* Four main approaches of Artificial Intelligence

1] Thinking Humanly: The cognitive modeling approach
- Make machines think like humans by mimicking how the human brain works.
- Understand and simulate human thought processes using cognitive science and psychology.
- ex. Modeling memory, learning, and problem-solving as human-do.
- Techniques:- Cognitive architecture, neural network

2] Acting Humanly (The turing test approach)
- Make machines behave like humans.
- The computer would need the following capabilities: NLP, knowledge representation automated reasoning, machine learning

3] Thinking Rationally (Laws of Thought)
- make machines think logically using clear rules.
- Use Formal logic and reasoning to solve problems and make decisions.
- ex. Expert systems using 'if-then' rules.

4] Acting Rationally (Rational Agent Approach)
- A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

Date / /
Page L&W

* Limits of AI

- The Argument from informality
- The Argument from Disability
- Lack of common sense
- Data Dependency
- Bias and fairness
- Lack of creativity
- Ethical concern
- Interpretability

* Ethics of AI

- Ethics of AI involves ensuring that the design, development, and use of artifical intelligence are responsible, fair and beneficial to society

1] Transparency
- AI system should be explainable and understandable.

2] Fairness and Non-Discrimination
- AI should not reinforce or amplify biases.

3] Privacy
- AI should respect user data and comply with data protection laws.

4] Safety and Security

5] Accountability

Date / /
Page L&W

* Structure of AI
- refers to the components and agent layers that together enable a machine to sense, reason, learn and act intelligently

1] Input layer
- This layer is responsible for sensing and collecting data from the environment.

2] knowledge layer (Representation and storage)
- This layer stores facts, rules and structured information that the AI system uses for reasoning and decision-making.

3] Reasoning and processing layer
- This is the core layer where the AI thinks, learns and makes decision

4] Action layers (Actuation)
- These layer enables the AI to interact with the environment

5] learning layer (Feedback and Adaptation)
- This layer allows AI to improve over time using experience or new data.

* Artificial life (A-life)
- Artificial life refers to computer simulations, robots, or algorithms that mimic or replicate life-like behaviours found in natural systems (e.g. insects, cells, animals)

- It focuses on creating autonomous agents that exhibit life like properties: self-replication, adaption, learning or evolution

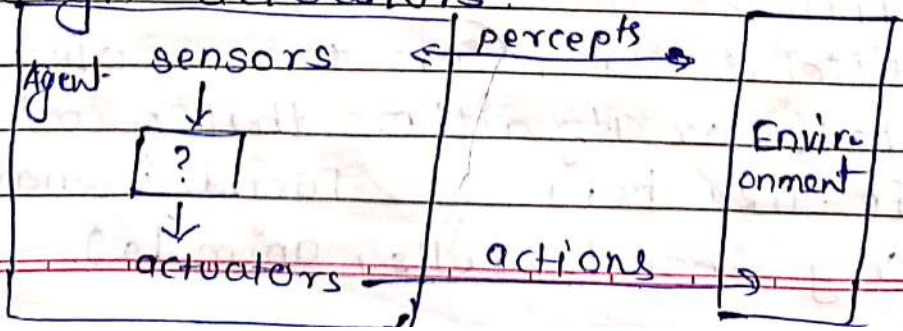
A-life systems can learn or evolve through:

1. Emergent Behaviours
Complex behavior that arises from simple rules followed by individual agents - without any central control.
- In A-life: Agents adapts over time through interactions - no learning algorithm is directly coded; the behavior evolves.

2] Rules :- Pre-defined if-then conditions that dictate the behavior of agents.

3] Expert Systems
- AI systems that use a knowledge base of rules and a reasoning engine to solve problems like a human expert.

* Agents and Environments
- An 'agent is anything that can be viewed as its perceiving its environment through sensors and acting upon that environment through actuators.



* Rationality

- What is rational at any given time depends on four things:
  - The performance measure that defines the criterion of success.
  - The agent's prior knowledge of the environment.
  - The actions that the agent can perform
  - The agent's percept sequence to date

- Rational-agent:- for each possible percept sequence, a ~~rontim~~ rational agent should select an action that is expected to maximizie its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

* PEAS :- Performance, Environment, Actuator and sensor.

- It is a framework used to define the task environment of an intelligent agent.

1] Performance Measure :- How we evaluate the agent's sucess. It's the goal of the agent.

2] Environment :- The surroundings in which the agent operates.

3] Actuators :- The tools/actions the agent uses to affect the environment

4] Sensors : The agent's perception tools to recieve data from the environment.

* Task Environment

- The task Environment refers to everything that affects the performance of an intelligent agent. It includes how the agent perceives and interacts with the world. To design a good agent must understand the properties of its task environment.

7 key properties.

(a) Observable vs. Partially Observable.

