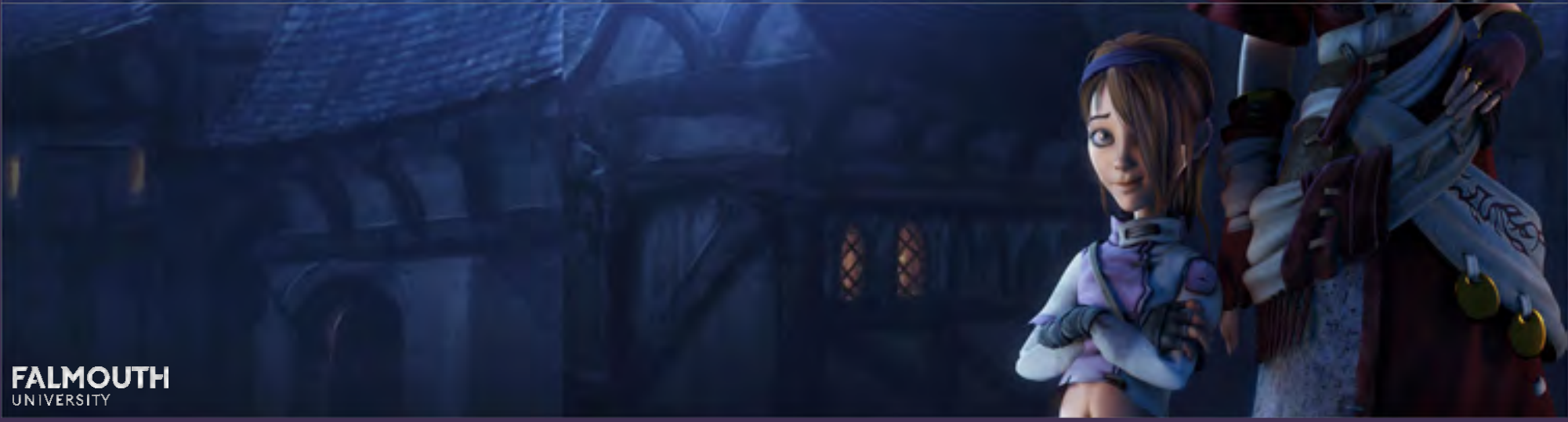


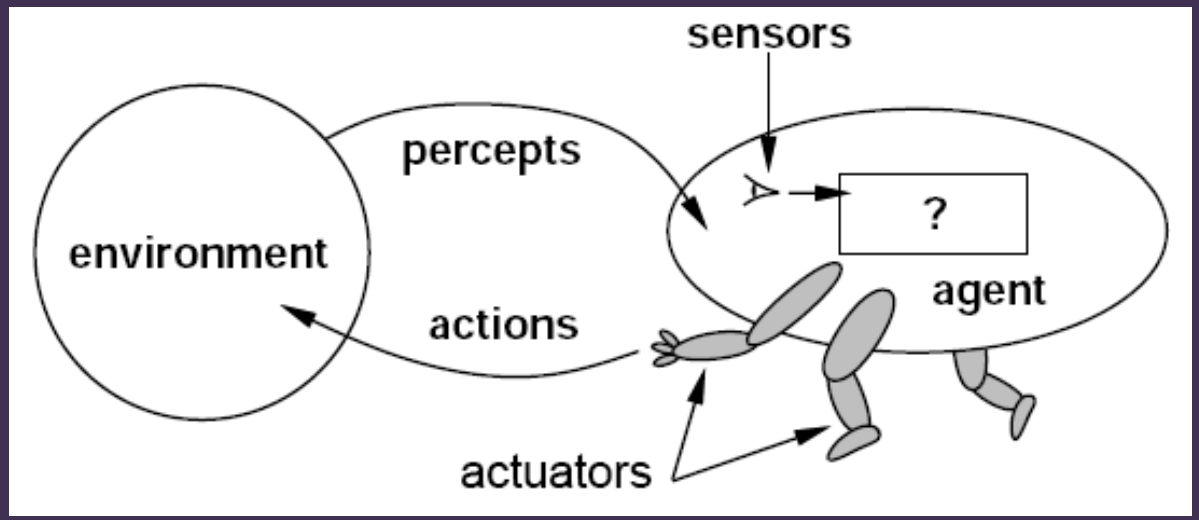
COMP702: Classical Artificial Intelligence

