Artificial Intelligence

Lecture 02:

Problem Solving & Reinforcement Learning

CS2AI 2025/26

Ferran Espuny-Pujol

Attendance Monitoring

Lecture 02: Learning Outcomes

By the end of this week/session, you should be able to

Explain foundational rational agent definitions and differentiate between the five main types of agent programs.

Formulate search problems and compare uninformed strategies against informed A* search to ensure optimal solution efficiency.

Analyse adversarial environments using MiniMax and alpha-beta pruning to determine optimal competitive strategies.

Explain reinforcement learning principles and implement Python solutions to maximise expected cumulative rewards.

Lecture 02: Problem Solving & RL

Today's Lecture

Rational Agents

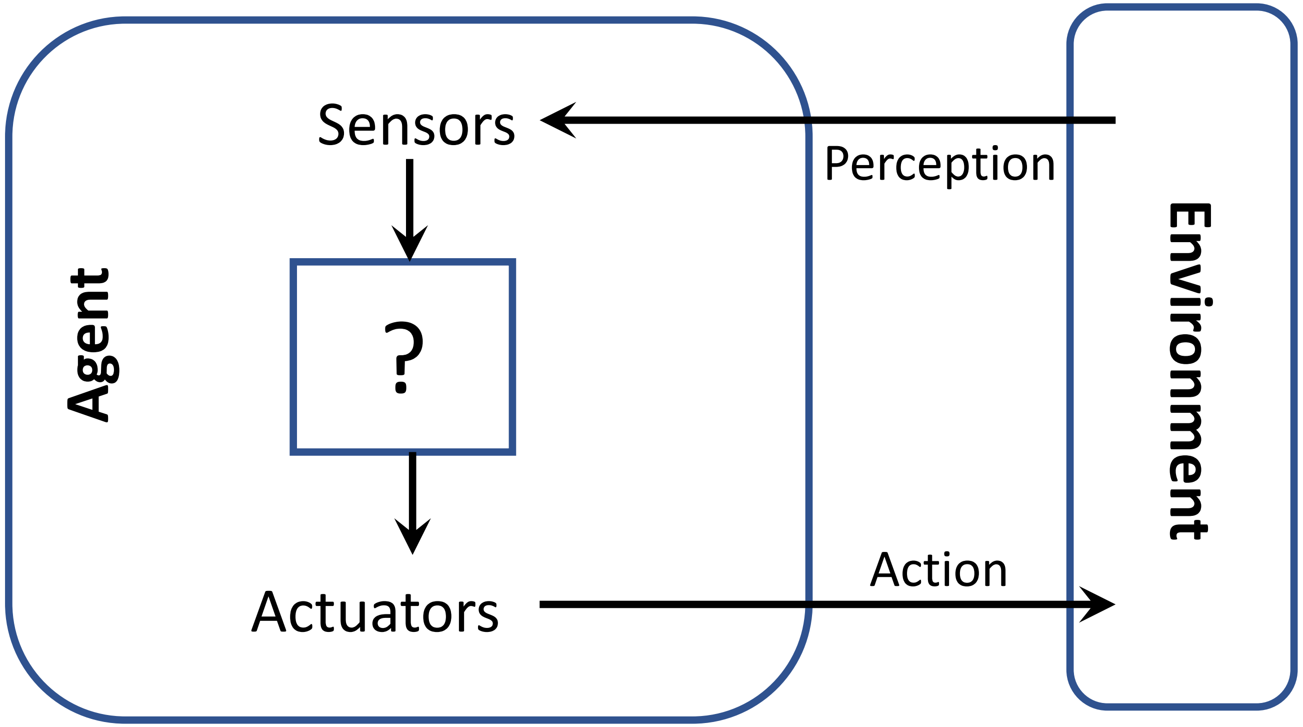
Problem Solving by Searching

Game Theory

Reinforcement Learning (RL)

Intelligent Agent

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



Examples:

Human agents (eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators)

Robotic agent (cameras and infrared range finders for sensors; various motors for actuators)

Remember: agent comes from the Latin *agere*, to do

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Rational Agent

- Designing an intelligent agent requires specifying a **task environment**:
  - **PEAS**: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors
  - Performance measure can be considered as an objective criterion for success of an agent's behaviour

- An **agent is rational** if for each possible percept sequence, it selects an action that is expected to maximize its performance measure
  - Information gathering: doing actions to modify future percepts
  - Learning from gathered information (unless environment known and predictable)
  - No omniscience (knowing the actual outcomes of actions and acting accordingly)

- **Main kinds of agent programs (by how actions are selected, i.e. decision-making):** Simple reflex, Model-based reflex, Goal-based, and Utility-based agents.

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Simple Reflex Agents

Simple reflex agents respond directly to percepts (ignoring full percept sequence).

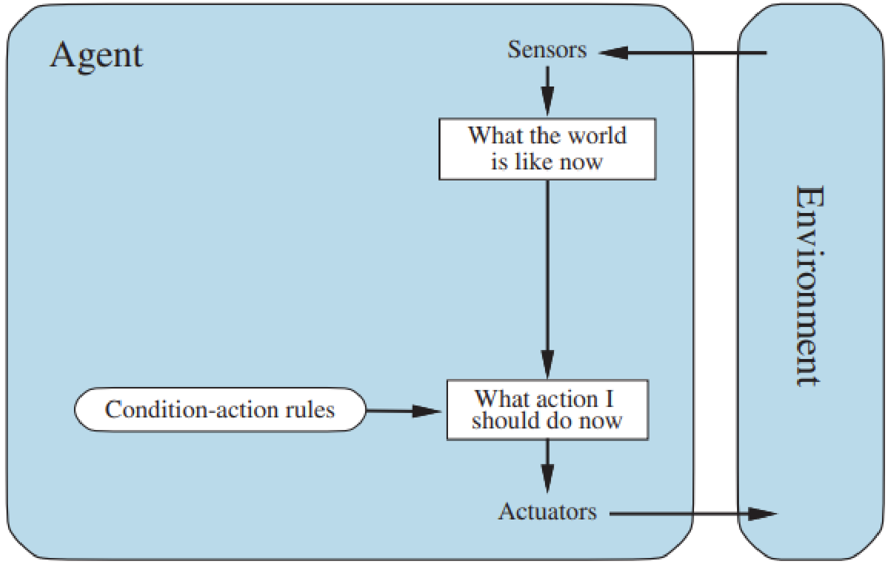
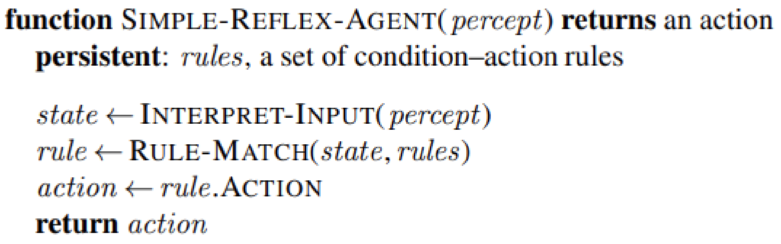


Figure 2.9 Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Use cases: environmental monitoring, quality control and inspection, process automation and resource allocation

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://www.ibm.com/think/topics/simple-reflex-agent

Model-based Reflex Agents

Model-based reflex agents maintain internal state to track aspects of the World that are not evident in the current percept.

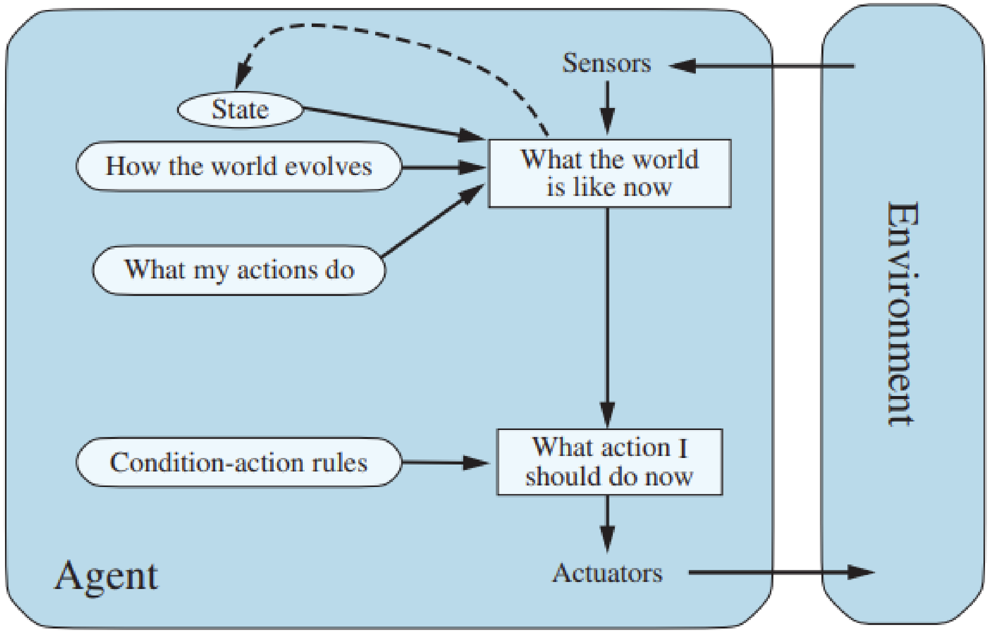
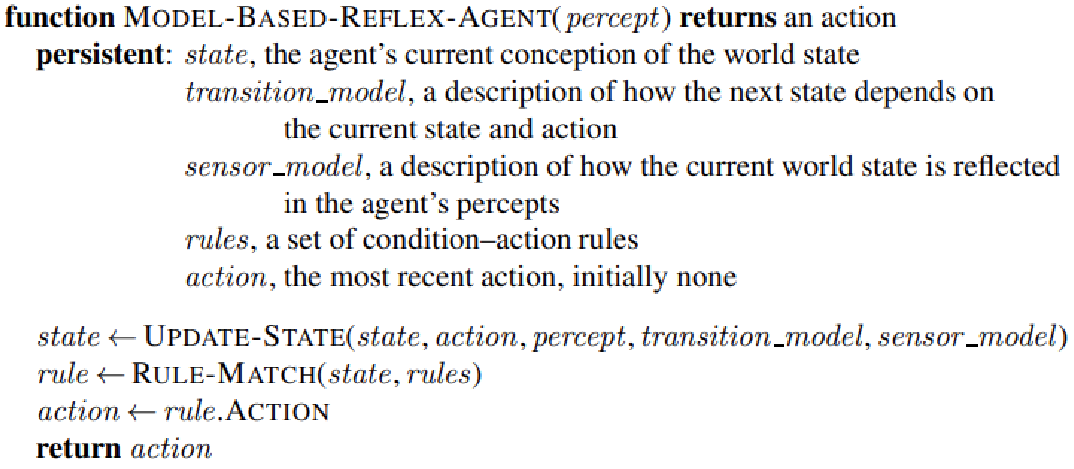