- Observable :- agent can access the complete state of the environment at each point in time. ex. chess (Board is fully visible)

(+) Partially Observable : The agent has incomplete information about the environment.
ex. self-driving car (can't see around corners)

(b) Deterministic vs stochastic

- Deterministic : The next state is completely determined by the current state and action
ex. puzzle game

- Stochastic : The next state involves randomness or uncertainty
ex. Driving in traffic (uncertain behaviour of other drivers)

(c) Episodic vs Sequential

- Episodic: Each task is independent of previous ex Image classification (one image at a time)
- Sequential: Current decisions affect future decisions.
  ex. Chess, navigation

(d) Static vs Dynamic

- Static: Environment does not change while the agent is thinking. ex. crossword puzzle
- Dynamic: Environment can change while the agent is choosing actions. ex. stockmarket

(e) Discrete or vs Continuous

- Discrete: A limited number of distinct, clearly defined states and actions.
  ex: Board games like Tic-Tac-Toe
- Continuous: State and actions are not countable; involve ranges ex: Robotic control, driving

(f) Single agent vs Multi-agent

- single agent - One agent is operating in the environment ex. sudoku solver
- Multi-agent - Multi-agents may cooperate or compete
  ex. Multiplayer games, auctions.

(9). known vs Unknown
- known - The agent knows the rules and consequences of actions ex chess (rules are predefined)
- Unknown ! The agent must learn how the environment works.
ex. learning a new game without instructions

* Agent Program

- The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behaviour from a smallish ~~prod~~ program rather than from a vast table
- There are 4 basic kinds of agent programs that embody the principles underlying almost all intelligent systems:

- Simple reflex agents
- Model-based reflex agents;
- Goal-based agents; and
- Utility-based agents.

- Each kind of agent program combines particular components in particular ways to generate actions.

(a) Simple reflex agent :- The simplest kind of agent is the simple reflex agent. These agents select actions on the basis of the

current percept, ignoring the rest of the percept history
ex. the vaccum agent who is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt.

(b) Model-based reflex agents
- The most effective way to handle partial observability is for the agent to keep track of the part of the world it can't see now. That is, the agent should maintain some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.

(c) Goal-based agents
- A goal-based agents is an intelligent agent that takes actions by considering future consequences of its actions with the objective of achieving a specific goal or set of goals. Unlike simple reflex agents that react to current conditions, a goal-based agent evaluates different possible future actions and chooses the one that best leads to the desired outcome.
- It performs search or planning to find actions that lead to the goal.

(d) Utility - Agent

- A utility based agent is an ite' intelligent agent that chooses actions based on a utility function that evaluates how desirable a particular state is. Instead of just aiming to achieve a goal, it tries to maximize overall satisfaction or performance, considering trade-offs and preferences.

(e) learning Agent

- A learning Agent is an intelligent agent that can improve its performance over time by learning from its experiences and environment. Unlike fixed-rule agents, learning agents adapt and evolve their behaviour based on feedback.

• key components of a learning Agent

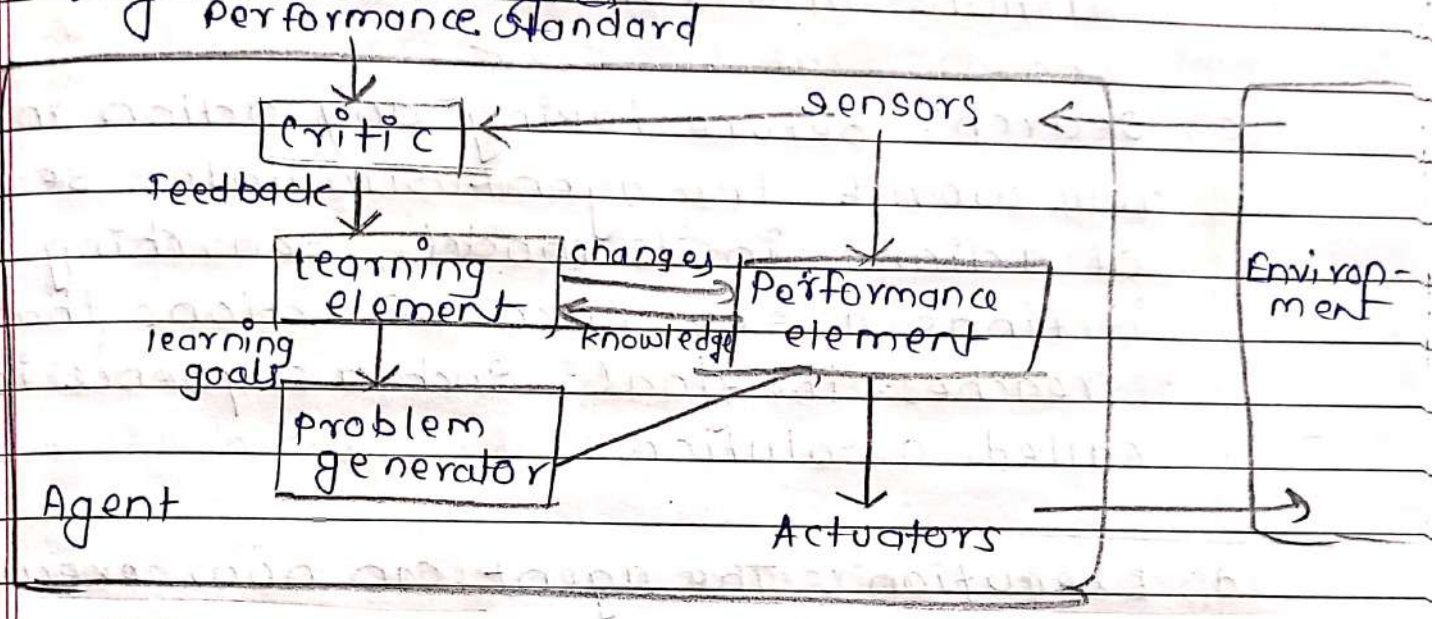
1] learning Element :- Which is responsible for making improvements.