2: Authored Behaviour

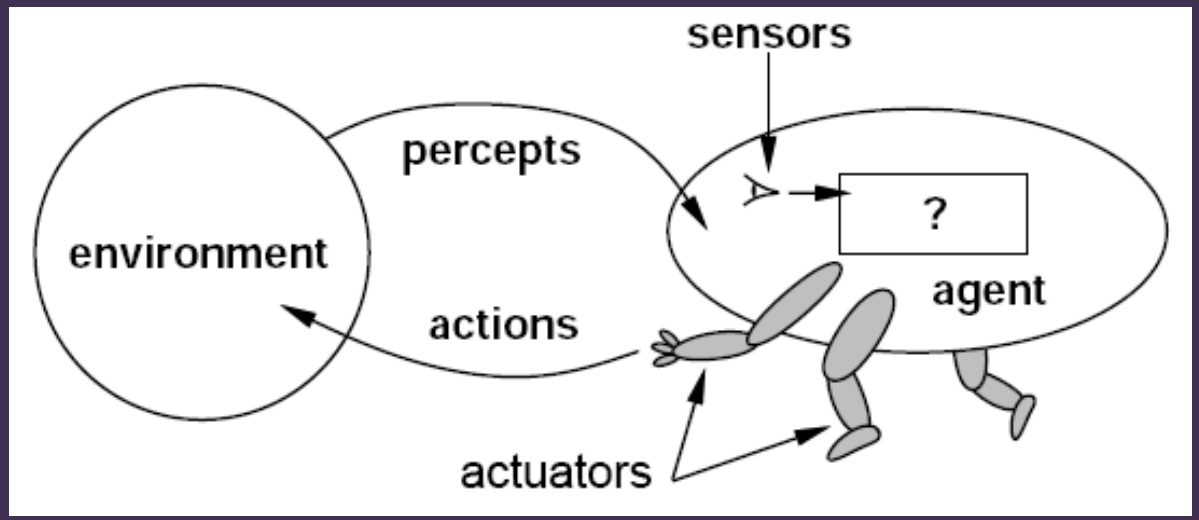
FALMOUTH
UNIVERSITY

Agents

Agents



Agents



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

Performance

Performance

- An "intelligent" agent moves towards some kind of **goal**

Performance

- An "intelligent" agent moves towards some kind of **goal**
- The goal is an **environment state** (or a set of states)

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?

PEAS

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

- A Roomba

PEAS

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

- A Roomba
- A self-driving car

PEAS

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

- A Roomba
- A self-driving car
- A chatbot

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

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

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

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

Types of environment

- Environments come with many different properties

Types of environment

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

Observability

Observability

- **Fully observable**: the agent's sensors give it full information about the state of the environment

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

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

Number of agents

- **Single agent**: our agent is the only one in the environment

Number of agents

- **Single agent**: our agent is the only one in the environment
- **Multi-agent**: there is more than one agent

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

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

Determinism

- **Deterministic**: the next state of the environment is completely determined by the current state and by the agent's action

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

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

Dynamicity

- **Static**: the environment does not change while the agent is deliberating

Dynamicity

- **Static**: the environment does not change while the agent is deliberating
- **Dynamic**: the environment changes constantly

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

Discreteness

- **Discrete**: time, percepts and actions are all discrete (from a finite set of possibilities or "integer valued")

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")

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

Known or unknown

- Are all the details of the environment **known** to the AI designer?

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)

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

Agents and AI

- The ideas of agents and environments are a useful frame for designing AI

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

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

FALMOUTH
UNIVERSITY

Rule-based AI

Rule-based AI

Rule-based AI

- Generally **reactive** to the state of the world

Rule-based AI

- Generally **reactive** to the state of the world
- Based on **if-then** triggers, basic **calculations**, etc.

