# Advanced Hands-On Course: Fundamentals of Artificial Intelligence

## Course Overview

This advanced, project-based course is designed for experienced learners to deepen their practical understanding of Artificial Intelligence fundamentals. It emphasizes hands-on implementation and structured applications of core AI concepts, enabling students to design and build intelligent systems capable of problem-solving, decision-making, and knowledge representation in complex, real-world scenarios.

## Weekly Summary

- **Week 1:** Explore diverse intelligent agent architectures and their interaction with various environments, focusing on how agents perceive, reason, and act.
- **Week 2:** Delve into advanced search algorithms, including informed and uninformed strategies, and master adversarial search for developing sophisticated game AI.
- **Week 3:** Understand Constraint Satisfaction Problems (CSPs) and formal logic, applying techniques for inference and automated reasoning to solve complex combinatorial problems.
- **Week 4:** Conclude by studying classical planning techniques for autonomous agents and exploring knowledge representation through ontological engineering for building intelligent systems.

## Week 1: Advanced Intelligent Agents and Environments

Concepts:

- **Intelligent Agent:*An autonomous entity that perceives its environment through sensors and acts upon that environment through actuators to achieve goals. Unlike simple programs, agents can exhibit rational behavior, aiming to maximize their performance measure.
- **Agent Architectures:*Different designs for intelligent agents. A** Simple Reflex Agent*reacts directly to current percepts. A **Model-Based Reflex Agent*maintains an internal model of the world to track unobserved aspects and predict action effects. A** Goal-Based Agent*uses explicit goals and planning to achieve desired

states. A **Utility-Based Agent*aims to maximize a "utility" function, which quantifies the desirability of outcomes, allowing for nuanced decision-making with trade-offs.

• **Rationality:*An agent is rational if it acts in a way that maximizes its expected utility, given its current percepts, knowledge, and available actions. This involves careful design of the performance measure to align with the desired behavior.

• **Environment Properties:*Environments can be characterized by their properties, which significantly influence agent design. Key properties include:

• **Fully Observable vs. Partially Observable:*Whether the agent can see the complete state of the environment.

• **Deterministic vs. Stochastic:*Whether the next state is entirely predictable from the current state and action.

• **Episodic vs. Sequential:*Whether actions in one episode affect future episodes.

• **Static vs. Dynamic:*Whether the environment changes while the agent is deliberating.

• **Discrete vs. Continuous:*Whether states and actions are finite/countable or take continuous values.

• **Single-Agent vs. Multi-Agent:*Whether the environment contains one or multiple intelligent agents.

• **State-Space Representation:*A formal model of the problem environment, comprising all possible states, actions the agent can perform, and the transitions between states caused by actions. This formalization is crucial for enabling agents to plan and reason.

Example: Consider a simulated robotic arm in a factory automation scenario. The environment is a grid where the arm can move, pick up, and place different colored items. A utility-based agent aims to sort items into designated bins with maximum efficiency, minimizing wasted moves and maximizing throughput. The environment is partially observable (the arm might only "see" its immediate vicinity), stochastic (items might appear randomly), and dynamic (items can be moved by other processes). The agent's utility function might weigh correct sorting heavily, while penalizing energy consumption and time taken. Its internal model would track the location of items it has seen and the current state of its arm.

Case Study: In advanced smart home energy management systems, utility-based agents are employed to optimize energy consumption. These agents learn user preferences, track energy usage patterns, and autonomously adjust thermostat settings, lighting, and appliance operation. Their utility function balances user comfort and energy cost, dynamically adapting to real-time electricity prices, weather forecasts, and user schedules. This system acts within a partially observable (e.g., cannot directly "see" all heat flows) and dynamic environment.

Exercise: Describe how a simple reflex agent would differ from a goal-based agent in the context of a self-driving car managing a traffic light. What specific percepts and internal structures would each agent prioritize?

Tip/Pitfall: A common pitfall in agent design is defining a performance measure that doesn't truly align with the desired behavior. For example, an agent rewarded solely for reaching a goal might ignore safety or efficiency. Ensure your performance measure incentivizes the *quality* of the outcome, not just its existence.

# Week 2: Advanced Search Techniques and Adversarial Gaming AI

Concepts:

- **Uninformed Search:*Search algorithms that explore a state space without using any information about the goal. Examples include** Breadth-First Search (BFS)**, which explores layer by layer, guaranteeing optimality for shortest paths in unweighted graphs;** Depth-First Search (DFS)**, which explores as deeply as possible before backtracking; and** Iterative Deepening Search (IDS)**, which combines the completeness and optimality of BFS with the memory efficiency of DFS.
- **Informed Search:*Search algorithms that use a** heuristic function*to estimate the distance or cost to the goal, guiding the search more efficiently.
- **Heuristic Functions:*A function, *h(n)*, that estimates the cost from state *nto the goal. A heuristic is** admissible*if it never overestimates the actual cost to the goal, crucial for A\search optimality. A heuristic is **consistent*if its estimate for a path segment is less than or equal to the actual cost plus the estimate from the next node. A** dominant*heuristic is generally better as it provides a more accurate estimate, leading to fewer nodes explored.
- **A\Search:*An optimal and complete informed search algorithm that combines the actual cost from the start node to the current node (*g(n)*) with the estimated cost from the current node to the goal (*h(n)*). It expands the node with the lowest *f(n) = g(n) + h(n)*. Optimality is guaranteed when the heuristic is admissible.
- **Adversarial Search (Game Theory):*The study of search problems where an agent's actions are influenced by the actions of an opponent who also acts optimally to minimize the agent's gain (or maximize their own). These problems are typically represented by** game trees**.
- **Minimax Algorithm:*A recursive algorithm used in two-player, zero-sum games to find the optimal move for a player, assuming the opponent plays optimally to minimize the first player's score. It works by evaluating all possible future game states up to a certain depth and propagates values up the game tree, alternating between maximizing for the agent and minimizing for the opponent.
- **Alpha-Beta Pruning:*An optimization technique for the Minimax algorithm that eliminates branches of the game tree that do not need to be evaluated because they cannot possibly affect the final decision. This significantly reduces the computational cost, allowing the search to explore deeper game trees.

Example: Consider developing an AI player for the classic board game Connect Four. The game state can be represented as a 2D array. Moves involve dropping a checker into a column. An effective Minimax algorithm with Alpha-Beta Pruning would be used. The AI would construct a game tree up to a certain depth, with nodes representing board states and edges representing moves. An **evaluation function** would assign a score to each terminal (or depth-limited) board state, considering factors like potential winning lines, blocked opponent lines, and central control. Minimax would then propagate these scores upwards, and Alpha-Beta Pruning would cut off branches where it's clear the opponent would choose a different path, leading to a better outcome for them.

Case Study: Beyond simple board games, adversarial search techniques are fundamental to the success of AI in more complex games like chess and Go. Deep Blue's victory over Garry Kasparov and AlphaGo's success against Go champions leveraged sophisticated search algorithms combined with advanced evaluation functions and, more recently, neural networks. These systems meticulously explore vast game trees, often employing

highly optimized Alpha-Beta variants, to identify optimal or near-optimal moves.

Exercise: For a simple Tic-Tac-Toe game, trace a portion of the game tree and demonstrate how Alpha-Beta Pruning would cut off a branch that doesn't need to be explored. Assume the AI is the maximizing player.

Tip/Pitfall: A common pitfall in adversarial search is designing a weak evaluation function. Even with perfect search (Minimax), if the function that scores the board state is inaccurate, the AI will make suboptimal decisions. The quality of the heuristic or evaluation function is paramount.

# Week 3: Constraint Satisfaction, Logic, and Inference Systems

Concepts:

- **Constraint Satisfaction Problem (CSP):*A problem defined by a set of** variables**, each with a defined** domain*of possible values, and a set of **constraints*that specify which combinations of values are allowed for subsets of variables. CSPs are a powerful framework for declarative problem-solving, where you describe *whata solution looks like rather than *howto find it.
- **CSP Algorithms:**
- **Backtracking Search:*A fundamental algorithm that incrementally assigns values to variables. If an assignment violates a constraint, it "backtracks" to the last valid assignment and tries a different value.
- **Constraint Propagation:*Techniques that reduce the domains of unassigned variables by enforcing local consistency.** Arc Consistency (AC-3)*is a common technique that ensures for every pair of variables connected by a constraint, any value in one variable's domain has a consistent value in the other's domain. This prunes inconsistent values early, dramatically speeding up search.
- **Heuristics for Backtracking:*Heuristics like** Minimum Remaining Values (MRV)**, which prioritizes variables with fewer legal values, and** Least Constraining Value (LCV)**, which chooses values that rule out the fewest choices for neighboring variables, improve search efficiency.
- **Local Search for CSPs:*Algorithms like** Min-Conflicts*start with a complete (but potentially inconsistent) assignment and iteratively modify one variable's value to reduce the number of constraint violations.
- **Knowledge-Based Agents:*AI systems that rely on a knowledge base (KB) to represent information about the world and use inference mechanisms to derive new conclusions or answer queries.
- **Propositional Logic:*A simple formal language for representing true/false statements (propositions) and combining them with logical connectives (AND, OR, NOT, IMPLIES).** Entailment*means one sentence logically follows from another.
- **First-Order Logic (FOL):*A more expressive logical language that allows representation of objects, properties (predicates), relationships, and quantifiers (universal 'for all' and existential 'there exists').
- **Inference in FOL:**
- **Unification:*The process of finding a substitution that makes two logical expressions identical. Crucial for matching rules and facts.
- **Resolution Theorem Proving:*A powerful inference rule that can derive new clauses from existing ones, used to prove logical statements by showing a contradiction with the negation of the statement.
- **Rule-Based Systems:*Systems that derive conclusions based on a set of `if-then` rules.** Forward Chaining*starts with known facts and applies rules to deduce new facts until a goal is reached. **Backward Chaining*starts with a goal and works backward to find facts and rules that support it.

