

Behavior Tree in Unreal Engine 4

`jaewan.huey.Park@gmail.com`

Contents

- What and Why?
- Theory
- Practice
- Q & A
- Reference

What?

What?

- Commonly used way to direct behaviors for AI in a game.
- They can be used for any decision-making(in systems that aren't just AI)

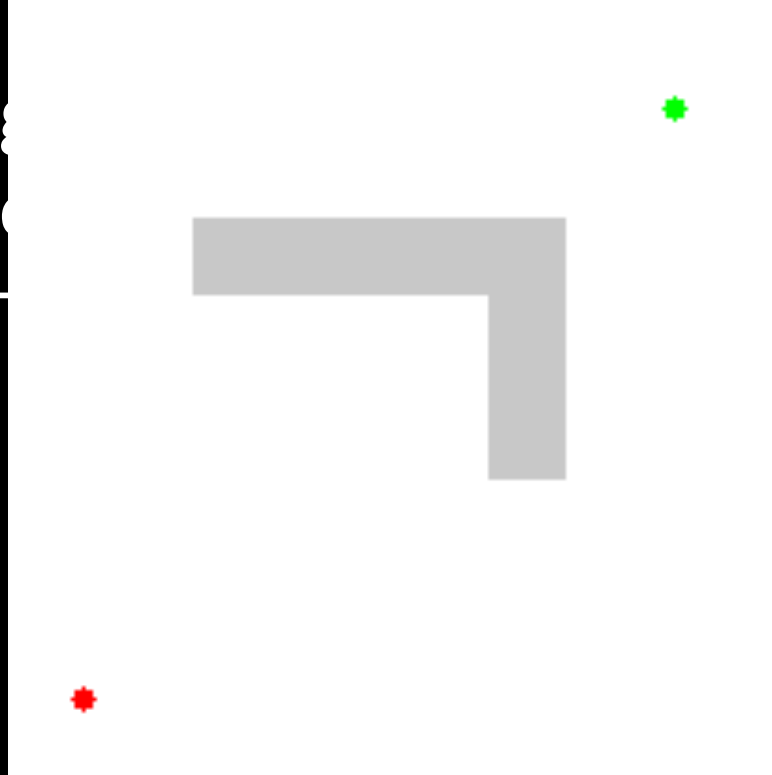
Why?

Why?

- Offer a good balance of supporting goal-oriented(*like a**) behaviors and reactivity(*like flocking*).

Why?

- Offer a goal-oriented reactivity



supporting
behaviors and

Why

- C
m
r



g
and

Why not other solutions?

Why not use other solutions?

- There are innumerable ways to implement AI or other decision making systems.
- Not necessarily always best, but they are frequently great for games. (LOL, Uncharted 2, ...)

Why not use other solutions?

State Machine?

- While state machines are reasonably intuitive for simple cases, as they become more complex they are hard to keep goal-oriented. As the number of states increases, the transitions between states become exponentially complex. Hierarchical state machines help a little, but many of the same issues remain.

10 Reasons the Age of Finite State Machines is Over

#1 They're Unorthodox

Problem : Building a FSM is a very different process from any other form of software engineering. Sure, the concept is “designer friendly” but a surprisingly small amount of mainstream programming knowledge applies to FSMs.

#1 They're Unorthodox

Reason : FSMs require each state to be wired with an **explicit transition** to the next state. No programming language requires this; everything is done implicitly on the **semantics** of the language itself.

#2 They're Low-Level

Problem : The process of editing the logic of a FSM is **very low-level** and **quite mechanical**. You often find rebuilding the similar behaviors over and over from scratch - which **takes a lot of time**.

#2 They're Low-Level

Reason : All you can do is edit transitions from a state to another. There's no way to capture higher-level patterns that reoccur frequently like sequences or conditionals. Meta-programming doesn't exist in the world of finite state machines.

#3 Their Logic is Limited

Problem : Finite state machines, as they are defined formally, are computationally limited. This means you can't do things like counting by default.

#3 Their Logic is Limited

Reason : If you consider events as symbols, you can only recognize regular grammars with a finite state automaton. Likewise, finite state machines can only act as transducers for regular language.

#4 They Require Custom Extensions

Problem : Game developers often use extensions to make FSMs useful in practice. However, these hacks aren't always easy to understand and aren't so well documented either – unlike the academic foundations of FSMs.

