

# CCPS 721 Artificial Intelligence I

Ilkka Kokkarinen

Chang School of Continuing Education  
Toronto Metropolitan University

Version of February 16, 2025

This document contains the outline and course management form for the course **CCPS 721 Artificial Intelligence I**, as prepared and taught by [Ilkka Kokkarinen](#) for Chang School of Continuing Education at Toronto Metropolitan University, Canada.

Ilkka Kokkarinen acts as the Subject Matter Expert for the purposes of virtualization of this course. The latest versions of the files and documents used in this course are always available in the public GitHub repository [ikokkari/AI](#).

Learning outcomes	3
Course outline	4
Part One: Thinking in states	4
Part Two: Coding in Prolog	4
Part Three: Thinking in symbols	4
Part Four: Acting in uncertainty	4
Reading material	5
Course textbook	5
Prolog material	6
Grading and policy on cheating	8
1. Python labs	8
2. Prolog labs	9
3. Midterm exam	10
4. Final exam	10
Individual modules	11
1. Agents and environments	12
2. Adversarial search	13
3. Constraint satisfaction	14
4. Prolog programming language	15
5. Solving problems with Prolog	16
6. Structure and interpretation of Prolog programs	17
7. Propositional logic	18
8. Predicate logic	19
9. Bayesian probability	20
10. One-shot decisions	21
11. Supervised learning	22
12. Reinforcement learning	23

# ***Learning outcomes***

Upon completion of this course, the students have acquired the following knowledge.

- Think of decision problems in terms of agents operating inside an environment to maximize its expected utility, and classify such problems according to the relevant aspects of the problem.
- Understand blind and informed search algorithms for problem solving in state spaces of one-player games of complete information.
- Use the minimax algorithm to play two player games of complete information, and know both general and game-specific techniques to speed up the search by pruning the game tree.
- Can model problems as constraint satisfaction problems to be solved with consistency and backtracking algorithms, and again know both general and problem-specific optimizations to prune branches of the search tree.
- Understand logical reasoning and programming in the Prolog programming language enough to solve non-trivial coding problems by converting them into Prolog logic formulas, especially problems in parsing and combinatorial search that would be problematic to solve within the traditional imperative programming paradigm.
- Know the syntax and semantics of propositional logic and predicate logic, and can apply mechanistic inference rules to logical formulas.
- Represent uncertainty with Bayesian probabilities, and perform both causal and diagnostic computations for two or three unknown by hand, and for a more realistic number of unknowns with belief networks.
- Combine utilities and probabilities for decision theory to be applied to both simple and complex decisions.
- Understand on a theoretical level the core ideas, principles and essential algorithms of machine learning, both supervised and reinforcement kind, so that they are ready to learn how these techniques are put in practical use in their work with [scikit-learn](#) and other modern machine learning frameworks.

# ***Course outline***

This course content consists of twelve modules. These modules have been naturally grouped into four parts of three modules each, each part consisting of three modules that together share and build on the same central theme.

## ***Part One: Thinking in states***

- Module 1: Agents and environments
- Module 2: Adversarial search
- Module 3: Constraint satisfaction

## ***Part Two: Coding in Prolog***

- Module 4: Prolog programming language
- Module 5: Solving problems with Prolog
- Module 6: Structure and interpretation of Prolog programs

## ***Part Three: Thinking in symbols***

- Module 7: Propositional logic
- Module 8: Predicate logic
- Module 9: Bayesian probability

## ***Part Four: Acting in uncertainty***

- Module 10: One-shot decisions
- Module 11: Supervised learning
- Module 12: Reinforcement learning

# *Reading material*

## *Course textbook*

The official textbook for this course is the **fourth edition** of the classic textbook "[Artificial Intelligence: A Modern Approach](#)", henceforth called **AIMA4**. ([amazon.ca hardcover](#), [Pearson online](#)) This fourth edition is a massively streamlined improvement over the already magnificent third edition that was used in the previous semesters of this course, and is recommended for all students without hesitation during and after this course. Keeping up with the times, AIMA4 comes with a public GitHub repository [aimacode](#) chock full of all kinds of useful material. The repository [aima-python](#) contains example code as Python modules and Jupyter notebooks that summarize the important material of AIMA4 in a convenient interactive format.

