Read it: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

Basics

- A behavior tree is a tree of hierarchical nodes that control the flow of decision making of an AI entity.
- At the extents of the tree, the leaves, are the actual commands that control the AI entity, and forming the branches are various types of utility nodes that control the AI's walk down the trees to reach the sequences of commands best suited to the situation.

Data Driven vs Code Driven

- There are many different possible implementations of behavior trees.
- A main distinction is whether the trees are defined externally to the codebase, perhaps in XML or a proprietary format and manipulated with an external editor, or whether the structure of the trees is defined directly in code via nested class instances.

Tree Traversal

- A core aspect of Behavior Trees is that unlike a method within your codebase, a particular node or branch in the tree may take many ticks of the game to complete.
- In the basic implementation of behavior trees, the system will traverse down from the root of the tree every single frame, testing each node down the tree to see which is active, rechecking any nodes along the way, until it reaches the currently active node to tick it again.
- This isn't a very efficient way to do things, especially when the behavior tree gets deeper as its developed and expanded during development.
- I'd say it's a must that any behavior tree you implement should store any currently processing nodes so they can be ticked directly within the behavior tree engine rather than per tick traversal of the entire tree.

Flow

- A behavior tree is made up of several types of nodes, however some core functionality is common to any type of node in a behavior tree.
- This is that they can return one of three statuses (depending on the implementation of the behavior tree, there may be more than three return statuses). The three common statuses are as follows:
    1) Success
    2) Failure
    3) Running
- This functionality is key to the power of behavior trees since it allows a node's processing to persist for many ticks of the game.

- These statuses then propagate and define the flow of the tree, to provide a sequence of events and different execution paths down the tree to make sure the AI behaves as desired.
- There are three main archetypes of behavior tree node:
    1) Composite
    2) Decorator
    3) Leaf

Composite

- A composite node is a node that can have one or more children.
- They will process one or more of these children in either a first to last sequence or random order depending on the composite node in question, and at some stage will consider their processing complete and pass either success or failure to their parent, often determined by the success or failure of the child nodes.
- The most used composite node is the Sequence, which simply runs each child in sequence, returning failure at the point any of the children fail, and returning success if every child returned a successful status.

Decorator

- A decorator node, like a composite node, can have a child node. Unlike a composite node, they can specifically only have a single child.
- Their function is either to transform the result they receive from their child node's status, to terminate the child, or repeat processing of the child, depending on the type of decorator node.
- A commonly used example of a decorator is the Inverter, which will simply invert the result of the child. A child fails and it will return success to its parent, or a child succeeds, and it will return failure to the parent.

Leaf

- These are the lowest level node type and are incapable of having any children.
- Leaves are however the most powerful of node types, as these will be defined and implemented by your game to do the game specific, or character specific tests or actions required to make your tree do useful stuff.
- An example of this, would be Walk. A Walk leaf node would make a character walk to a specific point on the map and return success or failure depending on the result.

Composite Nodes

- Here we will talk about the most common composite nodes found within behavior trees.

Sequences

- A sequence will visit each child in order, starting with the first, and when that succeeds will call the second, and so on down the list of children.
- If any child fails it will immediately return failure to the parent. If the last child in the sequence succeeds, then the sequence will return success to its parent.
- The most obvious usage of sequences is to define a sequence of tasks that must be completed in entirety, and where failure of one means further processing of that sequence of tasks becomes redundant.
- If a character fails to walk to the door, the sequence returns failure at the moment the walk fails, and the parent of the sequence can then deal with the failure gracefully.

Selector

- Where a sequence is an AND, requiring all children to succeed to return a success, a selector will return a success if any of its children succeed and not process any further children.
- It will process the first child, and if it fails will process the second, and if that fails will process the third, until a success is reached, at which point it will instantly return success. It will fail if all children fail.
- This means a selector is analogous with an OR gate, and as a conditional statement can be used to check multiple conditions to see if any one of them is true.
- Their main power comes from their ability to represent multiple different courses of action, in order of priority from most favorable to least favorable, and to return success if it managed to succeed at any course of action.

Random Selectors / Sequences

- Random sequences/selectors work identically to their namesakes, except the actual order the child nodes are processed is determined randomly.
- These can be used to add more unpredictability to an AI character in cases where there isn't a clear preferable order of execution of possible courses of action.

Decorator Nodes

Inverter

- They will invert or negate the result of their child node. Success becomes failure, and failure becomes success.
- They are most often used in conditional tests.

Succeeder

- A succeeder will always return success, irrespective of what the child node returned.

- These are useful in cases where you want to process a branch of a tree where a failure is expected or anticipated, but you don't want to abandon processing of a sequence that branch sits on.

Repeater

- A repeater will reprocess its child node each time its child returns a result.
- These are often used at the very base of the tree, to make the tree to run continuously.
- Repeaters may optionally run their children a set number of times before returning to their parent.

Repeat Until Fail

- Like a repeater, these decorators will continue to reprocess their child.
- That is until the child finally returns a failure, at which point the repeater will return success to its parent.

Data Context

- When a behavior tree is called on an AI entity, a data context is also created which acts as a storage for arbitrary variables that are interpreted and altered by the nodes.
- Nodes will be able to read or write into variables to provide nodes processed later with contextual data and allow the behavior tree to act as a cohesive unit.

Defining Leaf Nodes

- To provide functionality to leaf nodes, to allow for game specific functionality to be added into behavior trees, most systems have two functions that will need to be implemented.
  - *init* – Called the first time a node is visited by its parent during its parents' execution.
  - *process* – This is called every tick of the behavior tree while the node is processing. If this function returns Success or Failure, then its processing will end, and the result passed to its parent. If it returns Running it will be reprocessed next tick, and again and again until it returns a Success or Failure.
- Nodes can have properties associated with them, that may be explicitly passed literal parameters, or references to variables within the data context of the AI entity being controlled.

Stacks

- It is possible to implement stack operations as nodes in behavior trees.