#4 They Require Custom Extensions

Reason : Because FSMs are **theoretically limited**, developers must add functionality externally to implement certain features. This means leveraging the underlying programming language to implement things like counters, timers, or any form of memory.

#5 They Are Hard to Standardize

Problem : Unlike planners (HTN) or search algorithms (A*) which are implemented in relatively common ways, FSMs are very difficult to reuse across multiple games or in different parts of the engine.

#5 They Are Hard to Standardize

Reason : Since FSMs aren't turing complete, FSMs need customizing to be able to deal with the special cases of each problem. This makes them very narrowly applicable, unlike scripting languages which can easily be repackaged.

#6 They Are Not Deliberative

Problem : It takes a lot of work to use a FSMs to create goal-directed behaviors. This is an issue as most purposeful AI will require dealing with long-term goals.

#6 They Are Not Deliberative

Reason : FSMs operate in a reactive mode, only dealing with events and triggering transitions. They are not capable of searching ahead, so you have to edit the transitions for dealing with all the different goals manually.

#7 They Have Concurrency Nightmares

Problem : FSMs just don't like concurrency. When running multiple state machines in parallel, you either end up with deadlocks or you have to edit them all in a way they are compatible.

#7 They Have Concurrency Nightmares

Reason : FSMs have as much trouble dealing with external resource conflicts as they do storing information. Solutions to make state machines concurrent are typically external to the FSM itself.

#8 They Scale Poorly

Problem : Finite state machines, even hierarchical ones, don't scale very well. They often end up being edited as a large block of logic, instead of behaviors edited modularly.

#8 They Scale Poorly

Reason : FSMs are not built with the many mechanisms of programming languages that help them scale up to solve large problems, in particular indirection. Also, FSMs provide no easy way to synchronize multiple modular behaviors together.

#9 They Are Labor Intensive

Problem : It takes a lot of work to wire up a FSM to implement any design. Certain problems occur only because of the state machine itself!

#9 They Are Labor Intensive

Reason : Dealing with all the challenges mentioned previously takes a lot of time by the designers, and this ultimately becomes a source of bugs in the behaviors.

#10 Industry is Moving On

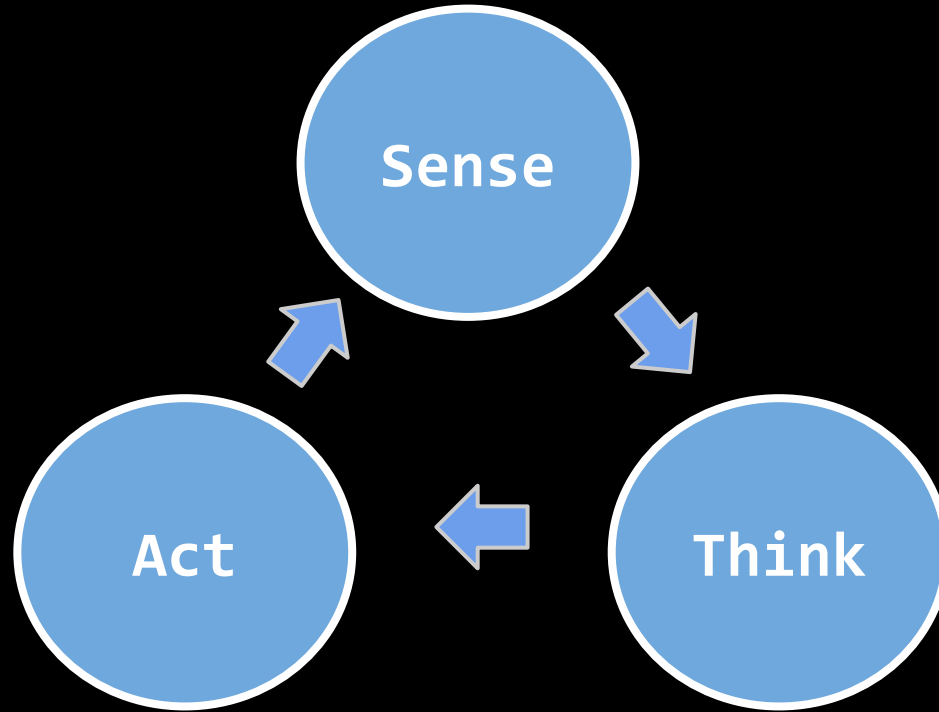
Fact : Experienced game developers are using finite state machines less and less, switching to alternatives like behavior trees.

