

# COMP250: Artificial Intelligence

## 3: Behaviour trees

# Research Journal



# Topic assignments

# Topic assignments

- ▶ Linked on LearningSpace

# Topic assignments

- ▶ Linked on LearningSpace
- ▶ Suggested jumping-off point: “Artificial Intelligence and Games” by Georgios Yannakakis and Julian Togelius

# Topic assignments

- ▶ Linked on LearningSpace
- ▶ Suggested jumping-off point: “Artificial Intelligence and Games” by Georgios Yannakakis and Julian Togelius
- ▶ ... but follow up the references as well!

# Topic assignments

- ▶ Linked on LearningSpace
- ▶ Suggested jumping-off point: “Artificial Intelligence and Games” by Georgios Yannakakis and Julian Togelius
- ▶ ... but follow up the references as well!
- ▶ **You** are initially responsible for your own assigned topics, but in the end **everyone** is responsible for **everything** on the wiki

# Topic assignments

- ▶ Linked on LearningSpace
- ▶ Suggested jumping-off point: “Artificial Intelligence and Games” by Georgios Yannakakis and Julian Togelius
- ▶ ... but follow up the references as well!
- ▶ **You** are initially responsible for your own assigned topics, but in the end **everyone** is responsible for **everything** on the wiki
- ▶ You should edit and improve each others’ work



# Topic assignments

- ▶ Linked on LearningSpace
- ▶ Suggested jumping-off point: “Artificial Intelligence and Games” by Georgios Yannakakis and Julian Togelius
- ▶ ... but follow up the references as well!
- ▶ **You** are initially responsible for your own assigned topics, but in the end **everyone** is responsible for **everything** on the wiki
- ▶ You should edit and improve each others’ work
- ▶ I will be checking the wiki in **week 5**, and I hope to see good progress!

# 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

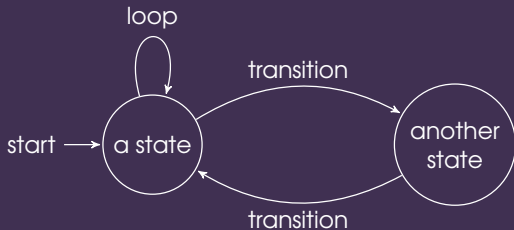
# Finite state machines

- ▶ 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

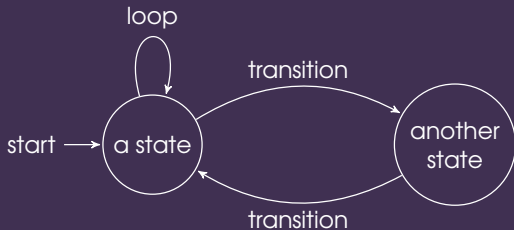
- ▶ A **finite state machine (FSM)** consists of:
  - ▶ A set of **states**; and
  - ▶ **Transitions** between states
- ▶ At any given time, the FSM is in a **single state**
- ▶ **Inputs** or **events** can cause the FSM to transition to a different state

# State transition diagrams



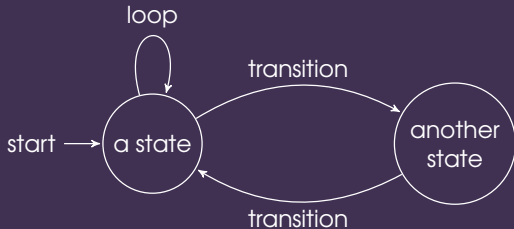


# State transition diagrams



- ▶ 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**

# FSMs for AI behaviour

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

# 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)

# 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

# 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

# FSMs for AI behaviour

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

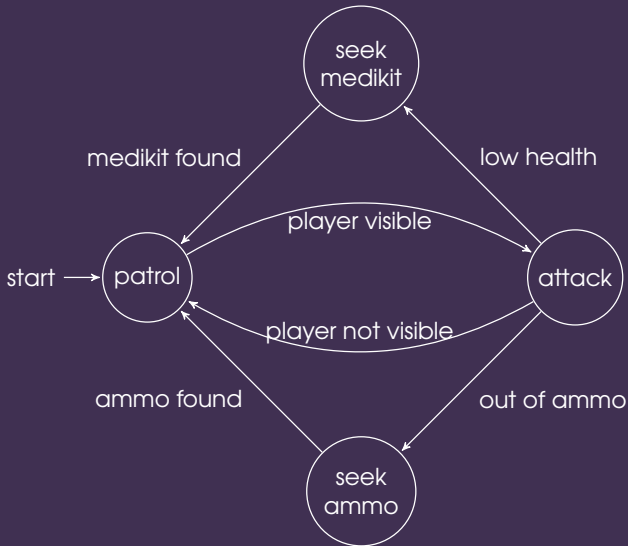
- ▶ By default, patrol (e.g. along a preset route)
- ▶ If the player is spotted, attack them
- ▶ If the player is no longer visible, resume patrolling
- ▶ If you are low on health, run away and find a medikit.  
Then resume patrolling

# 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





# 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
- ▶ ...

# Beyond FSMs

Some topics for you to research, for when plain old FSMs aren't enough...



# Beyond FSMs

Some topics for you to research, for when plain old FSMs aren't enough...

- ▶ Hierarchical FSMs
- ▶ Nested FSMs
- ▶ Stack-based FSMs
- ▶ Hierarchical task networks
- ▶ ...

