Games
5. Simulations
6. Adventure Games
7. Puzzle Games

Minimax Algorithm

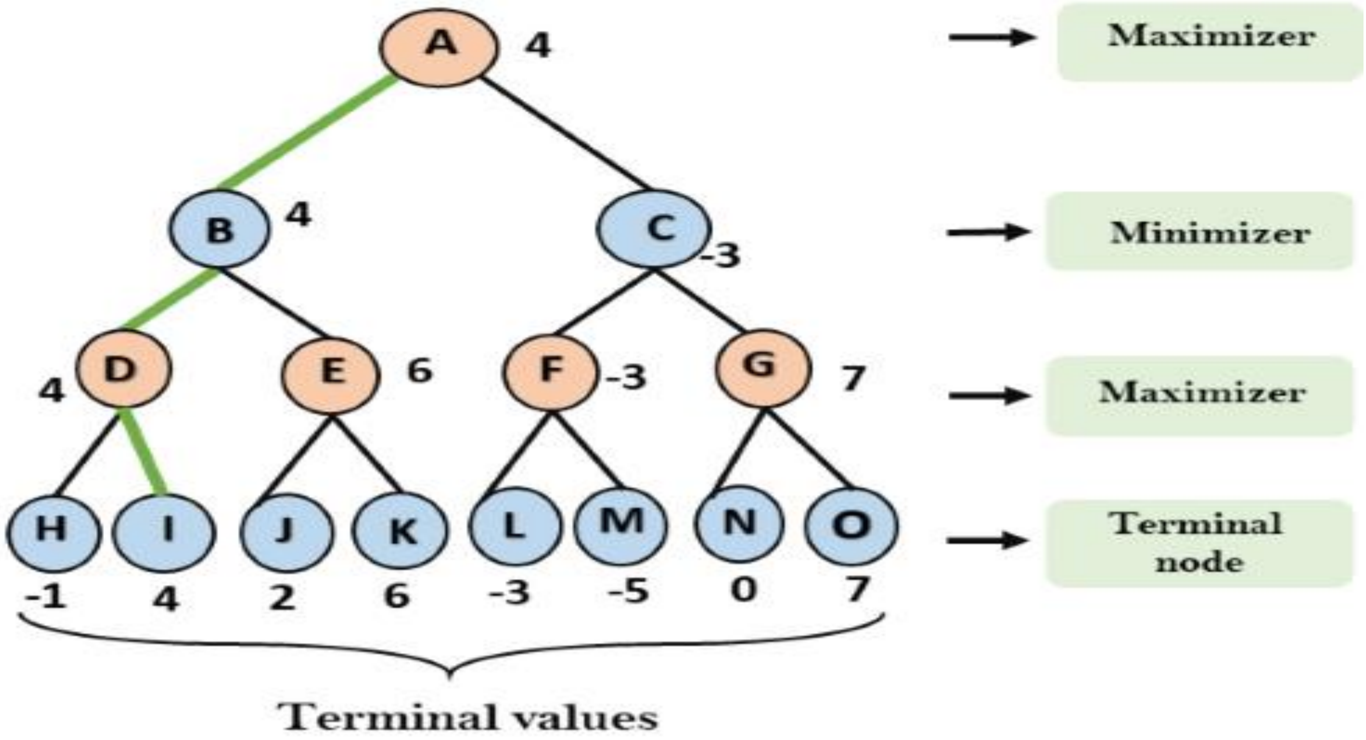
- Mini-Max **algorithm** uses recursion to search through the game-tree
- Min-Max **algorithm** is mostly used for game playing in **AI**. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game.
- Both the players fight it as the opponent player gets the **minimum** benefit while they get the maximum benefit.
- **Minimax** is a decision-making **algorithm**, typically used in a turn-based, two player games. The goal of the **algorithm** is to find the optimal next move. In the **algorithm**, one player is called the maximizer, and the other player is a minimizer.

Min-Max Algorithm

1. Construct the complete game tree
2. Evaluate scores for leaves using the evaluation function
3. Back-up scores from leaves to root, considering the player type:
    1. For max player, select the child with the maximum score
    2. For min player, select the child with the minimum score
4. At the root node, choose the node with max value and perform the corresponding move

- The **time complexity** of minimax is O(b^d) and the **space complexity** is O(bd), where b is the number of legal moves at each point and d is the maximum depth of the tree.

Min-Max Example



- **Properties of minimax algorithm:**
- **Complete?** **Yes (if tree is finite)**
- **Optimal?** **Yes (against an optimal opponent)**
- **Time complexity?** **$O(b^d)$**
- **Space complexity?** **O(bd) (depth-first exploration, if it generates all successors at once)**
-

For chess, b ≈ 35, m ≈ 80 for "reasonable" games

→ exact solution completely infeasible

d – maximum depth of the tree; b – legal moves

Mini-Max with Pruning

- Limitations of Original Mini-Max: Generally not feasible to traverse entire tree, Time limitations

- Key Improvements necessary: Use evaluation function instead of utility Evaluation function provides estimate of utility at given position

- Alpha/beta pruning

- The Alpha-beta **pruning** to a standard **minimax** algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision.
- The unnecessary visiting of nodes is reduced.
- Hence by **pruning** these nodes, it makes the algorithm fast.

Alpha Beta Pruning

Principle:

If a move is determined worse than another move already examined, then there is no need for further examination of the node.

1. Initialize **alpha** = -infinity and **beta** = infinity as the worst possible cases. ...
2. Start with assigning the initial values of **alpha** and **beta** to root and since **alpha** is less than **beta** we don't **prune** it.
3. Carry these values of **alpha** and **beta** to the child node on the left.

Minimax with alpha-beta pruning

- Rules:
- α is the best (highest) found so far along the path for Max
- β is the best (lowest) found so far along the path for Min
- Search below a MIN node may be alpha-pruned if
- its β <= α of some MAX ancestor
- Search below a MAX node may be beta-pruned if its
- α >= β of some MIN ancestor.

Advantages of Pruning

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning b(e.g., chess, try captures first, then threats, froward moves, then backward moves…)
- With "perfect ordering," time complexity =O($b^{d/2}$)
- *d* doubles depth of search that alpha-beta pruning can explore

References

- https://www.educative.io/edpresso/how-to-implement-a-breadth-first-search-in-python