They are trivial because the decision making they do is simple (uninteresting) agents:

- Thermostat – delegated goal is maintain room temperature
  - Actions are heat on/off
- UNIX biff program
  - Delegated goal is monitor for incoming email and flag it
  - Actions are GUI actions.

• They are trivial because the decision making they do is simple (uninteresting) agents:
Intelligent Agents

We typically think of an intelligent agent as exhibiting 3 types of behaviour:

- **Reactivity**
  - If a program's environment is fixed, a program can just execute blindly.
  - The real world is not like that: most environments are dynamic.
  - It is hard to build for dynamic domains:
    - Environment is hard to build for dynamic domains.
    - Programs must take into account possibilities of failure.
    - Some systems are one that maintain an ongoing interaction with its environment, and respond to changes that occur in it. (In time for the response to be useful.)
  - A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it. (In time for the response to be useful.)

- **Pro-activeness**
  - Recognising opportunities.
  - Initiating
  - Goal-directed: not driven solely by events: taking the initiative.
  - Goal-directed = generating and attempting to achieve goals, not driven solely by events: taking the initiative.
  - Hence goal-directed behaviour.
  - But we generally want agents to do things for us.
  - Response rules.
  - Reaching to an environment is easy (e.g., stimulus — response).

- **Social Ability**
  - The real world is a multi-agent environment: we cannot go around attempting to achieve goals without taking others into account.
  - Some goals can only be achieved by interacting with others.
  - Similarly for many computer environments: witness the Internet.
  - The real world is a multi-agent environment: we cannot go around attempting to achieve goals without taking others into account.
  - Some goals can only be achieved by interacting with others.
  - Similarly for many computer environments: witness the Internet.

We typically think of an intelligent agent as exhibiting 3 types of behaviour:

- **Reactivity**
- **Pro-activeness**
- **Social Ability**
Social ability is the ability to interact with other agents (and possibly humans) via cooperation, coordination, and negotiation.

- **Cooperation** is working together as a team to achieve a shared goal.
  - Obtain a better result (e.g., get result faster).
  - Often prompted either by the fact that no one agent can achieve the goal alone or that cooperation will allow both agents to achieve the goal.

- **Coordination** is managing the interdependencies between activities.
  - For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.

- **Negotiation** is the ability to reach agreements on matters of common interest.
  - A possible deal: watch football tonight and a movie tomorrow.
  - For example: you have one TV in your house; your housemate wants to watch a movie, you want to watch football.

Social ability is the ability to interact with other agents (and possibly humans) via cooperation, coordination, and negotiation...
Chapter 2: An Introduction to Multiagent Systems

Some Other Properties...

- Mobility
- Veracity
- Benevolence
- Rationality
- Learning/adaption

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

Agents and Objects

- Are agents just objects by another name?
- Objects do it for free.
- Agents do it for money.
- Objects do it because they want to.

Differences between Agents & Objects

- Agents are autonomous:
  - Agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent.
- Agents are smart:
  - Capable of flexible (reactive, pro-active, social) behavior – the OO model has nothing to say about such types of behavior.
- Agents are active:
  - Not passive service providers.

Objects do it for free...

- Agents do it because they want to;
- Agents do it for money.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Agents and Expert Systems

• Aren’t agents just expert systems by another name?
  - Expert systems typically disembodied ‘expertise’ about some (abstract) domain of discourse.
  - Example: MYCIN knows about blood diseases in humans.
    - MYCIN has a wealth of knowledge about blood diseases, in the form of rules.
    - A doctor can obtain expert advice about blood diseases, in humans.
    - Example: MYCIN knows about blood diseases in humans.
    - Expert systems significantly disembodied ‘expertise’

Differences between Agents & Expert Systems

• Agents are situated in an environment.
  - A useful agent
  - We do not have to solve all the problems of AI to build an agent.
    - Some real-time (especially process control) expert systems are agents.
    - Mycin does not operate on patients.
  - We simply want a system that can choose the right action to perform, typically in a limited domain.

Intelligent Agents and AI

• Aren’t agents just the AI project?
  - Isn’t building an agent what AI is all about?
    - AI aims to build systems that can (ultimately) understand concepts, use commonsense, think
      and understand natural language, recognize and understand emotions, etc.
    - So, don’t we need to solve all of AI to build an agent?
    - Aren’t agents just the AI project?

Intelligent Agents and AI

• When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain.
  - We do not have to solve all the problems of AI.
    - Some real-time (especially process control) expert systems are agents.
    - We simply want a system that can choose the right action to perform, typically in a limited domain.
  - We do not have to solve all the problems of AI.