#10 Industry is Moving On

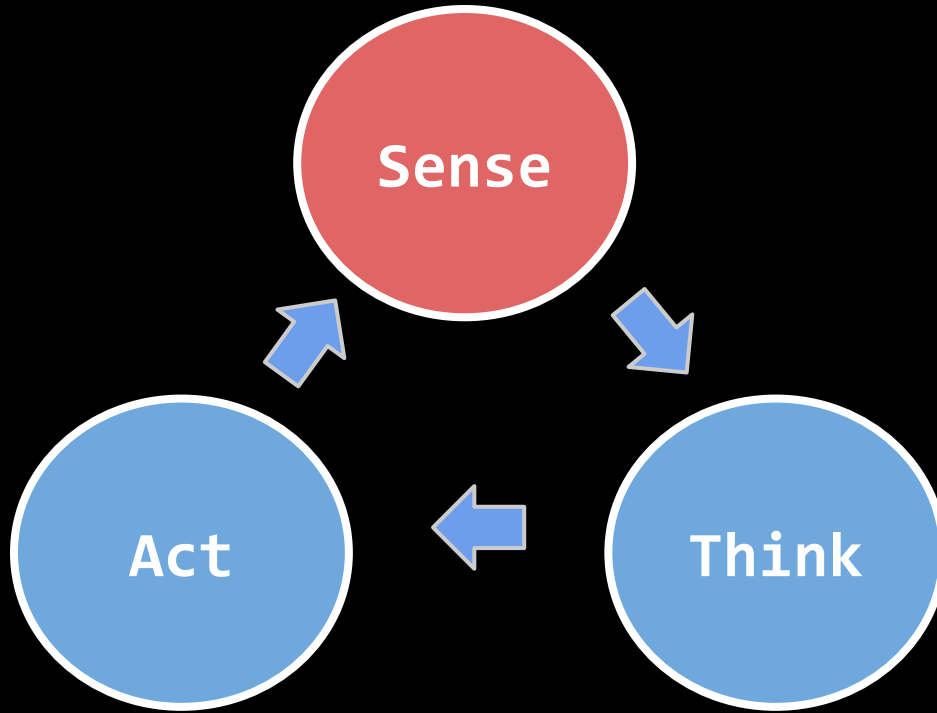
Fact : Middleware vendors for game AI are focusing their development efforts mainly on planners, and 2008 should see more of these becoming available off the shelf.

