



FALMOUTH  
UNIVERSITY



COMP280: Specialisms in Creative Computing

# 7: AI Architectures

# What is AI?



# What is AI?

# What is AI?

x Simulating human brains or human intelligence

# What is AI?

- ✗ Simulating human brains or human intelligence
- ✓ Performing tasks by machine (or by software) which would ordinarily require human intelligence

# What is AI?

- ✗ Simulating human brains or human intelligence
- ✓ Performing tasks by machine (or by software) which would ordinarily require human intelligence
- ✓ Making decisions to achieve goals

# What is AI?

# What is AI?

✗ Programming machines to learn by themselves



# What is AI?

- ✗ Programming machines to learn by themselves
- ✓ Machine learning is an important sub-field of AI, but there are many other AI techniques

# What is AI?

# What is AI?

- ✗ Programming machines to possess general intelligence, self-awareness, consciousness

# What is AI?

- ✗ Programming machines to possess general intelligence, self-awareness, consciousness
- ✓ Maybe one day, but for now this is pure sci-fi

# What is AI?

- ✗ Programming machines to possess general intelligence, self-awareness, consciousness
- ✓ Maybe one day, but for now this is pure sci-fi
- ✓ Programming machines to carry out (or learn to carry out) a specific type of task

# Computers vs brains

Discuss:

# Computers vs brains

Discuss:

- For what kinds of tasks are digital computers “better” than human brains?

# Computers vs brains

Discuss:

- ▶ For what kinds of tasks are digital computers “better” than human brains?
- ▶ For what kinds of tasks are human brains “better” than digital computers?



# Computers vs brains

Discuss:

- ▶ For what kinds of tasks are digital computers “better” than human brains?
- ▶ For what kinds of tasks are human brains “better” than digital computers?
- ▶ For what kinds of tasks are both “good”, but approach the task in different ways?

# Is it AI?

Discuss: are these examples of AI?

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE



# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)
- ▶ Amazon product recommendations

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)
- ▶ Amazon product recommendations
- ▶ Search function in a text editor

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)
- ▶ Amazon product recommendations
- ▶ Search function in a text editor
- ▶ Google search

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)
- ▶ Amazon product recommendations
- ▶ Search function in a text editor
- ▶ Google search
- ▶ C++ compiler

# Is it AI?

Discuss: are these examples of AI?

- ▶ Calculator
- ▶ Computer opponent in a chess program
- ▶ Enemy in a video game
- ▶ Facebook newsfeed
- ▶ Autocorrect in a text messaging app
- ▶ Autocompletion in an IDE
- ▶ Spellchecker
- ▶ Satellite navigation
- ▶ Virtual assistant (e.g. Siri, Alexa, Cortana etc.)
- ▶ Amazon product recommendations
- ▶ Search function in a text editor
- ▶ Google search
- ▶ C++ compiler
- ▶ Robot



# AI in games



# Applications of AI in games

# Applications of AI in games

- ▶ Enemies and other NPCs

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty
- ▶ Procedural content generation

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty
- ▶ Procedural content generation
- ▶ Content production tools



# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty
- ▶ Procedural content generation
- ▶ Content production tools
- ▶ Procedural narrative

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty
- ▶ Procedural content generation
- ▶ Content production tools
- ▶ Procedural narrative
- ▶ Agent-based simulations

# Applications of AI in games

- ▶ Enemies and other NPCs
- ▶ Opponents in {board, card, strategy} games
- ▶ Automated playtesting
- ▶ Directors, hints, adaptive difficulty
- ▶ Procedural content generation
- ▶ Content production tools
- ▶ Procedural narrative
- ▶ Agent-based simulations
- ▶ ...