This course covers approximately half of the material in the textbook. The lecture outline follows the usual, [these days almost quaint](#) outline for the first "Good Old-Fashioned Artificial Intelligence" course, covering the central topics listed in the above outline.

The repository [aima-exercises](#) offers a large collection of exercises, ranging from theoretical to practical, about the topics of each module. In fact, AIMA4 no longer contains any exercises inside the print edition. These exercises live and evolve in this repository so that those pages can be used for talking about the important concepts.

Each scheduled session nominally consists of three hours of lecture followed by one hour of lab. However, seeing that these virtualized courses liberate learning from the tyranny of time and place, our lectures and lab sessions are also integrated so that during these lectures, selected lab problems are examined at whichever point of the entire four hour session seems the most appropriate to bring in that problem.

## ***Prolog material***

As with all classic textbooks of computer science, the intention of AIMA4 is, of course, to soar above any particular programming language used to actually implement the low-level details of its high-level ideas. Therefore this tome does not contain any material on Prolog programming, beyond some cursory mentions of its connection to predicate logic and constraint satisfaction. The mainstream programming languages of the 2020's have generally reached the sweet spot between the imperative, functional and logical programming paradigms to make explorations on artificial intelligence topics natural. Therefore the same as in other applications of programming, AI programming these days is best done in high-level imperative languages such as Python, usually aided with powerful frameworks of [numpy](#) and [scikit-learn](#).

Originally discovered back in the very different world of the seventies, Prolog no longer holds much sway in artificial intelligence the way it did back when its practical alternatives were basically C and various other low-level string grinders designed for efficiency of low-level operations instead of clarity and brevity of thinking in higher levels of abstraction. However, due to its unique nature in that the Prolog language with its modern extensions is practically *sui generis* as its own [programming paradigm](#), Prolog still shines brightly in problems that consist of analyzing and processing symbolic structures, and also in discrete optimization problems that can be expressed and solved using the standard Constraint Logic Programming extension to the language. Prolog should still therefore be the language of choice for parsing and analyzing the meaning of both natural and artificial languages.

Even outside its applications in linguistic analysis and combinatorial search and optimization with constraints, the Prolog programming language still has value in expanding the concept of programming from the restrictive imperative form to more flexible logical and functional forms. In addition, Prolog makes otherwise ossified formal logic come to life when logical expressions get applied to arbitrary symbolic terms such as lists and integers. This drills into the worldview of computer programs as special cases of symbolic formulas and their entailed consequences in the domain of integers.

The instructor also has a series of YouTube videos on Prolog, collected in the playlist unimaginatively named "[Prolog](#)".

To supplement the lectures and example programs of this course, students may choose to study either one of the following Prolog tutorials. "[Learn Prolog Now!](#)" is simpler and behind that link even embedded with SWISH-Prolog for interactive play with examples. "[The Power of Prolog](#)" is more abstract and theoretical, but nobody can deny how well it makes its case, presented with the famous Teutonic efficiency and style that we all know and love. The author also maintains an eponymous YouTube channel "[The Power of Prolog](#)" of his lecture videos.

For additional clarification of all these new terms flying around here, students can also check out [The Prolog Dictionary](#). For the Constraint Logic Programming extension that makes Prolog unique among all programming languages in solving all sorts of discrete optimization problems, you can check out the [CLP\(FD\) Tutorial](#), a part of the larger site "[Real World Programming in SWI-Prolog](#)".

The collection "[Ninety-Nine Prolog Problems](#)" and their solutions to these problems can be used to practice problem solving in Prolog before you start the Prolog labs.