Intelligent Agents and AI

• When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain.
  - We do not have to solve all the problems of AI.
    - Some real-time (especially process control) expert systems are agents.
    - We simply want a system that can choose the right action to perform, typically in a limited domain.
  - We do not have to solve all the problems of AI.

Intelligent Agents and AI
Properties of Environments

- Accessible vs inaccessible. An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. Most moderately complex environments (including the Internet) are inaccessible.

- Deterministic vs non-deterministic. Non-deterministic environments present greater problems for the agent designer. In a non-deterministic environment, the agent cannot predict the outcome of its actions. Deterministic environments, on the other hand, have a single, guaranteed outcome for each action.

- Episodic vs non-episodic. In an episodic environment, the performance of the agent is dependent on a number of discrete episodes, with no link between the performance of an agent in one episode and the performance of an agent in another. In a non-episodic environment, the performance of the agent is dependent on the environment's state, which may change unpredictably.

- Static vs dynamic. A dynamic environment is one that can be assumed to remain unchanged except by the performance of actions by the agent. A static environment is one that cannot be assumed to remain unchanged.

- Deterministic vs non-deterministic. A physical world can be regarded as non-deterministic, because there is no certainty about the effects of an action. An agent in a deterministic world can predict the outcome of an action.

- Episodic vs non-episodic. Episodic environments are simpler from the agent designer's perspective because the agent can decide on the basis of the current episode, without having to reason about the interactions between episodes.

- Static vs dynamic. The physical world is a highly dynamic environment. In a dynamic environment, the agent's actions can result in changes to the environment. A static environment is one that remains unchanged except by the performance of actions by the agent.
Chapter 2
An Introduction to Multiagent Systems

2.1 Discrete vs Continuous

An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment. An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment. An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment.
What can be described with the intentional stance?

Consider a light switch:

'It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current when it believes that we want it transmitted and not otherwise. Clicking the switch transmits current when it believes that we want it transmitted, and not otherwise. What can be described with the intentional stance?'

Yoav Shoham

The intentional stance is such an abstraction.

As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations of the behavior of very complex systems may not be practical.

The intentional notions are thus abstraction tools.

The more we know about a system, the less we need to rely on simplistic, intentional explanations of its behavior. Essentially, to have a simpler, mechatanistic description of its behavior (Yoav Shoham) doesn't do us any good, since we use it. The more we know about a system, the less we need to rely on the intentional stance description of its behavior.

Most adults would find such a description absurd!

• Alice's and agents as intentional systems, represent
• M. P. D. conceptual abstraction.
• M. P. D. abstract data types.
• M. P. D. procedural abstraction.

Computing and systems are based on new abstractions.

Remember: most important developments in complex systems, describing, explaining, and predicting the behavior of systems.

The intentional notions are thus abstraction tools.

Remember: most important developments in computing are based on new abstractions. As computers systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations of the behavior of very complex systems may not be practical.

The intentional stance is such an abstraction.

As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations of the behavior of very complex systems may not be practical.

The intentional notions are thus abstraction tools.

• The intentional notions are thus abstraction tools.

Remember: most important developments in computing are based on new abstractions.

As computers systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations of the behavior of very complex systems may not be practical.

The intentional notions are thus abstraction tools.

Remember: most important developments in computer systems, describing, explaining, and predicting the behavior of systems. Essentials, to have a simpler, mechatanistic description of its behavior (Yoav Shoham) doesn't do us any good, since we use it. The more we know about a system, the less we need to rely on the intentional stance description of its behavior. Most adults would find such a description absurd!

What can be described with the intentional stance?
Points in favour of this idea:

### Characterising Agents

- It provides us with a familiar, non-technical way of understanding & explaining agents.

### Nested Representations

- It gives us the potential to specify systems that include representations of other systems.

### Post-declarative Systems

- It is widely accepted that such nested representations are essential for agents that must cooperate with other agents. It is widely accepted that such nested representations

### An aside...

- We find that researchers from a more mainstream computing discipline have adopted a similar set of ideas in knowledge-based protocols. The idea: when constructing protocols, one often encounters reasoning such as the following:

> IF process \( i \) knows process \( j \) has received message \( m \)

> THEN process \( i \) should send process \( j \) message \( m \)

> IF process \( j \) knows process \( i \) has received message \( m \)

> THEN process \( j \) should send process \( i \) message \( m \).

