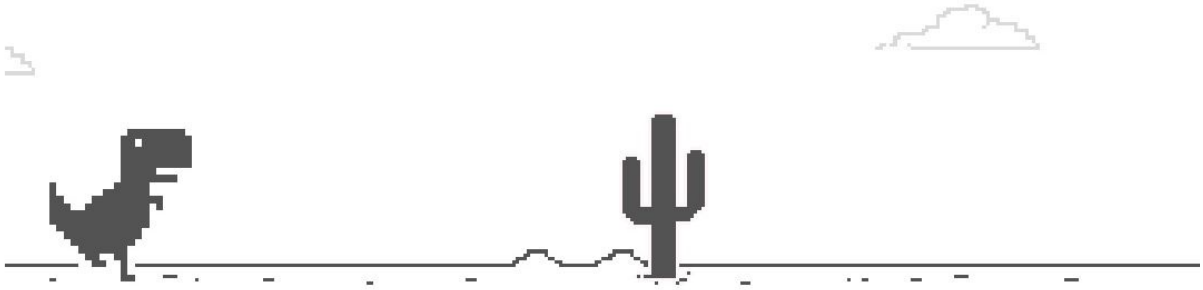
Artificial Intelligence 1

Charles University in Prague

Author:
Sabína Ságová



Date: October 16, 2024

Contents

1 Lecture 1 - History and Topics of AI

1.1 Definition of AI

Artificial intelligence is the science of making machines do things that would require intelligence if done by men. There are four views:

- `Acting Humanly`
- `Thinking Humanly`
- `Thinking Rationally`
- `Acting Rationally`

1.1.1 Acting Humanly

Turing Test: A computer passes the Turing test if a human interrogator cannot distinguish whether the responses come from a person or a computer. Required capabilities include:

- Natural Language Processing
- Knowledge Representation
- Automated Reasoning
- Machine Learning
- Computer Vision
- Robotics

Although advanced, `ChatGPT` cannot pass the Turing Test. Interestingly, `Eliza` passed the test in the 1970s.

Reverse Turing Test: In this variation, a computer attempts to recognize whether it communicates with a human or another computer.

1.1.2 Thinking Humanly

Cognitive Modelling:

- `Top-down approach` (Psychology): Follows human reasoning steps.
- `Bottom-up approach` (Neuroscience): Focuses on modelling the brain.

1.1.3 Thinking Rationally

Aristotle's Syllogisms: These are patterns for argument structures that always yield correct conclusions when given correct premises. Example:

- Socrates is a man.
- All men are mortal.

- Therefore, Socrates is mortal.

This study initiated the field of **logic**.

Major Obstacles: It is challenging to formalize reasoning when knowledge is uncertain. Moreover, there is a difference between solving a problem theoretically and solving it in practice.

1.1.4 Acting Rationally

Rational Behaviour: Involves doing the right thing, which is defined as achieving the best expected outcome even in the presence of uncertainty.

1.2 Foundations of AI

AI draws from various fields, including:

- **Philosophy**
- **Mathematics**
- **Economics**
- **Neuroscience**
- **Psychology**
- **Computer Engineering**
- **Control Theory**
- **Linguistics**

`Textbook: AI: A Modern Approach by S. Russell and P. Norvig`

1.3 Quiz Reminder

There will be a quiz at the end of this section.

2 Tutorial 1

If there are 7 sessions planned, one might be canceled, so there will be a total of 6.

2.1 Deadlines

- Submission deadline is the 3rd week.
- After the deadline, submissions are accepted for half points, with the final deadline at the end of September.

Python 3.11 works perfectly for this project.
We will use pure Python, with no additional libraries needed except for `pygame`.

2.2 Agent Programming

You will only modify the file: `dino -> agents -> myagent.py`.
The class inherits from the `Agent` class.
The `get_move` method is called on every tick of the game.
The task is for it to return the correct move for the dino.

2.3 Running Agents

To run the agents, use:

```
% python3 ./play_dino.py -a MyAgent
% python3 ./play_dino.py -a Dummy_Agent -s 50
```

Note that it is case-sensitive.
In Recodex, use `-s 50` for the score.
Submit the agent file to Recodex, ensuring you follow the naming conventions for classes and files.
You can make up to 50 submissions.

2.4 Hints

To speed up the game, use `-f 90`.
The game normally runs at 30 FPS, and `--delay 5000` will pause the screen for 5 seconds.
In `dino.py`, you'll find details about size and dinosaur states.
There are 9 types of obstacles.
Based on the type, you can decide whether to duck or jump, as they vary in size.
It's better to jump over two obstacles than to fall between them.

This is a simple reflex agent.

3 Lecture 2 - Agent Architectures

3.1 Rational Behaviour

Rational behaviour refers to behaviour that aims for ideal performance.

3.2 Agent

An agent receives percepts through sensors, processes this information, and sends it to actuators that influence the environment.

3.2.1 Examples

- Human agent: eyes, ears, nose, ... → hands, legs, mouth, ...
- Robotic agent: camera, infrared finder, ... → arms, wheels, ...
- Software agent: keyboard, network packets ... → screen, sending packets

3.3 Agent Function

The agent function is an abstract mathematical description.

3.3.1 Formal Definition

Formal definition: $V^* \rightarrow A$, where V is a set of percepts and A is a set of actions.

Internally, the agent function is implemented by an **agent program**.

3.4 Vacuum Cleaner

3.4.1 Percepts and Actions

- Percepts: location(A,B), property(clean, dirty)
- Actions: suck up, move left, move right, do nothing

3.5 Performance Measure

The performance measure describes the behaviour of an agent and defines the goals of the agent designer.