Rule-based AI

- Generally **reactive** to the state of the world
- Based on **if-then** triggers, basic **calculations**, etc.
- Generally hand-coded and only modifiable by a programmer

Case study: Ghosts in Pac-Man

Case study: Ghosts in Pac-Man

- Full details: `http://gameinternals.com/understanding-pac-man-ghost-behavior`

Case study: Ghosts in Pac-Man

- Full details: `http://gameinternals.com/understanding-pac-man-ghost-behavior`
- Each ghost has 3 states

Case study: Ghosts in Pac-Man

- Full details: `http://gameinternals.com/understanding-pac-man-ghost-behavior`
- Each ghost has 3 states
  - Chase: head for a specific position (see next slide)

Case study: Ghosts in Pac-Man

- Full details: `http://gameinternals.com/understanding-pac-man-ghost-behavior`
- Each ghost has 3 states
  - Chase: head for a specific position (see next slide)
  - Scatter: head for a specific corner of the level

Case study: Ghosts in Pac-Man

- Full details: `http://gameinternals.com/understanding-pac-man-ghost-behavior`
- Each ghost has 3 states
  - Chase: head for a specific position (see next slide)
  - Scatter: head for a specific corner of the level
  - Frightened: move randomly

Ghost “personalities”

Ghost "personalities"

- Red ghost: aim for Pac-Man

Ghost "personalities"

- Red ghost: aim for Pac-Man
- Pink ghost: aim for 2 spaces ahead of Pac-Man

Ghost "personalities"

- Red ghost: aim for Pac-Man
- Pink ghost: aim for 2 spaces ahead of Pac-Man
- Blue ghost: aim for position on the line between red ghost and 2 spaces ahead of Pac-Man

Ghost "personalities"

- Red ghost: aim for Pac-Man
- Pink ghost: aim for 2 spaces ahead of Pac-Man
- Blue ghost: aim for position on the line between red ghost and 2 spaces ahead of Pac-Man
- Orange ghost: aim for Pac-Man until 8 spaces away, then aim for corner

Ghost movement

Ghost movement

- No pathfinding — greedily move towards target

Ghost movement

- No pathfinding — greedily move towards target
- Can only change direction at an intersection

Ghost movement

- No pathfinding — greedily move towards target
- Can only change direction at an intersection
- Can't reverse or stay still

Ghost movement

- No pathfinding — greedily move towards target
- Can only change direction at an intersection
- Can't reverse or stay still
- Therefore can't get stuck, despite imperfect pathfinding

Ghost behaviour

Ghost behaviour

- Behaviour rules are very simple

Ghost behaviour

- Behaviour rules are very simple
- However, the combination of them leads to interesting gameplay and illusion of personality

Design lessons

Design lessons

- AI doesn't have to be complicated

Design lessons

- AI doesn't have to be complicated
- Simple AI, when interacting with a player and each other, can give engaging results

Design lessons

- AI doesn't have to be complicated
- Simple AI, when interacting with a player and each other, can give engaging results
- Bugs in AI don't always matter...

FALMOUTH
UNIVERSITY

Finite state machines

Finite state machines

Finite state machines

- A **finite state machine (FSM)** consists of:

Finite state machines

- A **finite state machine (FSM)** consists of:
  - A set of **states**; and

Finite state machines

- A **finite state machine (FSM)** consists of:
  - A set of **states**; and
  - **Transitions** between states