2] Performance Element :- Make decisions and takes actions based on current knowledge.

3] Critic :- Evaluates the agent's performance and gives feedback. Helps the learning element understand how well the agent is doing.

4] Problem Generator

- Suggests exploratory actions that help the agent learn better. Encourages the agent to try new strategies.



chapter 3 Solving problems by searching

- An agent may need to plan ahead: to consider a sequence of actions that form a path to a goal state. Such an agent is called is called a problem-solving agent. and the computational process it undertakes is called search.

- The agent can follow this four-phase problem-solving process:

(a) Goal formulation: The agent adopts the goal of reaching Bucharest. Goals organize behaviour by limiting the objectives and hence the actions to be considered.

(b) problem formulation:
clearly defining the problem by identifyin the initial stage state, goal state, available actions and constraints

(c) Search: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a solution.

(d) Execution: The agent can now execute the actions in the solutions, one at a time.

* Search problem in AI is a formal representatio of a task where an agent must find a sequence of actions that leads from a start state to a goal state.

components :-

(a) Initial state :- starting point of agent
(b) Goal state :- The desired final state (s) the agent wants to reach.
(c) Actions :- All possible moves or decisions the agent can make
(d) Transition model :- Rules that define the result of an action applied to a state.

(e) Path cost :- cost to travel from one state to another (e.g. time, steps, distance)

(f) ~~distance~~ : State space : All the possible states the agent can reach by applying actions

Note :- A sequence of actions forms a path, and a solution is a path from the initial state to a goal state.
- optimal soln :- : has the lowest path cost among all solutions.

* Uninformed search strategies (Blind search)
- Uninformed search algorithm / strategies do not use any domain-specific knowledge (heuristics). They only use the information available in the problem definition such as the start state, goal state and legal moves

+Characteristics :- (a) No knowledge of the goal's location beyond its definition.
(b) Explore the state space systematically
(c) focus on completeness, optimality, time and space complexity.

1. Breadth-first search :- In which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.

2] Dijkstra's algorithm
- algorithm is used to find the shortest path from a source anode to all other nodes in a weighted graph with non-negative edge weights

3] Depth-first search and the problem of memory
- DFS always expands the deepest node in the tree
- DFS is not cost-optimal; it returns the first soln it finds, even if it is not cheapest

4] Depth-limited and iterative deepening search
- a version of depth-first search in which we supply a depth ~~time~~, limit l and treat all nodes at depth l as if they had no successors. The time complexity of is $O(b^l)$ and space complexity is $O(bl)$

5] Bidirectional search
- An algorithms we have covered so far start at an initial state and can reach any one of multiple possible goal states. An alternative approach called bidirectional search simultaneously searches forward from the initial state and backwards from the goal state(s), hopping that two searches will meet.

* Informed (Heuristic) search strategies
- one that uses domain-specific hints about the locations of goals - can find solutions more efficiently than an uninformed strategy.
The hints come in the form of a heuristic function, denoted $h(n)$

$h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state.

(a) Greedy best-first search
- Greedy best-first search is a form of best-first search that expands first the node with the lowest $h(n)$ value - the node that appears to be closest to the goal - on the grounds that this is likely to lead to a solution quickly.
So the evaluation function $f(n) = h(n)$.