# Design considerations

# Design considerations

- ▶ Creating “perfect” AI is an interesting technical challenge, but may be bad game design

# Design considerations

- ▶ Creating “perfect” AI is an interesting technical challenge, but may be bad game design

**procedure** ENEMY SOLDIER AI

**while** player.isAlive **do**

        AIM AT(player.head)

        SHOOT( )

**end while**

**end procedure**

# Design considerations

- ▶ Creating “perfect” AI is an interesting technical challenge, but may be bad game design

**procedure** ENEMY SOLDIER AI

```
    while player.isAlive do
        AIM AT (player.head)
        SHOOT ( )
```

```
    end while
```

**end procedure**

- ▶ A common (and difficult) challenge: creating AI which is **imperfect**, but not obviously **stupid**

# AI architectures

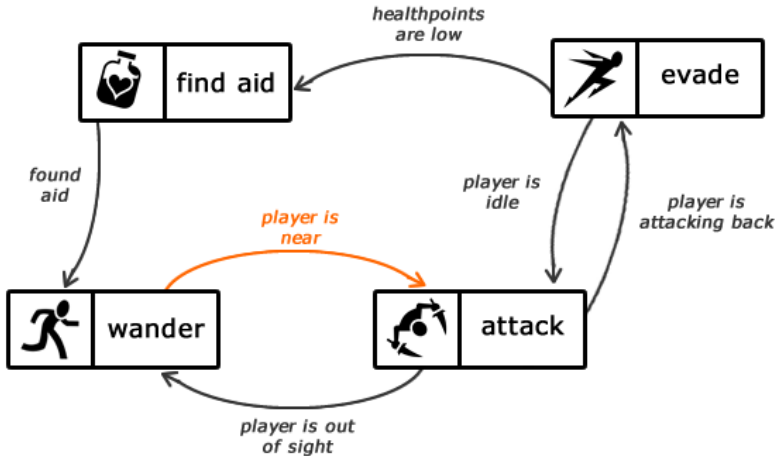




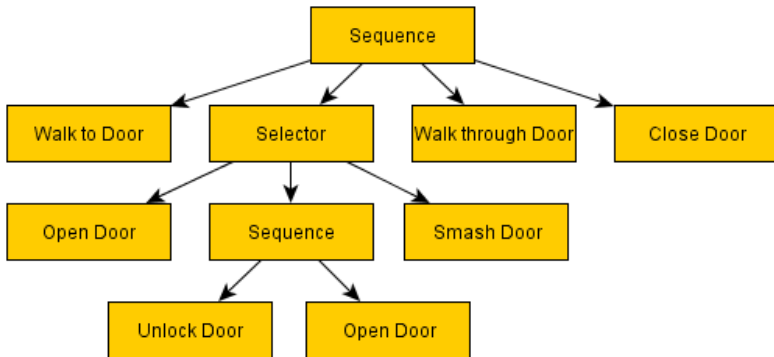
# Rule-based AI

Generally implemented as `if` statements or event-based triggers

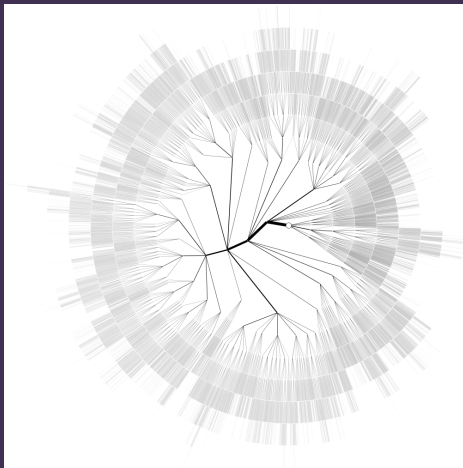
# Finite state machines



# Behaviour trees



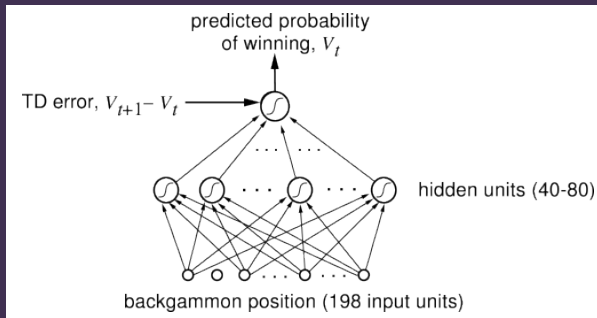
# Game tree search



# Multi-agent approaches (e.g. flocking)



# Machine learning



# AI architectures

# AI architectures

- ▶ Can roughly be divided into **hand-authored...**



# AI architectures

- ▶ Can roughly be divided into **hand-authored...**
  - ▶ Rule-based, FSM, behaviour trees