[SWI-Prolog](#) is the official Prolog environment used in this course. Students may choose to work with some other flavour of Prolog, but the lab submissions will be tested and graded in SWI-Prolog. Distributed under GNU General Public Licence, SWI-Prolog is completely free for [download](#) and unlimited use. Instead of installing this system locally on their computers, students can also use the free and interactive cloud-based version [SWISH-Prolog](#) that is used to demonstrate the language during the lectures.

Same as with working in any other programming language, you should keep the [SWI-Prolog reference manual](#) page open in one browser tab for quick access to the [documentation of built-in predicates](#). The page "[Notation of predicate descriptions](#)" explains the conventional notation used to document the instantiation expectations and behaviour of parameters.

# ***Grading and policy on cheating***

The grading for this course consists of the following four components:

1. Python labs, 30%
2. Prolog labs, 20%
3. In-person midterm exam, 20%
4. In-person final exam, 30%

The final course grade of the sum of the scores for these four components. Furthermore, as required by TMU School of Computer Science, to pass the course, the student must get at least half the marks for the final exam, regardless of their marks for the other three components.

This course has **zero tolerance** in students sharing their coding solutions in both the Python state space search labs and the Prolog labs. It is acceptable to discuss the problems at the level of ideas, but any sharing of code whatsoever will result in zero mark for those labs for all students who participate in sharing.

This course has **zero tolerance** in students submitting solutions generated by AI models. Any student submitting even one such solution will lose all the marks for those particular labs.

## ***1. Python labs***

This semester, we will try a new format for the Python labs. We will still use this author's Python problems from the documents "[109 Python Problems for CCPS 109](#)", "[109 Additional Python Problems](#)" and "[Third Python Problem Collection](#)" in the author's GitHub repository "[109 Python Problems for CCPS 109](#)". The students should start by downloading the entire repository [ikokkari/PythonProblems](#) on their computer to work on, including the automated fuzz tester script [tester109.py](#) for the problems to be solved.

However, to get the students immediately to the fun and interesting part of backtracking search, we provide non-optimized solutions to these problems in the file [solutions.py](#) in the Python subdirectory of the course GitHub repository. These solutions are correct in the sense that they will return correct answers to all test cases produced by the automated tester script `tester109.py`, but the non-optimized backtracking search is always far too slow to pass the test cases in reasonable time. It is therefore your task to speed up these solutions with various backtracking optimization techniques of their choice so that they pass the tester within the time limit.

From the following list of problems, **you get to freely choose up to five problems** whose solutions you will optimize in this manner. At the end of the course, submit the file `labs109.py`



that contains your optimized solutions, along with a PDF document that explains what optimizations you used to improve each solution, and actual measurement data of the running time achieved by your optimization steps.

Each of these five problems is worth up to six points, the mark determined by the speedup that you achieved and its documentation.

The list of problems for you to choose up to five problems is as follows:

- 108: Painted into a Corner
- 110: Reverse the Rule 110
- 111: Aye, eye, I
- 114: Stepping Stones
- 120: Count the permorsetations
- 122: Forks on the Gin Lane
- A78: Independent Dominating Set
- A99: The Sharpest Axes
- A107: The Round Number Round
- A105: Blocking Pawns
- A114: Bandwidth Minimization
- A115: Set Splitting
- A116: Domino Poppers
- A122: Unity Partition
- T92: Cutting Sticks
- T109: Colonel Blotto and the Spymaster
- T112: Post Correspondence Problem
- T114: No Dogs, No Masters
- T120: Fox and Hounds
- T121: Life Is Like a Box of Dominoes

**Only the problems listed above are accepted for these labs. Other problems from the same collections will not be accepted.**

Students should start working on these labs some time around Module Three after the state space technique lectures have been fully digested, and from there continue thinking about and working on these problems on their own time and convenience during the semester.

