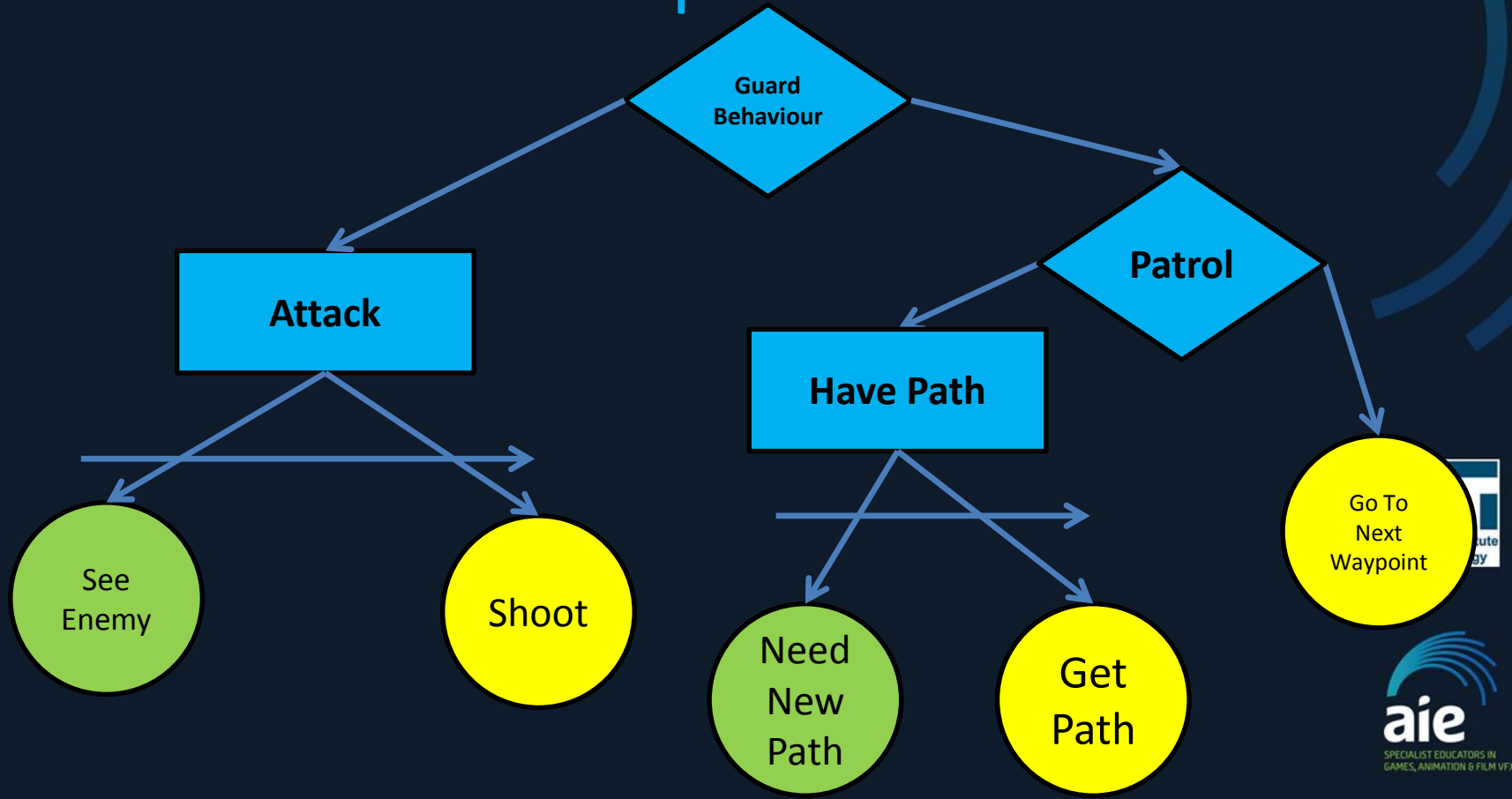
Behaviour Trees – Part 1

Complex decision making through discrete modular behaviours

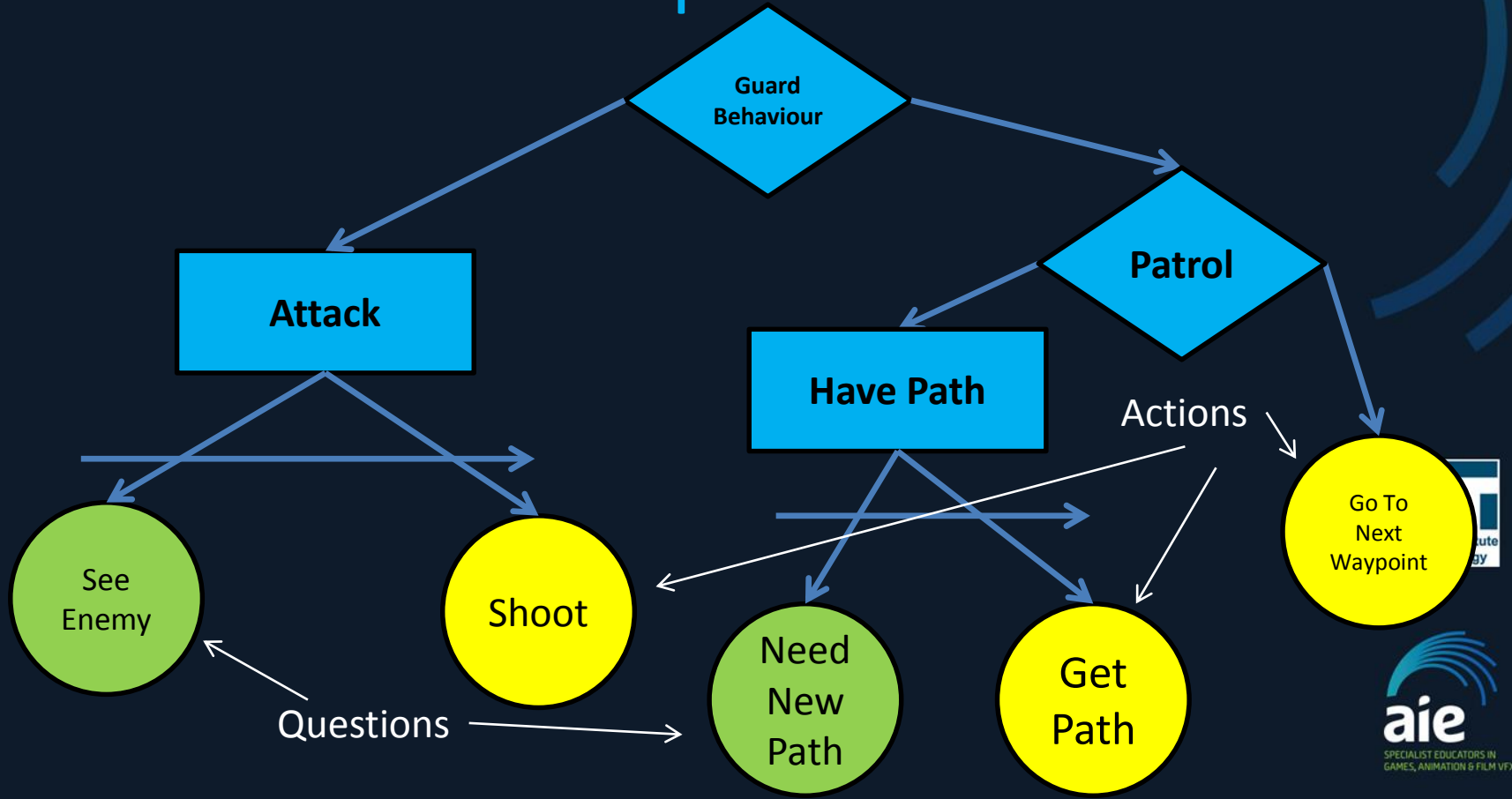
Behaviour Trees

- Behaviour Trees are a more complicated form of Decision Tree
- Decisions are made based on a composite of “questions” and “actions” rather than a single **YES** / **NO** question
  - Allows for **AND** and **OR** logic
- Actions are typically broken down into discrete parts
  - **Animate** would be a separate discrete action, as would **Move**, rather than an action that both **Moves and Animates** the A.I. agent
  - Multiple Actions may be triggered in one Decision

Behaviour Tree Example



Behaviour Tree Example



Behaviour Tree Nodes

- Behaviour Trees are a tree structure made up of nodes called **Behaviours**
- Behaviours typically come in three types
  - **Composites**, which are Branch nodes that contain child Behaviours
  - **Actions**, which are Leaf nodes in the tree
  - **Conditions**, which are also Leaf nodes in the tree
- All Behaviours, when executed, return if the behaviour was successful
  - **Success** and **Failure** are typical results
  - Some behaviours can return “on-going”

```
enum BehaviourResult
   Success
   Failure

class Behaviour
   func execute(Agent agent) = 0
```

```
class Agent
   Behaviour behaviour

   func update()
      behaviour.execute(this)
```

Behaviour Tree Actions and Conditions

- An **Action** Behaviour is a behaviour that “does” something
  - Like “Animate”, or “Move a step forward”
  - Typically always return **Success**


```
class AttackAction : Behaviour
   integer damageToApply
   float range

   func execute(Agent agent)
      for each enemy within range of agent