Figure 2.11 A model-based reflex agent.



```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              transition_model, a description of how the next state depends on
                  the current state and action
              sensor_model, a description of how the current world state is reflected
                  in the agent's percepts
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, transition_model, sensor_model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

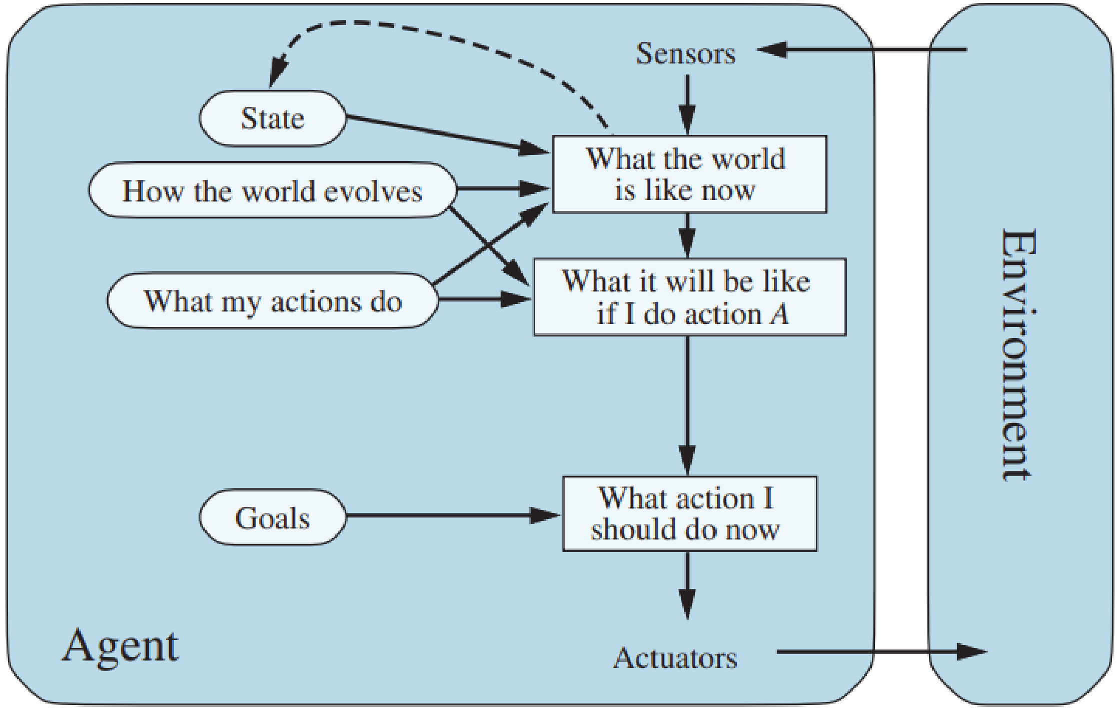
Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

Use cases: robotics and autonomous vehicles (dynamic environments), game-playing AI, large enterprise automation

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://www.ibm.com/think/topics/model-based-reflex-agent

Goal-based Agents

Goal-based agents act to achieve their goals (approach useful for problem solving).


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action *A*
Goals
What action I should do now
Agent
Actuators
Environment

Requires: goal definition, planning, action selection, and execution

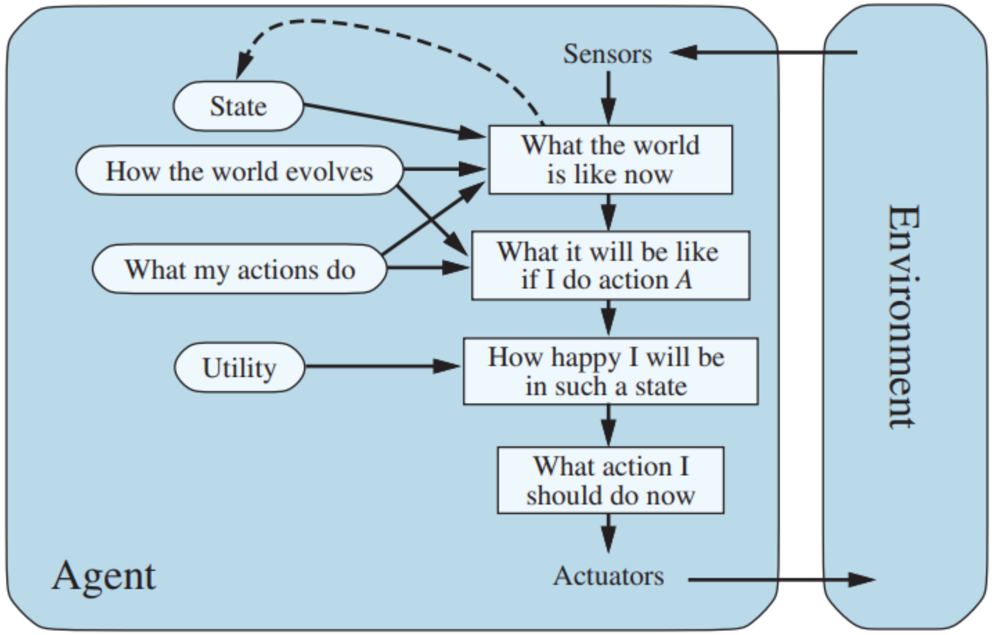
Use case: robotics warehouse automation

Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://www.ibm.com/think/topics/goal-based-agent

Utility-based Agents

Utility-based agents try to maximize their own expected *happiness* (overall utility benefit vs single goal achievement)



Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Requires: perception, internal modelling, action generation, outcome prediction, utility assessment, action prediction, action

Use cases: self-driving cars, e-commerce (pricing optimisation and product recommendation),

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://www.ibm.com/think/topics/utility-based-agent

Learning Agents

Learning: process of modification of each component of the agent to bring the components into closer agreement with available feedback information, thereby improving the overall performance of the agent

Structure of Learning Agents

- **Critic**: evaluates agent's actions according to a performance standard (agent's success or failure information, e.g. as reward or penalty) and provides feedback
- **Performance element**: how actions are selected (e.g. goal-based, utility-based)
- **Learning element**: using feedback and experience, determine how to modify the performance element to do better in future
- **Problem generator**: suggests actions that will lead to new and informative experiences (e.g. suboptimal actions in the short term)

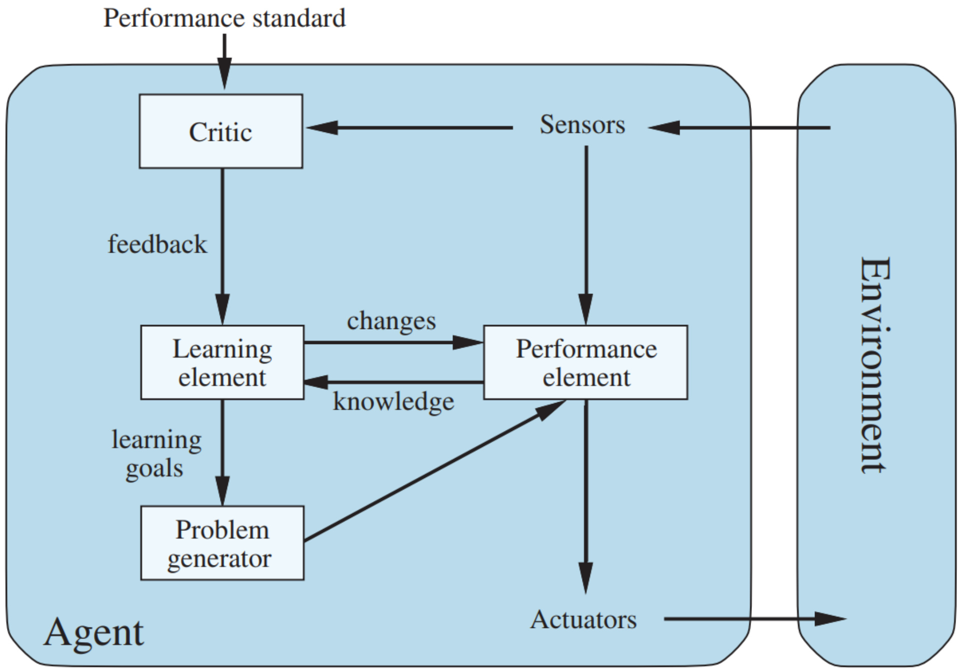


Figure 2.15 A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Multi-Agent Systems

There are 5 main types of AI agents: simple reflex agents, model-based reflex agents, goal-based agents, utility-based agents and learning agents.

Each type has strengths and applications, ranging from basic automated to highly adaptable systems.

All 5 types can be deployed together as part of a multi-agent system, with each agent specializing in handling the part of the task for which they are best suited.

Further Reading

AI Agent Use Cases (IBM) https://www.ibm.com/think/topics/ai-agent-use-cases

Agentic Architecture (IBM) https://www.ibm.com/think/topics/agentic-architecture

Agents White Paper, 2025 (Google) https://www.kaggle.com/whitepaper-agents

Agents Companion White Paper, 2025 (Google) [multi-agent systems and advanced topics] https://www.kaggle.com/whitepaper-agent-companion

Lecture 02: Adversarial Search, Game Theory & RL

Today's Lecture

Rational Agents

Problem Solving by Searching

Game Theory

Reinforcement Learning (RL)

Problem Solving by Searching

Problem solving

- The theory and technology of building **rational agents** that can plan to **solve problems**
  - a **problem-solving agent** can find ahead a sequence of actions that achieves its goals when no single action will do

Searching

- The process of looking for a sequence of actions that reaches the goal is called search
  - A search algorithm takes a problem as input and returns a solution in the form of an action sequence
  - Once a solution is found, the actions it recommends can be carried out (execution phase)

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Solving Problems by Searching

- Goal formulation is the first step in problem solving which is based on the current situation (or *state*) and the agent's performance measure

- Problem formulation is the process of deciding what actions and states to consider, given a goal
  - ▸ Initial state
  - ▸ Possible actions available
  - ▸ Transition model (state given action)
  - ▸ Goal test
