

Week 1 Seminar: What Is AI?

AI or not AI, four puzzles, and AI around us | Solutions

This seminar uses **no computers**. Work in groups of 3 or 4 with paper and a pen. It is **not assessed**. The four puzzles in Section B preview methods you will meet later in the module: search (Week 2), logic (Week 3), planning (Week 4) and constraint satisfaction (Week 6). For each puzzle, **how** you solved it matters more than the answer.

Timing: about 2 hours 15 minutes, with a 10-minute break after Activity 3.

A AI or not AI?

Activity 1 | Vote

15 minutes

For each item, decide as a group: is it AI, not AI, or are you not sure?

Item	AI	Not AI	Not sure	Item	AI	Not AI	Not sure
A spam filter	☐	☐	☐	Netflix recommendations	☐	☐	☐
A thermostat	☐	☐	☐	A chess app	☐	☐	☐
A sat-nav finding a route	☐	☐	☐	A calculator	☐	☐	☐
A spreadsheet formula	☐	☐	☐	Face unlock on a phone	☐	☐	☐
ChatGPT	☐	☐	☐	Automatic doors	☐	☐	☐

- What rule did your group use to decide?
- The lecture defined an *agent* as an entity that perceives its environment and acts upon that environment. Is a thermostat an agent? Is it intelligent?
- Does it matter what we call "AI"? Who gains from calling a product AI?

Answers

There is no single right answer: the discussion matters more than the votes. A reasonable set of answers, with a reason for each:

Item	Verdict	Why
A spam filter	AI	It learns from thousands of emails marked “spam” or “not spam”, then decides about emails it has never seen (learning, Week 7).
A thermostat	Not AI	One fixed rule: if it is colder than the setting, switch the heating on. A “smart” thermostat that learns your routine is closer to AI.
A sat-nav finding a route	AI	It searches a huge road map for the best route (search, Weeks 2 and 3). Many people now call it “just software”: see (a).
A spreadsheet formula	Not AI	It does exactly the calculation it was given, and nothing else.
ChatGPT	AI	A large language model: a neural network trained on huge amounts of text (Weeks 8 and 9).
Netflix recommendations	AI	It learns what you are likely to enjoy from what you and similar viewers have watched (Week 7).
A chess app	AI	It looks ahead at possible moves and the opponent’s replies (adversarial search, Week 2).
A calculator	Not AI	Fixed rules of arithmetic. Very fast and always right, but it never chooses or learns.
Face unlock on a phone	AI	A neural network recognises your face from the camera, in different light and at different angles (Week 8).
Automatic doors	Not AI	A sensor triggers a motor: one fixed rule, like the thermostat.

- (a) Good rules: *it learns from data; it copes with situations it has never seen; it chooses between options in a way that would need intelligence if a person did it.* Rules such as “it uses a computer” or “it is clever” are too vague: they would make the calculator AI. The line moves over time: route finding and chess were headline AI once, and now many people call them ordinary software. This is sometimes called the *AI effect*.
- (b) Yes, it is an agent: it perceives its environment (the temperature) and acts on it (switching the heating on or off). But it is not intelligent: it follows one rule, never compares options, never plans ahead and never learns. It is the simplest kind of agent, a *simple reflex agent*. So “agent” and “intelligent” are not the same thing.
- (c) It matters. Companies call products “AI” because it sells, which links back to the hype and the AI winters in the lecture. The label can hide what a product really does, and it can make people trust it more than they should. Someone buying or using “AI” should ask: what exactly does it do, what data does it use, how often is it wrong, and who checks it? The EU AI Act (2024) uses its own definition of AI for exactly this reason: the law needs to know which systems it covers.