         enemy.damage -= damageToApply
      return BehaviourResult.Success
```

- A **Condition** Behaviour is a behaviour that “asks” something
  - For example, “is health almost empty”, “can see enemy”
  - Returns **Success** or **Failure**


```
class EnemiesCloseCondition : Behaviour
   float range

   func execute(Agent agent)
      for each enemy
         if distance(enemy, agent) < range
            return BehaviourResult.Success
      return BehaviourResult.Failure
```

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Behaviour Tree Composite Nodes

- **Composite** Behaviours have multiple child Behaviours
  - For example, a Decision in a Decision Tree would be a Composite with 2 children
  - Composites return a result based off their child nodes
  - There are a few types of Composite, two common types being **Selector** and **Sequence**

- Composites are what differentiate a behaviour Tree from a Decision Tree
  - They add the **AND** and **OR** logic
  - Allow multiple conditions and actions

```
class CompositeBehaviour : Behaviour
   list childBehaviours

   func execute(Agent agent) = 0
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Selector Composite Nodes

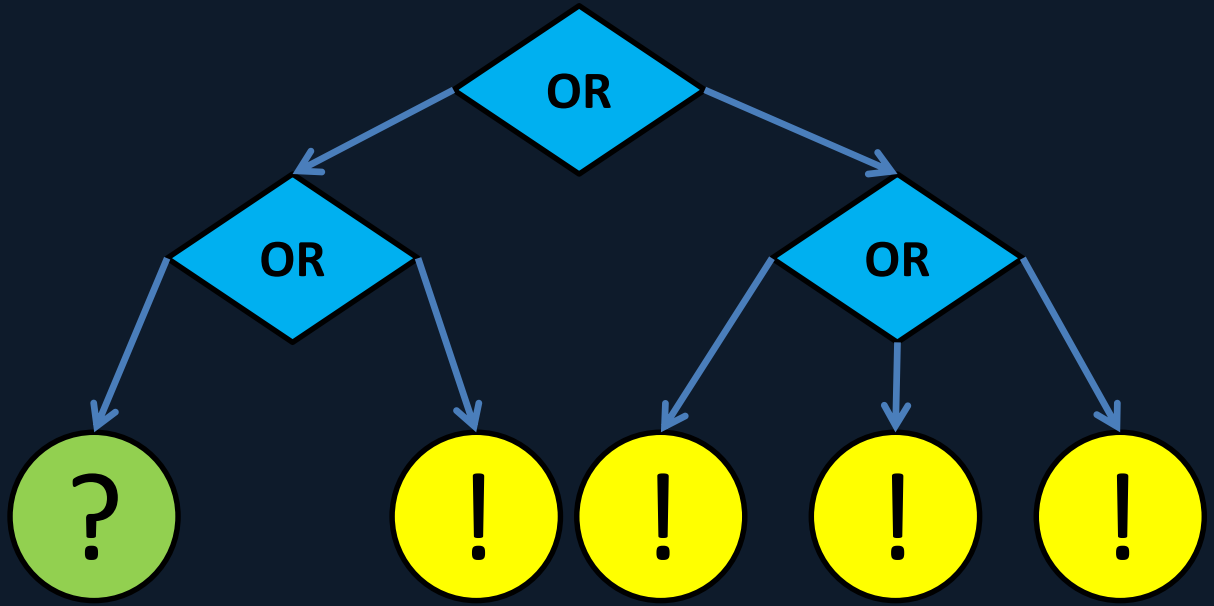
- A **Selector** is a **Composite** node that returns Success if one of its child nodes returns Success
  - For example:
    - If a child returns Success then the Selector returns Success without executing its remaining child Behaviours
    - If a child returned Failure then it would execute the next child Behaviour
    - If all child behaviours return Failure then the Selector returns Failure

- A **Selector** acts as an **OR**
  - Result = A **OR** B **OR** C **OR** etc etc