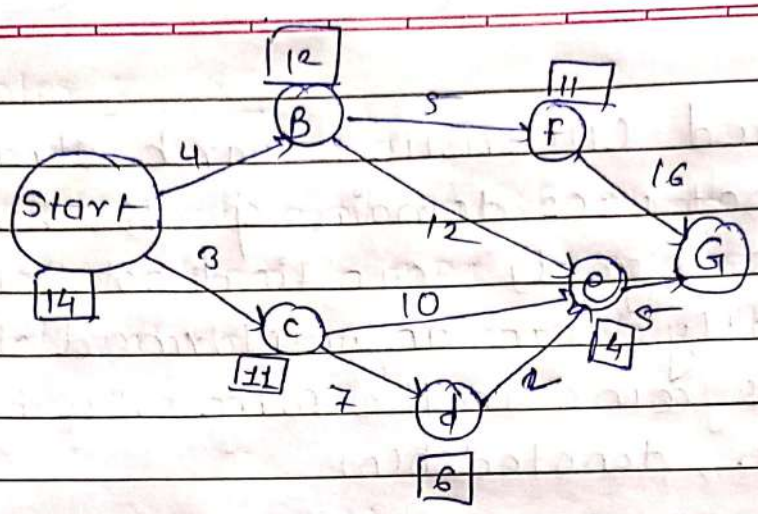
(b) A* search
- The most common informed search algorithm is A* search & is best-first search that uses the evaluation function

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the path cost from the initial state to node n, and $h(n)$ is the estimated cost of the shortest path from n to a goal state, so we have

$f(n)$ = estimated cost of the best path that continues from n to a goal.

Ex.



$f(S) = 0 + 14 = 14$

S → B : 4 + 12 = 16

S → C : 3 + 11 = 14

SC → e : 3 + 10 + 4 = 17

SC → d : 3 + 7 + 6 = 16 → SCd → e : 3 + 7 + 2 + 4 = 16

SB → f : 9 + 5 + 11 = 20

Sb → e : 4 + 12 + 4 = 20

Scde → G : 12 + 5 + 0 = 17

(c) Weighted A*

- Weighted A* is a variant of the A* search algorithm used in Artificial Intelligence and pathfinding. It speeds up the search by prioritizing paths that are estimated to be closer to the goal, even if they might not be optimal. This is done by increasing the weight of the heuristic function.

$$f(n) = g(n) + w \cdot h(n)$$

A* search : $g(n) + h(n)$ $(w = 1)$

Uniform-cost search: $g(n)$ $(1 \times 1 = 0)$

Suboptimal paths: $g(n) + w \times (h(n))$ $(w > 1)$

Greedy best-first search: $h(n)$ $(w = \infty)$

* Memory-bounded search

- Memory-bounded search algorithms are search methods that limit memory usage to avoid exhaustion while still aiming to find complete and, if possible, optimal solutions in large or infinite search spaces.

Types

(a) IDA* (Iterative Deepening A*)

- Combines the space-efficiency of DFS with the optimality of A* by repeatedly deepening the f-cost threshold.

(b) RBFS (Recursive Best-first search):

- Uses limited memory by exploring paths recursively and backtracking when a better alternative path is found.

(c) SMA* (Simplified Memory-based bounded A*)

- Works like A* but uses a fixed amount of memory by discarding the least promising nodes and regenerating them if needed

Chapter 5: CSP

- A CSP (constraint satisfaction problem) is a type of problem in AI and CS where the goal is to find values for a variables that satisfy a set of constraint

- CSP consists of three components, X, D and C
  - X is set of variables, $\{x_1, \ldots, x_n\}$
  - D is a set of domains, $\{D_1, \ldots D_n\}$. One for each variable
  - C is set of constraints that specify allowable combinations of ~~variable~~ values.

A ~~se~~ domain, D_i, consists of a set of allowable values, $\{v_1, \ldots v_k\}$, for variable x_i.

- Different variables can have different domains of diff. sizes. Each constraint C_j consists of pair $<$scope, rel$>$, where scope is a tuple of variables that participate in the constraints and rel is a relation that defines the values that ~~those satisfy the constraint~~ those variables can take on.

ex x_1, x_2 → variable

domain → $\{1, 2, 3\}$

constraint → x_1 must be greater than x_2

written as

$< (x_1, x_2), \{(3,1), (3,2), (2,1)\} >$

or

$< (x_1, x_2), x_1 > x_2 >$

* CSPs deal with assignments of values to variables.

- consistent :- an assigment that does not violate any constraints
- complete assignment :- in which every variable is assigned a value.
- soln :- a consistent and complete assignment
- Partial assigment :- is one that leaves some variables unassigned, ~~and a partial soln~~
- Partial soln :- is partial assigment that is consistent

* Constraint propagation:
It is the process of inferring variable domain reductions by enforcing constraints to make the CSP more consistent.

purpose :- (a) Reduce the search space
(b) Detect inconsistencies early
(c) speed up csp solving

ex. $x \in \{1, 2, 3\}$ $Y \in \{2, 3\}$
constraint : $x \neq y$

Constraint propagation will revise the domains:
- If $x = 2$, then $y \neq 2$
- This eliminates 2 from Y's domain, and vice versa.