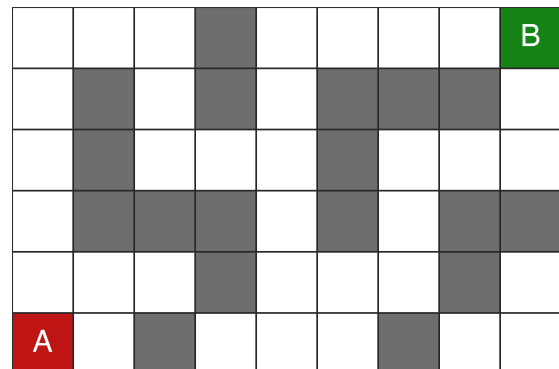
B Four puzzles

Activity 2 | The maze

15 minutes

Find your way from A (red) to B (green). You can move up, down, left or right, but not through the grey squares.

- (a) Find a path from A to B. How many steps is it?
- (b) Write pseudocode that finds a path through *any* maze, even one you have never seen. Assume your program can see the whole maze.
- (c) Does your pseudocode always find a path if there is one? Does it always find the shortest?
- (d) What does your program need to remember as it goes?



Answers

5	6	7		13	14	15	16	B
4		8		12				
3		9	10	11				
2								
1								
A								

The shortest path, with its 17 steps numbered.

- (a) **17 steps:** up 5 to the top-left corner, right 2, down 2, right 2, up 2, then right 4 to B. There are six ways through altogether, of 17, 19, 21, 23, 23 and 25 steps, so the other five are all longer. There are two dead ends (the short corridor just to the right of A, and the square at the bottom in the middle). The three squares in the bottom-right corner cannot be reached at all.

- (b) One good answer:

```
frontier = [A]; explored = empty
repeat:
  if frontier is empty: there is no path
  take a square s out of the frontier
  if s is B: follow the "came from" notes back to A
  add s to explored
  for each open neighbour n of s not seen before:
    note that n came from s; add n to the frontier
```

Two details make it work for *any* maze: the "came from" notes, so the path can be rebuilt at the end, and the list of squares already seen, so the program never goes round in circles. If you always take the *newest* square from the frontier, this is depth-first search (DFS). If you always take the *oldest*, it is breadth-first search (BFS). If you take the square *closest to B*, it is greedy best-first search (Week 3).

- (c) With a list of explored squares, both DFS and BFS always find a path if there is one. Only

BFS always finds the shortest. On this maze, DFS (looking up, down, left, right in that order) explores 25 squares and finds a 21-step path; BFS explores 29 squares and finds the 17-step path. "Keep one hand on the wall" finds a way through some mazes but not all (it can loop round an island of walls for ever), and it rarely gives the shortest path.

- (d) Three things: the squares still to try (the *frontier*); the squares already tried (the *explored set*); and where each square was reached from, so the path can be rebuilt. In Week 2 these become the frontier, the explored set and the *parent* of each node.

Activity 3 | The river crossing

15 minutes

A farmer must take a wolf, a goat and a cabbage across a river. The boat carries the farmer and at most one other. If the wolf and the goat are left alone together, the wolf eats the goat. If the goat and the cabbage are left alone together, the goat eats the cabbage.

- Find a way to get all four across safely.
- Write down every state as *left bank* | *right bank*. The start is F W G C | and the goal is | F W G C.
- Which states are not allowed?
- What is the smallest number of crossings?

Answers

- (a) and (b): one solution, with every state:

Step	Crossing	State after it
Start		F W G C
1	Farmer takes the goat over	W C F G
2	Farmer comes back alone	F W C G
3	Farmer takes the wolf over	C F W G
4	Farmer brings the goat back	F G C W
5	Farmer takes the cabbage over	G F W C
6	Farmer comes back alone	F G W C
7	Farmer takes the goat over	F W G C

Taking the cabbage at step 3 and the wolf at step 5 also works. These are the only two 7-crossing solutions: the two paths in the diagram below.