```
class Selector : CompositeBehaviour

   func execute(Agent agent)
      for each child in childBehaviours
         if child.execute(agent) == BehaviourResult.Success
            return BehaviourResult.Success
      return BehaviourResult.Failure
```

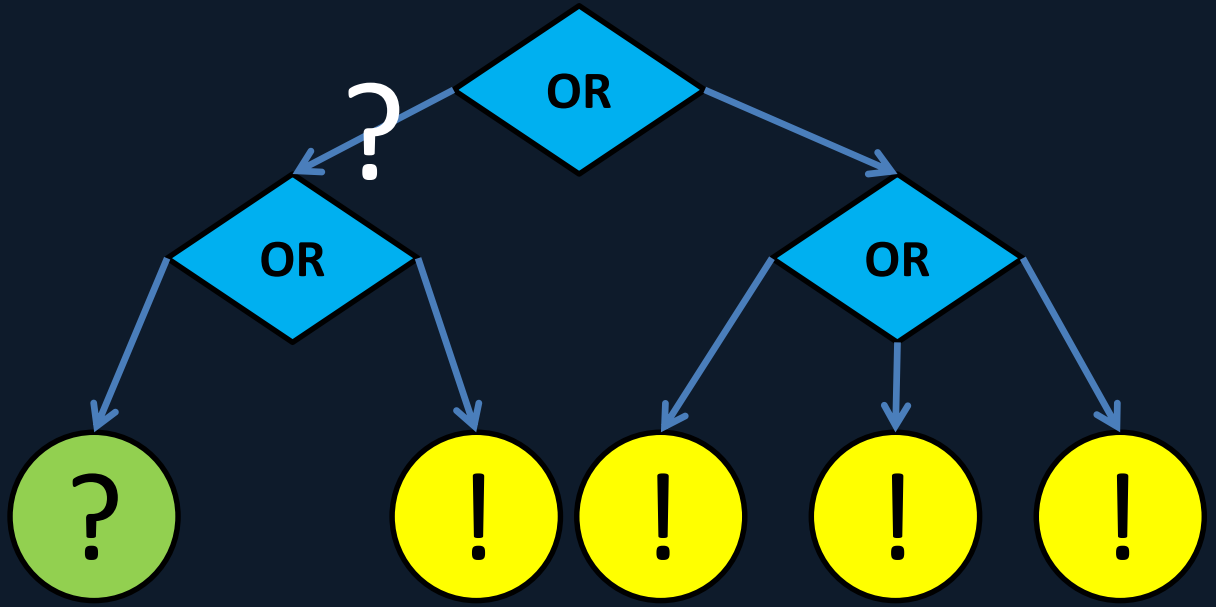
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

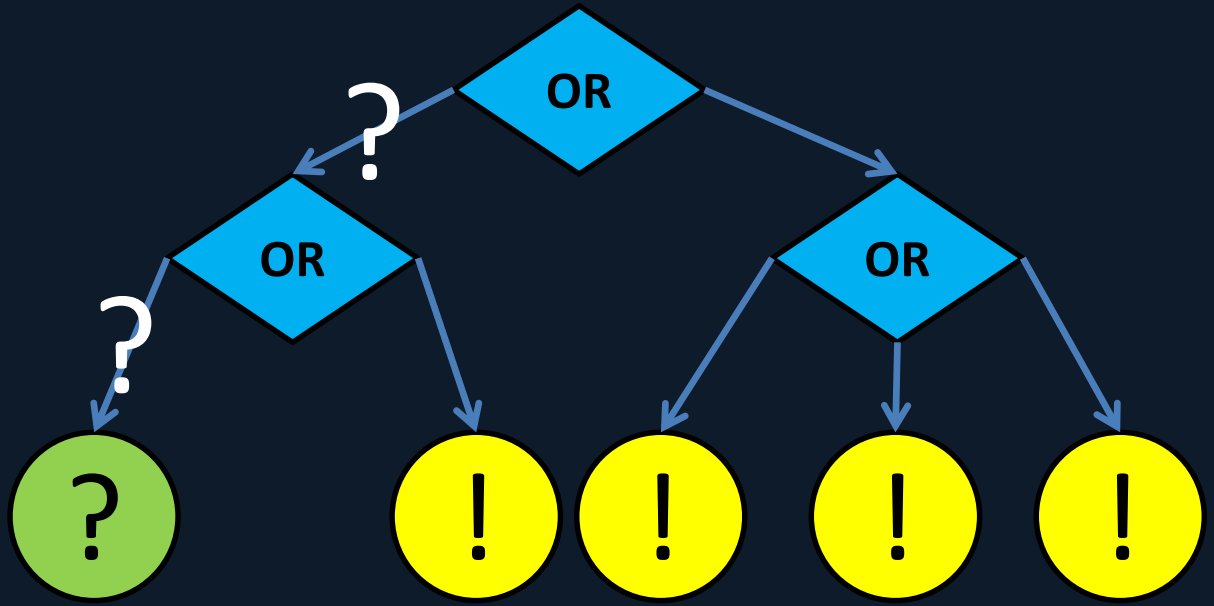
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

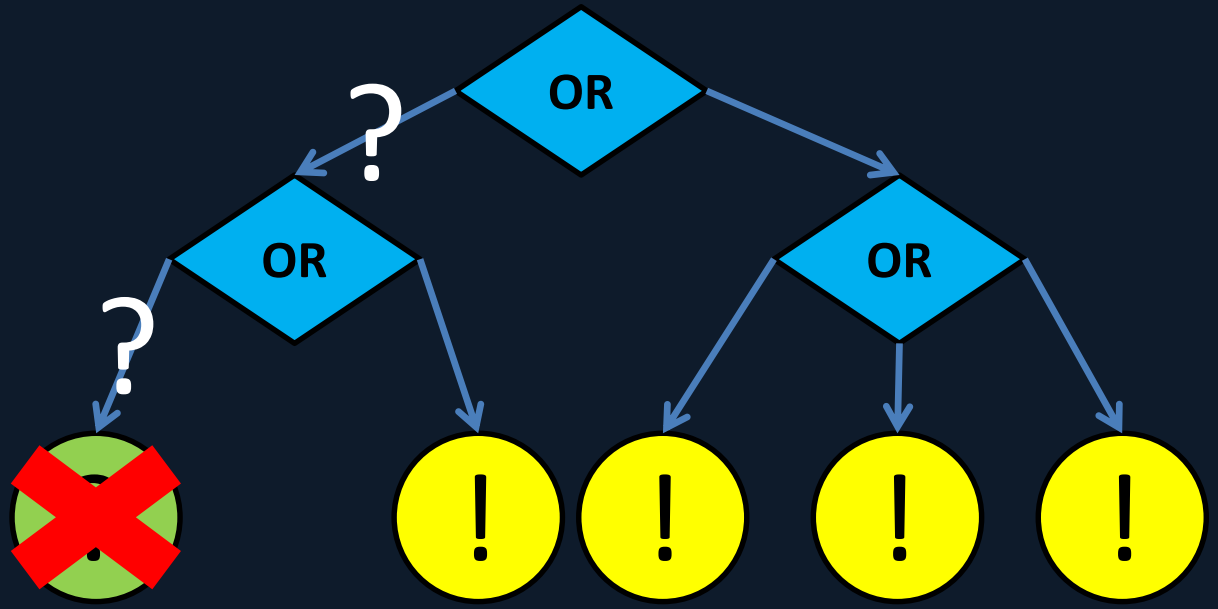
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