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**

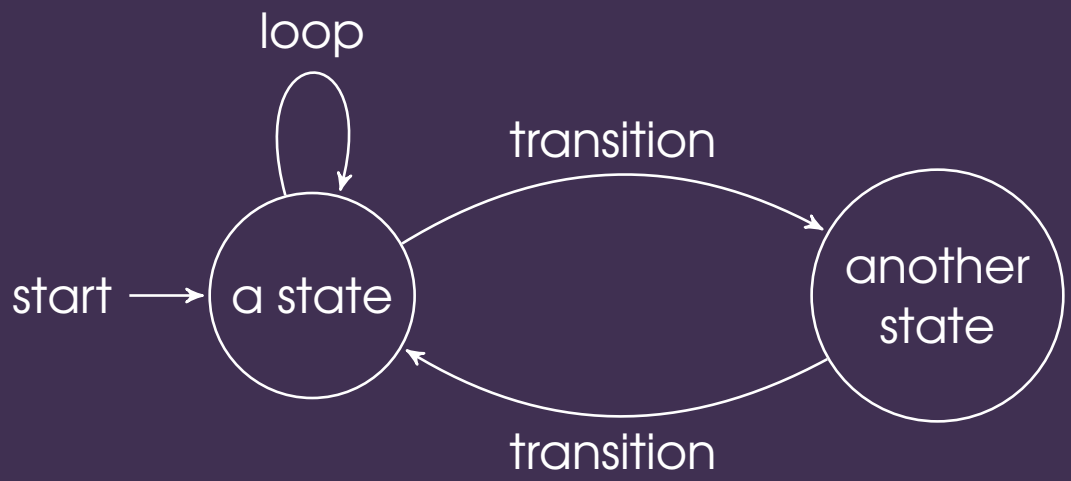
Finite state machines

- 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

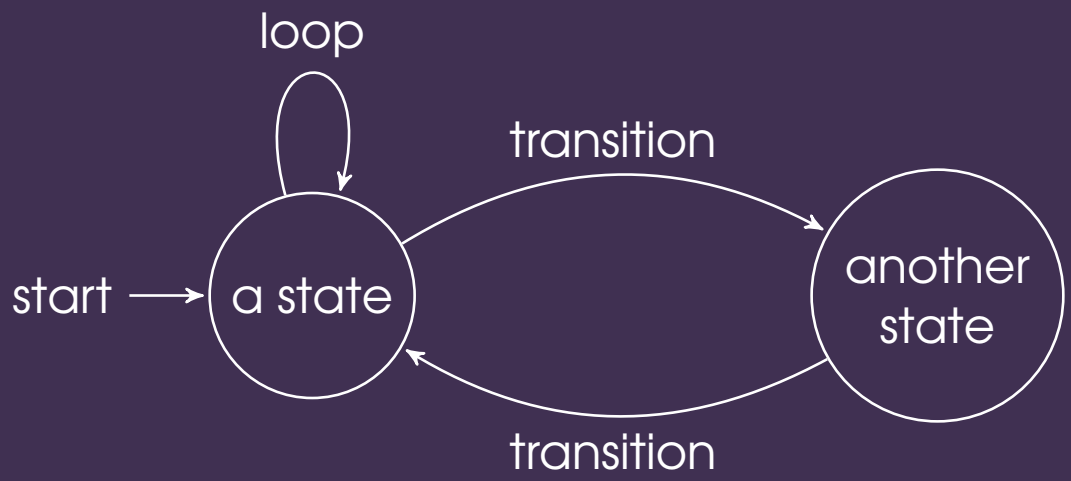
Finite state machines

- 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

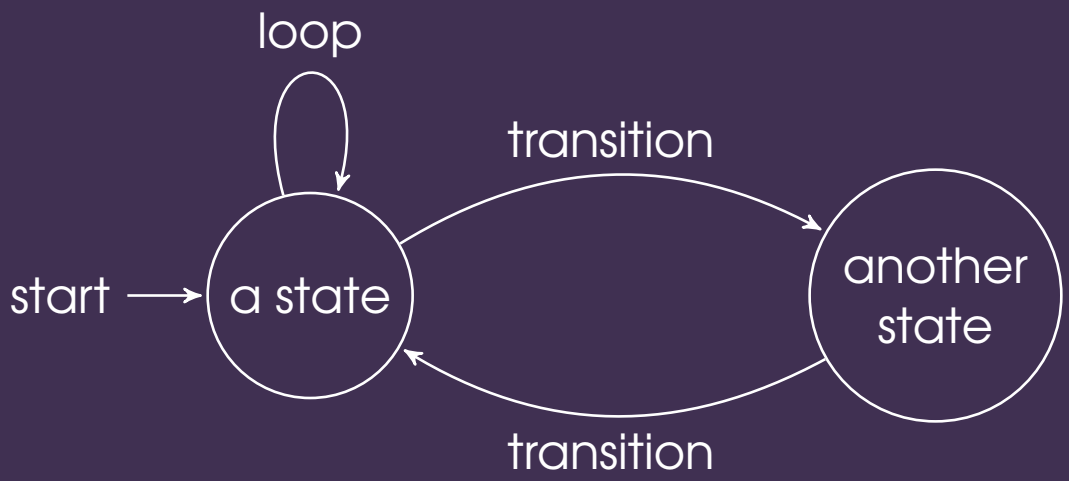


State transition diagrams



- FSMs are often drawn as **state transition diagrams**

State transition diagrams



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

Other uses of FSMs

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

Other uses of FSMs

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

- Animation

Other uses of FSMs

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

- Animation
- UI menu systems

Other uses of FSMs

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

- Animation
- UI menu systems
- Dialogue trees

Other uses of FSMs

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

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

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

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)