## ***2. Prolog labs***

The Prolog labs consist of a collection of problem specifications given in the PDF document "[CCPS 721 Prolog Problems](#)". Of the collection of problems given, **students get to freely**

**choose up to ten problems to solve.** Each problem is to be solved by defining the appropriate Prolog predicate whose name and arity are specified in the title of the problem specification. Working on these labs makes the students practice and apply their Prolog skills and techniques that were taught in Part Two of this course outline.

Every correctly solved problem that passes the instructor's test predicate (as given in the Prolog file [tester721.pl](#)) with flying colours is worth **two points** to the course grade, up to the maximum of thirty points for successfully solving ten Prolog problems.

Students should start working on these labs some time around Module Six after the Prolog lectures have been fully digested, and from there continue thinking about and working on these problems on their own time and convenience. These Prolog labs are due all as a single bunch during Module 12, to be uploaded to D2L.

### ***3. Midterm exam***

The midterm exam will take place in person **Monday February 23**. The midterm exam will be a multiple choice exam covering the course material from modules 1 to 3. The Prolog materials will not be included in the midterm. The midterm exam is closed book, writing equipment only, and lasts two hours.

### ***4. Final exam***

The final exam will take in person **Monday April 13**, during the last scheduled session of the course. The final exam will be a multiple choice exam covering the course material from modules 7 to 12. The final exam is closed book, writing equipment only, and lasts three hours.

## ***Individual modules***

In the table presentation for each module, the textbook section titles listed in the row "AIMA 4" summarize the important ideas and concepts that form the learning goals of that module in this course. The links in the row "Central terms" lead to the corresponding Wikipedia articles for additional information about the central terms of each module.

Some of these related concepts are close enough to the actual course topics to warrant inclusion in these tables. However, such topics are marked with the symbol ♣ to indicate that they are not part of the main material.

Note that the chapter numbering of [aima-exercises](#) follows the third edition of AIMA, and can therefore be off by one compared to the chapter numbers of AIMA4.

## 1. Agents and environments

Central terms	<a href="#">Intelligent agent (classes)</a> <a href="#">Rational agent</a> <a href="#">Depth-first search</a> <a href="#">Breadth-first search</a> <a href="#">Iterative deepening</a> <a href="#">A*</a> <a href="#">Bidirectional search</a>
AIMA 4	2.1, "Agents and Environments" 2.2, "Good Behaviour: The Concept of Rationality" 2.3, "The Nature of Environments" 2.4, "The Structure of Agents" 3.1, "Problem-Solving Agents" 3.3, "Search Algorithms" 3.4, "Uninformed Search Strategies" 3.5, "Informed (Heuristic) Search Strategies"
<a href="#">aima-python</a>	<a href="#">agents4e.py</a> <a href="#">agents.ipynb</a> <a href="#">search.py</a> <a href="#">search.ipynb</a>
<a href="#">aima-exercises</a>	<a href="#">2. Intelligent agents</a> <a href="#">3. Solving problems by searching</a>

## 2. Adversarial search

Central terms	<a href="#">Minimax algorithm</a> ( <a href="#">negamax</a> ) <a href="#">Alpha-beta pruning</a> <a href="#">Transposition table</a> <a href="#">Principal variation search</a> 🏰 <a href="#">Expected value</a> <a href="#">Solved game</a> 🏰 <a href="#">Combinatorial game theory</a> 🏰
AIMA 4	5.1, "Game Theory", 5.2, "Optimal Decisions in Games" 5.3, "Heuristic Alpha-Beta Tree Search" 5.4, "Monte Carlo Tree Search" 5.5, "Stochastic Games"
<a href="#">aima-python</a>	<a href="#">games4e.py</a> <a href="#">games4e.ipynb</a>
<a href="#">aima-exercises</a>	<a href="#">5. Adversarial search</a>