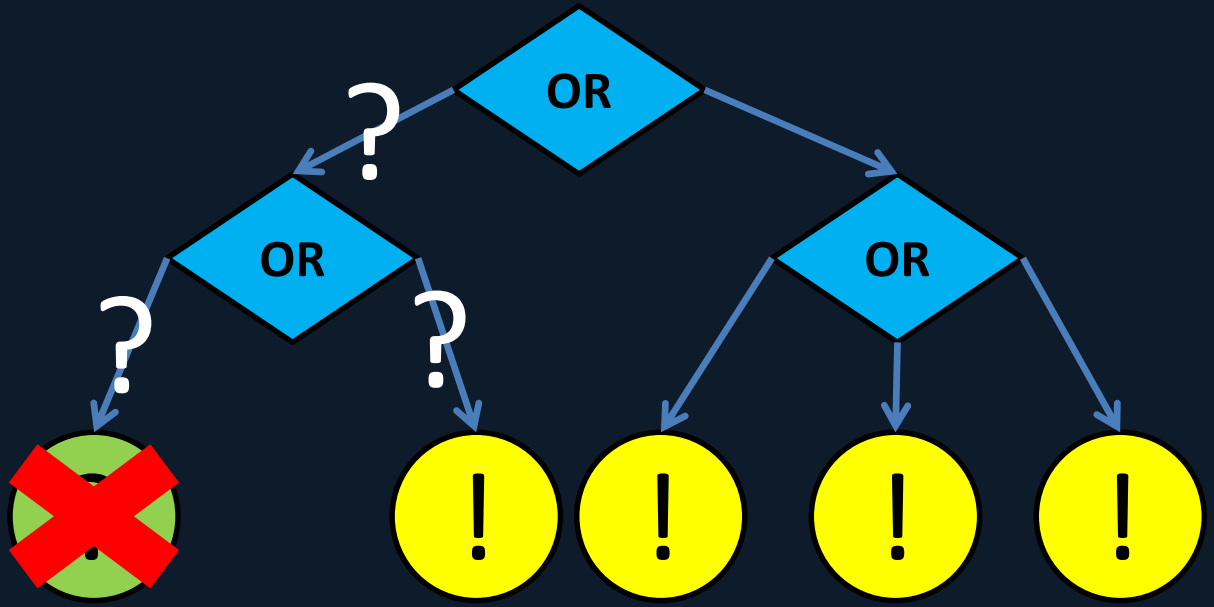
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

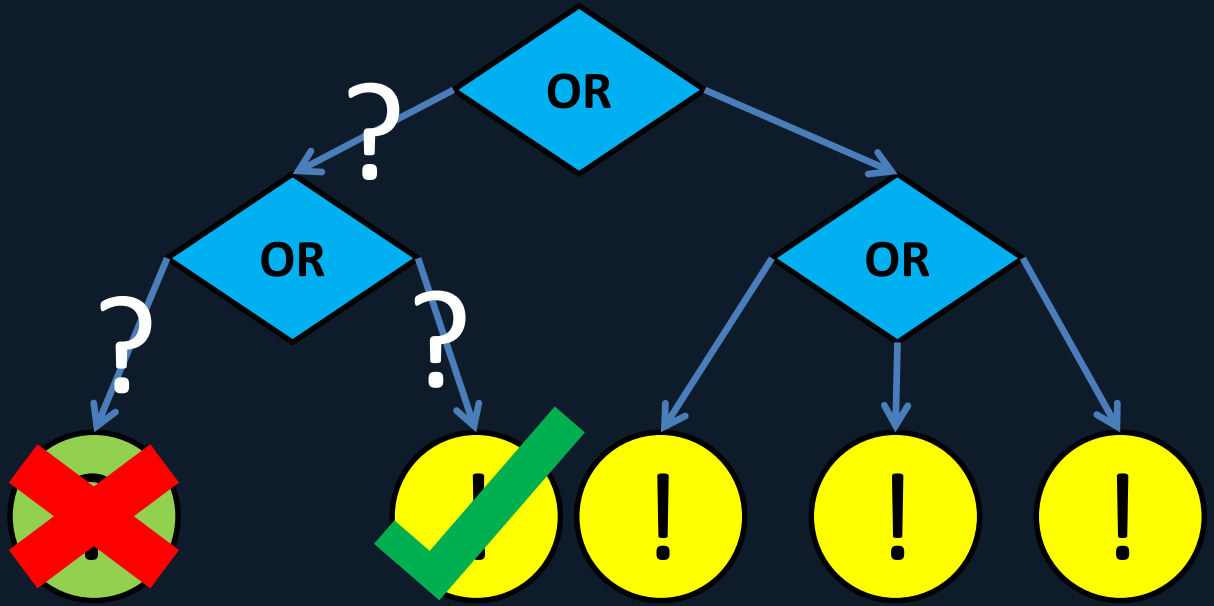
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

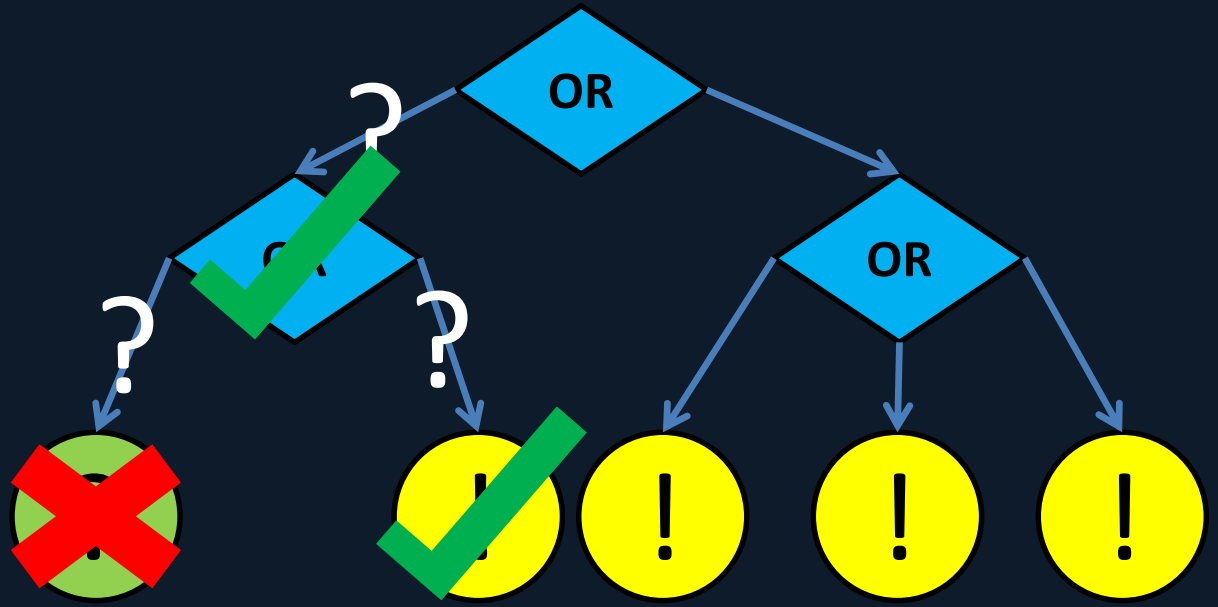
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

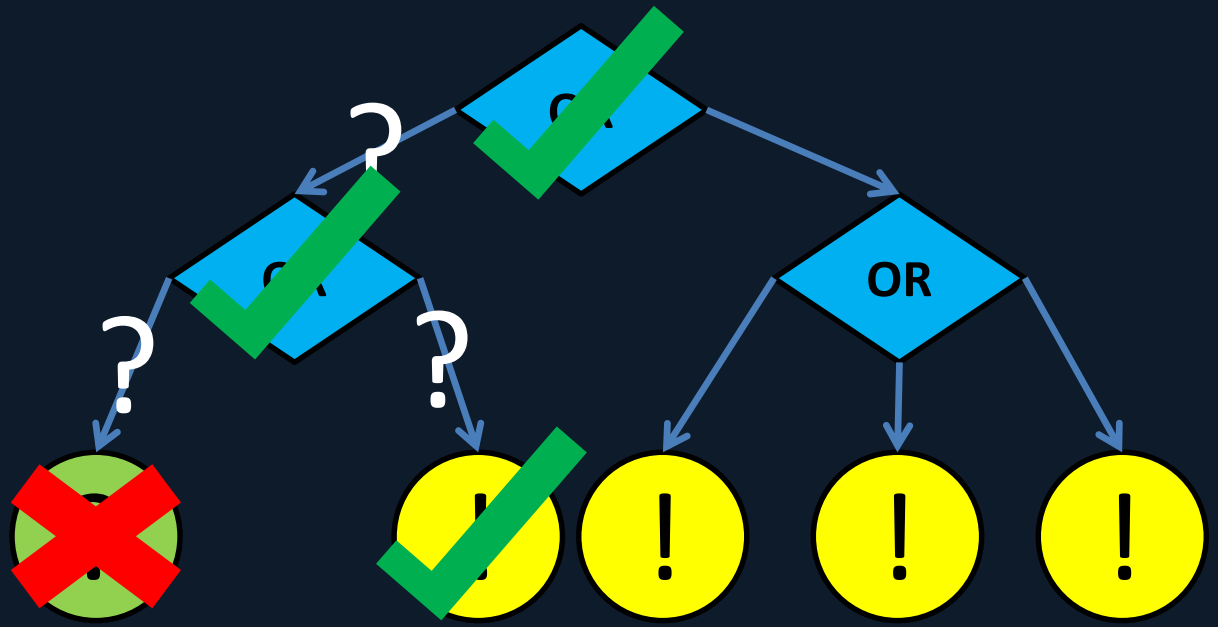
Selector Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Selector Composite Nodes

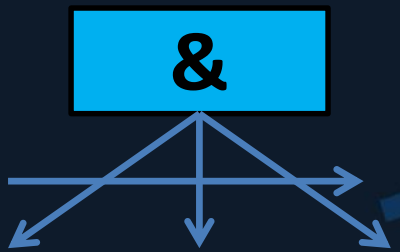


Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Sequence Composite Nodes