Eventually
$X = \{1, 2\}$
$Y = \{3\}$

(a) Node consistency

- A single variable is node-consistent if all the values in the variables domain satisfy the variable's unary constraints.

ex: $x \in \{1,2,3,4\}$

Unary constraint : $x > 2$

applying node consistency :

- Remove values that don't satisfy the constraint
- ∴ so, the revised domain of x becomes $\{3,4\}$

(b) Arc consistency

- A variable x is arc consistent with another variable y (denoted as an arc $x \to y$) if and only if:

For every value in the domain of x, there is atleast one compatible value in the domain of y that satisfies the ~~array~~ binary constraint bet^n x and y

ex: $x \in \{1,2,3\}$

$y \in \{2,3\}$

constraint $x < y$

but for $x = 3 \to y = ?$

so, $x = 3$ is removed

$x \in \{1,2\}$

$y \in \{2,3\}$

Now the ~~arc~~ $x \to y$ is arc consistent.

(c) Path consistency

- If CSP is path consistent if for any pair of variables (x, Y) and for every pair of values (x ∈ Dom(x), y ∈ Dom(y)) that satisfy the the constraint between x and y, there exists a value z ∈ Dom(z) such that the constraints bet' x and z, y and z are also satisfied.

eg. x, y, z

x ∈ {1,2}

y ∈ {1,2}

z ∈ {1,2}

- constraints
  - x = y
  - y = z

The path consistency checks that:

x = z (since x = y and y = z implies x = z)

(d) K-consistency

- A csp is strongly k-consistent if it is k-consistent and is also (k-1) consistent, (k-2) consistent, all the way down to 1 consistent.

* Backtracking search for CSPs

key functions:

(a) select_unassigned_variable: choose the next variable to assign (can use heuristics lik MRV - Minimum Remaining value)

(b) order_domain_values: Return the domain values in an order (can use heuristics like least constraining value)

(c) is consistent: check if the value assignment doesn't violate any constraints with the current partial assignment

- Enhancements to improve Efficiency:

1. Forward checking: After assigning a variable, eliminate inconsistent values from neighbours domains.
2. Constraint Propagation (like AC-3): Enforce arc consistency to reduce domains before and during search.
3. Variable ordering:
   - MRV: choose the variable with the fewest legal value.
   - Degree Heuristic: choose the variable involved in most constraints
4. Value ordering:
   - Least Constraining value: Try value that constrain others the least.

chapter 6 : Adversarial search and games

* Two player zero-sum games

- A two-player zero-sum game is a type of game in which.

- There are two players competing against each other.
- One player's gain is exactly the other player's loss
- The total payoff is always zero.

* Optimal Decision in Games

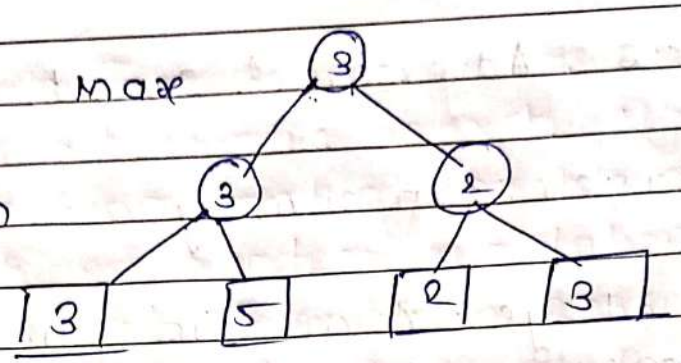
- In game theory, an optimal decision is a strategy that maximizes a player's outcome assuming that the other player is also playing optimally.

(a) Minimax search algorithm

- The minimax algorithm is used in two-player, turn-based, zero-sum games like Tic-Tac-Toe, chess, etc to find the optimal move for a player assuming that the oppenent also plays optimally.

- max tries to maximize the score (best for itself)
- Min tries to minimize the score (best for opponent)
- It explores all possible moves to a certain depth, assigns a score, and then backtracks to choose the best move

ex. max



* Alpha-Beta Pruning

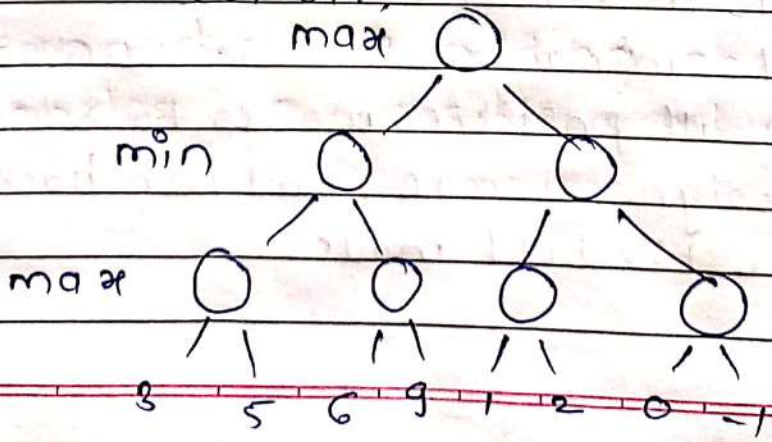
- Alpha-Beta Pruning is an optimization technique for the Minimax algorithm used in two-player, zero-sum games like chess or tic-tac-toe. It reduces the number of nodes evaluated in the game tree by pruning branches that cannot influence the final decision.