### 3. Constraint satisfaction

Central terms	<a href="#">Hill climbing</a> <a href="#">Simulated annealing</a> <a href="#">Tabu search</a> <a href="#">Genetic algorithm</a> <a href="#">Backtracking</a> <a href="#">Look-ahead</a> <a href="#">Local consistency</a> <a href="#">AC-3 algorithm</a> <a href="#">Eight queens puzzle</a> <a href="#">Verbal arithmetic</a> <a href="#">SWI-Prolog Constraint Logic Programming</a> 🏰
AIMA 4	4.1, "Local Search and Optimization Problems" 6.1, "Defining Constraint Satisfaction Problems" 6.2, "Constraint Propagation: Inference in CSP's" 6.3, "Backtracking Search for CSP's" 6.4, "Local Search for CSP's" 6.5, "The Structure of Problems"
<a href="#">aima-python</a>	<a href="#">csp.py</a> <a href="#">arc_consistency_heuristics.ipynp</a>
<a href="#">aima-exercises</a>	<a href="#">4. Beyond classical search</a> <a href="#">6. Constraint satisfaction problems</a>

## ***4. Prolog programming language***

The reading material of modules Four to Six consists of the example programs listed in each module, in addition to the Prolog tutorial chosen by the student from the earlier list of Prolog materials.

Example programs	<a href="#">listdemo.pl</a> <a href="#">builtindemo.pl</a> <a href="#">pokerhand.pl</a>
------------------	---

## ***5. Solving problems with Prolog***

Example programs	<a href="#">tailrecdemo.pl</a> <a href="#">bst.pl</a> <a href="#">fractions.pl</a>
------------------	--



## ***6. Structure and interpretation of Prolog programs***

Example programs	<a href="#">crypt.pl</a> <a href="#">homoiconic.pl</a> <a href="#">clpfddemo.pl</a>
------------------	---

## 7. Propositional logic

Central terms	<a href="#">Logic</a> <a href="#">Propositional logic (argument forms)</a> <a href="#">What the Tortoise said to Achilles</a> 🐢 <a href="#">Law of excluded middle</a> (not to be confused with the principle of bivalence) <a href="#">List of rules of inference</a> 🐢 (these are only for reference for interested students, since the only inference rules that we need in this course are <i>Modus Ponens</i> and <i>resolution</i> ) <a href="#">Conjunctive normal form</a> ( <a href="#">Tseytin transformation</a> 🐢) <a href="#">Satisfiability</a> ( <a href="#">2-Satisfiability</a> 🐢) <a href="#">Forward chaining</a> <a href="#">Backward chaining</a> <a href="#">Resolution</a> <a href="#">DPLL algorithm</a>
AIMA 4	7.3, "Logic", 7.4, "Propositional Logic: A Very Simple Logic" 7.5, "Propositional Theorem Proving" 7.6, "Effective Propositional Model Checking"
<a href="#">aima-python</a>	<a href="#">logic4e.py</a> <a href="#">logic.ipynp</a>
<a href="#">aima-exercises</a>	<a href="#">7. Logical agents</a>

## 8. Predicate logic

Central terms	<a href="#">First-order predicate logic</a> <a href="#">Interpretation</a> <a href="#">Horn clause</a> <a href="#">Skolem normal form</a>
AIMA 4	8.1, "Representation Revisited" 8.2, "Syntax and Semantics of First-Order Logic" 8.3, "Using First-Order Logic" 9.1, "Propositional vs. First-Order Inference" 9.2, "Unification and First-Order Inference" 9.3, "Forward Chaining" 9.4, "Backward Chaining" 9.5, "Resolution"
<a href="#">aima-python</a>	<a href="#">logic4e.py</a> <a href="#">logic.ipynp</a>
<a href="#">aima-exercises</a>	<a href="#">8. First-order logic</a> <a href="#">9. Inference in first-order logic</a>