3.5.1 Example

Example: vacuum performance measure - have a clean floor.

3.6 Rational Agent

A rational agent should select an action that is **expected** to maximize its performance measure. It should be **autonomous** and learn what it can to compensate for partial or incorrect prior knowledge.

3.6.1 Example: Automated Taxi Driver

- Agent: taxi driver
- Performance measure: safe, fast, comfortable, profit
- Environment: roads, other traffic, pedestrians, customers
- Actuators: steering, accelerator, brake, signal, horn
- Sensors: camera, sonar, lidar, odometer, GPS

3.7 Properties of Task Environments

- Fully observable / partially observable
- Deterministic / stochastic
- Episodic / sequential
- Static / dynamic

- Discrete / continuous
- Single agent / multi-agent

The simplest environment is fully observable, deterministic, episodic, static, discrete with a single agent.

3.8 Agent Composition

An **agent** consists of architecture + program.

3.9 Types of Agents

3.9.1 Simple Reflex Agent

The agent selects an action based on the **current percept**. It is implemented as condition-action rules, leading to a significant reduction in the number of possibilities (to the number of percepts).

3.9.2 Model-Based Agent

The most commonly used type, which maintains an internal state to track aspects of the world that are not evident in the current percept.

3.9.3 Goal-Based Agent

Action selection is based not only on the current state but also on what the agent is trying to achieve. This involves search and planning for future actions.

3.9.4 Utility-Based Agent

The utility (long-term benefit) function is an internalization of the performance measure. It is useful when there are conflicting goals or unequal chances of achieving different goals.

3.10 Detour - Representation of Environment

- Atomic representation: treats each state as a black box
- Factored representation: each state is divided into a fixed set of variables, each with a value
- Structured representation: each state consists of a set of objects with various and varying relationships

3.11 General Learning Agent

- Performance element: the initial agent structure responsible for action selection
- Learning element: responsible for making improvements
- Critic: provides feedback
- Problem generator: explores new actions

4 Lecture 3 - Problem Solving and Uniformed Search

Problem solving consists of four steps:

- Goal formulation
- Problem formulation
- Problem solving
- Solution execution

4.1 Well-defined problems

Well-defined problems consist of:

- The initial state
- Transition model (putting together actions and states)
- The goal test
- Path cost

A solution to a problem is an action sequence that leads from the initial state to a goal state.

4.2 Abstraction

- **Abstraction:** Removing unnecessary details. It should be valid and useful.
- **Abstraction of world states:** (e.g., we ignored the weather).
- **Abstraction of actions:** (e.g., we ignored turning on the radio).

State space is different from a search tree – a world state is different from a search node.

4.3 Node Components

A node consists of:

- A current state
- A link to its parent
- An action
- A cost
- Depth (how many steps/actions)

4.4 Fringe (Frontier)

The fringe (or frontier) is a set of nodes that have not yet been expanded. Nodes from the frontier are called leaf nodes. The fringe is managed as a queue.

4.5 Algorithm's Performance Evaluation

- Completeness
- Optimality
- Time complexity
- Space complexity

Search cost = how much time and space is needed to find a solution.
Total cost = search cost + path cost.

4.6 Uninformed (Blind) Search

Uninformed search is the opposite of informed (heuristic) search.

4.6.1 Breadth-First Search (BFS)

Breadth-first search is a complete method. It is optimal but may not always result in the lowest path cost. In BFS:

- All nodes are expanded at a given depth.
- The shallowest unexpanded node is chosen for expansion, using FIFO for the frontier.

BFS can visit the same state multiple times.
Time complexity: $O(b^{d+1})$
The memory requirements are a bigger problem than execution time.

4.6.2 Uniform-Cost Search (UCS) / Dijkstra's Algorithm

Uniform-cost search is a modified version of BFS for finding optimal solutions. In UCS:

- Expand the node n with the lowest path cost $g(n)$.
- Beware of zero-cost steps, as they can cause cycling!

It guarantees completeness with a lower bound. However, the time/space complexity can be much worse than BFS.

4.6.3 Depth-First Search (DFS)

Depth-first search expands the deepest node using LIFO (a recursive approach). In DFS:

- DFS is not complete and may not find an optimal solution.

Time complexity: $O(b^m)$, where m is the maximum reachable depth.
Space complexity: $O(b \cdot m)$.
Space complexity can be decreased via backtracking.

4.6.4 Iterative Deepening Search

This method combines the benefits of BFS and DFS:

- It is complete and optimal.
- It has low memory consumption $O(bd)$ and time complexity $O(bd)$.

Iterative deepening is the preferred uninformed search method when the search space is large, and the depth of solutions is unknown.

4.6.5 Bidirectional Search

Bidirectional search involves searching forward from the initial state and backward from the goal state simultaneously.

4.6.6 Backward Search

Search starts from the goal and moves backward towards the initial state.

4.6.7 Tree Search vs. Graph Search

- **Tree Search:** Revisits the same states without checking.
- **Graph Search:** Keeps track of already expanded states (closed).

4.7 Search with Partial Information

In cases with partial information:

- No sensors: The agent does not know the current state and works with belief states representing possible real states.
- Non-deterministic actions: The agent may need plans with alternatives.
- Unknown actions: Some actions are not pre-defined.

5 Tutorial 2

The first part involves programming the algorithms, and the second part involves programming an agent. We focus on search algorithms like BFS and DFS. If no solution exists, return `noSol`.

5.1 Graph Search vs. Tree Search

- **Graph Search:** Check if the state has already been visited before proceeding.
- **Tree Search:** Does not perform this check.

Command to run:

```
% python3 ./play_pacman.py -a MyAgent
```

Run the command in the Pacman folder.