# AI architectures

- ▶ Can roughly be divided into **hand-authored...**
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**

# AI architectures

- ▶ Can roughly be divided into **hand-authored...**
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning

# AI architectures

- ▶ Can roughly be divided into **hand-authored**...
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning
- ▶ Do you want to **design** the AI behaviours yourself, or do you want them to **emerge** from the system?

# AI architectures

- ▶ Can roughly be divided into **hand-authored**...
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning
- ▶ Do you want to **design** the AI behaviours yourself, or do you want them to **emerge** from the system?
- ▶ Predictability and authorial control versus adaptability and novelty

# AI architectures

- ▶ Can roughly be divided into **hand-authored**...
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning
- ▶ Do you want to **design** the AI behaviours yourself, or do you want them to **emerge** from the system?
- ▶ Predictability and authorial control versus adaptability and novelty
- ▶ Can also combine the two

# AI architectures

- ▶ Can roughly be divided into **hand-authored**...
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning
- ▶ Do you want to **design** the AI behaviours yourself, or do you want them to **emerge** from the system?
- ▶ Predictability and authorial control versus adaptability and novelty
- ▶ Can also combine the two
  - ▶ E.g. use a rule-based system to constrain a CI system

# AI architectures

- ▶ Can roughly be divided into **hand-authored**...
  - ▶ Rule-based, FSM, behaviour trees
- ▶ ... and **computational intelligence**
  - ▶ Search, multi-agent, machine learning
- ▶ Do you want to **design** the AI behaviours yourself, or do you want them to **emerge** from the system?
- ▶ Predictability and authorial control versus adaptability and novelty
- ▶ Can also combine the two
  - ▶ E.g. use a rule-based system to constrain a CI system
  - ▶ E.g. flocking — individual agents are usually rule-based, but overall flock dynamics are emergent



# 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**

# Node types

# Node types

- ▶ There are **two main types** of BT node

# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes



# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children

# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent **tasks** (i.e. the AI agent actually doing something)

# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent **tasks** (i.e. the AI agent actually doing something)
- ▶ **Composite** nodes

# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent **tasks** (i.e. the AI agent actually doing something)
- ▶ **Composite** nodes
  - ▶ One or more children

# Node types

- ▶ There are **two main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent **tasks** (i.e. the AI agent actually doing something)
- ▶ **Composite** nodes
  - ▶ One or more children
  - ▶ Control which of the children run on each tick

# Leaf nodes

# Leaf nodes

- ▶ Represent **atomic actions**

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions



# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"
  - ▶ Running means "the action is still in progress"

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"
  - ▶ Running means "the action is still in progress"
- ▶ Leaf nodes can also be used to represent **conditions**

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"
  - ▶ Running means "the action is still in progress"
- ▶ Leaf nodes can also be used to represent **conditions**
  - ▶ E.g. "is my health below 10%?"

# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"
  - ▶ Running means "the action is still in progress"
- ▶ Leaf nodes can also be used to represent **conditions**
  - ▶ E.g. "is my health below 10%?"
  - ▶ Returns success for true, failure for false



# Leaf nodes

- ▶ Represent **atomic actions**
  - ▶ I.e. actions which can't sensibly be broken down into smaller actions
- ▶ E.g. walk to, crouch, attack, open door
- ▶ Status:
  - ▶ Success means "the action is done"
  - ▶ Failure means "the action cannot be done"
  - ▶ Running means "the action is still in progress"
- ▶ Leaf nodes can also be used to represent **conditions**
  - ▶ E.g. "is my health below 10%?"
  - ▶ Returns success for true, failure for false
- ▶ ... although this is not recommended in Unreal — conditionals should be implemented as **decorators** instead