(c) Each of the four is on one bank or the other, so there are $2^4 = 16$ possible states. Six are not allowed:

Left bank unsafe (farmer on the right):

W G | F C G C | F W W G C | F

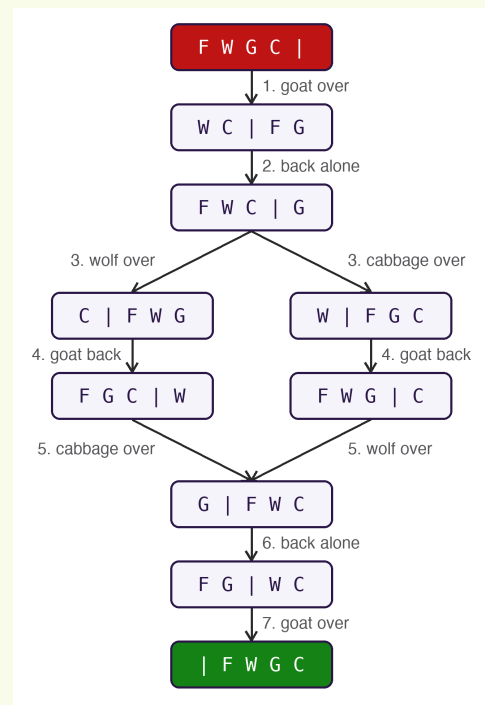
Right bank unsafe (farmer on the left):

F C | W G F W | G C F | W G C

That leaves 10 safe states, and all 10 appear in the diagram.

(d) **Seven.** The first crossing *must* take the goat: going alone leaves all three together, taking the wolf leaves the goat with the cabbage, and taking the cabbage leaves the wolf with the goat. The diagram shows every safe state and every crossing between them. Working through it level by level (exactly what breadth-first search does in Week 2), the goal is first reached at crossing 7, so no shorter plan exists. The trick is crossing 4: bringing the goat *back* looks like going backwards, but it is the only way on.

What this shows: each line of the table is a *state*, each crossing is an *action*, some states are not allowed, and there is a *goal*. That is a search problem (Week 2), and the diagram is its *state space*. The list of crossings is a *plan* (Week 4).



Every safe state, and the crossings between them.

Activity 4 | Knights and knaves

15 minutes

On an island, every person is either a *knight* or a *knave*. Knights always tell the truth. Knaves always lie.

- A says: "I am both a knight and a knave." What is A?
- A says: "We are both knaves." B says nothing. What are A and B?
- Write your reasoning for (b) as "if ... then ..." rules.
- Harder.* A says: "We are the same kind." B says: "We are of different kinds." What are A and B?

Answers

A safe method for every part: list every possible world (who is a knight and who is a knave), and keep only the worlds where every knight's statement is true and every knave's statement is false. A computer does exactly this in Week 3 (*model checking*).

- A is a knave.** Nobody can be both, so the statement is false. A knight never says something false, so A cannot be a knight. A knave saying something false fits.

A is	"I am both" is...	Possible?
knight	false	no: a knight said something false
knave	false	yes: a knave lies

- (b) **A is a knave and B is a knight.** If A were a knight, "we are both knaves" would be true, so A would be a knave: impossible. So A is a knave, and the statement is false, so they are not both knaves. A is a knave, so B must be a knight.

A	B	"We are both knaves" is...	Possible?
knight	knight	false	no: a knight said something false
knight	knave	false	no: a knight said something false
knave	knight	false	yes: a knave lies
knave	knave	true	no: a knave said something true

- (c) For example:

- Every person is a knight or a knave, but not both.
- If A is a knight, then what A says is true: A and B are both knaves.
- If A is a knave, then what A says is false: A and B are *not* both knaves.

The first "if" leads to a contradiction (A would be a knight and a knave), so A is a knave. The second "if" then tells us B is not a knave, so B is a knight. In Week 3 these rules are written in logic, for example $A_{\text{knight}} \rightarrow (A_{\text{knave}} \wedge B_{\text{knave}})$.

- (d) **A is a knave and B is a knight.** If B were a knave, B's statement would be false, so they would be the same kind: A would be a knave too. But then A's statement "we are the same kind" would be true, and a knave cannot say something true. So B is a knight, they are of different kinds, and A is a knave.