- A **Sequence** is a **Composite** node that returns Success if all of its child nodes returns Success
  - For example:
    - If a child returned Failure then it would return Failure and not execute the remaining child behaviours
    - If all child behaviours return Success then the Sequence returns Success
  - Typically executes in a set order, commonly denoted with an arrow across the branches

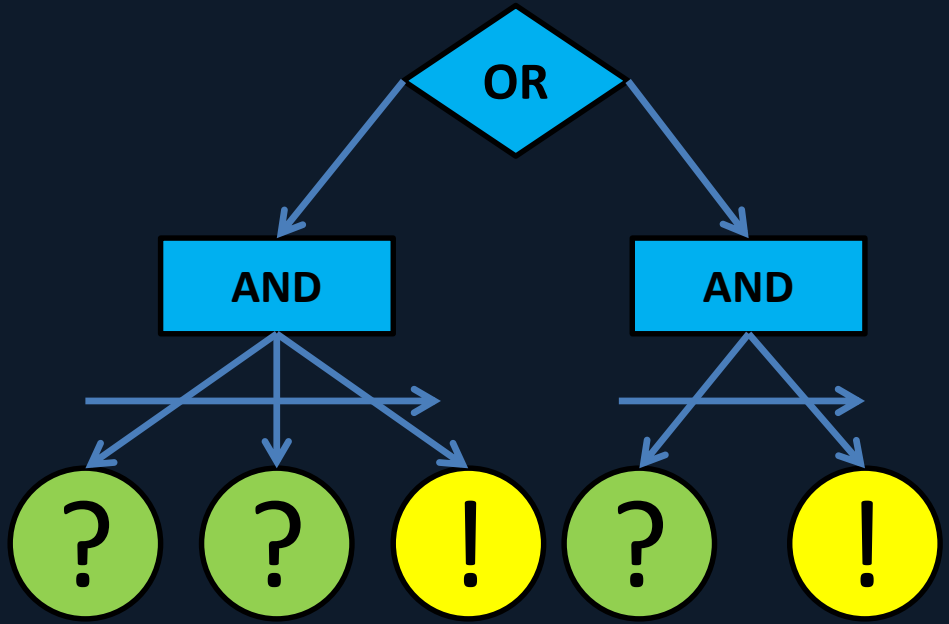


- A **Sequence** acts as an **AND**
  - Result = A **AND** B **AND** C **AND** etc