  - ▸ Path/Action cost function

These implicitly define the **state-space** of the problem: set of all states reachable from the initial state by any sequence of actions

- The state space forms a directed network or graph in which the leaf nodes are states and the links between nodes are actions.

Example – Navigation Problem

- Initial state: *In(Arad)*

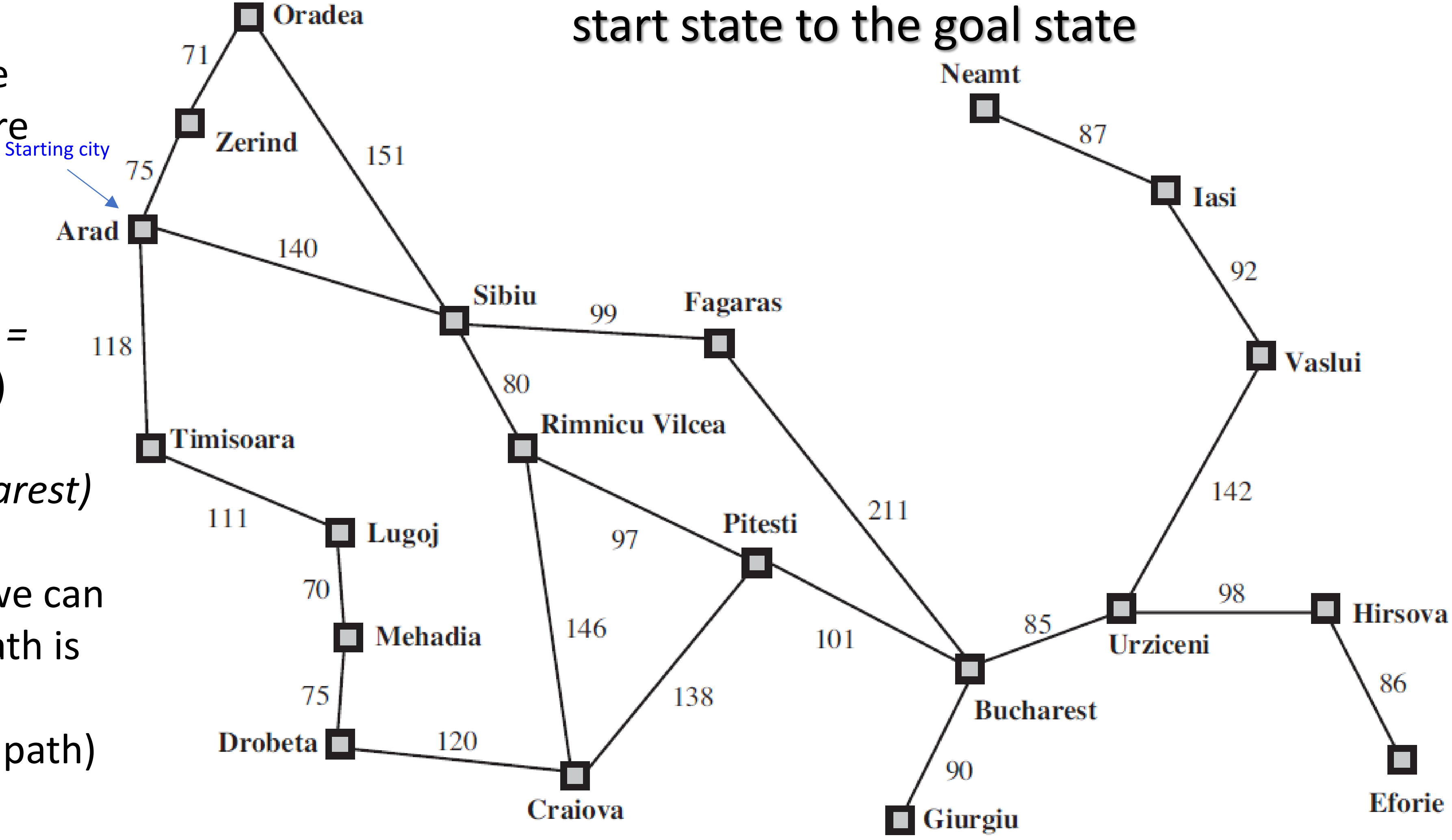
- Possible actions: In the state *In(Arad)*, the valid actions are {*Go(Sibiu), Go(Timisoara), Go(Zerind)*}

- Transition model: *In(Zerind)* = RESULT(*In(Arad),Go(Zerind)*)

- Goal test: If state is *In(Bucharest)*

- Path cost: Length in miles (we can assume that the cost of a path is the sum of the costs of the individual actions along the path)

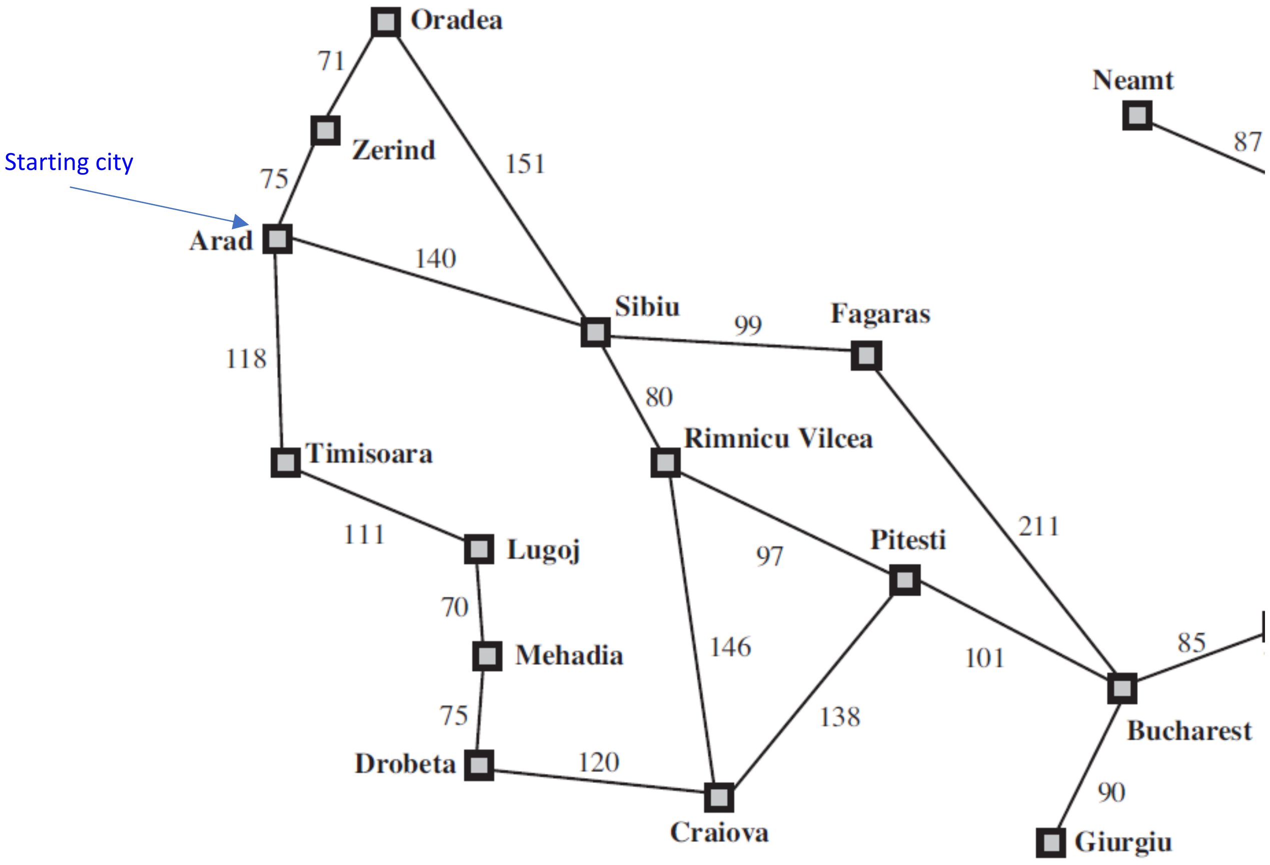
Given a map, pick the shortest route from start state to the goal state



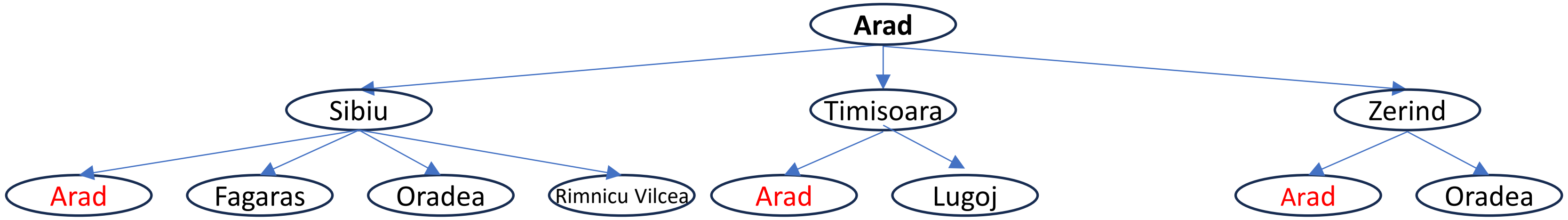
Example – Navigation Problem

- Initial state: *In(Arad)*
- Possible actions: In the state *In(Arad)*, the valid actions are {*Go(Sibiu), Go(Timisoara), Go(Zerind)*}
- Transition model: *In(Zerind)* = RESULT(*In(Arad),Go(Zerind)*)
- Goal test: If state is *In(Bucharest)*
- Path cost: Length in miles (we can assume that the cost of a path is the sum of the costs of the individual actions along the path)

Given a map, pick the shortest route from start state to the goal state



Search Tree (depth 2)



Arad

Sibiu

Timisoara

Zerind

Arad

Fagaras

Oradea

Rimnicu Vilcea

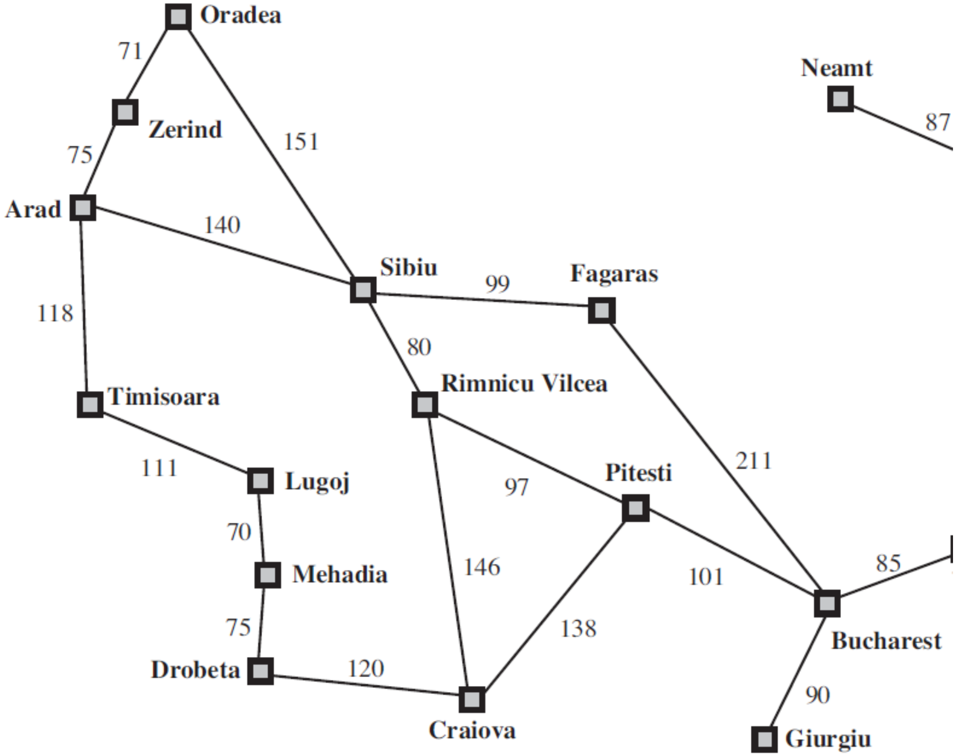
Arad

Lugoj

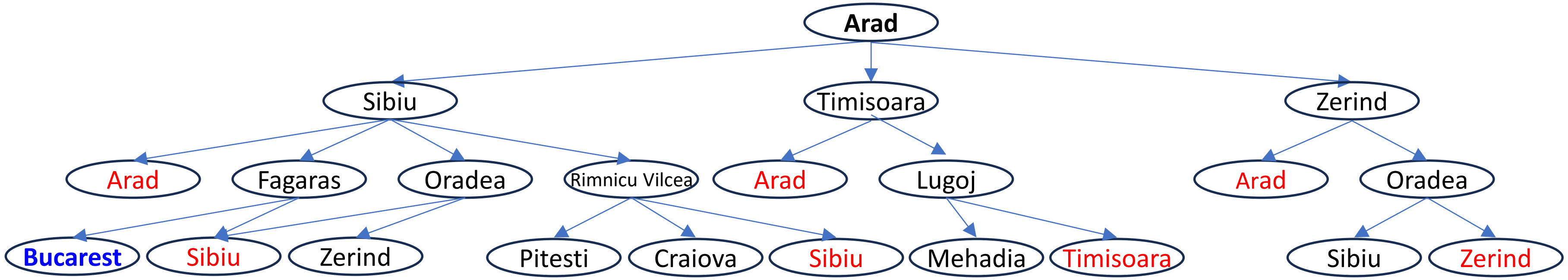
Arad

Oradea

Text in red indicates a cycle (the search tree returns to a previous state)



Search Tree (depth 3)



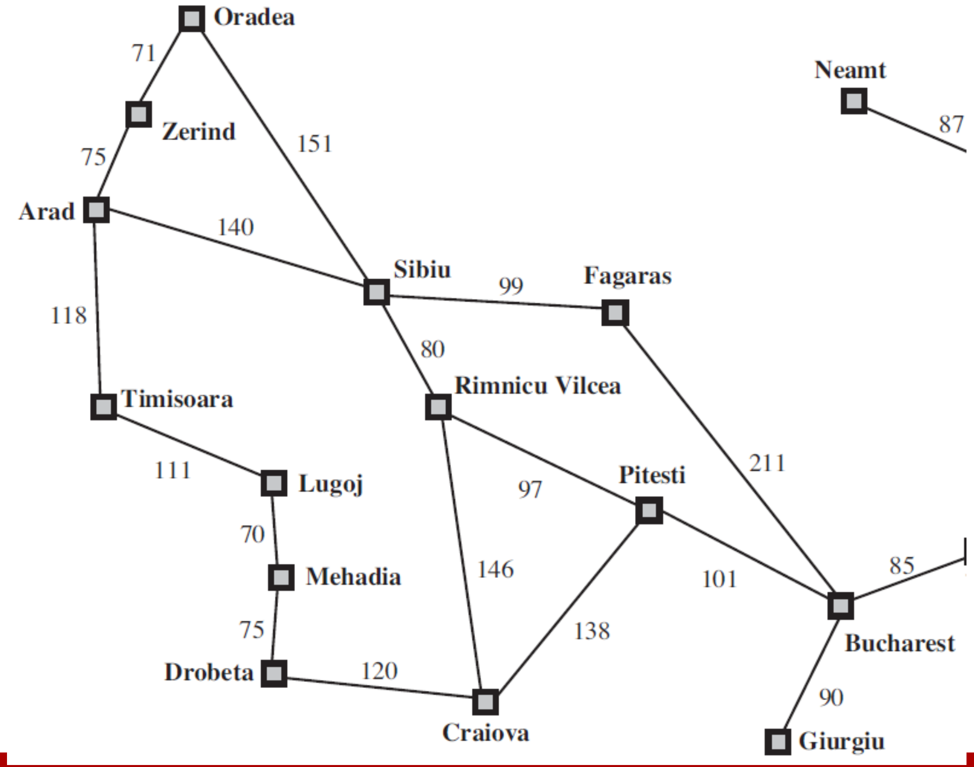
Arad

Sibiu Timisoara Zerind

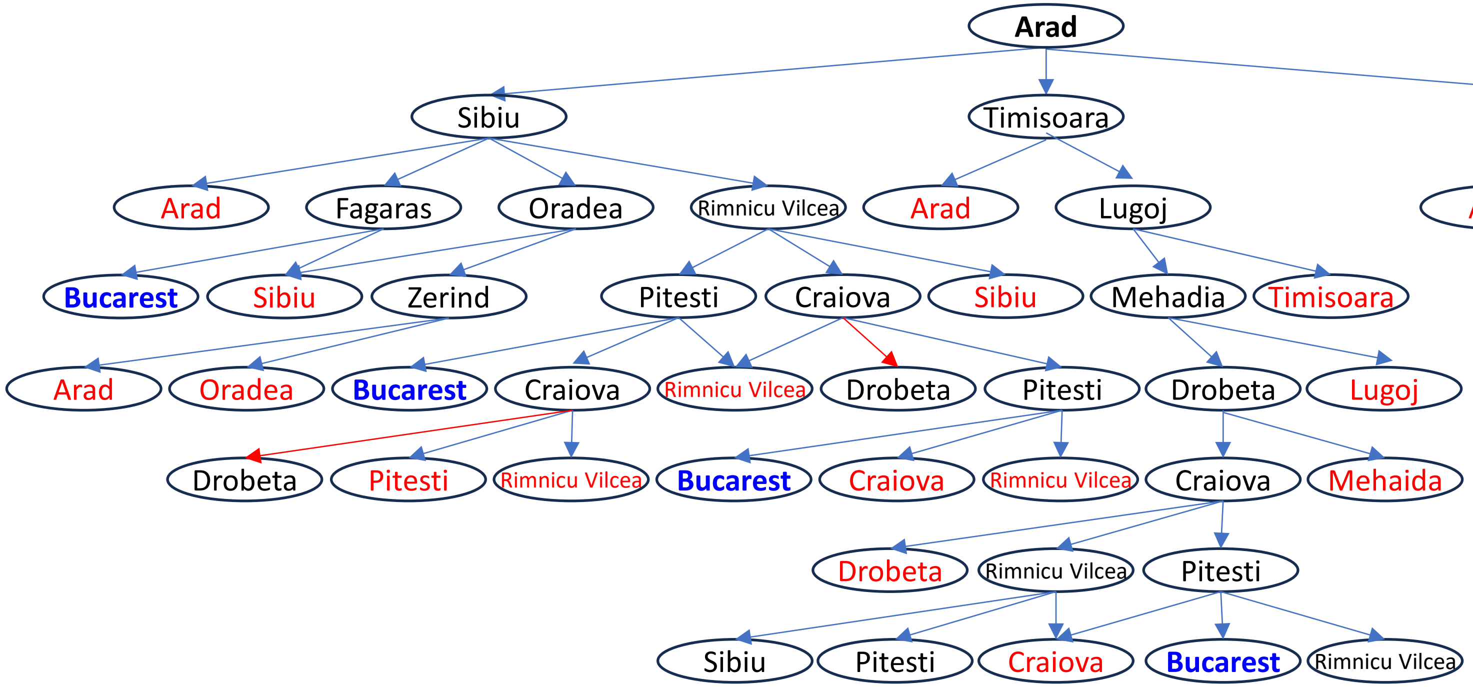
Arad Fagaras Oradea Rimnicu Vilcea Arad Lugoj Arad Oradea

Bucarest Sibiu Zerind Pitesti Craiova Sibiu Mehadia Timisoara Sibiu Zerind

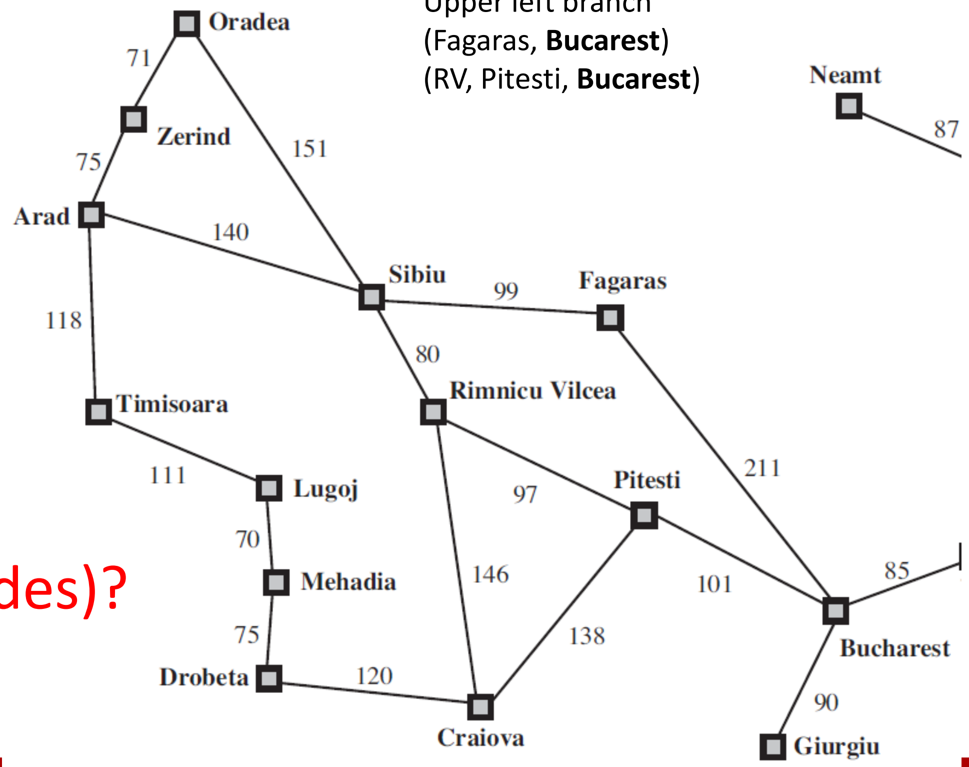
Text in red indicates a cycle (the search tree returns to a previous state)



Search Tree (depth 7)



Upper left branch
(Fagaras, **Bucarest**)
(RV, Pitesti, **Bucarest**)



Text in red indicates a loop (the search tree returns to a previous state)
The red arrow is followed by a loop returning to Arad (not made explicit)

Do we need to explore the full search tree (expand all nodes)?
What is the best search strategy?

Uninformed Search Strategies

Uninformed Search Strategies

Use only the information available in the problem definition (also called Blind Search)

- **Depth-first** search (Uses Stack)

  Always selects the *deepest* frontier node from the start node for expansion (low memory requirement)
  It can get trapped in cycles, otherwise finds the "leftmost" solution in the search tree (ignores costs)
  The "depth-limited search" variant fixes max depth, overcoming cycles at computational expense
  The "iterative deepening search" variant searches iteratively over increasing max depth thresholds

- **Breadth-first** search (Uses Queue)

  Always selects the *shallowest* frontier node from the start node for expansion
  It will find the "shallowest" solution in the search tree (ignores costs)

- **Uniform-cost** search (Uses Priority Queue) (also called Dijkstra's algorithm)

  Always selects the *lowest cost* frontier node from the start node for expansion
  It will find the lowest cost path in the search tree (computationally expensive), hence the optimum

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://inst.eecs.berkeley.edu/~cs188/textbook/search/uninformed.html

Informed (Heuristic) Search Strategies

Informed (Heuristic) Search Strategies

Use information from a heuristic function estimating the cost of each path from a node to the goal state

- **Greedy best-first** search

  Always selects the frontier node with the *lowest heuristic value* for expansion, which corresponds to the state it believes is nearest to a goal

  It is not guaranteed to find a solution or the optimal goal, and it is highly dependent on the heuristic

- **A*** search

  Always selects the frontier node with the *lowest estimated total cost* for expansion

  A* combines the total backward cost (sum of edge weights in the path to the state) used by uniform-cost search with the estimated forward cost (heuristic value) used by greedy search by adding these

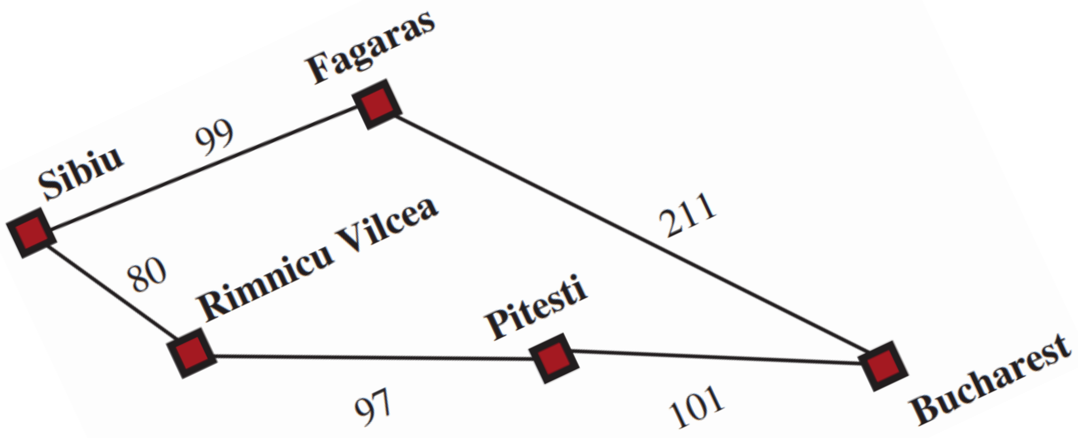
  It combines the generally high speed of greedy search with the optimality and completeness of uniform-cost search

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)
https://inst.eecs.berkeley.edu/~cs188/textbook/search/informed.html

Group Activity