A	B	A: "same kind"	B: "different"	Possible?
knight	knight	true	false	no: knight B said something false
knight	knave	false	true	no: knight A said something false
knave	knight	false	true	yes: A lies, B tells the truth
knave	knave	true	false	no: knave A said something true

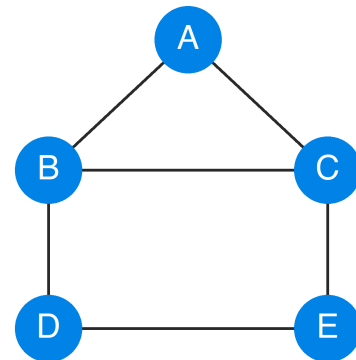
Exactly one world survives each time, so each puzzle has one answer. They are puzzles 0, 1 and 2 of CS50's "Knights" project, which you can try after Week 3.

Activity 5 | The exam timetable

15 minutes

Five exams, A to E, must fit into three days: Monday, Tuesday and Wednesday. A line between two exams means they share students, so they cannot be on the same day.

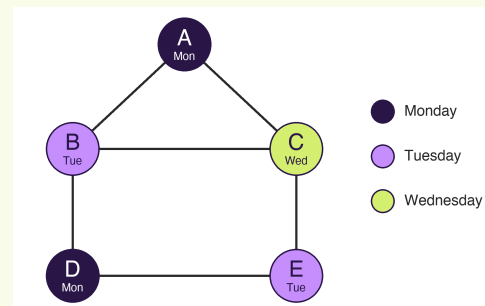
- Find a timetable that works.
- Describe your method, step by step.
- Did you ever have to undo a choice? When?
- Wednesday is cancelled. Can the exams fit into two days? Why or why not?



Exam	A	B	C	D	E
Day					

Answers

- (a) For example: **A Monday, B Tuesday, C Wednesday, D Monday, E Tuesday** (shown in the picture). Check each line: A–B Mon/Tue, A–C Mon/Wed, B–C Tue/Wed, B–D Tue/Mon, C–E Wed/Tue, D–E Mon/Tue: no clashes. There are **18** timetables that work altogether. In every one of them A, B and C are on three different days, because each pair of them shares students.



- (b) A good method is *backtracking*:
- Choose an exam that has no day yet. Start with the exams that have the most lines: B and C (three lines each).
 - Give it the first day that does not clash with any exam already placed.
 - If no day is left, go back to the last exam you placed and try its next day.
 - Repeat until every exam has a day, or until there is nothing left to try (then no timetable exists).

Worked through in the order B, C, A, D, E: B Monday; C Tuesday (not Monday, B); A Wednesday (not Monday, B; not Tuesday, C); D Tuesday (not Monday, B); E Monday (not Tuesday, C or D). Done, with no going back.

- (c) With three days you may never need to, but it can happen. Suppose a group puts A on Monday, B on Tuesday and then E on Wednesday. Every choice so far is allowed, but now C clashes with all three days, because A, B and E all share students with C. The group must go back and change a choice, for example move E to Monday. Then C goes on Wednesday and D on Wednesday (D shares students with B and E, not with C).
- (d) **No.** A, B and C are all joined to each other, so they need three different days. Backtracking shows this: A Monday, B Tuesday, then C has no day; go back to B, which has no other day; go back to A and try Tuesday; B Monday, and again C has no day. Nothing is

left to try, so no two-day timetable exists. To fit into two days, the university would have to change something else, such as moving students between exams or using a different week.

What this shows: the exams are *variables*, the days are their *domain*, and the lines are *constraints*. This is a constraint satisfaction problem, and “go back and change an earlier choice” is *backtracking* (Week 6).

Activity 6 | What did you just do?

10 minutes

- (a) Would your maze method still work on a maze with a million squares? Would your timetable method work for 500 exams?
- (b) In each puzzle, what did you have to keep track of?