Theory

Theory

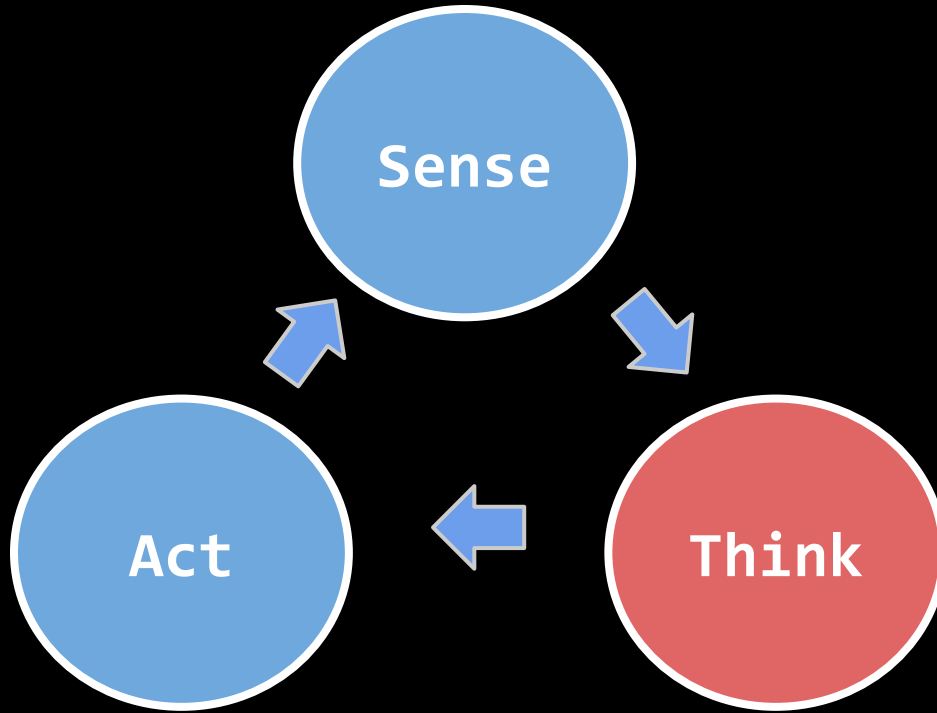


Theory



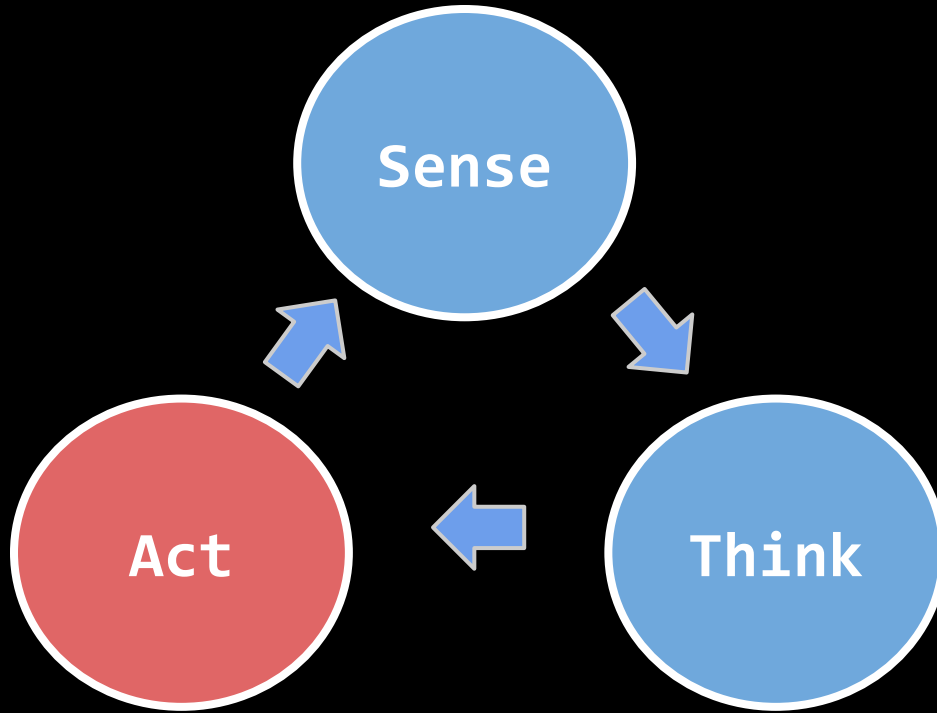
- Generally rely on physics engine
- Usually very expensive
- Use infrequently

Theory



- Decision Logic
- Generally quite simple
- Design intensive

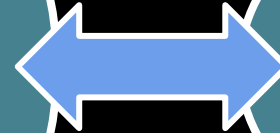
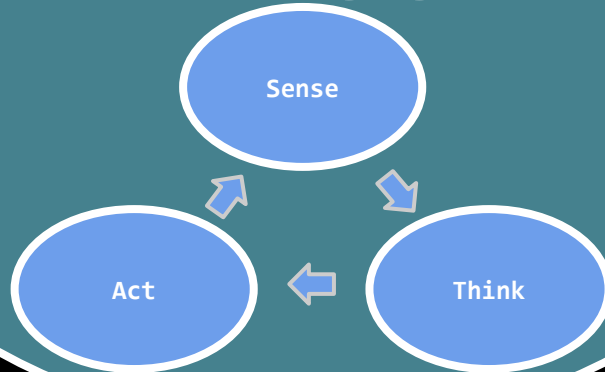
Theory