Problem. Find the lowest distance path from Sibiu to Bucharest

- Represent the search tree of depth 3, sorting nodes in alphabetical order
- For each search method, give the solution (e.g. SFB) and expanded nodes



Heuristic: Straight line distances to Bucharest

Bucharest	0		
Fagaras	176	Rimnicu Vilcea	193
Pitesti	100	Sibiu	253

Post in Padlet: search tree and each method solution & expanded nodes (hint in 5 minutes, discussion in 10 minutes)

Sibiu-Bucharest Search Tree (depth 3)

Sibiu-Bucharest Solutions: Uninformed Search

Sibiu-Bucharest Solutions: Greedy Search

(limited view (ending up finding S.F.B as solution))

Sibiu-Bucharest Solutions: A* Search

A Note on Heuristics

- A* search is guaranteed to find the lowest cost path if the heuristic function is admissible
- A heuristic *h(n)* is admissible if for every node *n*,

$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to reach the goal state from n

- An admissible heuristic never overestimates the true cost, i.e., it is optimistic
  - ▸ Example: straight line distance never overestimates the actual road distance

Further Reading

“Solving Problems by Searching”, Chapter 3 in S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

“Search”, Chapter 1 in UC Berkeley’s Introduction to Artificial Intelligence https://inst.eecs.berkeley.edu/~cs188/textbook/search/

Lecture 02: Problem Solving & RL

Today's Lecture

Rational Agents

Problem Solving by Searching

Game Theory

Reinforcement Learning (RL)

Adversarial Search and Game Theory

- In **adversarial search** problems (commonly called games or economies), agents need to plan a goal while adversarial agents are planning against it.
  - Only competitive environments considered: multiple agents' goals are in conflict.
  - Adversarial search returns a **strategy**, or **policy**, which recommends the best possible action(s) given some configuration of our agent(s) and their adversaries, and possibly given a time limit (forcing an approximate solution)
  - Problem solving by search returned a plan to achieve the goal (no adversaries!).

- Mathematical **game theory** views any multiagent environment (be it competitive or collaborative) as a game if the impact of each agent on the others is "significant".

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Perfect Information Deterministic Zero-Sum

- We will consider **deterministic zero-sum games**, in which outcomes are deterministic and an agent's gain is directly equivalent to the opponent's loss and vice versa. Additionally, we will assume **perfect information**, i.e. fully observable game states.
  - Examples: Tic Tac Toe, Chess, Checkers, Go

- Example probabilistic zero-sum games not considered:
  - Perfect information, e.g. Monopoly, Backgammon
  - Imperfect information, e.g. Scrabble, Poker, Bridge

- Other not considered games:
  - Player vs environment games (not zero-sum), e.g. pac-man,
  - Team games, e.g. football, rugby, tennis, etc.

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Two-Player Zero-Sum Games

- MAX and MIN players competing, with MAX moving first.
  - At the end, points are awarded to the winning player and penalties are given to the loser

- Formulation of a game as a kind of search problem :
  - S0: The initial state, which specifies how the game is set up at the start
  - PLAYER(s): Defines which player has the move in a state s
  - ACTIONS(s): Returns the set of legal moves in a state s
  - RESULT(s, a): The transition model, which defines the result of a move (action a)
  - TERMINAL-TEST(s): A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states
  - UTILITY(s, p): A utility function (also called an objective or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p

- The initial state, action and result functions define the state space graph (leaf nodes are states and edges are moves). We can build the game tree for MAX (conditional on MIN moves)…

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Tic-Tac-Toe Game Tree Example

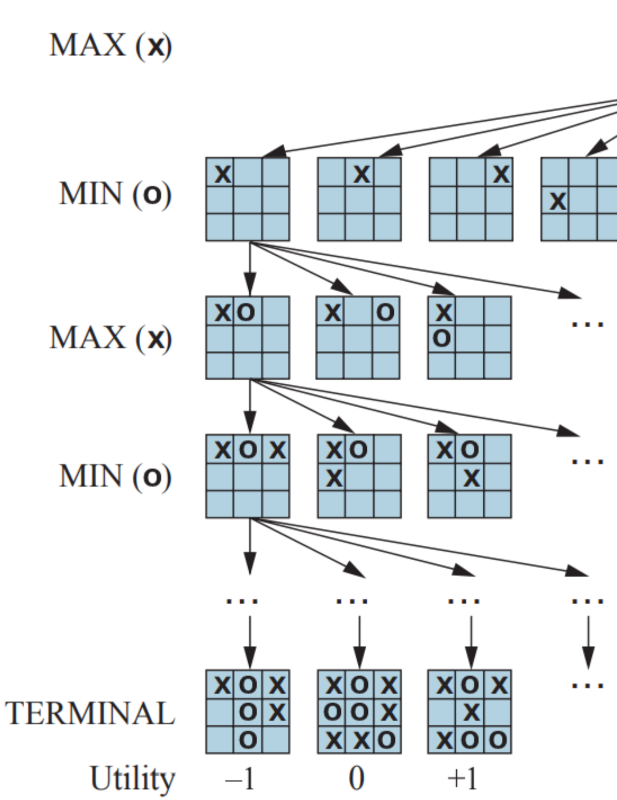


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

- Game ends when we reach leaf nodes corresponding to **terminal states**: *either one player has three in a row or all the squares are filled*
- The number on each leaf node indicates the **utility value** of the terminal state from the point of view of MAX
- High utility values are assumed to be good for MAX and bad for MIN (which is how the players get their names)

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

MiniMax Search

- The minimax value of each state in the tree is the (optimal) utility for MAX assuming <u>both players play optimally</u> from that state to the end of the game