- We find that researchers from a more mainstream computing discipline have adopted a similar set of ideas in knowledge-based protocols. The idea: when constructing protocols, one often encounters reasoning such as the following:

> IF process \( i \) knows process \( j \) has received message \( m \)

> THEN process \( i \) should send process \( j \) message \( m \)

> IF process \( j \) knows process \( i \) has received message \( m \)

> THEN process \( j \) should send process \( i \) message \( m \).

### Abstract Architectures for Agents

- A run \( r \) of an agent in an environment is a sequence:

\[
\{e_0, \alpha_0 \rightarrow e_1, \alpha_1 \rightarrow e_2, \alpha_2 \rightarrow \cdots \}
\]

- The environment and the repertoire of all actions available to it, which transform the state of the environment, may be in any of a finite set of discrete, instantaneous states.

- Agents are assumed to have a repertoire of possible actions, with a high-level description of the goal, and the control mechanism figure out what to do.

- Theorem proving figure out what to do: build-in control mechanism (e.g., goal-directed built-in control mechanism between objectives).

- In declarative programming, we specify something that system should do.

- In procedural programming, we say exactly what a system should do.
Chapter 2
An Introduction to Multiagent Systems

Runs

Let $R$ be the set of all runs. A run is a function which maps runs to actions:

- $R : \mathbb{N} \rightarrow \mathcal{A}$

Each run $r$ is a sequence of actions $(a_1, a_2, \ldots)$.

Environments

- An environment $Env$ is a triple $Env = \langle E, e_0, \tau \rangle$ where:
  - $E$ is the set of environment states,
  - $e_0 \in E$ is the initial state,
  - $\tau$ is the state transformer function.

- $\tau : R \rightarrow \wp(E)$ is the state transformer function, where $\wp(E)$ is the power set of $E$.

- If $\tau(r) = \emptyset$, there are no possible successor states to $r$.

Agents

- An agent $A$ is a function which maps runs to actions:

$A : R \rightarrow \mathcal{A}$

A run $r$ is a sequence of actions $(a_1, a_2, \ldots)$.

- History-dependent.
- Non-deterministic.

So we say the run has ended ("game over") if $\tau(r) = \emptyset$, there are no possible successor states to $r$.

Note that environments are:

- History-dependent.
- Non-deterministic.

Let $E$ be the set of all possible environment states, $e \in E$, $e_0 \in E$ is initial.
A system is a pair containing an agent and an environment.

Any system will have associated with it a set of possible runs, which we denote the set of runs of agent $Ag$ in environment $Env$.

Formally, a sequence $(\eta_0, \eta_1, \eta_2, \ldots, \eta_n)$ represents a run of an agent $Ag$ in environment $Env$ if:

1. $\eta_0$ is the initial state of $Env$.
2. $\eta_1 \in \tau(\eta_0, \eta_2, \ldots, \eta_{n-1})$.
3. For $n \geq 0$,
   \[ \eta_n = \begin{cases} \text{off} & \text{if } \eta_n = \text{temperature OK} \\ \text{on otherwise} \end{cases} \]

A thermostat is a purely reactive agent.

We call such agents purely reactive.

Now introduce perception system:

A purely reactive agent.

Perception

Assume $R(\eta_0, \eta_1, \ldots, \eta_n)$ contains only runs that have environment $Env$ by $R(\eta_0, \eta_1, \ldots, \eta_n)$.

Any system will have associated with it a set of systems.

A system is a pair containing an agent and an environment.
Agents with State

The action-selection function \( \text{action} \) is now defined as a mapping from \( I \) to \( Ac \), where \( I \) is the set of all internal states of the agent.

\[ \text{action} : I \rightarrow Ac \]

These agents have some internal data structure, \( I \), which is typically used to record information about the environment state and history.

\[ I \]

We now consider agents that maintain state:

 Agents with State

Perception

The perception function \( \text{see} \) for a state-based agent is unchanged:

\[ \text{see} : E \rightarrow Per \]

Let \( E \) be the set of all environment states of the agent.

\[ E \]

\( Per \) is typically used to record information about the environment.

\[ Per \]

\( \Rightarrow \) The agent's decision-making process.

\[ \Rightarrow \]

\( \Rightarrow \) The agent's ability to observe its environment.
A function $\text{next}$ is introduced, which maps an internal state and percept to an internal state:

$$\text{next} : I \times \text{Per} \rightarrow I$$

A function $\text{next}$ is introduced, which maps an internal state to an internal state.