## 9. Bayesian probability

Central terms	<a href="#">Probability</a> (axioms) <a href="#">Dutch book</a> <a href="#">Conditional probability</a> ( <a href="#">Confusion of the inverse</a> , <a href="#">Base rate fallacy</a> ) <a href="#">Independence</a> <a href="#">Conditional independence</a> <a href="#">Bayes' theorem</a> <a href="#">Monty Hall problem</a> ( <a href="#">Principle of Restricted Choice</a> 🏰) <a href="#">Bayesian network</a> <a href="#">Probability interpretations</a> 🏰
AIMA 4	12.1, "Acting Under Uncertainty", 12.2, "Basic Probability Notation" 12.3, "Inference Using Full Joint Distributions" 12.4, "Independence" 12.5, "Bayes' Rule and Its Use" 13.1, "Representing Knowledge in Uncertain Domain" 13.2, "The Semantics of Bayesian Networks" 13.4, "Approximate Inference for Bayesian Networks"
<a href="#">aima-python</a>	<a href="#">probability4e.py</a> <a href="#">probability4e.ipynp</a>
<a href="#">aima-exercises</a>	<a href="#">13. Quantifying uncertainty</a> <a href="#">14. Probabilistic reasoning</a>

## 10. One-shot decisions

Central terms	<a href="#">Preference</a> <a href="#">Utility (cardinal)</a> <a href="#">Allais paradox</a> <a href="#">Decision network</a> ♣ <a href="#">Value of perfect information</a> <a href="#">Game theory</a> <a href="#">Simultaneous game</a> <a href="#">Nash equilibrium</a> (online <a href="#">bimatrix solver</a> for arbitrary matrix games ♣) <a href="#">Matching pennies</a> , <a href="#">Prisoner's Dilemma</a> , <a href="#">Chicken</a> (list of other ♣)
AIMA 4	16.1, "Combining Beliefs and Desires under Uncertainty" 16.2, "The Basis of Utility Theory" 16.3, "Utility Functions" 16.6, "The Value of Information" 18.1, "Properties of Multiagent Environments" 18.2, "Non-Cooperative Game Theory" 18.3, "Co-operative Game Theory"
<a href="#">aima-python</a>	<a href="#">making_simple_decision4e.py</a>
<a href="#">aima-exercises</a>	<a href="#">16. Decision theory</a>

## 11. Supervised learning

Central terms	<a href="#">Machine learning</a> <a href="#">Supervised learning</a> ( <a href="#">Inductive bias</a> ) <a href="#">Training, test and validation sets</a> <a href="#">Bias-variance tradeoff</a> <a href="#">Decision tree learning</a> ( <a href="#">Random forest</a> ) <a href="#">Overfitting</a> <a href="#">Probably approximately correct learning</a> <a href="#">Naive Bayes classifier</a> <a href="#">Ensemble learning</a>
AIMA 4	19.1, "Forms of Learning" 19.2, "Supervised Learning" 19.3, "Learning Decision Trees" 19.4, "Model Selection and Optimization" 19.5, "The Theory of Learning" 19.8, "Ensemble Learning" 20.1, "Statistical Learning"
<a href="#">aima-python</a>	<a href="#">learning4e.py</a>
<a href="#">aima-exercises</a>	-

## 12. Reinforcement learning

Central terms	<a href="#">Markov decision process (MDP)</a> <a href="#">Reinforcement learning</a> <a href="#">Softmax function</a> <a href="#">Temporal difference learning</a> <a href="#">Q-learning</a> <a href="#">TD-Gammon</a> 🏆 <a href="#">AlphaGo</a> 🏆 <a href="#">Kelly criterion</a> 🏆
AIMA 4	17.1, "Sequential Decision Problems" 17.2, "Algorithms for MDP's" 22.1, "Learning from Rewards" 22.2, "Passive Reinforcement Learning" 22.3, "Active Reinforcement Learning" 22.4, "Generalization in Reinforcement Learning"
<a href="#">aima-python</a>	<a href="#">mdp4e.py</a> <a href="#">reinforcement-learning4e.py</a> <a href="#">reinforcement-learning4e.ipynp</a>
<a href="#">aima-exercises</a>	<a href="#">17. Making complex decisions</a> <a href="#">21. Reinforcement learning</a>