  - ▸ The minimax value of a terminal state is just its utility

$$\text{MINIMAX}(s) =$$

$$\begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

- The minimax search finds the best move for MAX by recursively first calculating the game tree and the utilities for each terminal status, and then backs up the minimax values (which assume that both players play optimally)
  - ▸ It returns the optimal solution (against optimal adversary) if the tree is finite

S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Interpreting MiniMax Values

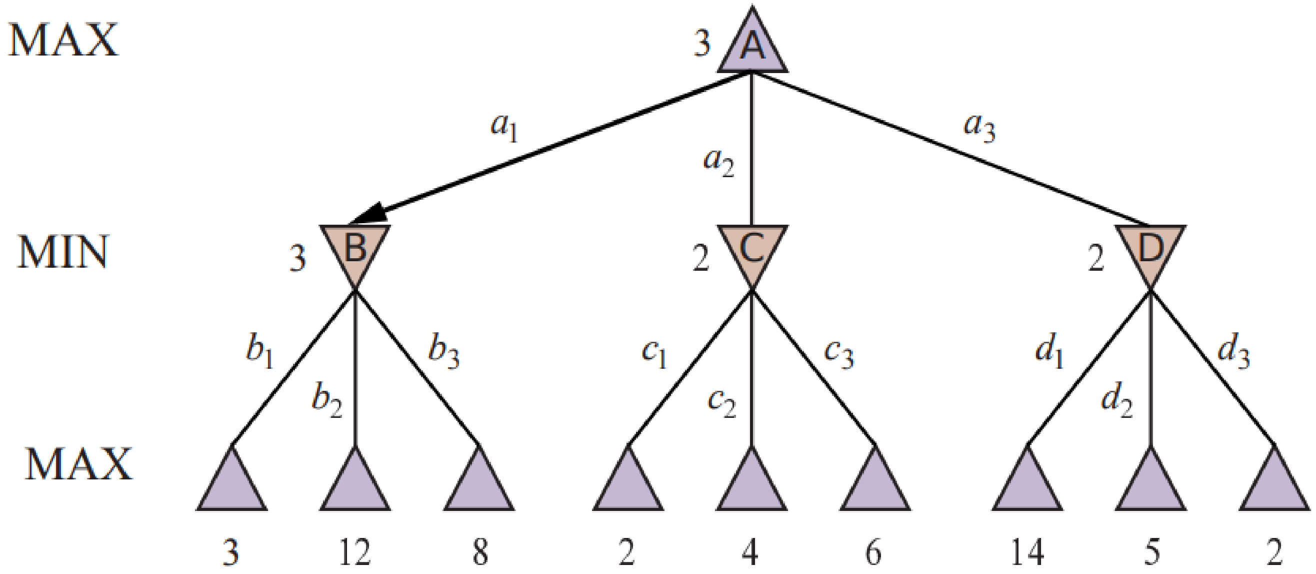


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

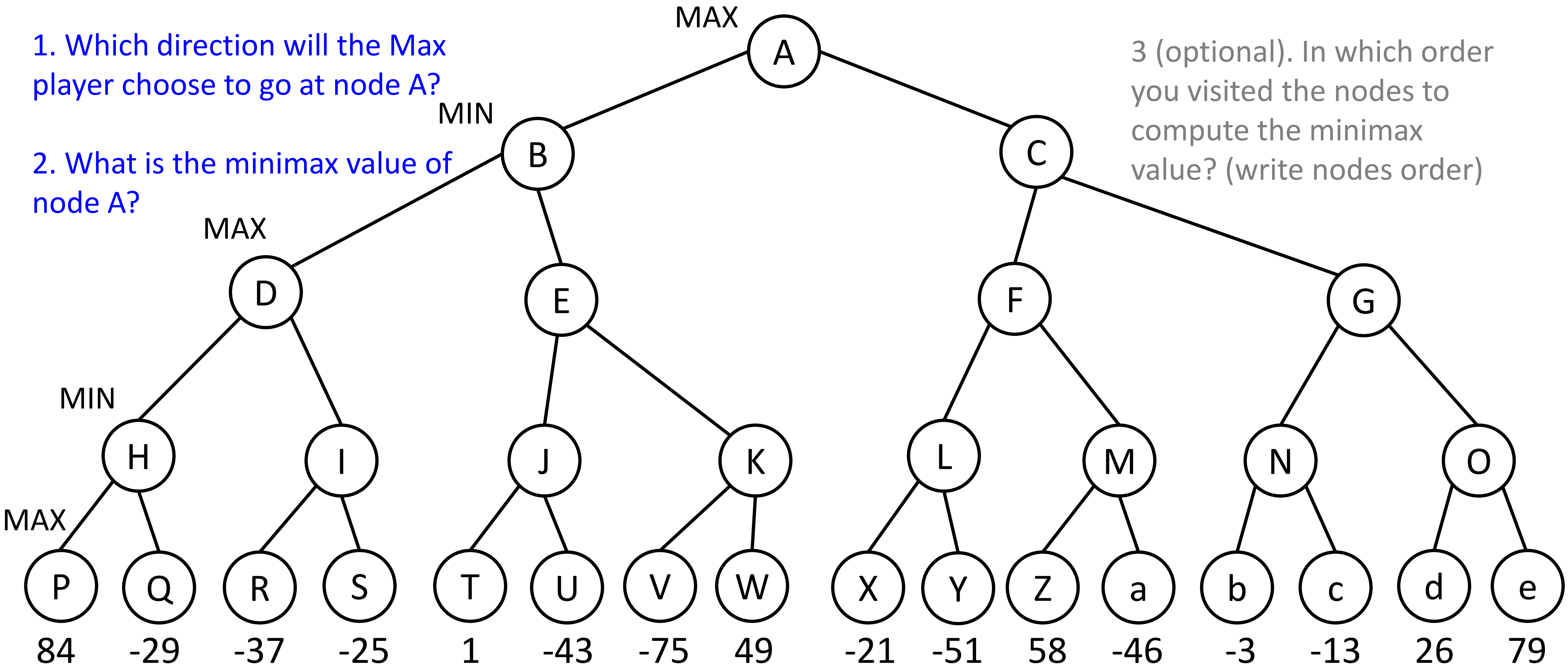
S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

Group Activity (5-10 mins): MiniMax Algorithm

1. Which direction will the Max player choose to go at node A?

2. What is the minimax value of node A?

3 (optional). In which order you visited the nodes to compute the minimax value? (write nodes order)



MiniMax Algorithm Group Activity Solution

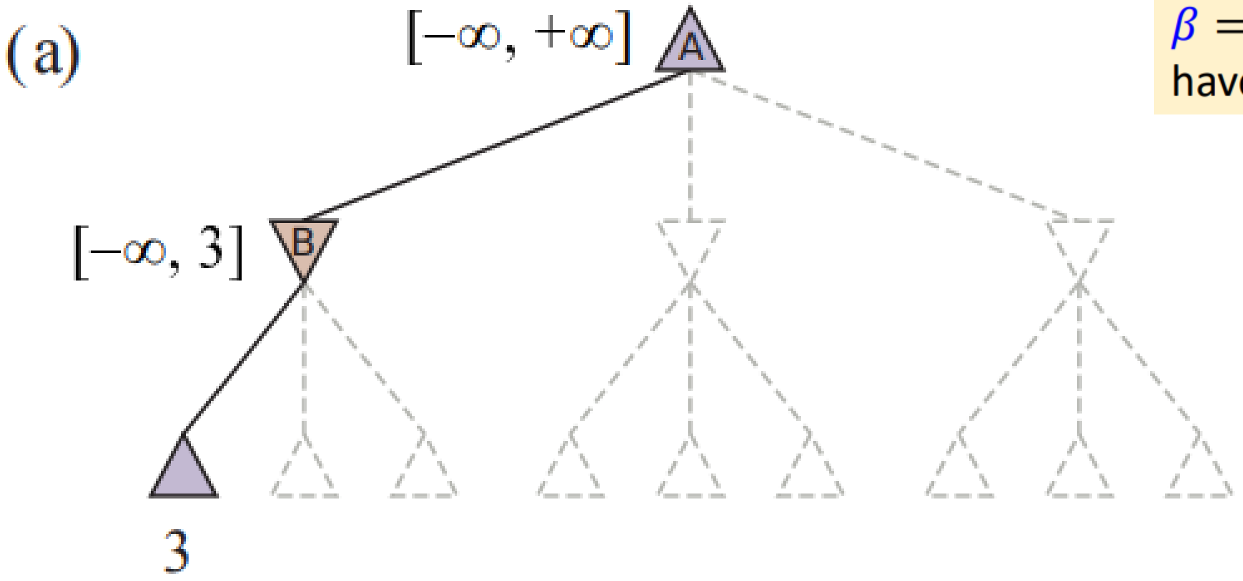
Limitations of MiniMax Search

- Time complexity $O(b^m)$, $\begin{cases} b \text{ – number of legal moves at each point (branching factor)} \\ m \text{ – maximum depth of the tree} \end{cases}$

- Unfortunately, we can’t eliminate the exponent, but it turns out we can effectively cut it in half

- The trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree
  - ▸ i.e., we can use the idea of pruning to eliminate large parts of the tree from consideration
  - ▸ When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision
  - ▸ This is known as alpha-beta prunning

Alpha-Beta Pruning: 2-ply Example

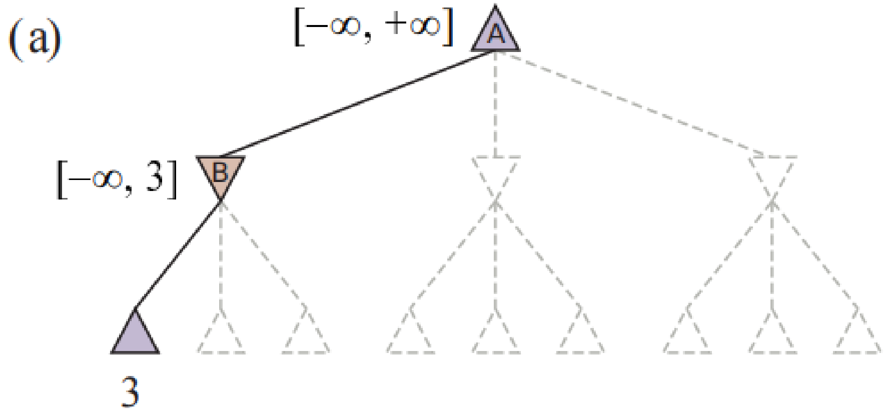


α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN

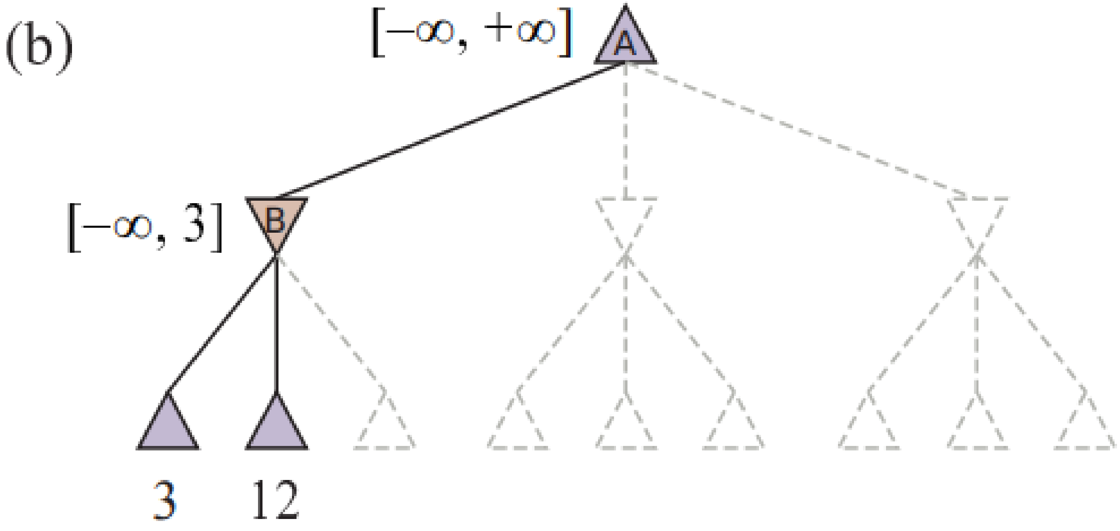
The first node below B has the value 3.
Hence, B, which is a MIN node, has value **at most** 3.

Alpha-Beta Pruning: 2-ply Example



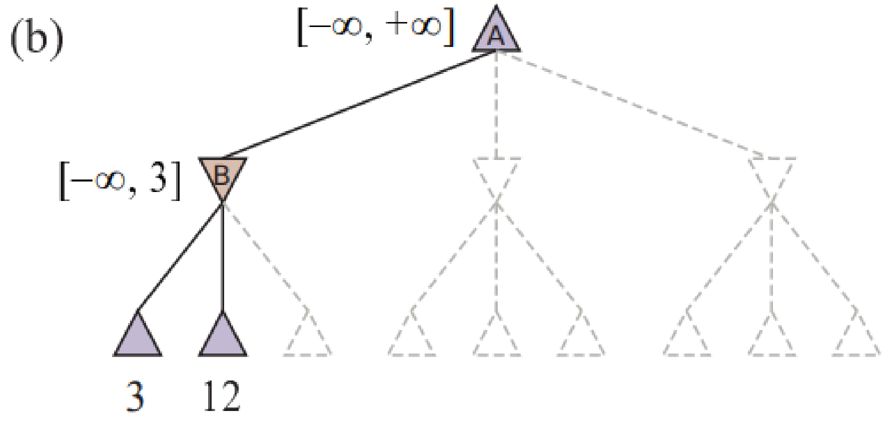
α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN



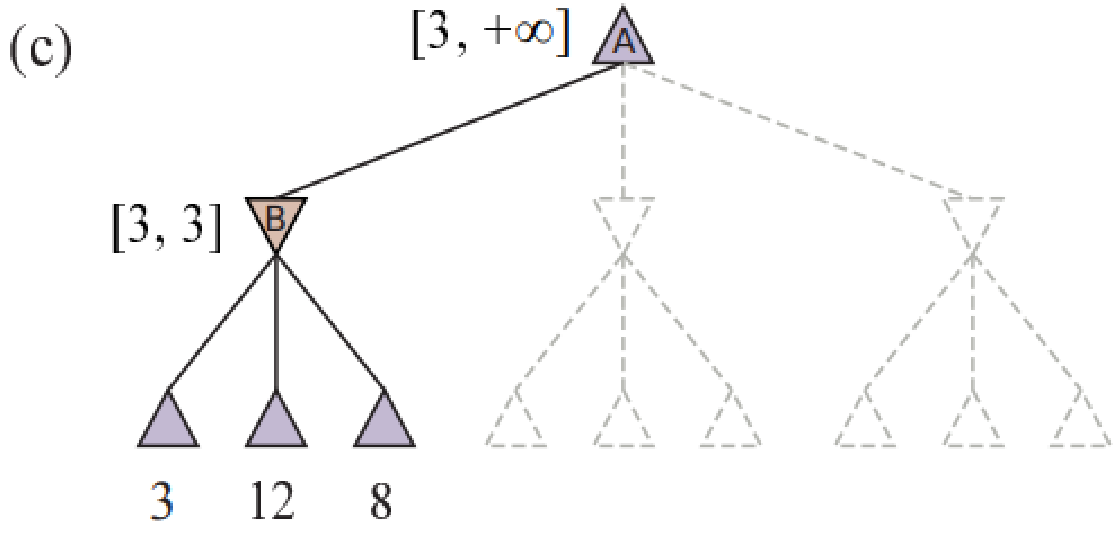
The second node below B has value greater than 3
MIN would avoid this move,
so the value of B is still **at most** β=3.

Alpha-Beta Pruning: 2-ply Example



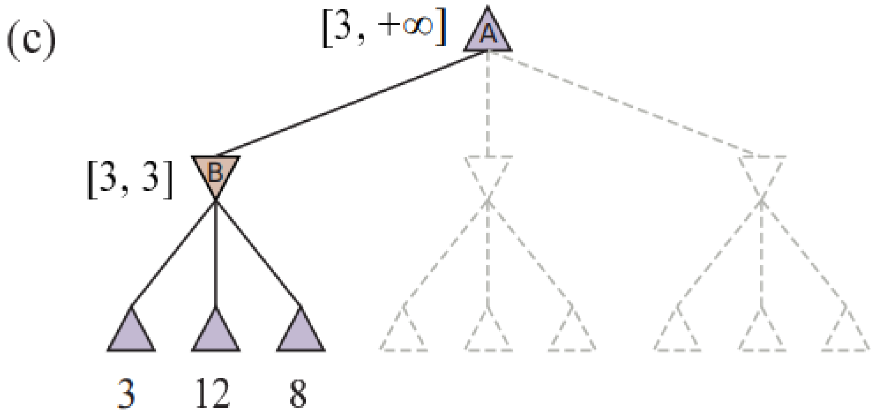
α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN



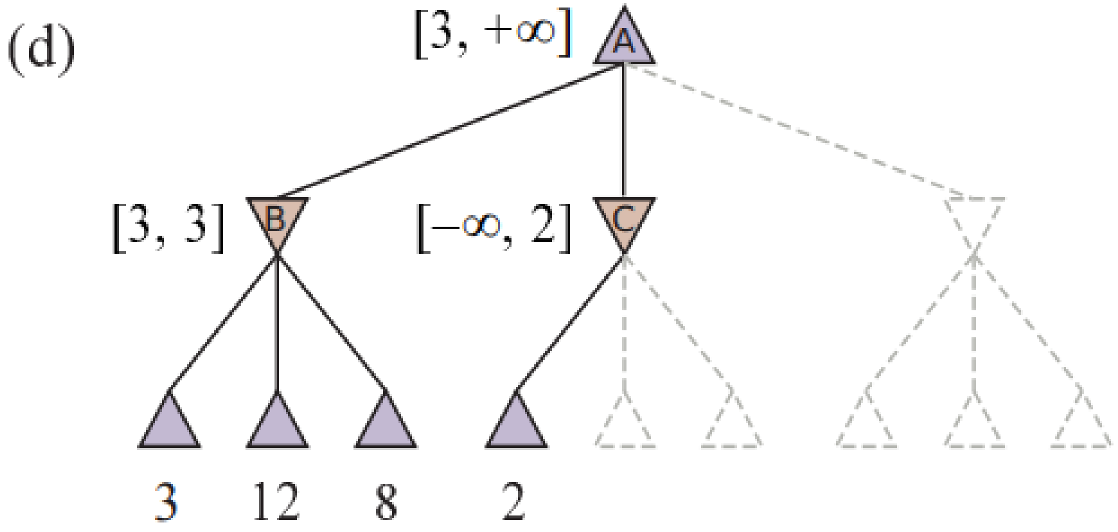
The third node below B has value greater than 3
MIN would avoid this move,
so the value of B is still **at most** β=3.

Alpha-Beta Pruning: 2-ply Example



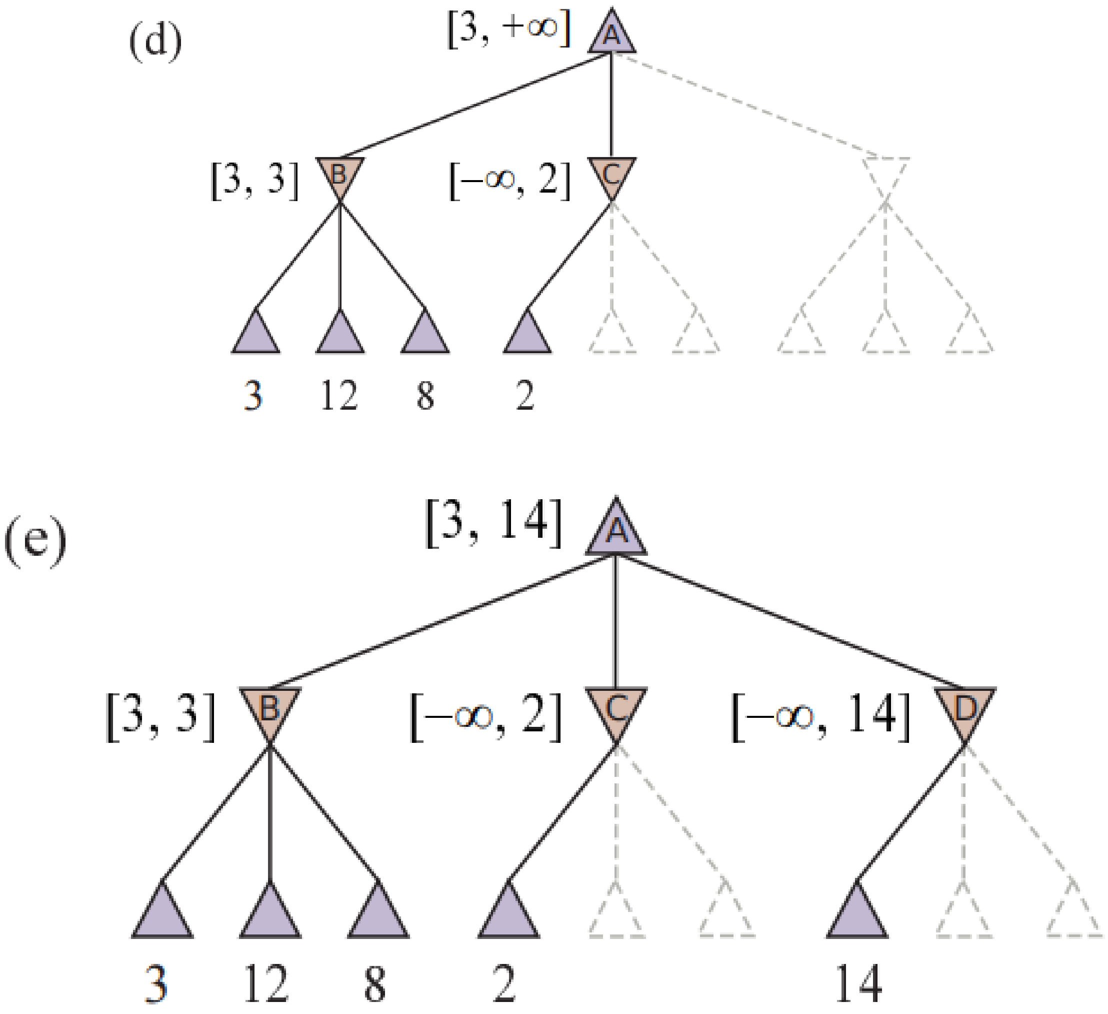
α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN



The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of at most 2. **But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C.** This is an example of alpha–beta pruning.

Alpha-Beta Pruning: 2-ply Example

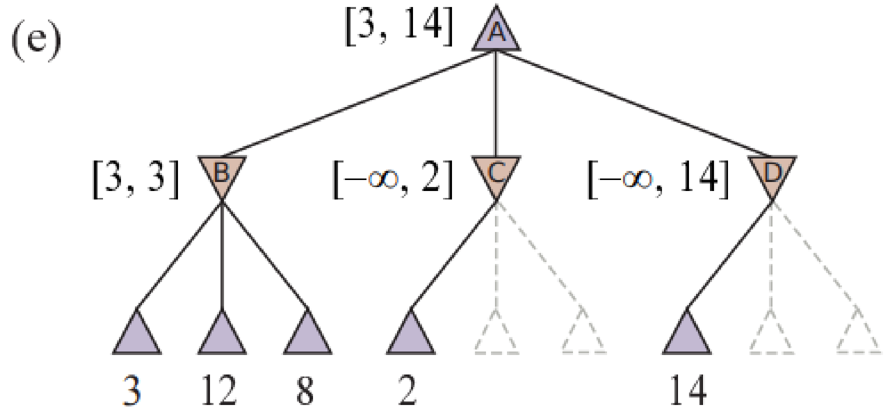


α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN

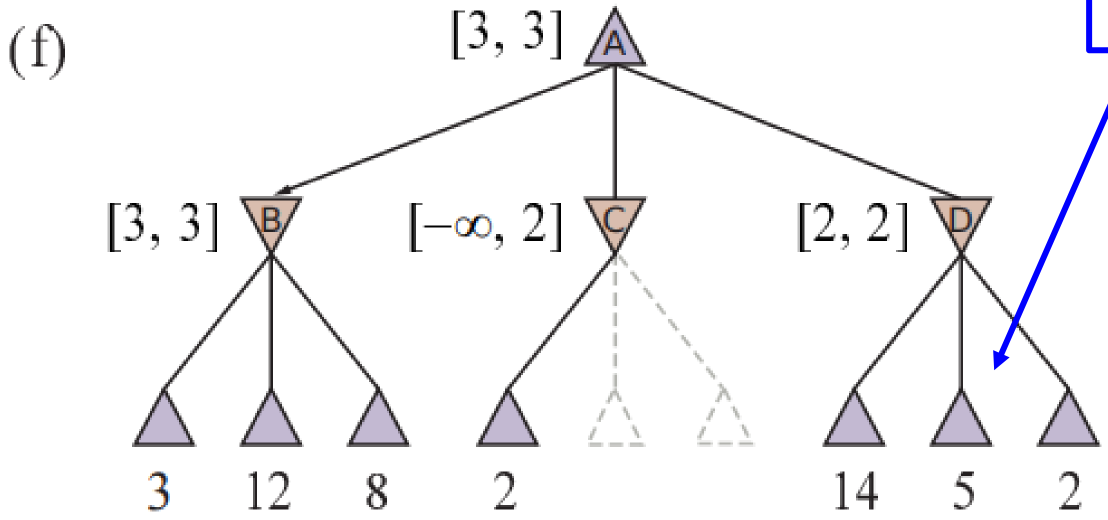
The first leaf below D has the value 14, so D is worth at most 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states.
Now have bounds on all of the successors of the root, so the root's value is also at most 14.

Alpha-Beta Pruning: 2-ply Example



α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN

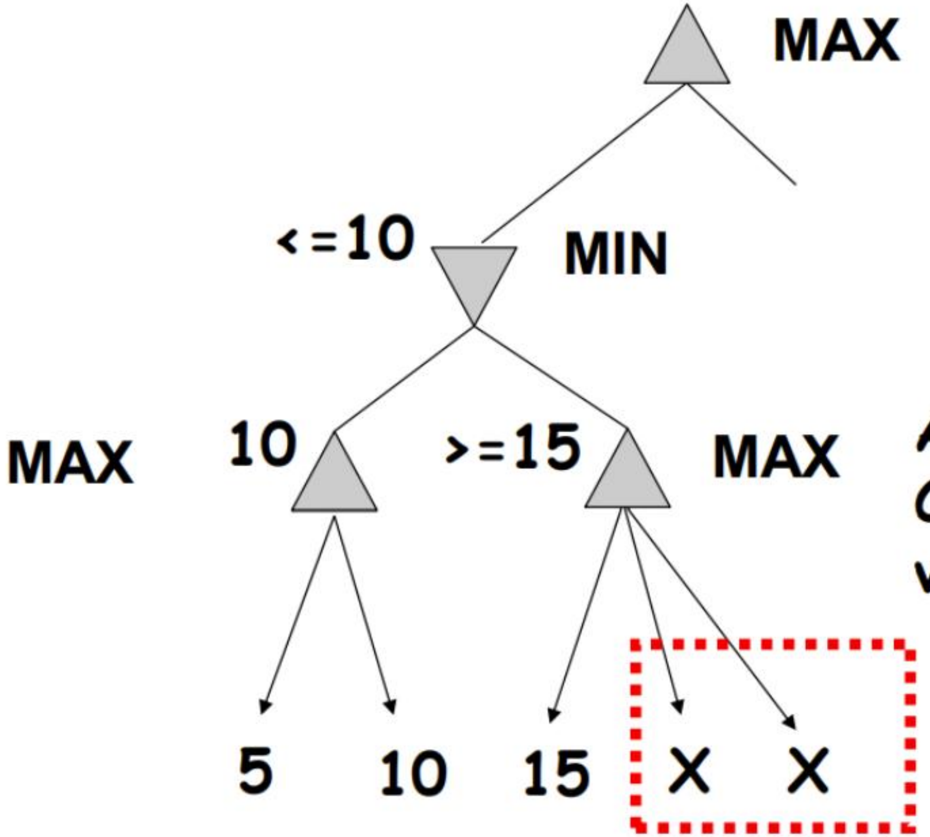


Ordering matters! (e.g. reverse D successors)
"Perfect ordering" can double depth of search

The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2.

MAX's decision at the root is to move to B, giving a value of 3.

Alpha-Beta Pruning: 3-ply Example



At MAX node:
Current best MIN
value β = 10 < 15

No need to look at these nodes!! (these nodes can only increase MAX value from 15)

Cutoff Test for Real-Time Decisions

Motivation

- The minimax algorithm generates the entire game search space.
  - E.g. Chess has b~35 moves/position and m~80 levels deep tree

    Seach space ~10^{150} states >> 10^{20} atoms in observable universe!
- The alpha–beta algorithm allows us to prune large parts of the search space, but still has to search all the way to terminal states for at least a portion of the search space
- Programs should cut off the search earlier and apply a heuristic evaluation function to states in the search, effectively turning nonterminal nodes into terminal leaves

Potential Solution

Alter minimax or alpha–beta in two ways:

- Replace the utility function by a heuristic evaluation function EVAL, which estimates the (non-terminal) position's utility, and
- Replace the terminal test by a cutoff test that decides when to apply EVAL

Example Chess Evaluation Function