```
class Sequence : CompositeBehaviour

   func execute(Agent agent)
      for each child in childBehaviours
         if child.execute(agent) == BehaviourResult.Failure
            return BehaviourResult.Failure
      return BehaviourResult.Success
```

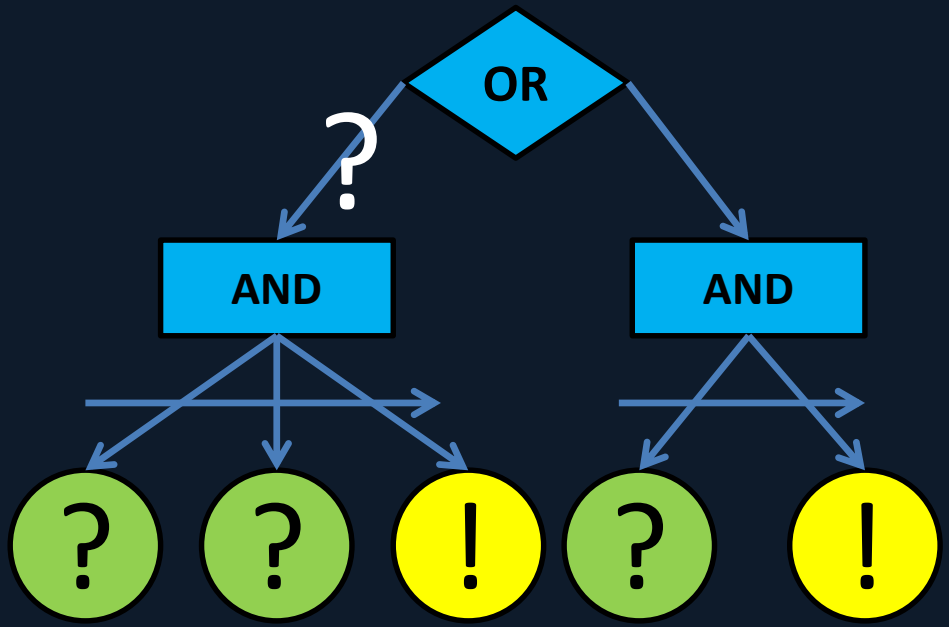
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

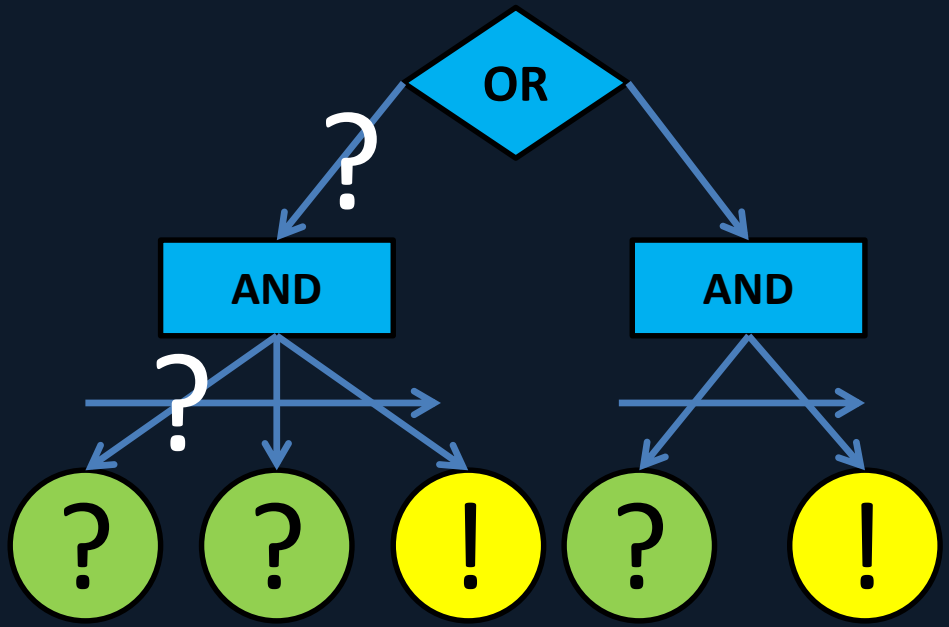
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

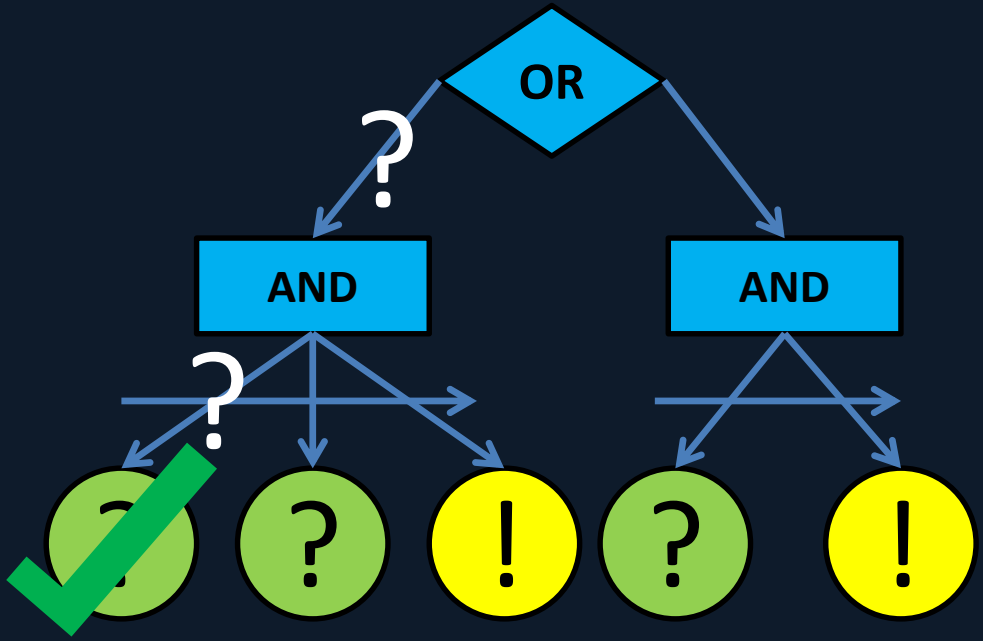
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

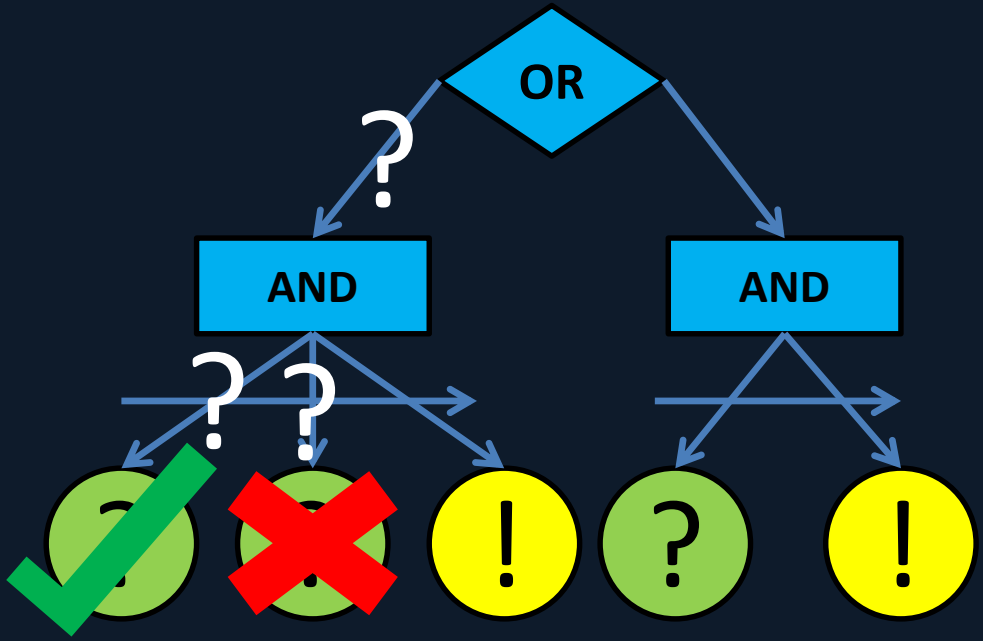
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

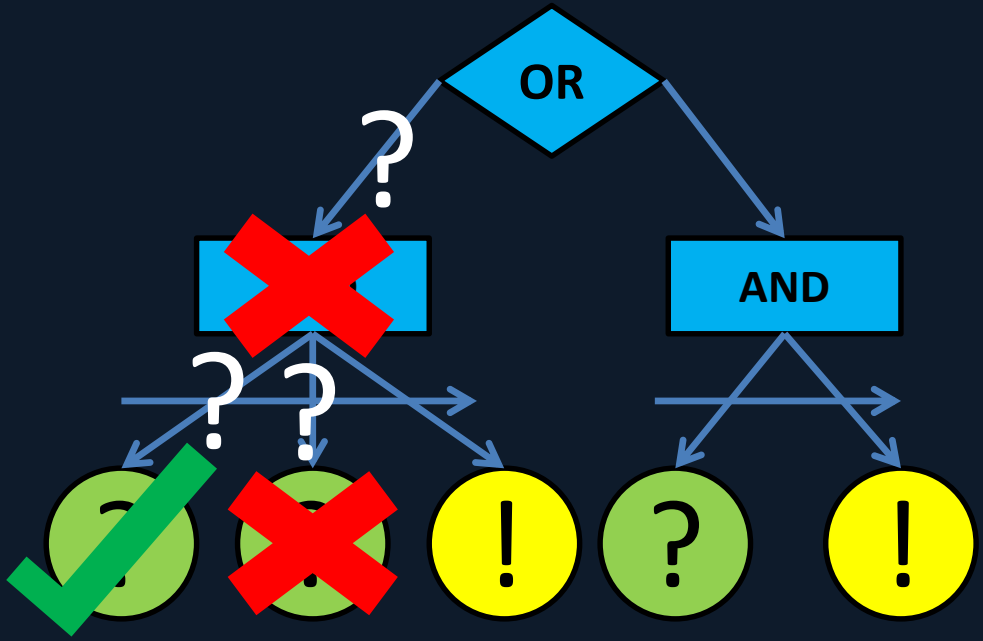
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

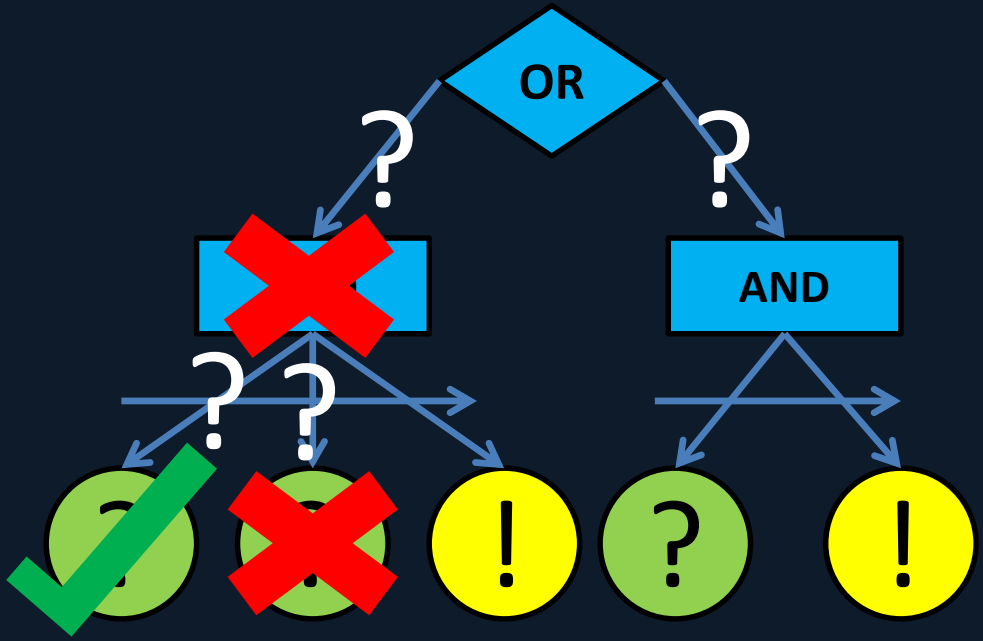
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

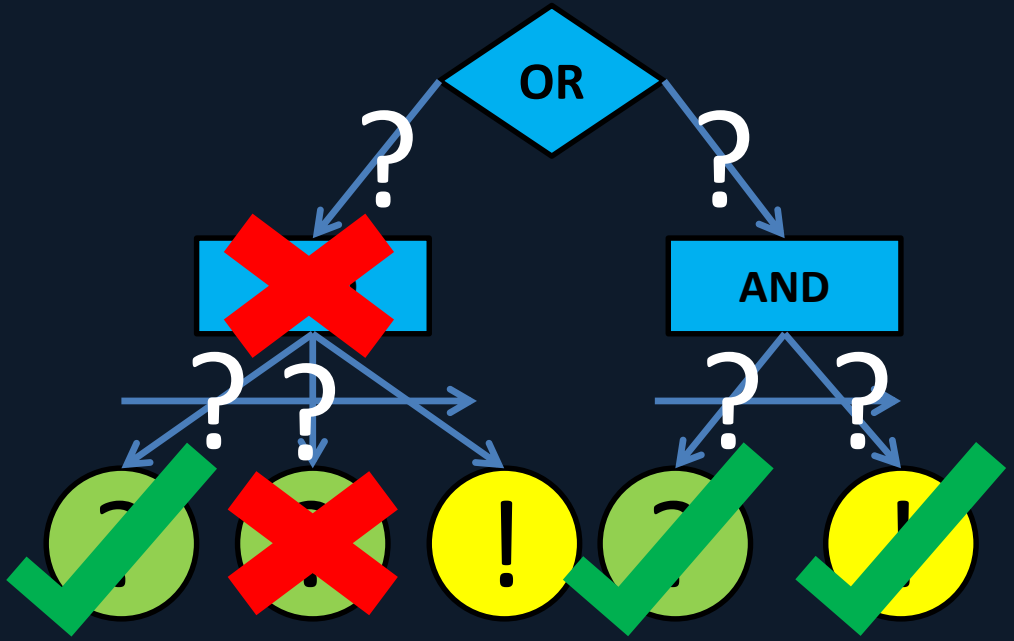
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

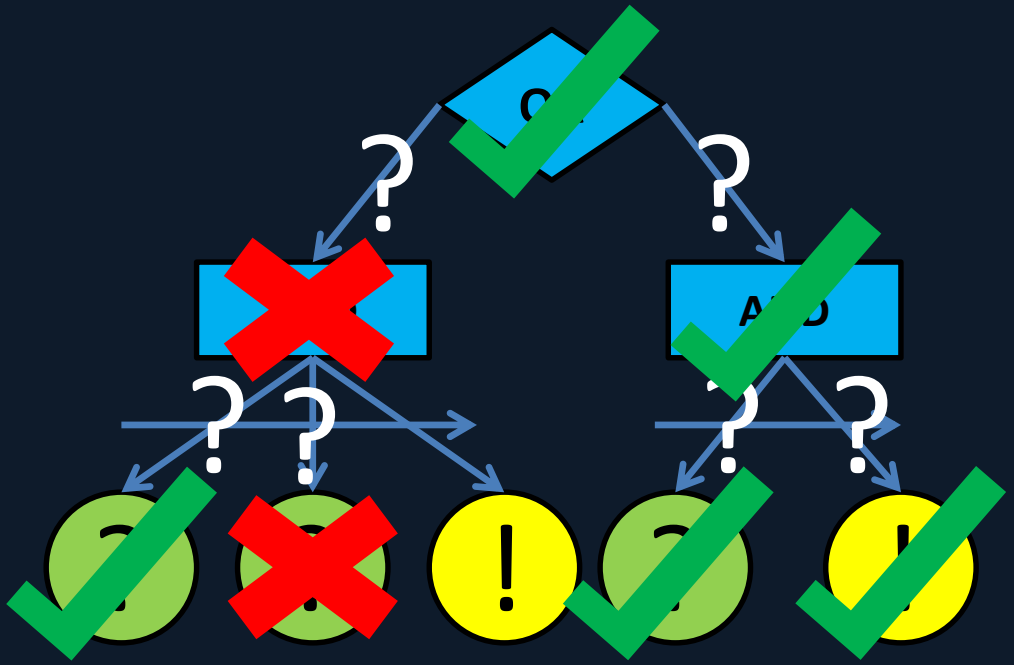
Sequence Composite Nodes



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Sequence Composite Nodes

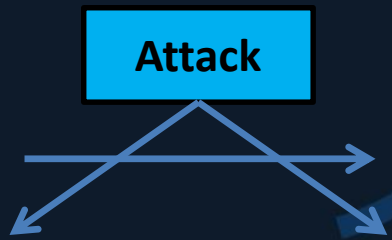