Plus the topic we will be looking at today: **behaviour trees**

# 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 **three main types** of BT node



# Node types

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

# Node types

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

# Node types

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

# Node types

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

# Node types

- ▶ There are **three main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent actions (i.e. the AI agent actually doing something)
- ▶ **Decorator** nodes
  - ▶ One child

# Node types

- ▶ There are **three main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent actions (i.e. the AI agent actually doing something)
- ▶ **Decorator** nodes
  - ▶ One child
  - ▶ Modify the execution of the child

# Node types

- ▶ There are **three main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent actions (i.e. the AI agent actually doing something)
- ▶ **Decorator** nodes
  - ▶ One child
  - ▶ Modify the execution of the child
- ▶ **Composite** nodes

# Node types

- ▶ There are **three main types** of BT node
- ▶ **Leaf** nodes
  - ▶ No children
  - ▶ Represent actions (i.e. the AI agent actually doing something)
- ▶ **Decorator** nodes
  - ▶ One child
  - ▶ Modify the execution of the child
- ▶ **Composite** nodes
  - ▶ Control which of the children are executed 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
- ▶ Leaf nodes often have **parameters** to allow for reuse in different situations

# 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

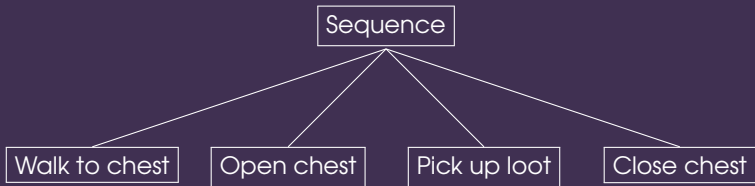
# Composite nodes: sequence

- ▶ 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



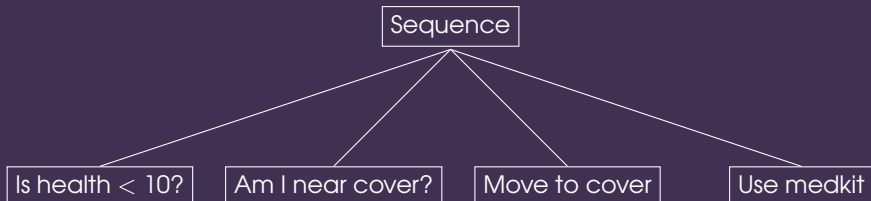
# Sequence nodes and conditions

# 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

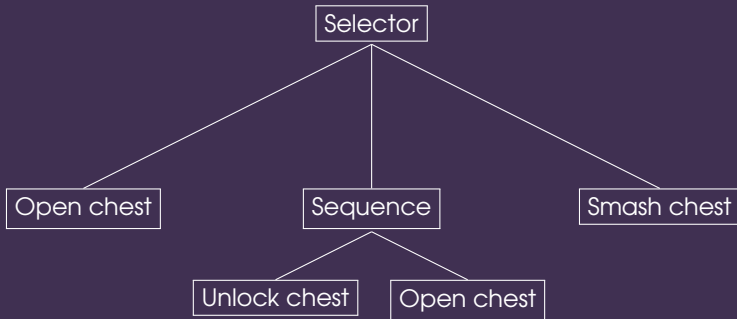
# Composite nodes: selector

- ▶ 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



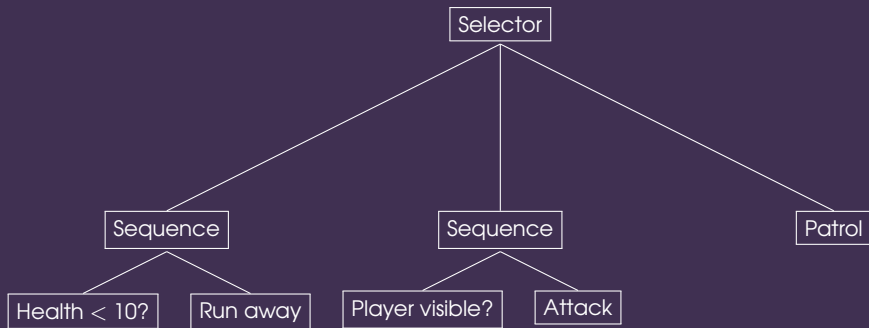
# Selectors and priority

# 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**
- ▶ Most BT frameworks allow programmers to create custom composite nodes

# Decorator nodes

# Decorator nodes

- ▶ **Inverter:** if child returns success then return failure, and vice versa

# Decorator nodes

- ▶ **Inverter:** if child returns success then return failure, and vice versa
- ▶ **Repeater:** run the child a number of times, or forever

# 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



# 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

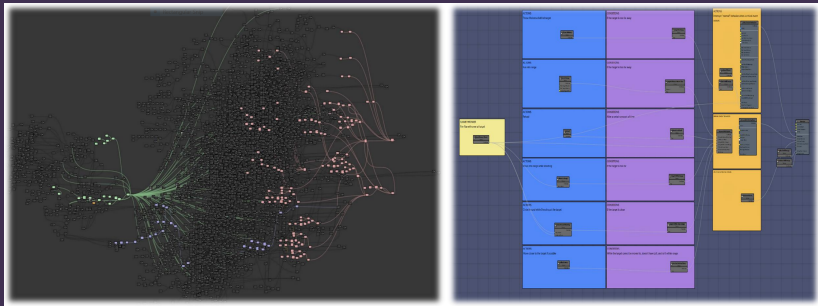
# Blackboard

- ▶ 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](http://www.gdcvault.com/play/1023382/AI-Behavior-Editing-and-Debugging)

# Activity

**Unreal users:** follow the tutorial at

<https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/QuickStart/>

**Unity users:** download “Behaviour Machine Free” from the Asset Store, and follow the tutorial at

<https://youtu.be/ZV11FM24OXg>



# Portfolio task proposals