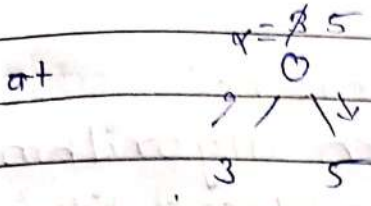
(a) Alpha: The maximum score that the maximizing player (usually max) is gauranteed so far.

(b) Beta: The maximum score that the minimizing player (usually min) is gauranteed so far.

(c) Prunning: If one branch proves to be worse than a previously explored one, it is cut off.

ex





- at 3 max sets alpha = 3
- at 5 $\alpha = \max(3, 5) = 5$
- This max returns 5 to min.
- at 6 - alpha = 6
- at 9 - alpha = 9
- this max returns 9 to min
- Now min(5, 9) = 5
- so left subtree returns 5 to root

for Right substree

- at 1 $\alpha = 1$
- at 2 $\alpha = 2$
- this max returns 2 to min
- ~~at 0 = $\alpha = 0$~~

Now, at the second min node, Beta = 2 (so far)
Now min needs to evaluate the right max, but it already knows the opponent-(max) can gaurantee 5, which is greater than 2 → Prune!

* Heuristic Alpha-Beta search

- Heuristic Alpha-Beta search is an optimization of the minimax algorithm that uses alpha-beta pruning to eliminate unnecessary branches and a heuristic evaluation funct^n to estimate the value of non-terminal nodes at a limited search depth. It is commonly used in complex two-player games where exploring the game tree is infeasible. This approach balances decision quality and computational efficiency.

* Monte carlo Tree search (MCTS) -

- MCTS is a heuristic search algorithm used in ~~dece~~ decision-making problems like games. It combines random simulations with tree-based search to explore possible moves and outcomes. MCTS builds a partial game tree by simulating many random playouts and selecting moves that maximize long-term success. It balances exploration of new moves and ~~explorat~~ exploitation of known good moves.

- Monte Carlo tree search does that by maintaining a search tree and growing it on each iteration of the following four steps.

(a) Selection:

Starting from the root node, select child nodes recursively using a selection policy (like UCT - Upper Confidence Bound for trees) until a leaf node is reached (a node not fully explored)

(b) Expansion

If the selected node is not a terminal state, expand it by adding one or more child nodes (i.e, possible next moves)

(c) Simulation:
- From the newly added node, simulate a random playout (game is played randomly unit it ends) to get a possible outcome (win / loss / draw)

(d) Backpropagation:
- Propagate the result of the simulation back up the tree, updating the statistics (e.g. visit count, win rate) of each node on the path

* (Upper Confidence Bound) UCT
- is a selection strategy used in Monte Carlo tree search (MCTS) to balance exploration and exploitation. It helps choose which node to explore next by assigning a score to each child node based on both:
  - The avg reward (exploitation)
  - The no. of times the node ~~for~~ has been visited (exploration).

$$UCT(i) = \bar{x}_i + c \cdot \sqrt{\frac{\ln N}{n_i}}$$

$\bar{x}_i$ = Avg reward of child i
N: Total visits to the parent node
n_i: Visits to child node i
c: Exploration constant (e.g. $\sqrt{2}$)

* Stochastic games

- A stochastic game is a generalization of both Markov decision processes (MDPs) and repeated games. It models a situation where multiple players interact in a dynamic environment, and the outcome of their actions influences future states of the game probabilistically.

• key components:

- States (S): Different possible situations the game can be in.
- Players (N): Two or more players who make decisions.
- Actions (A): Each player choose an action from a set of possible actions.
- Transitions function (T): Determines the probability of moving from one state to another based on current state and actions.
- Reward function (R): Each player gets a reward depending on the ~~same~~ state and chosen actions.
- Discount factor (γ): Future rewards may be discounted by to reflect time preference

* Limitation of Game Search Algorithms

- Exponential time complexity make deep search impractical for large games.
- Limited lookahead can miss long-term strategies or traps.
- Relies on heuristic functions, which may be inaccurate or game-specific.
- Struggles with randomness and games with hidden information.
- High memory consumption due to large search trees.
- Does not learn from experience unless combined with learning methods.
- Difficult to parallelize efficiently, limiting scalability.

Chapter 7 Logical Agents

* Knowledge-Based Agents (KBA)

- A knowledge-Based Agent is an Intelligent agent that uses a knowledge base of facts and rules to make decisions. It operates by:

1. Representing knowledge about the world.
2. Reasoning using inference to derive new knowledge.
3. Acting based on the conclusions.

key terms:

1. knowledge Representation language
- A formal language used to represent knowledge in a structured and understandable format for machines.
ex. prepositional logic, first order logic

2. Axiom:
- A statement or rule that is assumed to be true without proof. These are the foundational truths in the knowledge base.