Tasks for Agents

- We build agents in order to carry out tasks for us.
- But we want to tell agents what to do without telling them how to do it.
- The task must be specified by us.
- One possibility: associate utilities with individual states.
Chapter 2: An Introduction to Multiagent Systems

Utilities over Runs

- Minimum utility of state on run?
- Maximum utility of state on run?
- Sum of utilities of states on run?
- Minimum utility of state on run?
- Average?
- But what is the value of a run?

Problems with Utility-based Approaches

- Where do the numbers come from? (Peter Cheeseman)
- People don't think in terms of utilities — it's hard for people to specify tasks in these terms.
- Nevertheless, works well in certain scenarios...
- Although assigning utilities to individual states is difficult to specify a long term view.
- Where do the numbers come from? (Peter Cheeseman)
- People don't think in terms of utilities — it's hard for people to specify tasks in these terms.
- Nevertheless, works well in certain scenarios...

TILEWORLD

Simulated two-dimensional grid environment on which there are agents, tiles, obstacles, and holes.

An agent can move in four directions: up, down, left, right. An agent scores points by filling holes with tiles, and tiles have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, and tiles have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, and tiles have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, and tiles have to be filled up with tiles by the agent.

Utility in the TILEWORLD

utility( ): \( R \rightarrow R \)

Other variations: incorporate probabilities of different states emerging. Incorporate probabilistic states of different states' emergences. Another possibility: assigns a utility not to individual states, but to runs themselves.

utility over runs

Chapter 2: An Introduction to Multiagent Systems
Utility function

Utility function defined as follows:

\[ u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r} \]

Thus:
- If agent fills all holes, utility = 1.
- If agent fills no holes, utility = 0.

Expected Utility

\[ P(r | Ag, Env) \]

\[ \sum_{r \in R(Ag, Env)} P(r | Ag, Env) = 1. \]

The expected utility of agent \( Ag \) in environment \( Env \) (given \( P, u \)) is defined as follows:

\[ \text{EU}(Ag, Env) = \sum_{r \in R(Ag, Env)} u(r) P(r | Ag, Env). \]

\[ \text{EU}(Ag, Env) = \begin{cases} 0 & \text{if no holes} \\ 1 & \text{if all holes} \end{cases} \]

\[ \text{EU}(Ag, Env) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r} \]

Utility in the Tileworld

Consider the environment \( Env_1 = \langle E, e_0, \tau \rangle \) defined as follows:

\[ E = \{ e_0, e_1, e_2, e_3, e_4, e_5 \} \]

\[ \tau(e_0, \alpha_0) = \{ e_1, e_2 \} \]

\[ \tau(e_0, \alpha_1) = \{ e_3, e_4, e_5 \} \]

There are two agents possible with respect to this environment:

\[ Ag_1(e_0) = \alpha_0 \]

\[ Ag_2(e_0) = \alpha_1 \]

The probabilities of the various runs are as follows:

\[ P(e_0, \alpha_0 \rightarrow e_1 | Ag_1, Env_1) = 0.4 \]
\[ P(e_0, \alpha_0 \rightarrow e_2 | Ag_1, Env_1) = 0.6 \]
\[ P(e_0, \alpha_1 \rightarrow e_3 | Ag_2, Env_1) = 0.1 \]
\[ P(e_0, \alpha_1 \rightarrow e_4 | Ag_2, Env_1) = 0.9 \]
\[ P(e_0, \alpha_1 \rightarrow e_5 | Ag_2, Env_1) = 0.7 \]

Utility function defined as follows:

\[ u_1(e_0, \alpha_0 \rightarrow e_1) = 8 \]
\[ u_1(e_0, \alpha_0 \rightarrow e_2) = 11 \]
\[ u_1(e_0, \alpha_1 \rightarrow e_3) = 70 \]
\[ u_1(e_0, \alpha_1 \rightarrow e_4) = 9 \]
\[ u_1(e_0, \alpha_1 \rightarrow e_5) = 10 \]

Whataretheexpectedutilitiesoftheagentsforthisutilityfunction?
An optimal agent

\[
\text{\(A_{\text{opt}}\) in an environment \(E\) is the one that maximizes expected utility:}
\]

\[
\text{\(A_{\text{opt}} = \arg \max_{A \in \mathcal{A}} \mathbb{E}[U(A, E)]\)}
\]

Of course, the fact that an agent is optimal does not mean that it will be best; only that on average, we can expect it to do best.

\[
\text{\(A_{\text{opt}} \) in an environment \(E\) is the one that maximizes expected utility:}
\]

\[
\text{\(A_{\text{opt}} = \arg \max_{A \in \mathcal{A}} \mathbb{E}[U(A, E)]\)}
\]

Bounded optimal agents

A bounded optimal agent, \(A_{\text{bopt}}\), is the agent that can be implemented on some machine (computer) \(m\), with respect to \(m\): the bounded optimal agent, \(A_{\text{bopt}}\), is defined as follows:

\[
\{ \{0, 1\} \} \rightarrow \mathbb{R}^+ \in A_{\text{bopt}}
\]

where \(\{0, 1\}\) is an environment, and \(\mathbb{R}^+\) is a set of all task environments.