- Action execution
- Often long running
- Can fail to complete

Theory

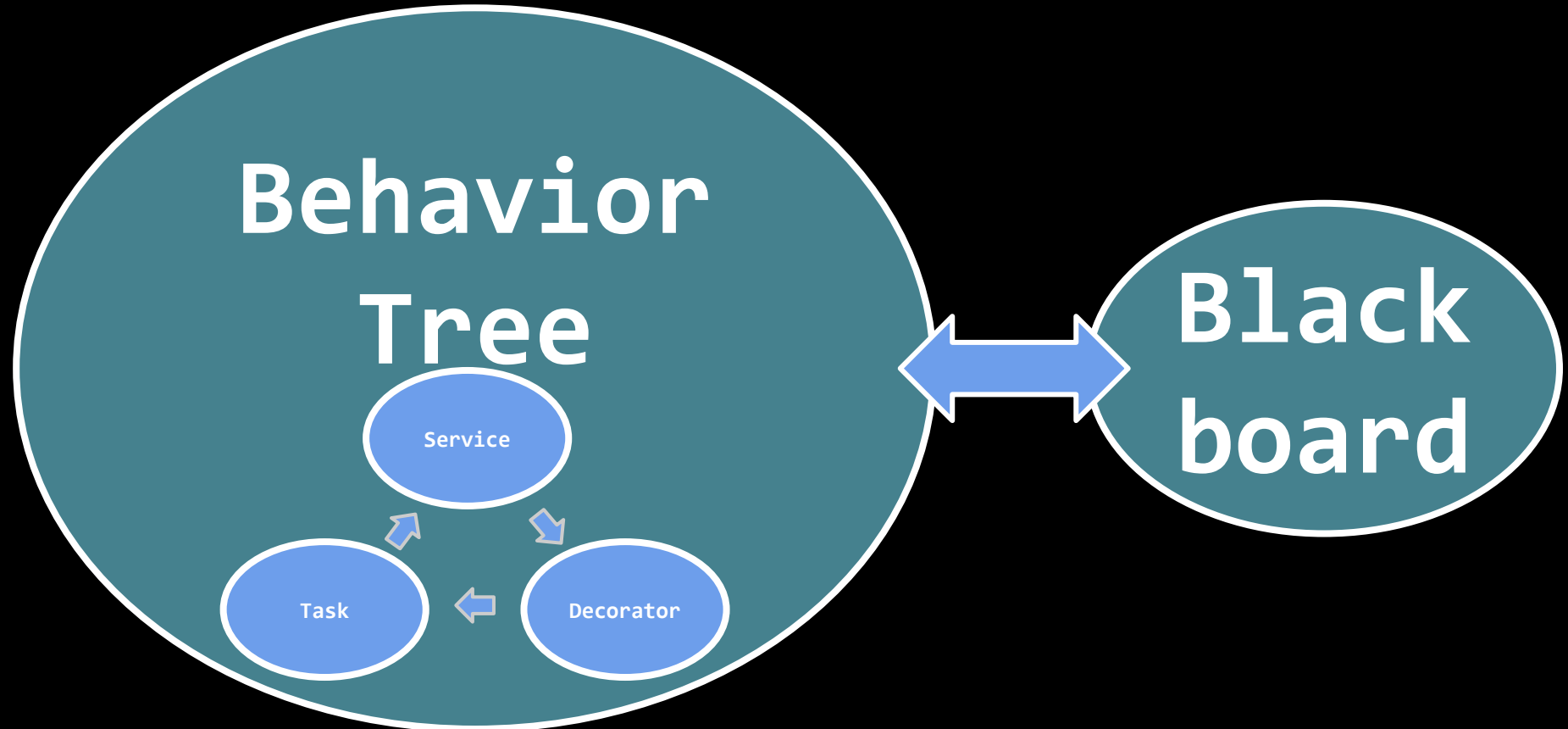
Behavior Tree



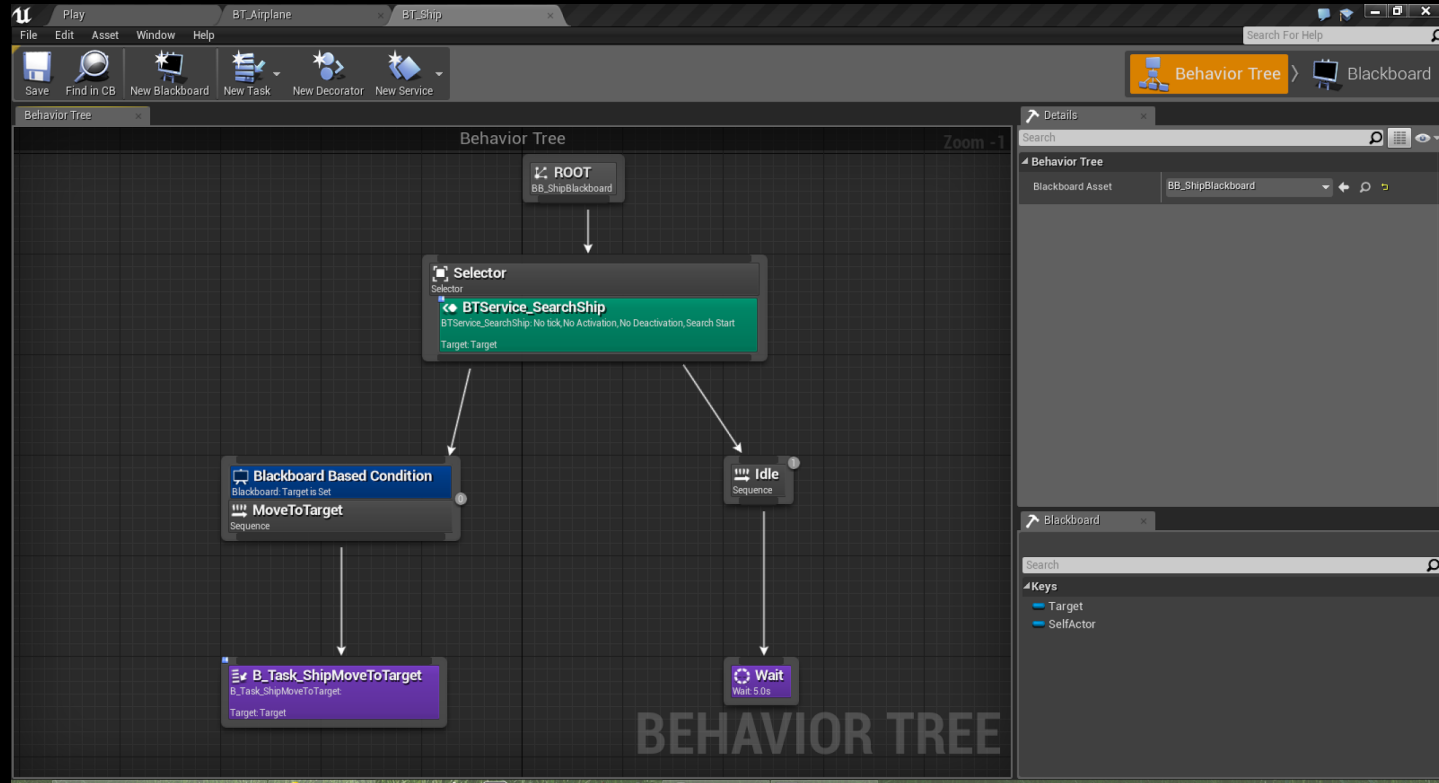
Memory

Unreal Engine 4 Behavior Tree

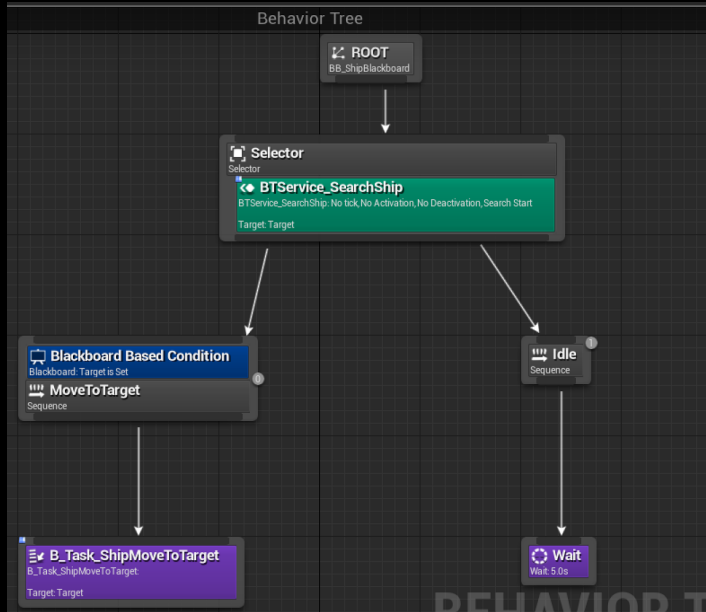
Unreal Engine 4 Behavior Tree



Unreal Engine 4 Behavior Tree



Unreal Engine 4 Behavior Tree



Root
Composite
Service
Decorator
Task

Unreal Engine 4 Behavior Tree

Root

- The starting execution node for the Behavior Tree.
- Every Behavior Tree has one.
- You cannot attach Decorators or Services to it.