3. Inference:
- The process of deriving new facts or conclusions from known facts and axioms using logical reasoning.

4. Background knowledge:
- General facts or common-sense knowledge that the agent uses in addition to perceptual input.

* Building knowledge based Agent

Two Approaches
1. Declarative
2. procedural

1] Declarative Approach.
- Focuses on what is true
- knowledge is expressed as facts and rules in a logical format.
- Uses inference engines to derive conclusion from the knowledge base

ex.
R1 : All humans are mortal
R2 : Socrates is a human

Infer :- Socrates is mortal

2] Procedural Approach:
- Focuses on how to do things
- knowledge is embedded in procedures, rules, or functions (like in a program)
- less general-purpose; optimized for specific types of queries or tasks.

ex. if x == 'socrates':
    return True

* Wumpus World
- The Wumpus World is a classic environment used in AI to test knowledge-based agents
It's a 4x4 grid world here:
  - The agent explore cells.
  - The goal is to find gold and avoid pits and the Wumpus (a monster).
  - The agent can sense things like breeze (near pit), stench (near Wumpus), and glitter (gold).

In Wumpus world

1] Performance measure
- Safely grapping the gold
- Avoiding death (by Wumpus or pit)
- Using minimal actions
- Coming out of the cave alive

2] Environment (E)
- The 4x4 grid of rooms
- Gold (gold), Pits (danger), Wumpus (enemy)
- Walls (boundaries)

3] Actuators (A)
- Move forward
- Turn left/right
- grab gold
- shoot an arrow
- climb out of the cave

4] Sensors
- stench (Wumpus nearby)
- Breeze (Pit nearby)
- Glitter (Gold in current cell)
- Bump (Ran into a wall)
- Scream (Wumpus is dead)

* Logic:- Logic is formal study of reasoning and valid inference. It provides a structured way to ~~repeat~~ represent

Facts, rules, and relationships and allows deriving new conclusions from existing information

Key features of logic:

1. Syntax: Rules for forming valid statements
2. Semantics: Meaning of those statements
3. Inference rules:- Methods to derive new truth

1] Prepositional logic
- is a simple form of logic where:
  - statement are either true or False
  - These prepositions are combined using logical connectives like AND, OR, NOT, etc.

a] syntax: defines rules for forming valid sentences in prepositional logic.
(i) prepositional symbols: P, Q, R (represent simple statements)
(ii) connectives:

~ (NOT)
∧ (AND)
∨ (OR)
→ (IMPLIES)
↔ (IFF or if and only if)

eg (~P ∧ (P ∧ Q) → R

b] Semantics : tells us what each sentence means in terms of truth values

ex.

P = True Q = True R = false.

Evaluate : $(P \wedge Q) \rightarrow R$

$P \wedge Q$ = True $\wedge$ True = True

True $\rightarrow$ false = false

c] Inference Rule :- define to derive conclusions from known facts

ex $P \rightarrow Q$

= If P then Q

* Effective propositional Model checking

- Propositional Model checking is a method use to determine if a given propositional logic sentence is true in all possible models. It's mainly used in knowledge-based agents and AI reasoning systems

Steps

1. Enumerate all possible models (truth assignment for all variables).
2. Evaluate KB and α in each model
3. check entailment: If α is true in all models where KB is true, then KB $\models \alpha$.

Technique : DPLL

* DPLL Algorithm (Davis - Putham - Longemann - Loveland)
- is a complete, backtracking - based search algorithm used for solving SAT esatisfiability problems in prepositional logic)
- It improves on basic ~~is~~ truth table checking by using smart pruning techniques, making it highly efficient for checking whether a formulae in CNF (conjuctive Normal form) is satisfiable.

key components of DPLL:

a] Backtracking Search :
- Try assignments, and backtrack if conflict is found.

b] Unit Propagation :
- IF a clause has only one unassigned literal, assign it to a satisfy the clause.

c] Pure literal Elimination :-
- If a variable appears with only one polarity (only P or only ~P), assign it accordingly.

d] Splitting Rule :-
- choose a variable, assign a value (True/ False), and recurse.

* Agent based on propositional logic
- An agent based on propositional logic uses symbols and logic inference to make decisions. It works in a percept-action loop where it receives percepts, updates knowledge, and then acts using logical reasoning

• Current state of the World
- The agent maintains a knowledge Base (KB) that stores facts about the world in propositional logic.
- Based on current percepts, it updates the KB and infers new facts about the world

• Hybrid agent
- combines logic-based reasoning with planning and reactive behavior

• Logical state Estimation
- Logical state estimation is the process of using inference to estimate the state of the world when not all facts are directly observable.