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

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

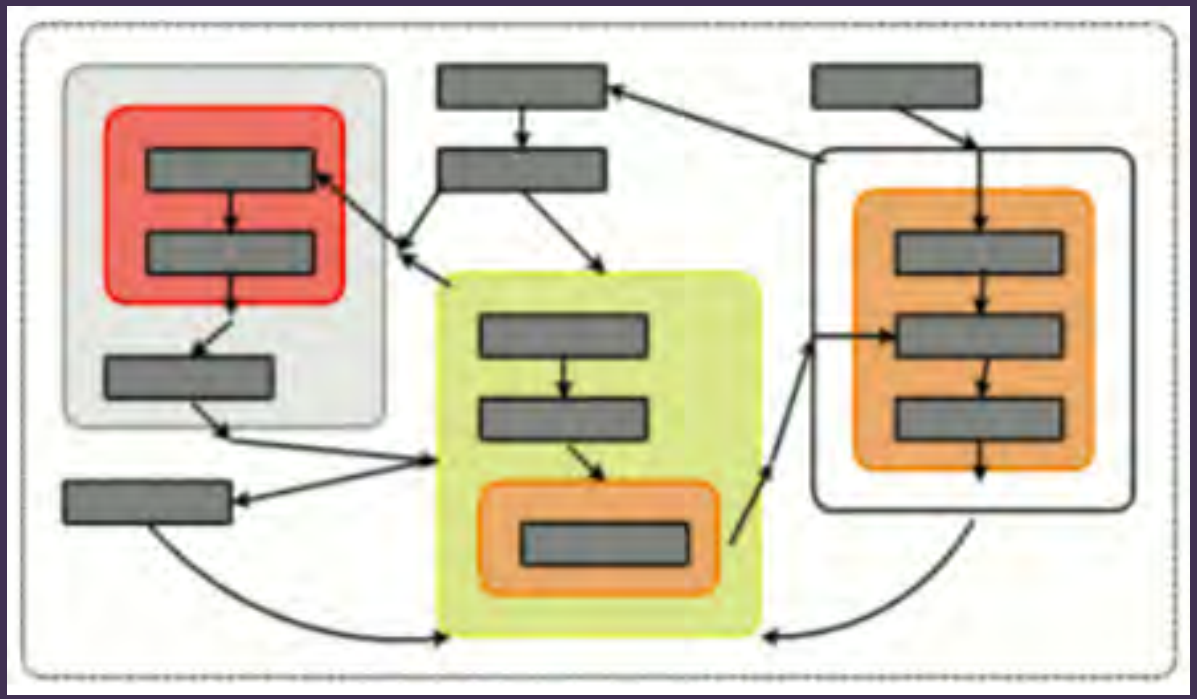
Implementing FSMs

- 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

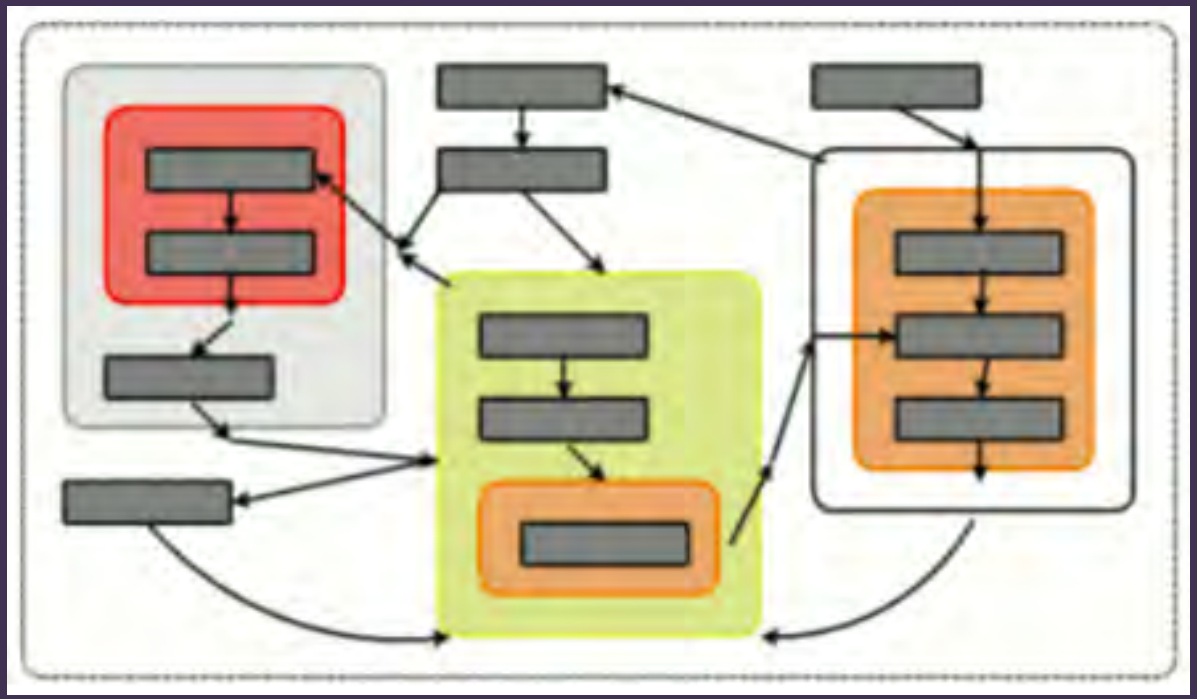
Implementing FSMs

- 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

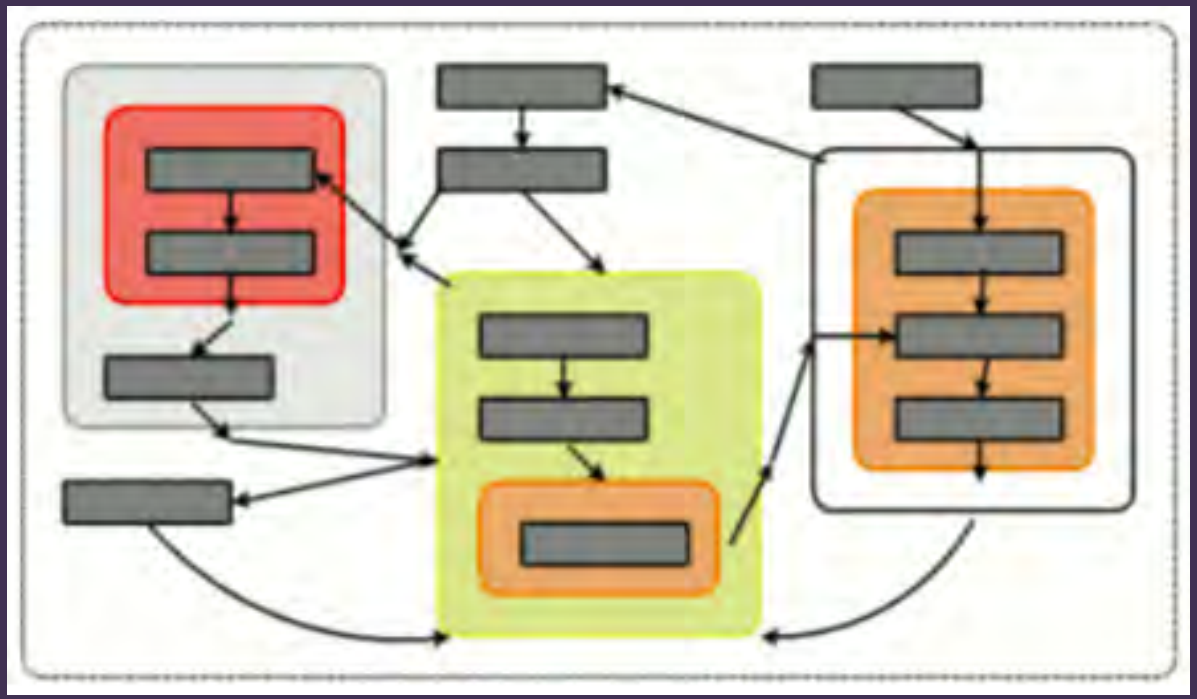


Hierarchical FSMs



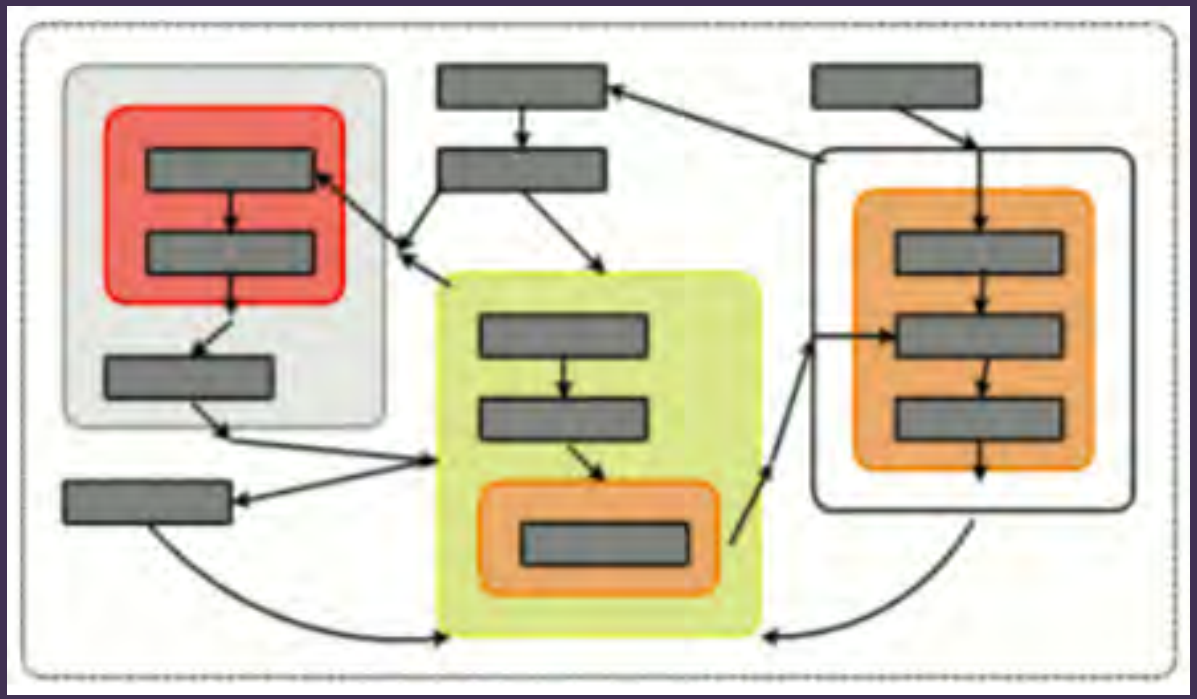
- An FSM with N states has potentially N^2 transitions

Hierarchical FSMs



- An FSM with N states has potentially N^2 transitions
- Designing complex behaviour with FSMs quickly gets unwieldy

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?

Should you use FSMs?

- FSMs are useful for designing simple AI behaviours

Should you use FSMs?

- FSMs are useful for designing simple AI behaviours
- Historically an important technique for game AI

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

FALMOUTH
UNIVERSITY

Behaviour Trees