Answers

- (a) Methods that rely on “looking at it” do not scale: nobody can look at a million-square maze and see the way through. A written-down method (an *algorithm*) does scale, because a computer can follow it, and we can say when it works: BFS always finds the shortest path, and backtracking always finds a timetable if one exists. Scale still matters, though. BFS on a million squares is quick for a computer, but simple backtracking on 500 exams can take far too long, which is why Week 6 adds smarter ideas, such as choosing the most constrained exam first.
- (b)
 - **Maze:** the squares still to try, the squares already tried, and where each square was reached from.
 - **River:** the current state, which states are safe, and which states have been visited (so you do not cross back and forth for ever).
 - **Knights:** which combinations of knight and knave are still possible.
 - **Timetable:** which days are still free for each exam, and the choices made so far (so you can undo them).

Keeping track of possibilities is at the heart of Weeks 2 to 6.

Activity 7 | AI in an organisation

25 minutes

Choose one organisation: a supermarket, a bank, the NHS or a university.

- Where does it use AI, or where could it? Find three uses.
- For each use, give one benefit and one risk, and say who is affected.
- Choose your best example to share with the class. Which of the module's eight topics do you think it uses?

Use of AI	Benefit	Risk	Who is affected?

Answers

Some ideas, with the module topic each one uses.

- **Supermarket:** forecasting how much stock to order (less waste; mistakes mean empty shelves) (*Learning*); personalised offers from loyalty-card data (more sales; privacy) (*Learning*); cameras at self-checkouts (less theft; customers wrongly accused) (*Neural networks*).
- **Bank:** fraud detection (money saved; genuine payments blocked) (*Learning, Uncertainty*); credit scoring (faster decisions; unfair refusals, bias) (*Learning*); chatbots (open all hours; wrong answers) (*Language*).
- **NHS:** reading scans (faster diagnosis; mistakes and over-reliance) (*Neural networks*); predicting demand for beds (better planning) (*Learning, Uncertainty*); scheduling appointments and theatre time (fewer gaps; patient data privacy) (*Optimisation*).
- **University:** timetabling (fewer clashes, like the puzzle) (*Optimisation*); chatbots for student questions (quick answers; wrong advice) (*Language*); spotting students who may drop out (early support; students labelled unfairly) (*Learning*).

A complete answer for one organisation looks like this:

Use of AI (bank)	Benefit	Risk	Who is affected?
Fraud detection on card payments	Stolen cards are stopped within seconds; less money lost.	Genuine payments are blocked, for example on holiday.	Customers, the bank, shops.
Credit scoring for loans	Decisions in minutes instead of days.	Unfair refusals if the training data reflects past bias; hard to explain a "no".	Applicants, especially groups treated unfairly in the past.
Chatbot on the website	Answers questions at any time; shorter queues.	Wrong or made-up answers; customers who cannot reach a person.	Customers, staff who handle complaints.

Links to learning outcome (d) and to Part 2 of the portfolio. The "AI in practice" activities from Week 4 build on this.

D Before Week 2

Setup checklist

- ☐ Install Python 3 from python.org (or check it is installed: `python3 --version`).
- ☐ Download the module code from github.com/sakibanwar/comp5001-ai-code (green **Code** button, then **Download ZIP**) and unzip it.
- ☐ In a terminal, go into the `week02-search` folder and run `pip install -r requirements.txt`.
- ☐ Run `python maze/maze.py maze/maze1.txt` and open the picture `maze.png` it makes.

If something goes wrong, bring the error message to the Week 2 practical.

Answers

Common setup problems and fixes:

- `python: command not found`: on a Mac try `python3`; on Windows try `py`. Use the same word in every command after that.
- `pip: command not found`: use `python3 -m pip install -r requirements.txt` (or `py -m pip ...` on Windows).
- `No module named 'PIL'`: the install step was skipped or ran in another folder. Run it again from inside `week02-search`.
- `No such file or directory: 'maze/maze1.txt'`: the terminal is in the wrong folder. Use `cd` to go into `week02-search` first.
- The picture does not open: `maze.png` is saved in the folder you ran the command from; open it from the file browser.