Chapter 8 First Order logic

- is also called as Predicate logic, is an extension of Propositional logic that is more expressive

- while propositional logic deals with whole statements, FOL breaks them down into objects, properties and relations bet^n objects.
- It allows reasoning about individuals and their relationships.
- It can express statements like "Every human is mortal" which propositional logic cannot do easily

* Symbols and interpretations

- The basic syntactic elements of first-order logic are the symbols that stand for objects, relations and functions. The symbols, therefore, come in three kinds:
  (a) constant symbols - stands for object
  (b) predicate symbols → stands for relation
  (c) function symbols → stand for function.
- In addition to its objects, relations, and functions each model includes an interpretation that specifies exactly which objects, relation and function are referred to by ~~which~~ by symbols

* Term: is a logical expression that refer to an object. Constant symbols are terms, but it is not always convenient to have a distinct symbol to name every object.

* Quantifiers in first-order logic.
- Quantifiers allow us to express statemen about multiple objects in the domain rather than individual ones.
Two types

(a) Universal Quantifiers (∀)
- is usually pronounced "for all..."
"For all x, if x is a king, then x is person"
The symbol x is called a variable. A variab is a term all by itself, and as such can also serve as the argument of a function
E
ex. ∀x (P(x))
Meaning :- P(x) is true for all x in the domain

(b) Existential qualification (∃)
- Universal qualification makes statements about every object. Similarly, we can make a statement about some object without naming it, by using an existentia quantifier.
ex. ∃x (P(x))
∴ There is at least one x in the domain for which P(x) is true

* Equality :- First order logic includes one more way to make atomic sentences, other than using a predicate and term as described earlier. we can use the equality symbol to signify that

two terms refer to the same object
ex. Father(John) = Henry

* key terms in first-order logic

(a) Domain: In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge.

(b) Assertions
- Assertions are sentences added to the knowledge base to represent facts, rules, or relationships.
- They are assumed to be true and help the agent build knowledge about the world.
- This operation is sometimes referred to as "Tell"

ex
Facts: Human(Socrates)
Father(John, Mary)
Rules: $\forall x$ (Human(x) → Mortal(x)

(c) Queries
- A ~~questio~~ query is a question posed to the knowledge base.
- The system uses inference to determine if the query can be logically concluded from the assertions in the KB.
- This is called an 'ASK' operation

ex. Mortal (Socrates)?
→ Is socrates mortal?

- $\exists x$ (Human(x) $\wedge$ mortal(x))?
→ Is there a human who is mortal?

* kinship Domain.
- refers to the set of relationships among individuals based on family ties such as parent, child, sibling, spouse, etc.
- It is often used as a standard example domain in logic and AI to demonstrate knowledge representation, reasoning and inference.

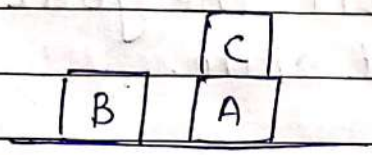
ex. Parent(x, y) → x is parent of y
Mother(x, y) → x is mother of y

* Chapter 11 Automated planning

* Classical planning.
- is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.
- we have seen two approaches to this task.
(a) problem solving agent
(b) hybrid propositional logical agent.
- Both have limitation.
(1) require a heuristic ~~scar~~ function and hand-written rules.
(2) require large state space.

- to overcome above limitations planning researchers have invested in a factored representation using a family of languages called PDDL, the planning Domain Definition Language

ex. three Box A, B and C.



correct plan.
Unstack (C, A)
Put Down (C)
Pick up (B)
Stack (B, C)
Pick up (A)
Stack (A, B)

* Algorithms for classical planning

(a) forward state-space search for planning
- forward state-space search is a classical planning algorithm that starts from the initial state and applies actions forward until a goal state is reached.

(b) backward state-space search ~~in~~ for planning
- is planning technique that starts from the goal state and works backward to find a sequence of actions that lead to the initial state.

- 1. Start from the goal condition
2. Look for actions that can achieve the goal (i.e actions whose effects satisfy some part of the goal)
3. Then regress the goal through that action replace the goal with the preconditions of that action
4. Repeat this until the goal becomes true in the initial state.

* Planning as Boolean satisfiability.
- Planning as SAT is a powerful technique that translates a classical planning problem into a Boolean satisfiability problem (SAT). It uses propositional logic to encode the planning domain and goal, and then applies a SAT solver to find a sequence of actions (a plan) that achieves the goal.

* Planning as Search
- is a method in AI where the planning problem is formulated as a search problem. The goal is to find a sequence of actions (plan) that transforms an initial state into a goal state, by searching through the space of possible states using classical search algorithms like Breadth-first search (BFS) or A*

* STRIPS planning
- is a classical planning approach where the world is represented as a set of logical predicates, and actions are described by their preconditions, and the effects they have on the world when executed.

Each action is defined as:

Action (name,
precondition: {$P_1, P_2 \ldots$},
add effects: {$Q_1, Q_2, \ldots$},
delete_effects: {$R_1, R_2 \ldots$})

* Inference and Resolution for problem solving: ppt
Dec 18