Example: Imagine creating a system to manage a complex event schedule. You have variables for each event (e.g., "Event A Start Time", "Event A Room"), domains for possible times and rooms, and constraints (e.g., "Event A must finish before Event B starts", "Room 1 cannot host more than 50 people", "Speaker X is only available on

Tuesdays"). This is a CSP. A **backtracking search** algorithm, augmented with **Arc Consistency (AC-3)**, would systematically try to assign times and rooms. AC-3 would prune impossible combinations early (e.g., if Room 1 is too small for Event A, it's removed from Event A's domain for that specific combination, preventing fruitless search branches).

Case Study: The development of expert systems in areas like medical diagnosis or financial advising heavily relies on logical inference. For instance, a diagnostic system might have a knowledge base of symptoms and associated diseases (e.g., "If patient has fever AND cough AND fatigue THEN patient MAY have flu"). Given a patient's symptoms, a **forward chaining** mechanism could infer possible diseases, or a **backward chaining** approach could test a hypothesis (e.g., "Does the patient have flu?") by seeking evidence for its supporting conditions.

Exercise: Formulate the classic N-Queens problem (placing N chess queens on an NxN board so no two queens attack each other) as a Constraint Satisfaction Problem. Define the variables, their domains, and the constraints.

Tip/Pitfall: A common pitfall with CSPs is over-constraining the problem, making it insoluble or very difficult to find a solution. Conversely, under-constraining can lead to too many trivial solutions. Carefully define your constraints to accurately model the problem without introducing unnecessary restrictions.

# Week 4: Classical Planning and Knowledge Representation

Concepts:

- **Classical Planning:*The process of finding a sequence of actions that will achieve a specific goal state from an initial state in a known, deterministic, and fully observable environment. It's about devising a course of action.
- **STRIPS Representation:*A widely used formal language for defining planning problems. It specifies a** state*(a set of facts true in the world), a **goal*(a set of facts that must be true for the goal to be achieved), and** actions**. Each action has:
- **Preconditions:*Facts that must be true for the action to be executed.
- **Effects:*Facts that become true (add list) or false (delete list) after the action is executed.
- **Planning as State-Space Search:*Classical planning problems can be solved using search algorithms.** Forward (Progression) Search*starts from the initial state and applies valid actions to explore reachable states until a goal state is found. **Backward (Regression) Search*starts from the goal state and finds actions that could have led to it, working backward to the initial state.
- **Planning Graphs (GraphPlan):*A data structure that helps find plans and generate informed heuristics. It layers propositional facts and actions, showing what facts can be achieved and which actions are possible over time, considering mutual exclusion.
- **Multi-Agent Planning and Coordination:*Addresses the challenges of coordinating actions among multiple intelligent agents to achieve collective goals, often involving communication, negotiation, or distributed planning strategies.
- **Knowledge Representation (KR):*The field of AI dedicated to representing information about the world in a form that an AI system can use to reason and make intelligent decisions.
- **Ontological Engineering:*The process of designing and building** ontologies**. An ontology is a formal specification of a shared conceptualization, essentially a structured way to represent knowledge about a domain (e.g., classes of objects, their properties, and relationships between them).
- **Categories and Objects:*Fundamental KR concepts, focusing on how to classify things in the world into categories and represent individual instances (objects) belonging to those categories.
- **Events and Time:*Representing dynamic aspects of the world, including actions, occurrences, and their temporal relationships (e.g., "before," "after," "during").
- **Reasoning with Default Information:*Handling common-sense reasoning and exceptions, where conclusions are drawn based on what is typically true, but can be overridden by specific evidence.

Example: Imagine a simple delivery robot tasked with picking up a package from Room A and delivering it to Room B. Using **STRIPS representation**, the initial state might be `(RobotAt RoomC), (PackageAt RoomA)`. The goal is `(PackageAt RoomB)`. Actions would be `(MoveFrom X Y)` with preconditions `(RobotAt X)` and effects `(RobotAt Y), (Not RobotAt X)`. And `(PickUp Package Room)` with preconditions `(RobotAt Room), (PackageAt Room)` and effects `(Carrying Package), (Not PackageAt Room)`. A **forward search planner** would then search for a sequence of these actions to achieve the goal.

Case Study: Robotics task planning heavily relies on classical planning. For instance, a robotic arm assembling a product might use a planner to determine the precise sequence

of pick-and-place, screw-driving, or welding operations. Each operation is an action with preconditions (e.g., "tool in hand," "part in position") and effects (e.g., "parts assembled," "screw tightened"). Furthermore, knowledge representation is critical in enabling service robots to understand their environment, such as the layout of a building, the function of objects (e.g., "coffee machine," "elevator"), and human commands, allowing them to reason about tasks and interact intelligently.

Exercise: Given the initial state `(On A B), (On B Table), (Clear A), (Clear C)` and the goal `(On C A)`, define the necessary STRIPS actions (e.g., `Stack`, `Unstack`, `MoveToTable`) to solve this Blocksworld problem.

Tip/Pitfall: A significant challenge in classical planning is the "frame problem" – explicitly stating all facts that *do not* change for every action. STRIPS' add and delete lists simplify this, but implicitly assuming things remain unchanged can lead to issues in complex, real-world environments. Be mindful of unintended side effects when defining actions.