# Composite nodes: sequence

# Composite nodes: sequence

- ▶ Run each child, in order

# Composite nodes: sequence

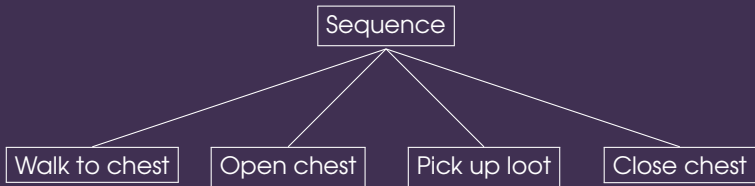
- ▶ Run each child, in order
- ▶ If **any** child returns failure, stop and return failure

# Composite nodes: sequence

- ▶ Run each child, in order
- ▶ If **any** child returns failure, stop and return failure
- ▶ If **all** children return success, stop and return success

# Composite nodes: sequence

- ▶ Run each child, in order
- ▶ If **any** child returns failure, stop and return failure
- ▶ If **all** children return success, stop and return success



# Sequence nodes and conditions

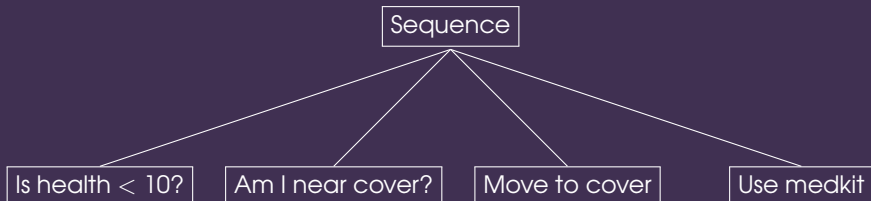
# Sequence nodes and conditions

- ▶ A sequence node can be used like an `if (cond1 && cond2)` statement



# Sequence nodes and conditions

- ▶ A sequence node can be used like an `if (cond1 && cond2)` statement



# Composite nodes: selector

# Composite nodes: selector

- ▶ Run each child, in order

# Composite nodes: selector

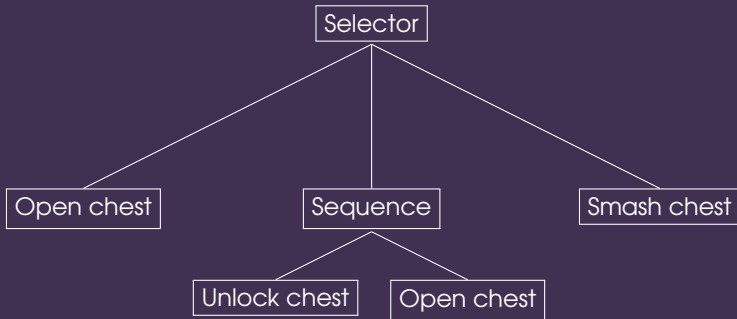
- ▶ Run each child, in order
- ▶ If a child returns failure, move onto the next one

# Composite nodes: selector

- ▶ Run each child, in order
- ▶ If a child returns failure, move onto the next one
- ▶ If **any** child returns success, stop and return success

# Composite nodes: selector

- ▶ Run each child, in order
- ▶ If a child returns failure, move onto the next one
- ▶ If **any** child returns success, stop and return success



# Selectors and priority

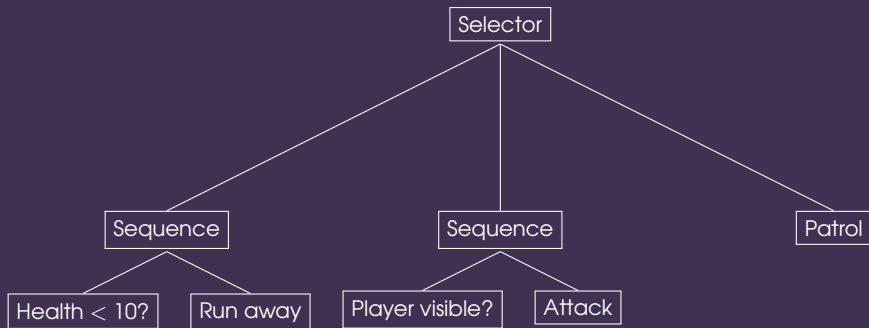
# Selectors and priority

- ▶ Order of selector children represents the **priority** of different alternatives