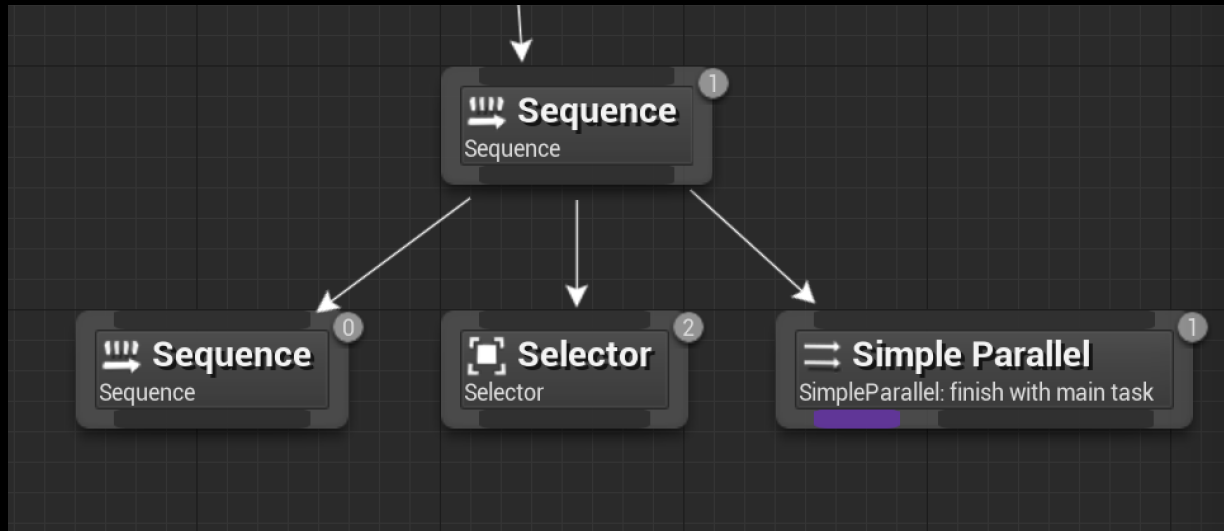
Unreal Engine 4 Behavior Tree

Composite

- These are nodes that define the root of a branch and define the base rules for how that branch is executed.
- Sequence, Selector, Simple Parallel