Behaviour trees (BTs)

Behaviour trees (BTs)

- A **hierarchical** model of decision making

Behaviour trees (BTs)

- A **hierarchical** model of decision making
- Allow **complex behaviours** to be built up from **simple components**

Behaviour trees (BTs)

- A **hierarchical** model of decision making
- Allow **complex behaviours** to be built up from **simple components**
- Allow for **more complex** behaviours than FSMs

Behaviour trees (BTs)

- A **hierarchical** model of decision making
- Allow **complex behaviours** to be built up from **simple components**
- Allow for **more complex** behaviours than FSMs
- First used in Halo 2 (2005), now used extensively

Behaviour trees (BTs)

- A **hierarchical** model of decision making
- Allow **complex behaviours** to be built up from **simple components**
- Allow for **more complex** behaviours than FSMs
- First used in Halo 2 (2005), now used extensively
- Also used in robotics and other non-game AI applications

Using BTs

Using BTs

- Fairly easy to implement; plenty of resources online

Using BTs

- Fairly easy to implement; plenty of resources online
- **Unreal**: an advanced BT system is built in

Using BTs

- Fairly easy to implement; plenty of resources online
- **Unreal**: an advanced BT system is built in
- **Unity**: numerous free and paid options on the Asset Store e.g. Behavior Machine, Behavior Designer, Behave, RAIN

BT basics

BT basics

- A BT is a **tree** of **nodes**

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root
- A ticked node returns one of three **statuses**:

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root
- A ticked node returns one of three **statuses**:
  - Success

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root
- A ticked node returns one of three **statuses**:
  - Success
  - Running

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root
- A ticked node returns one of three **statuses**:
  - Success
  - Running
  - Failure

BT basics

- A BT is a **tree** of **nodes**
- On each game update (i.e. each frame), the root node is **ticked**
  - When a node is ticked, it might cause some or all of its **children** to tick as well
  - So ticks propagate down the tree from the root
- A ticked node returns one of three **statuses**:
  - Success
  - Running
  - Failure
- "Running" status allows nodes to represent operations that **last multiple frames**

Blackboard

Blackboard

- It is often useful to **share** data between nodes

Blackboard

- It is often useful to **share** data between nodes
- A **blackboard** (sometimes called a **data context**) allows this

Blackboard

- It is often useful to **share** data between nodes
- A **blackboard** (sometimes called a **data context**) allows this
- Blackboard defines **variables**, which can be **read** and **written** by nodes

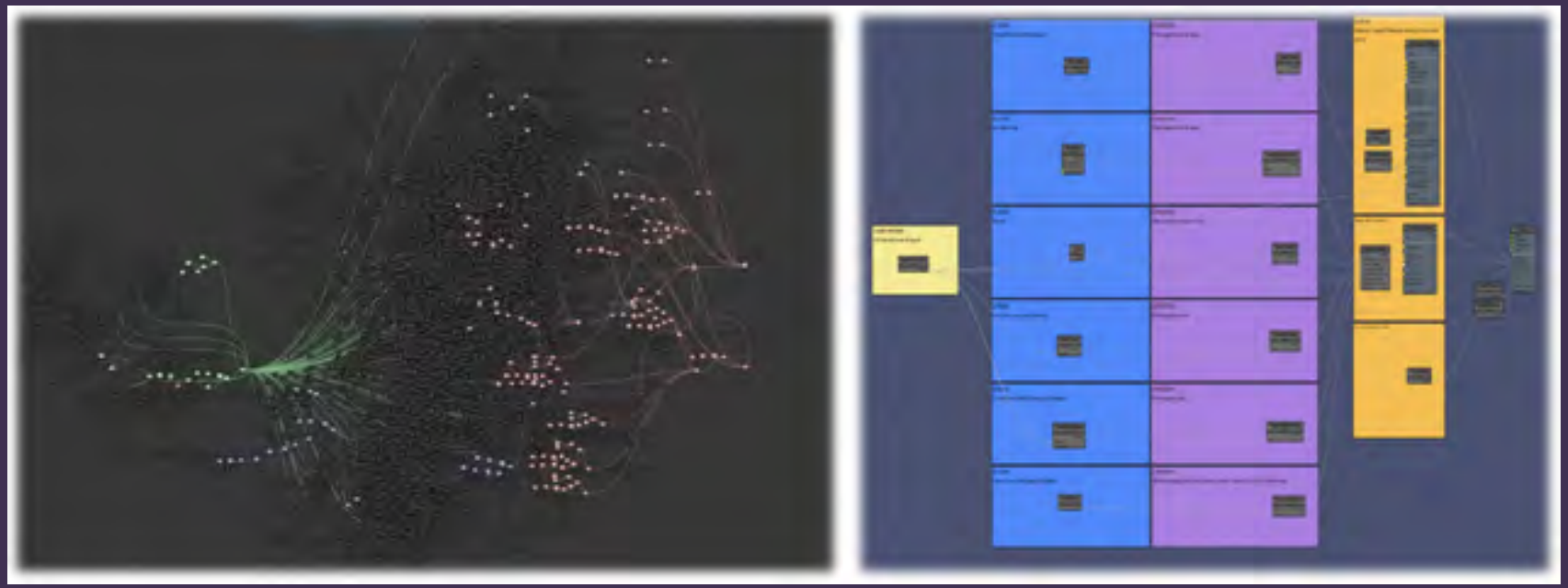
Blackboard

- It is often useful to **share** data between nodes
- A **blackboard** (sometimes called a **data context**) allows this
- Blackboard defines **variables**, which can be **read** and **written** by nodes
- Blackboard can be **local** to the AI agent, **shared** between several agents, or **global** to all agents

Blackboard

- It is often useful to **share** data between nodes
- A **blackboard** (sometimes called a **data context**) allows this
- Blackboard defines **variables**, which can be **read** and **written** by nodes
- Blackboard can be **local** to the AI agent, **shared** between several agents, or **global** to all agents
- (Shared blackboards mean that your AI has "telepathy" — this may or may not be desirable!)

BTs in The Division



http://www.gdcvault.com/play/1023382/AI-Behavior-Editing-and-Debugging