# Selectors and priority

- Order of selector children represents the **priority** of different alternatives



# Sequence vs selector

# Sequence vs selector

- ▶ Sequence: perform a list of actions; if one of them fails then abandon the task

# Sequence vs selector

- ▶ Sequence: perform a list of actions; if one of them fails then abandon the task
- ▶ Selector: try a list of alternatives; stop once you find one that works

# Sequence vs selector

- ▶ Sequence: perform a list of actions; if one of them fails then abandon the task
- ▶ Selector: try a list of alternatives; stop once you find one that works
- ▶ Sequence works like **and**, selector works like **or**

# Other composite nodes

# Other composite nodes

- ▶ Execute children in **random** order

# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**



# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**
- ▶ **Decorator** nodes

# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**
- ▶ **Decorator** nodes
  - ▶ **Inverter**: if child returns success then return failure, and vice versa

# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**
- ▶ **Decorator** nodes
  - ▶ **Inverter**: if child returns success then return failure, and vice versa
  - ▶ **Repeater**: run the child a number of times, or forever

# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**
- ▶ **Decorator** nodes
  - ▶ **Inverter**: if child returns success then return failure, and vice versa
  - ▶ **Repeater**: run the child a number of times, or forever
  - ▶ Most BT frameworks allow programmers to create custom decorator nodes

# Other composite nodes

- ▶ Execute children in **random** order
- ▶ Execute children in **parallel**
- ▶ **Decorator** nodes
  - ▶ **Inverter**: if child returns success then return failure, and vice versa
  - ▶ **Repeater**: run the child a number of times, or forever
  - ▶ Most BT frameworks allow programmers to create custom decorator nodes
- ▶ Some BT frameworks allow programmers to create custom composite nodes

# 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

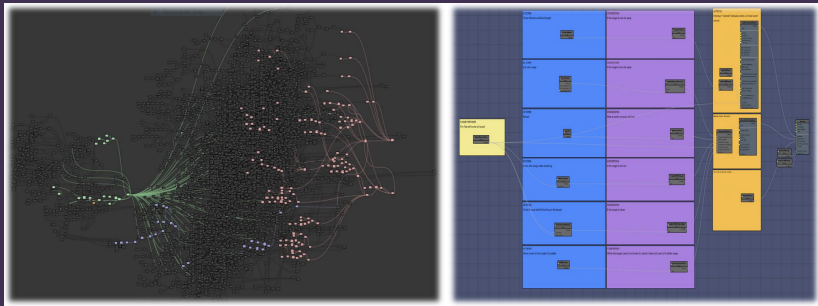
# Blackboard

- ▶ 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
- ▶ Some BT frameworks allow blackboards to 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
- ▶ Some BT frameworks allow blackboards to 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](http://www.gdcvault.com/play/1023382/AI-Behavior-Editing-and-Debugging)

# Worksheet



# COMP280 worksheet 2

# COMP280 worksheet 2

- Implement AI ghost behaviours for a Pac-Man game

# COMP280 worksheet 2

- ▶ Implement AI ghost behaviours for a Pac-Man game
- ▶ Brief on LearningSpace



# COMP280 worksheet 2

- ▶ Implement AI ghost behaviours for a Pac-Man game
- ▶ Brief on LearningSpace
- ▶ Template project on GitHub

# Workshop

- ▶ Make a start on the worksheet!
- ▶ Follow the tutorial linked in the worksheet to implement a simple **behaviour tree based ghost AI**
- ▶ Start experimenting with modifying your AI