$$\begin{aligned} eval(s) = \; & w1 * material(s) + \\ & w2 * mobility(s) + \\ & w3 * king\ safety(s) \\ & w4 * center\ control(s) \\ & + \ldots \end{aligned}$$

Example of weighted linear function (simple evaluation in chess)

- Material worth: pawn=1, knight =3, bishop=3, rook=5, queen=9
- Other features: king safety=0.5, good pawn structure=0.5
- Weights **normalised** so that eval is in the utility range
- Note: utility in chess is 0=loss, ½=draw, 1=win

- Does it work in practice? Time complexity $O(b^m)$
  - If b=35, m=4 then $b^m \approx 10^6$
  - 4 ply lookahead is a hopless player! (human novice)
  - 8 ply usual PC or human master; 12 ply Kasparov or Deep Blue (some games 40 ply search!)

b – number of legal moves at each point

m – maximum depth of the tree

Further Reading

"Adversarial Search and Games", Chapter 6 in S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach [2022] (available online at UoR Library)

"Games", Chapter 3 in UC Berkeley's Introduction to Artificial Intelligence https://inst.eecs.berkeley.edu/~cs188/textbook/games/

Simple chess program https://github.com/thomasahle/sunfish/blob/master/nnue/sunfish2.py

Lecture 02: Problem Solving & RL

Today’s Lecture

Rational Agents

Problem Solving by Searching

Game Theory

Reinforcement Learning (RL)

Sequential Decision Making

Sequential decision making

An agent decides the best next action based on its current state, by learning behaviours that will **maximise the reward (or reinforcement)** via exploratory trial-and-error or simulation, essentially learning a policy or set or rules for action.

The goal is to select actions to maximise (future) total cumulative reward.

Reward Hypothesis

"All goals can be described as maximisation of expected cumulative reward"

Do you agree with this statement?

It will work for this module!

https://davidstarsilver.wordpress.com/teaching/

Reinforcement Learning (RL) & Planning

Two fundamental problems in sequential decision making:

- **Reinforcement Learning (RL)**
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy

    a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Examples of RL Problems and Rewards

- Fly stunt manoeuvres in a helicopter
  - Reward for following desired trajectory, penalty for crashing
  - Refuelling might prevent future crash
- Manage an investment portfolio
  - Reward for each £ in bank
  - Investments may take months to mature
- Control a power station
  - Positive reward for producing power, penalty for exceeding safety thresholds
  - Optimizing immediate profit alone may deplete reserves or damage equipment

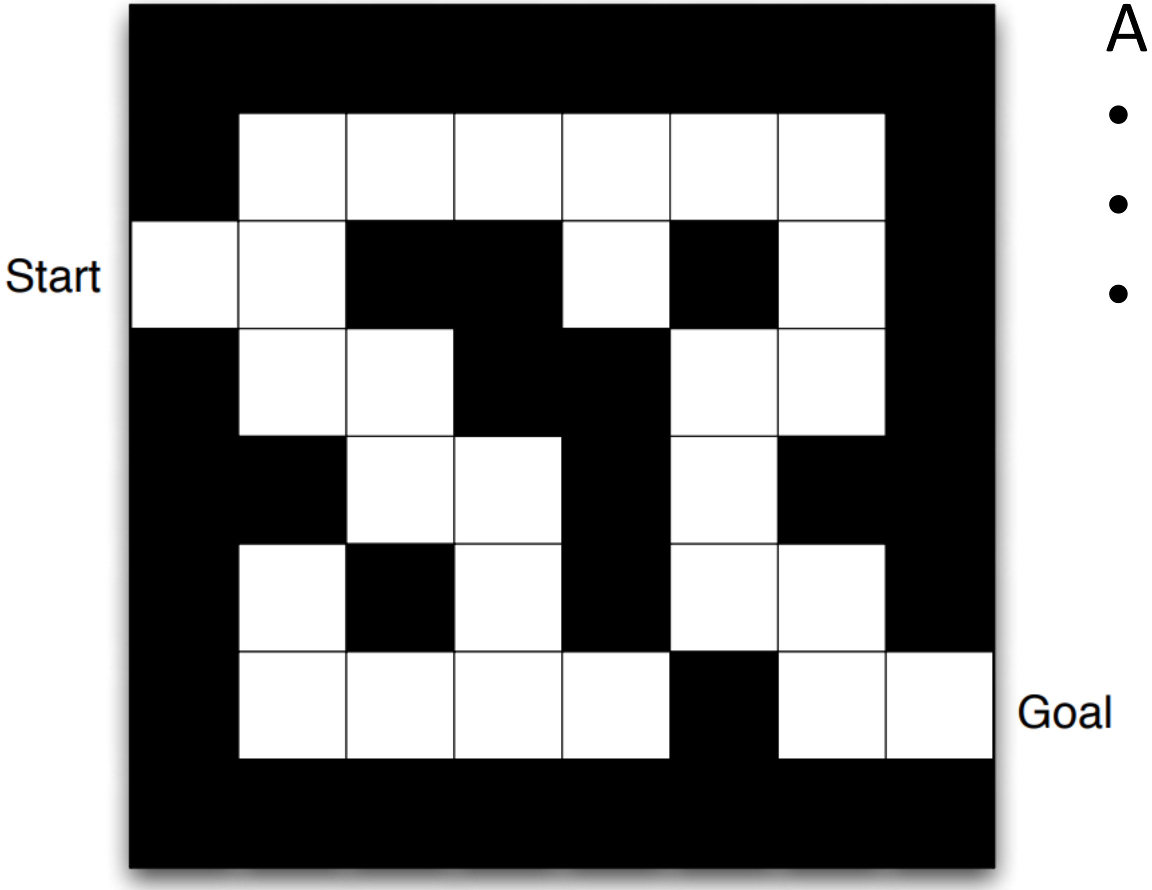
Major Components of an RL Agent

- **Policy** function (agent’s behaviour)
  - Map from state to (best) action
  - It can be deterministic, $a = \pi(s)$, or stochastic, $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

- **Value** (a.k.a. Utility) function (agent’s prediction of future reward)
  - Map to evaluate the (long term) goodness/badness of states
  - Expected future rewards can be used to select between actions
    $\mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$

- **Model** (agent’s representation of the environment)
  - It predicts what the environment will do next
  - Immediate state and reward are predicted for a current state and action
    $\mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$, $\mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

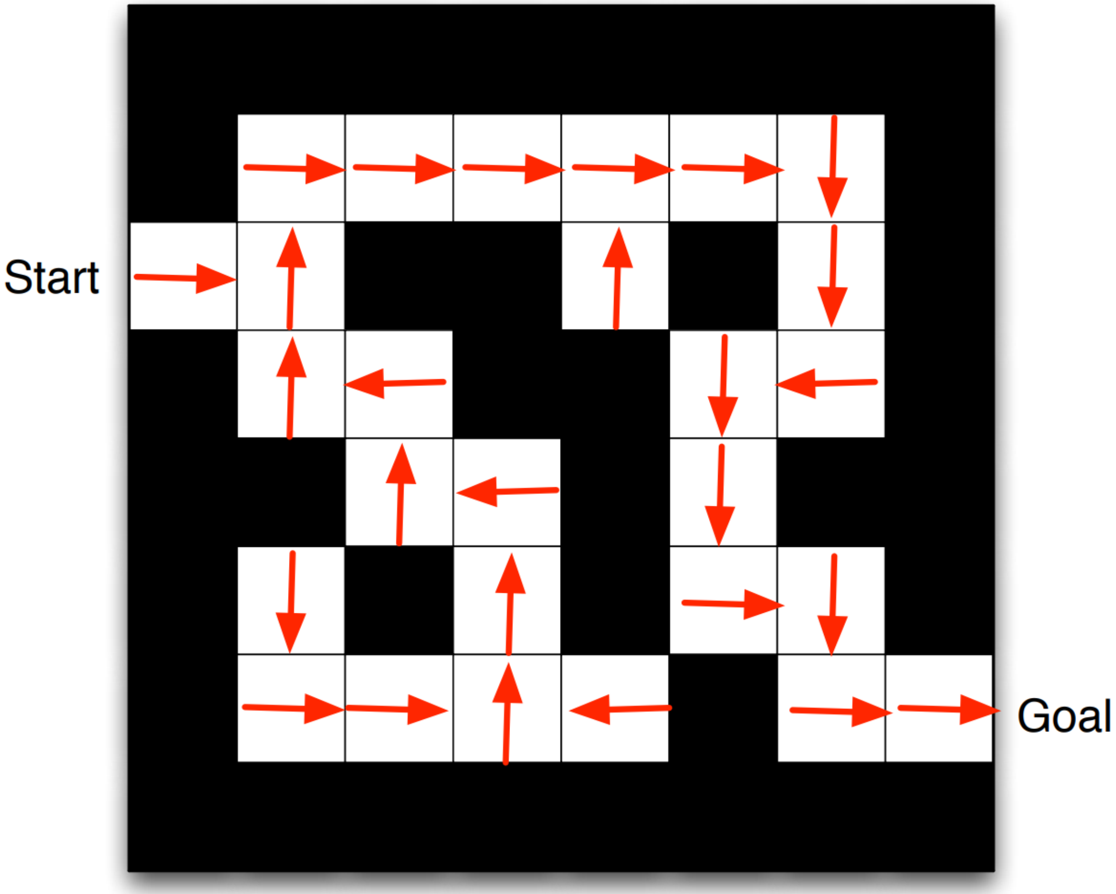
Maze Example



Aim: to reach the Goal as quickly as possible

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: agent’s location

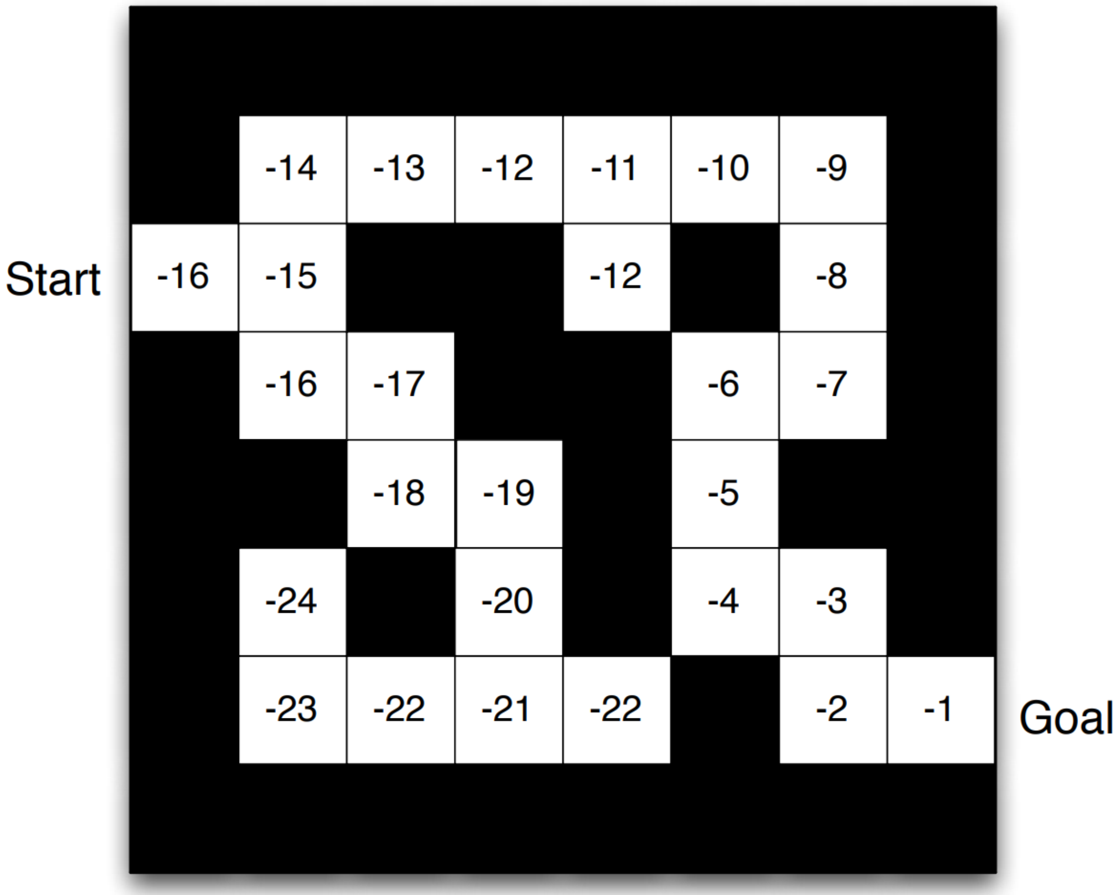
Maze Example: Policy (Deterministic)



Arrows represent policy $\pi(s)$ for each state s

Mapping from state to (best) actions

Maze Example: Value Function

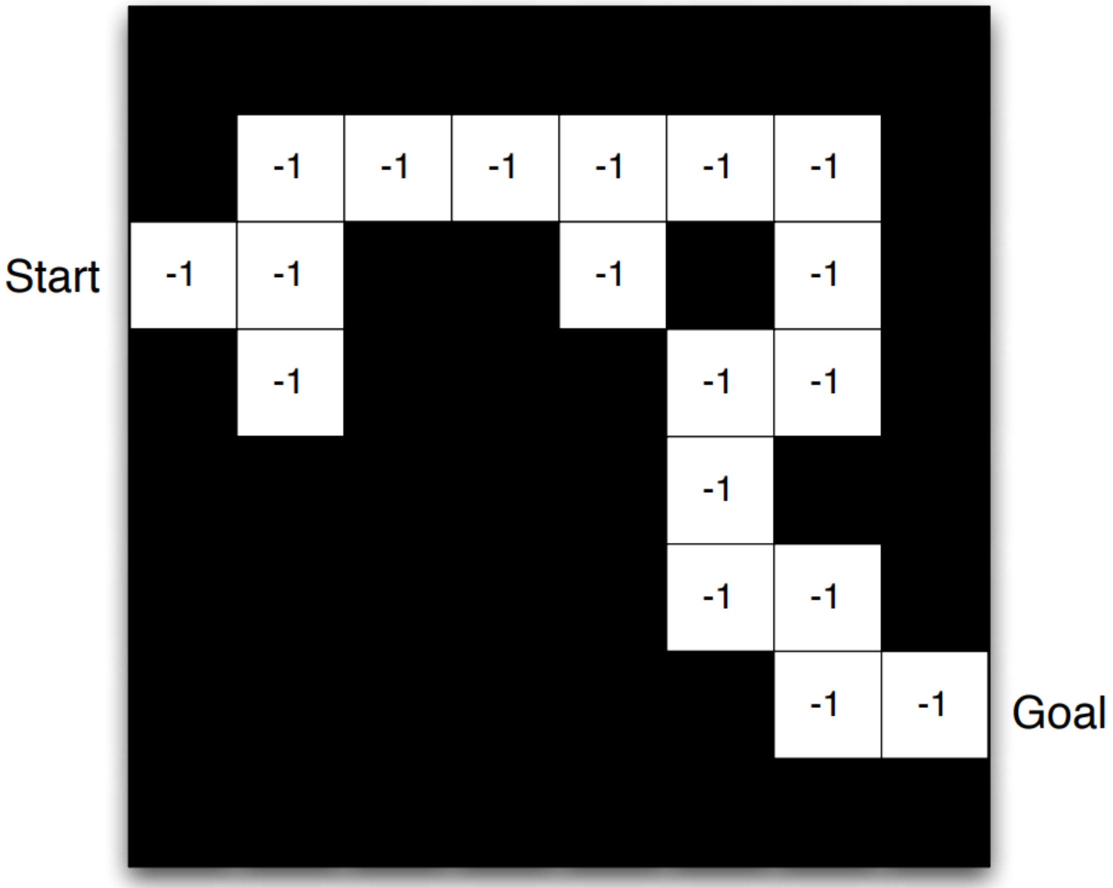


		-14	-13	-12	-11	-10	-9		
Start	-16	-15			-12		-8		
		-16	-17			-6	-7		
			-18	-19		-5			
		-24		-20		-4	-3		
		-23	-22	-21	-22		-2	-1	Goal

Numbers represent the value $v_\pi(s)$ of each state s (predicted future rewards)

Easy to derive a policy by looking at the value function at each state

Maze Example: Model



Agent may have an internal model of the environment

- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

In the example, the agent may have learnt its own map and that it gets -1 per step (immediate state and reward prediction)

RL Agent Types

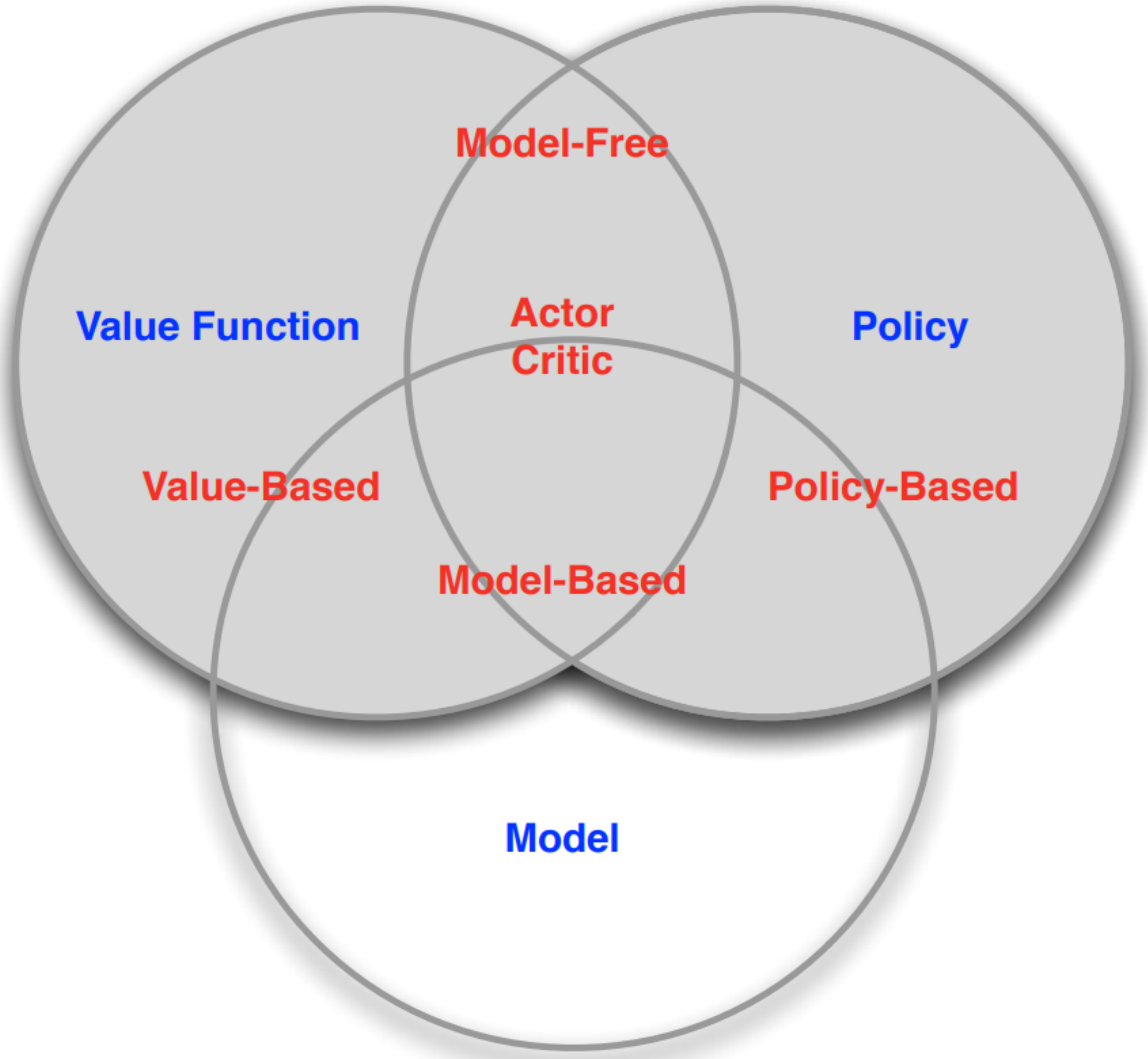
Not all three components (Policy, Value, Model) need to be present in RL agents

- **Policy-based**
  - Policy is trained directly with no Value function estimation
- **Value-based**
  - Policy is implicit, trained indirectly guided by Value maximisation
- **Actor Critic**
  - Both Policy and Value function are part of the RL agent

RL agents can yet be classified as:

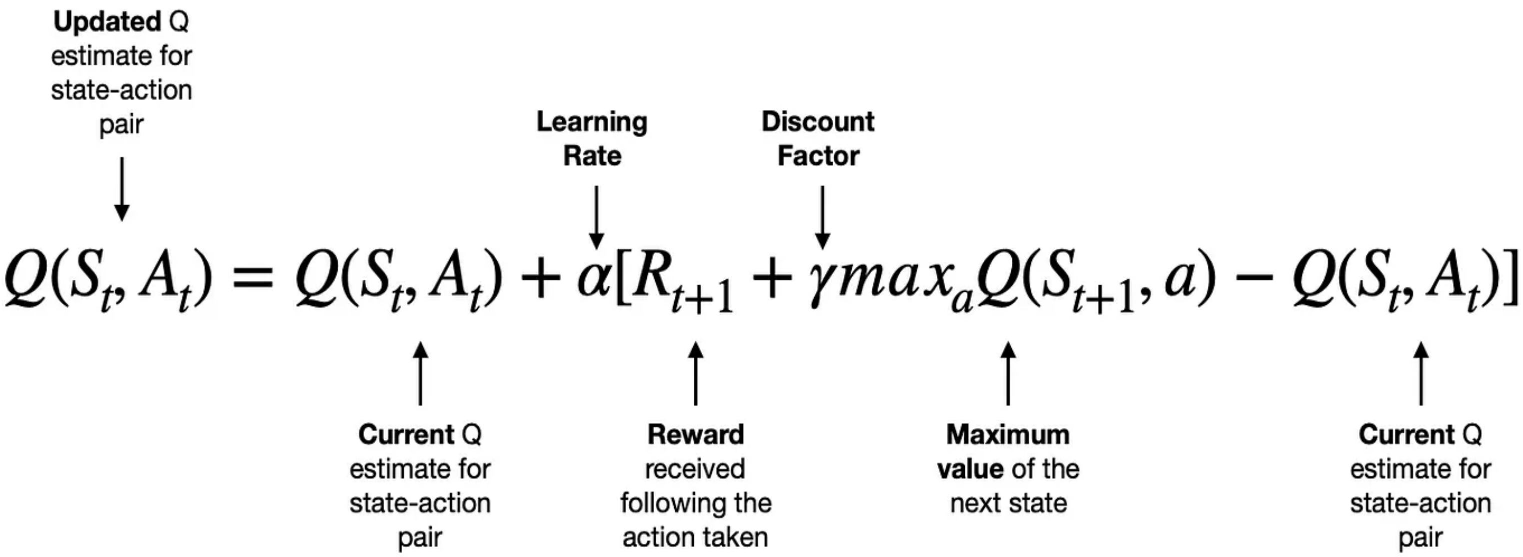
- Model-Free
  - No Model
- Model-based
  - Model

RL Agent Types



Q-Learning Overview

- **Q-learning** is a **model-free value-based method**
  - ▸ The Value function is called a **Q-function**
  - ▸ Inside, it will have a Q-table containing every state-action pair
  - ▸ Training a Q-function is simply finding the values associated with each state-action pair stored in a Q-table
  - ▸ Knowing these values enables the agent to choose the best action at each state
- Updating Q-table

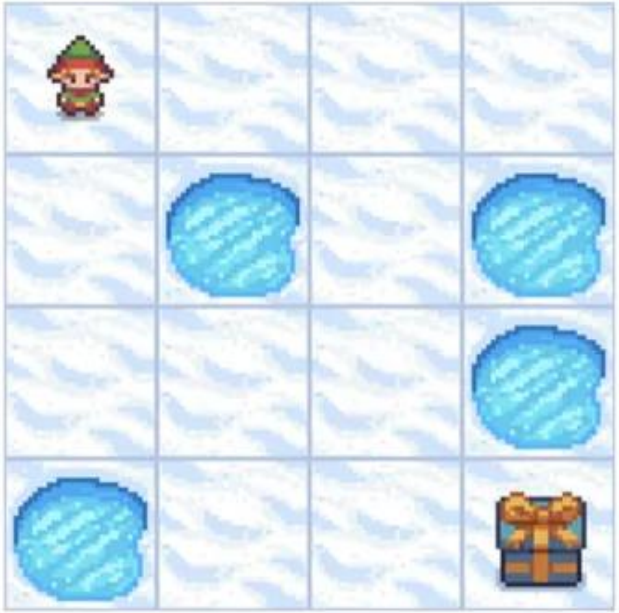


$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Updated Q estimate for state-action pair
Learning Rate
Discount Factor
Current Q estimate for state-action pair
Reward received following the action taken
Maximum value of the next state
Current Q estimate for state-action pair

Example: Frozen-Lake environment

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

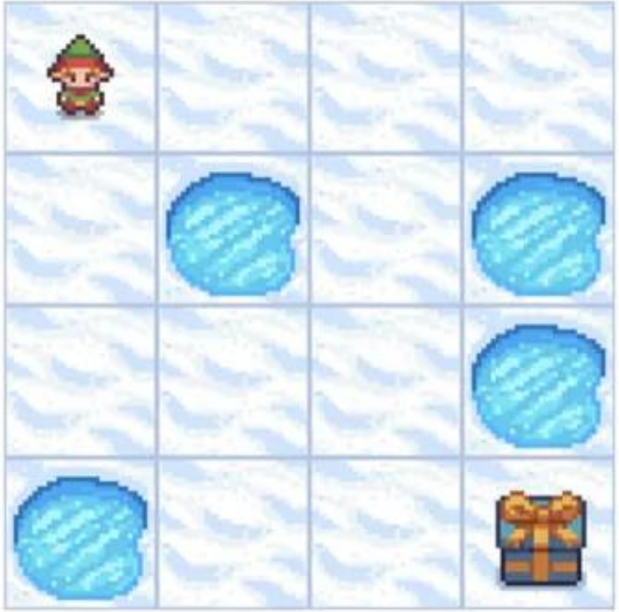
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(14,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

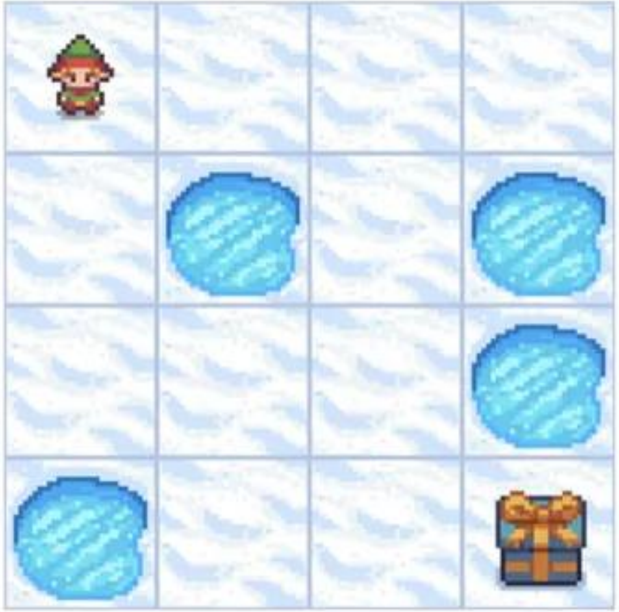
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(14,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

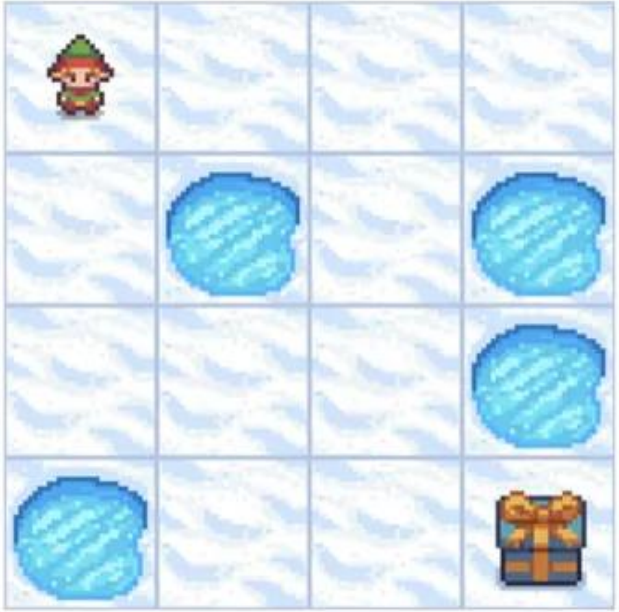
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(13,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_{a} Q(S_{t+1}, a)$$

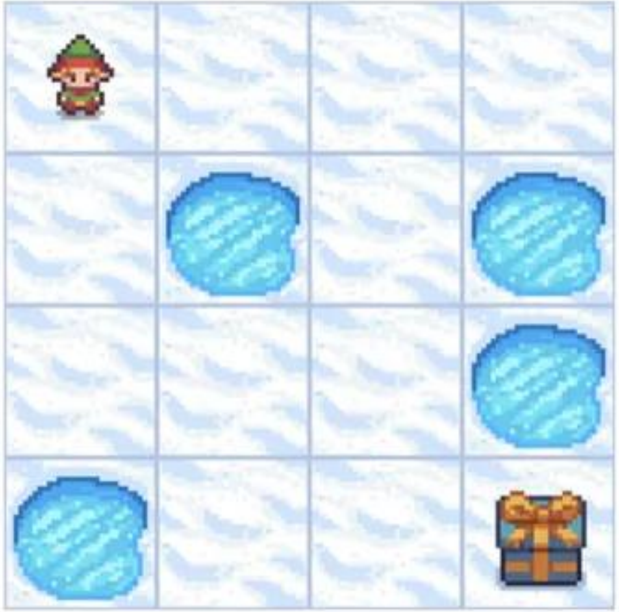
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(13,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

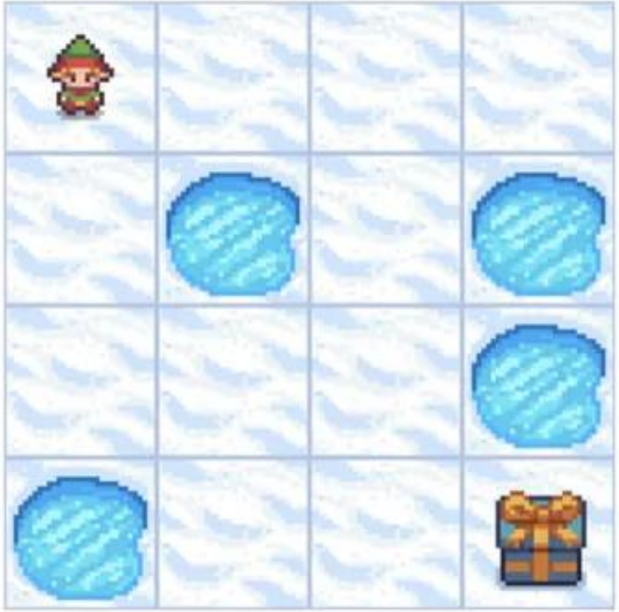
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(9,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

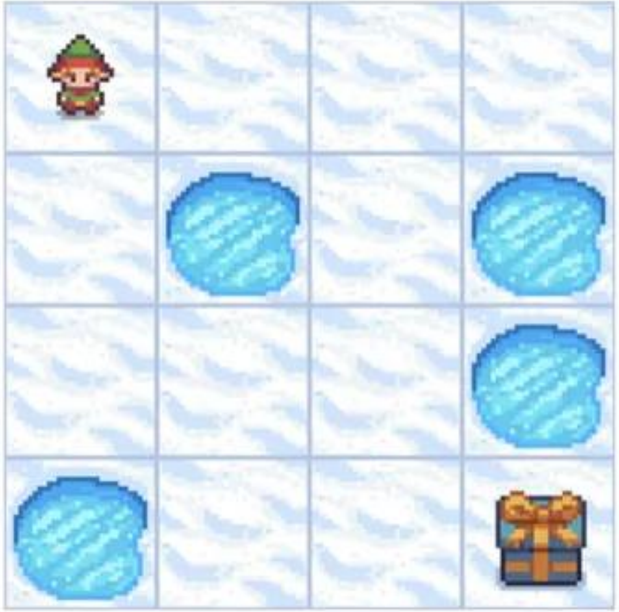
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(9,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Action space

State space

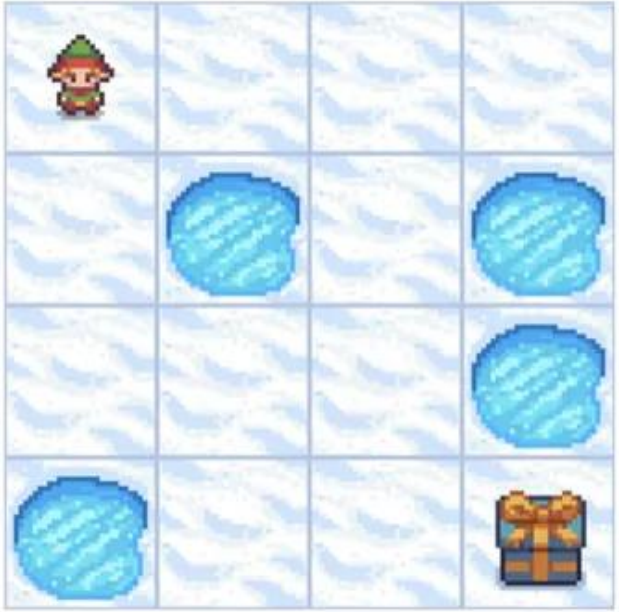
Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

If reward of reaching the goal state is set as +1, then what would be the Q(8,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

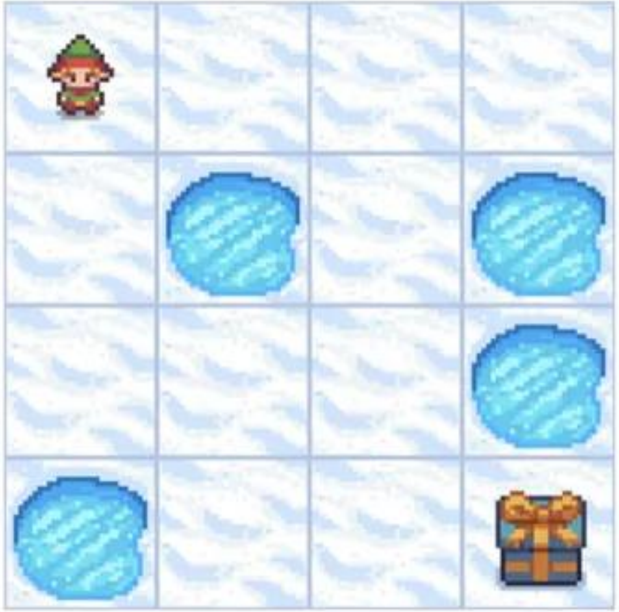
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(8,Right)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

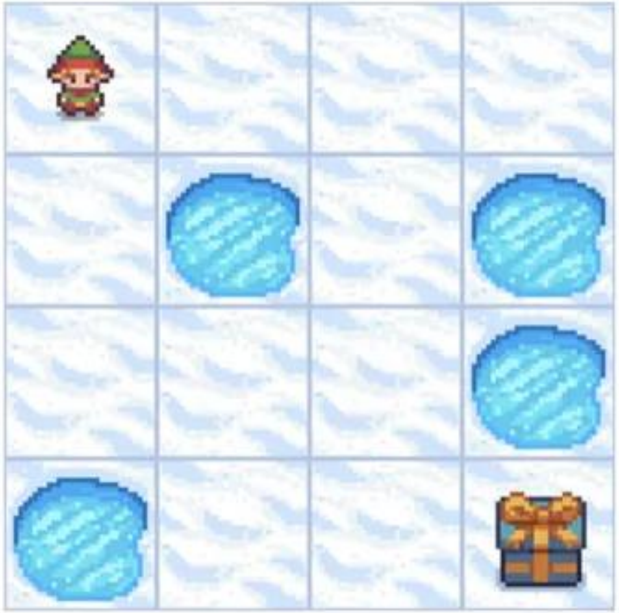
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	1	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(4,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

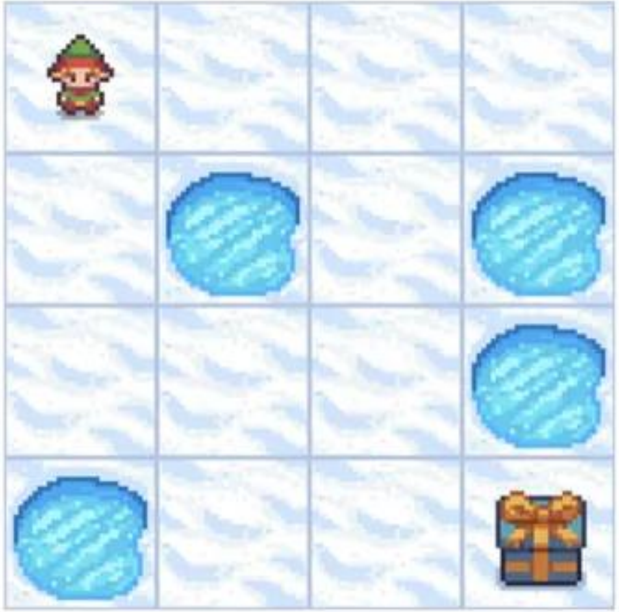
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	1	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(4,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

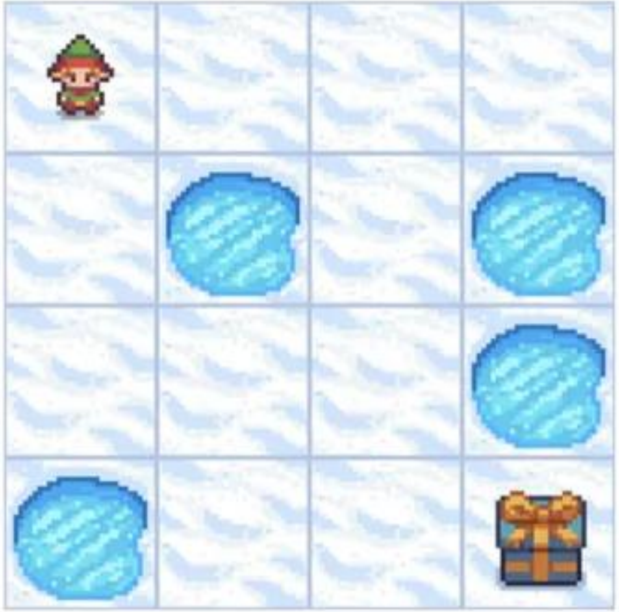
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	1
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	1	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(0,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

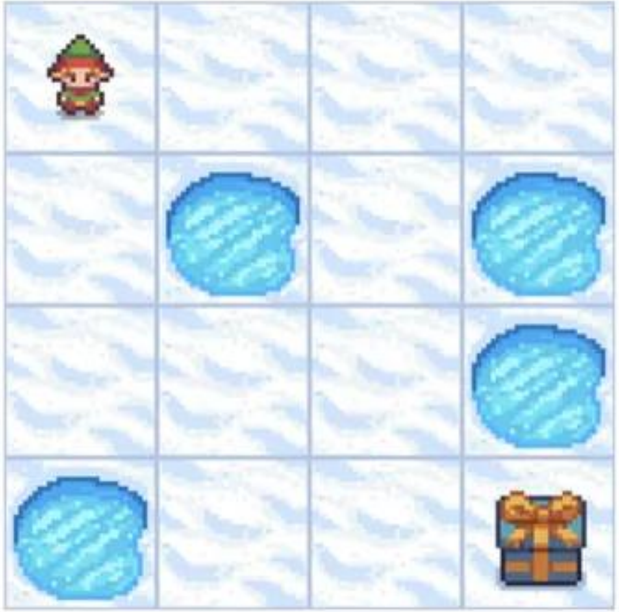
Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	1
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	1	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

If reward of reaching the goal state is set as +1, then what would be the Q(0,Down)?

State Space



Numbers assigned to each state

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

With $\alpha = 1$ and $\gamma = 1$, we get

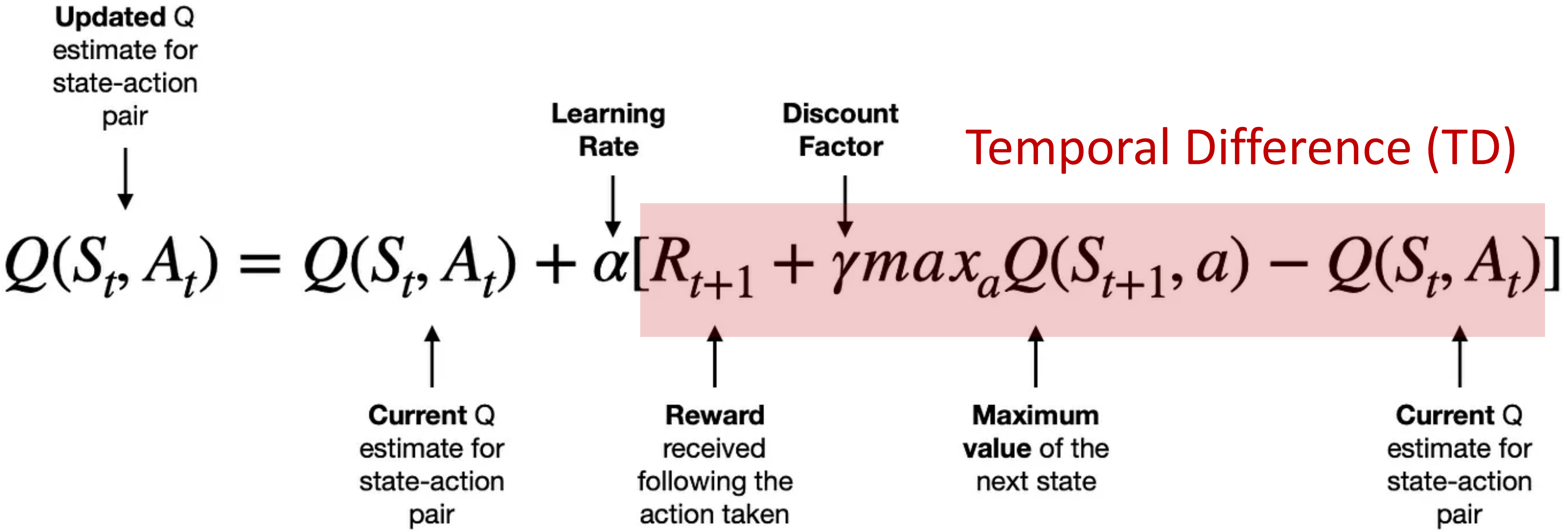
$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

Action space

State space

Q-table	Left	Right	Up	Down
0	0	0	0	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	1
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	1	0	0
9	0	0	0	1
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	0

Q-Learning Algorithm: Updating Q-Table



$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Updated Q estimate for state-action pair

Learning Rate

Discount Factor

Temporal Difference (TD)

Current Q estimate for state-action pair

Reward received following the action taken

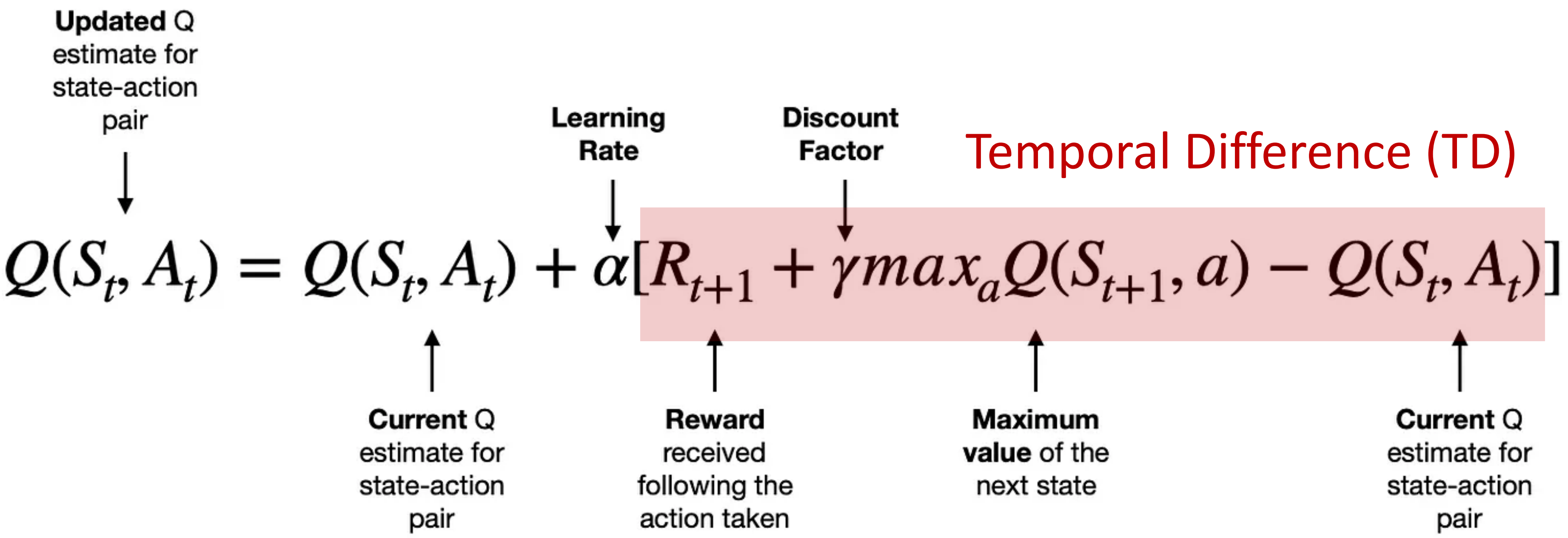
Maximum value of the next state

Current Q estimate for state-action pair

With $\alpha = 1$ and $\gamma = 1$, we get

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

Q-Learning Algorithm: Adopts TD Approach



Updated Q estimate for state-action pair

Learning Rate

Discount Factor

Temporal Difference (TD)

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Current Q estimate for state-action pair

Reward received following the action taken

Maximum value of the next state

Current Q estimate for state-action pair

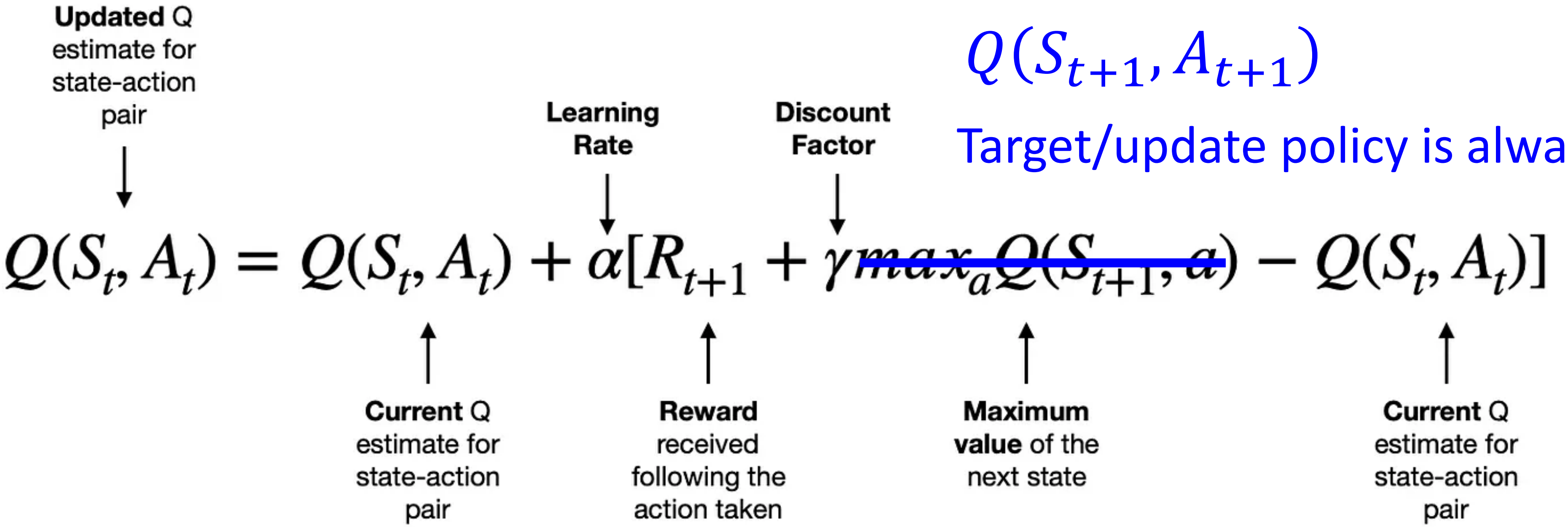
– Q-Learning uses a TD approach which means that during training, it updates the Q-function after each step

Q-Learning Algorithm: An Off-Policy Approach

State-Action-Reward-State-Action (SARSA)

$Q(S_{t+1}, A_{t+1})$

Target/update policy is always same



Updated Q estimate for state-action pair

Learning Rate

Discount Factor

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \cancel{\max_a Q(S_{t+1}, a}) - Q(S_t, A_t)]$$

Current Q estimate for state-action pair

Reward received following the action taken

Maximum value of the next state

Current Q estimate for state-action pair

- Q-Learning is an off-policy algorithm, which means that during training, we use different policies for the agent to act (acting/behavioural policy) and to update the Q-function (updating policy)
- Alternative to this is on-policy which means that the same policy is used for acting and updating (SARSA model, SARSA standing for State-Action-Reward-State-Action)

Summary (Lecture Closure)

Rational Agents

- A rational agent selects actions expected to maximise its performance measure based on its percept sequence and built-in knowledge
- PEAS task environment (Performance measure, Environment, Actuators, Sensors) for agent design
- Main 4 agent types (by how they select actions): Simple reflex (direct response), Model-based reflex (tracking world states), Goal-based (planning a simple goal achievement), Utility-based (maximising “happiness”)
- Learning agents feature a critic for feedback, a performance element, a learning element for improvements, and a problem generator for exploration
- In multi-agent systems, each agent type can specialise in different task components to handle complex tasks

Problem Solving by Searching

- Problem solving involves finding a sequence of actions ahead of time to achieve a goal, single actions being insufficient
- Problem formulation defines the initial state, available actions, a transition model, a goal test, and a path cost function
- Uninformed search strategies like Depth-first, Breadth-first, and Uniform-cost search use only problem definitions
- Informed searches utilize a heuristic function to estimate costs to the goal
- A* search combines backward path costs used by uniform search with forward heuristics for optimal efficient solutions
- A heuristic is admissible if it never overestimates the true cost to reach a goal (e.g. straight line distance in maps)

Summary (Lecture Closure)

Game Theory

- Adversarial search applies to competitive environments where agents have conflicting goals, returning a strategy/policy
- In deterministic, zero-sum games with perfect information (e.g., Chess, Go), one player's gain equals the other’s loss
- MiniMax search determines the best move by assuming both players play optimally from a given state
- Alpha-beta pruning increases efficiency by eliminating branches that cannot influence the final decision
- Real-time decisions require a cutoff test and a heuristic evaluation function to estimate utility for non-terminal states

Reinforcement Learning (RL)

- Reinforcement Learning involves agents learning behaviours through trial-and-error to maximise cumulative rewards
- The Reward Hypothesis states all goals can be described as the maximisation of expected cumulative rewards.
- Unlike planning, using a known model, RL agents interact with initially unknown environments to improve their policies
- RL components: policy (behaviour mapping), value function (future reward prediction), and model (of the environment)
- Q-learning is a model-free, value-based method using a Q-table (state-action) and a Temporal Difference (TD) off-policy approach to update estimates after each step.

Further Reading

S. Russell & P. Norvig (2022) book, “**Artificial Intelligence: A Modern Approach**”
Available online at UoR Library https://ebookcentral.proquest.com/lib/reading/detail.action?docID=6563568

- Chapter 2: “Intelligent Agents”
- Chapter 3: "Solving Problems by Searching"
- Chapter 6: "Adversarial Search and Games"
- Chapter 23: "Reinforcement Learning"

UC Berkeley’s **Introduction to Artificial Intelligence** https://inst.eecs.berkeley.edu/~cs188/textbook/

- Chapter 1: “Search”
- Chapter 3: “Games”
- Chapter 5 “RL”

D. Silver (2015) **Lectures on Reinforcement Learning** https://davidstarsilver.wordpress.com/teaching/

S. Raschka & V. Mirjalili, **Python Machine Learning**, Chapter 18
https://ebookcentral.proquest.com/lib/reading/detail.action?docID=6005547

Challenging the Reward Hypothesis https://neurips.cc/virtual/2022/65594