Canberra Institute of Technology

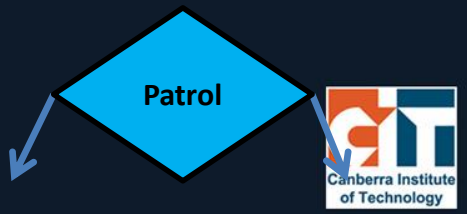
aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Sub-Behaviours and Reusability

- Branches in a Behaviour Tree typically represent sub-behaviours
  - Abstract idea
  - **Attack**, **Patrol** etc

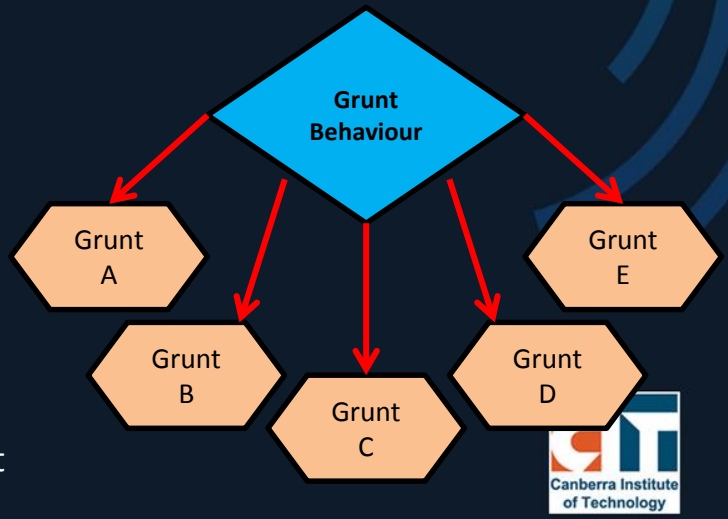


- Sub-behaviours can be designed and inserted in to multiple trees and reused
  - For example, a Patrol branch could be used in a **GruntSoldier's** Behaviour Tree and also a **BossSoldier's** Behaviour Tree



Sub-Behaviours and Reusability

- A Behaviour Tree can also be used by multiple agents
  - Only works if the variables within a behaviour don't change!
    - Changeable data should be stored within the agents themselves
    - For example, the amount of damage that an attack action causes can be within the behaviour or, if it depends on the weapon being carried, the value comes from the attacking agent and not the behaviour
    - Or another example, if an Attack action has a timeout then the timer should be a part of the agent, and the length of the timeout duration would be a part of the behaviour



Advantages

- Responds to interruptions
- Re-usable Behaviours
  - Sub-Trees of Behaviours (like a patrol tree) can be re-used within other trees (for example, in an **AngrySoldier** tree and a **ScaredGuard** tree)
- New Behaviours can be written as individual pieces of code without reliance on other behaviours
- Easily understood logic for designers
  - A tool could be written and given to a designer to build complex behaviours, with very little training required

Disadvantages

- For small behaviours can be a complex initial setup
  - A simple FSM might be better
- Difficult to setup without a tool
  - Coding a Behaviour Tree is fine, but a designer would usually be setting up Behaviour Trees for Agents
- Executing a large tree from the top can take a lot of processing

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- Behaviour Trees have become the primary tool for A.I. in AAA games
  - Easily understood logic for designers and coders alike

- Behaviours can be broken down into discrete actions, sub-behaviours and constructed into overall behaviours

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX