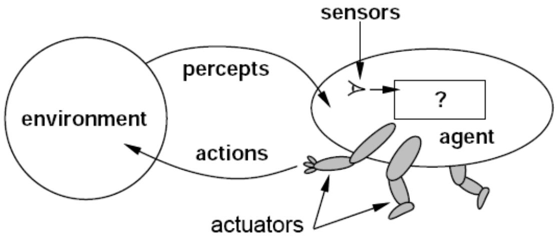
COMP250: Artificial Intelligence

2: Designing AI behaviours

Agents

Agents



An **agent** is anything which perceives an **environment** through **sensors**, and acts upon that environment through **actuators**.

Performance

- An “intelligent” agent moves towards some kind of **goal**
- The goal is an **environment state** (or a set of states)
- A **performance measure** evaluates a given state for how well it fits the goal

PEAS

For each example of an agent, what are the Performance measure, Environment, Actuators and Sensors?

- A Roomba
- A self-driving car
- A chatbot
- A factory robot
- An enemy in an FPS game
- A chess AI
- A human

Types of environment

- Environments come with many different properties
- These properties influence the choice of AI architecture we use to build agents

Observability

- **Fully observable**: the agent's sensors give it full information about the state of the environment
- **Partially observable**: some aspects of the environment state are not visible to the agent's sensors
- E.g. a chess game is fully observable, a poker game is partially observable

Number of agents

- **Single agent**: our agent is the only one in the environment
- **Multi-agent**: there is more than one agent
- **Cooperative**: all agents share the same performance measure
- **Competitive**: agents' performance measures are in opposition to each other (i.e. if one agent "wins", another "loses")

Determinism

- **Deterministic**: the next state of the environment is completely determined by the current state and by the agent's action
- **Stochastic**: there is some aspect of randomness in determining the next state
- E.g. chess is deterministic; any board game involving dice rolls or random card draws is stochastic

Dynamicity

- **Static**: the environment does not change while the agent is deliberating
- **Dynamic**: the environment changes constantly
- E.g. most board games are static, most (non turn-based) video games are dynamic

Discreteness

- **Discrete**: time, percepts and actions are all discrete (from a finite set of possibilities or “integer valued”)
- **Continuous**: at least one of these is not discrete (“float valued”)
- Continuous problems are hard so we sometimes **discretise** them

Known or unknown

- Are all the details of the environment **known** to the AI designer?
- For a game or simulation: probably **yes** (unless someone else made it and we don't have the source code)
- For the real world: technically **no** (but we have physics, sociology, economics etc to give us good approximations)

Agents and AI

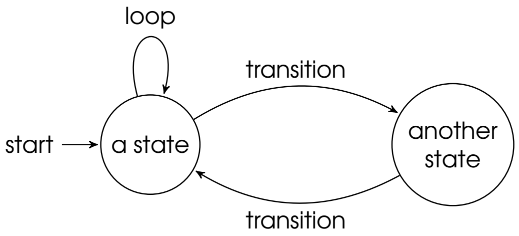
- The ideas of agents and environments are a useful frame for designing AI
- All(?) AI problems can be expressed in terms of creating an agent that optimises some performance measure in some environment
- Agent design boils down to: given a **percept** (and possibly some **memory** of past percepts/actions), choose the best **action** to take now

Finite state machines

Finite state machines

- A **finite state machine (FSM)** consists of:
  - A set of **states**; and
  - **Transitions** between states
- At any given time, the FSM is in a **single state**
- **Inputs** or **events** (**percepts**) can cause the FSM to transition to a different state
- Which state the FSM is in dictates what **actions** the agent takes

State transition diagrams

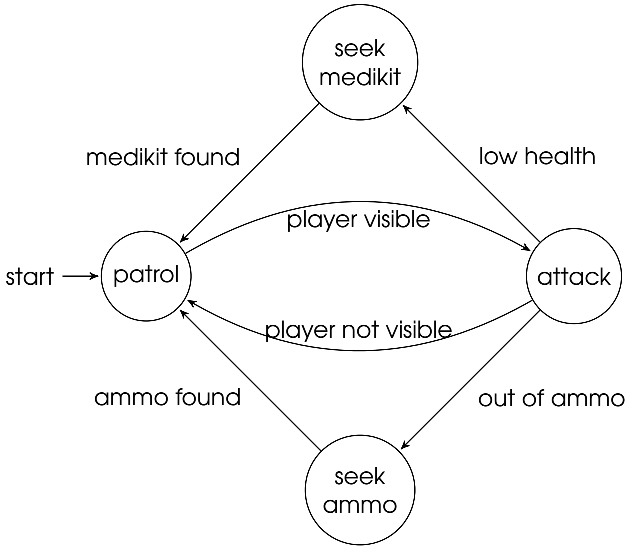


- FSMs are often drawn as **state transition diagrams**
- Reminiscent of **flowcharts** and certain types of **UML diagram**

FSMs for AI behaviour

The next slide shows a simple FSM for the following AI behaviour, for an enemy NPC in a shooter game:

- By default, patrol (e.g. along a preset route)
- If the player is spotted, attack them
- If the player is no longer visible, resume patrolling
- If you are low on health, run away and find a medikit. Then resume patrolling
- If you are low on ammo, run away and find ammo. Then resume patrolling


seek medikit
medikit found
low health
player visible
start
patrol
attack
player not visible
ammo found
out of ammo
seek ammo

Other uses of FSMs

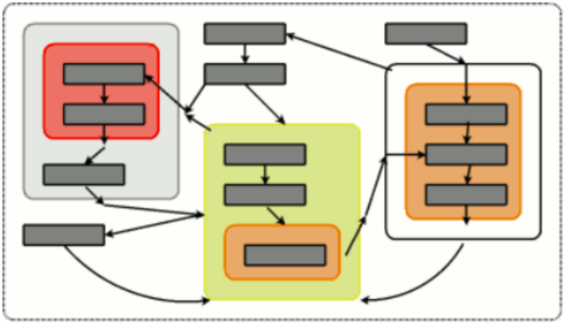
As well as AI behaviours, FSMs may also be used for:

- Animation
- UI menu systems
- Dialogue trees
- Token parsing
- ...

Implementing FSMs

- Implementation needs to keep track of current state, and execute some code dependent on the state (this code itself possibly changing the current state)
- Most common approach: a big **`switch–case`** statement, with an **`enum`** type for the state
- Object-oriented approach: a `State` class, which your FSM states inherit from
- Functional approach: represent state by a function delegate
- Coroutine approach: encode your FSM logic as a procedure which runs as a coroutine (requires either refactoring logic into structured loops, or using **`goto`**...)

Hierarchical FSMs



- An FSM with N states has potentially N^2 transitions
- Designing complex behaviour with FSMs quickly gets unwieldy
- Hierarchical FSMs allow to group states into **super-states** to simplify defining transitions

Should you use FSMs?

- FSMs are useful for designing simple AI behaviours
- Historically an important technique for game AI
- However other techniques such as behaviour trees are more flexible and better suited to designing complex behaviours