A bounded task environment is a pair \((E, \mathbb{S})\), where \(\mathbb{S}\) is an environment and \(\mathbb{S}\) is a predicate over runs.

\[
\text{\(\mathbb{S} : \mathbb{R}^+ \rightarrow \{0, 1\}\)}
\]

A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
Chapter 2
An Introduction to Multiagent Systems

2.1 The Probability of Success

The agent succeeds in a particular environment if

\[ \Psi(\text{run}) = 1 \land \text{run is a good run} \]

A maintenance goal is specified by a set \( G \) of "bad" states.

A maintenance task is specified by a set \( G \) of "good" states.

An achievement task is specified by a set \( G \) of "good" states.

\[ \Psi(\text{run}) = 1 \land \text{run avoids all bad states} \]

1. Achievement tasks are those of the form "achieve state of affairs \( \phi \)."

2. Maintenance tasks are those of the form "maintain state of affairs \( \psi \)."

The Probability of Success

\[ \Pr(\text{run}) = \sum_{\text{run} \in R} \Pr(\text{run}) \]

We then say that an agent \( A \) succeeds in task \( \Psi \) if \( \Psi(\text{run}) = 1 \) for all runs \( \text{run} \) of the environment.

\[ \Pr(\text{run}) = \sum_{\text{run} \in R} \Pr(\text{run}) \]

\[ \Pr(\text{run}) = \sum_{\text{run} \in R} \Pr(\text{run}) \]

\[ \Pr(\text{run}) = \sum_{\text{run} \in R} \Pr(\text{run}) \]

\[ \Pr(\text{run}) = \sum_{\text{run} \in R} \Pr(\text{run}) \]

Achievement & Maintenance Tasks

Two most common types of tasks are achievement and maintenance tasks:

1. Achievement tasks are those of the form "achieve state of affairs \( \phi \)."

2. Maintenance tasks are those of the form "maintain state of affairs \( \psi \)."

An achievement task is specified by a set \( G \) of "good" states:

\[ G \subseteq E \]

The agents succeed if it is guaranteed to bring about at least one of those states (we do not care which one).

An achievement task is specified by a set \( G \) of "good" states:

\[ G \subseteq E \]

A maintenance goal is specified by a set \( B \) of "bad" states:

\[ B \subseteq E \]

The agents succeed if it is guaranteed to avoid all states in \( B \) — if it never performs actions which result in any state in \( B \) occurring.

A maintenance task is specified by a set \( B \) of "bad" states.

\[ B \subseteq E \]

The agents succeed if it is guaranteed to avoid all states in \( B \) — if it never performs actions which result in any state in \( B \) occurring.

The agents succeed if it is guaranteed to avoid all states in \( B \) — if it never performs actions which result in any state in \( B \) occurring.
Agent synthesis

• Agent synthesis is automatic programming: goal is to have a program that will take a task environment and generate an agent that succeeds in this environment.

\[ \text{syn} : \text{TE} \rightarrow (\text{AG} \cup \{\bot\}) \]

(Think of \( \bot \) as being like \text{null} in \text{JAVA}.)

\[ \{(T) \cap \exists \forall \} \leftarrow L \wedge \text{syn} \]

• Agent synthesis is automatic programming: goal is to have a program that will take a task environment and generate an agent that succeeds in this environment.

Soundness and Completeness

• Synthesis algorithm \( \text{syn} \) is sound if it satisfies the following condition:

\[ \text{syn} (\langle \text{Env}, \Psi \rangle) = \text{Ag} \implies R(\text{Ag}, \text{Env}) = R(\Psi, \text{Env}) \]

• Synthesis algorithm is complete if:

\[ \exists \text{Ag} \in \text{AG} \text{ s.t. } R(\text{Ag}, \text{Env}) = R(\Psi, \text{Env}) \implies \text{syn} (\langle \text{Env}, \Psi \rangle) \neq \bot \]