Unreal Engine 4 Behavior Tree

Composite : Sequence



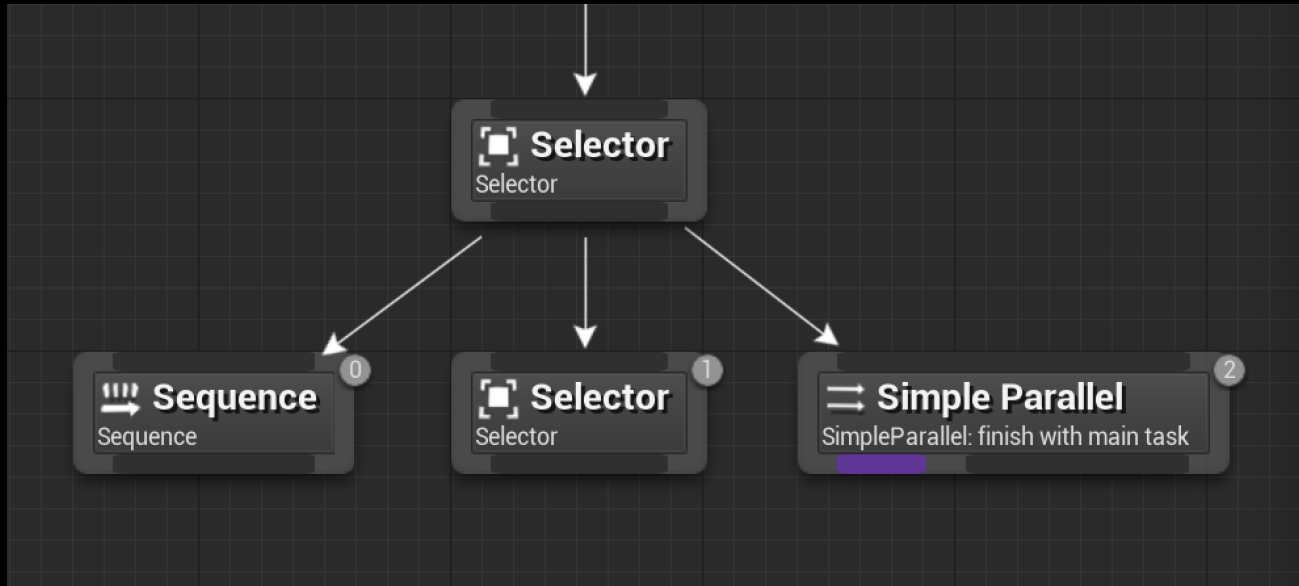
Unreal Engine 4 Behavior Tree

Composite : Sequence

- Sequence Node execute their children from left to right, and will stop executing its children when one of their children **Fails**. If a child fails, then the Sequence fails.

Unreal Engine 4 Behavior Tree

Composite : Selector



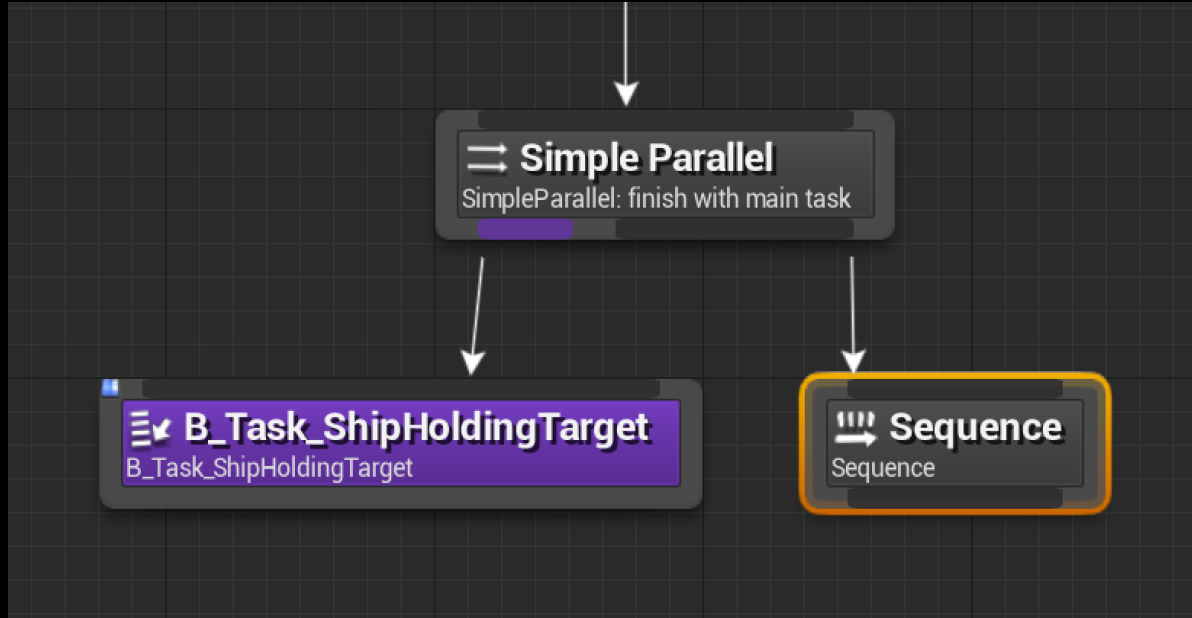
Unreal Engine 4 Behavior Tree

Composite : Selector

- Selector Nodes execute their children from left to right, and will stop executing its children when one of their children **Succeeds**. If a Selector's child succeed, the Selector succeeds.

Unreal Engine 4 Behavior Tree

Composite : Simple Parallel



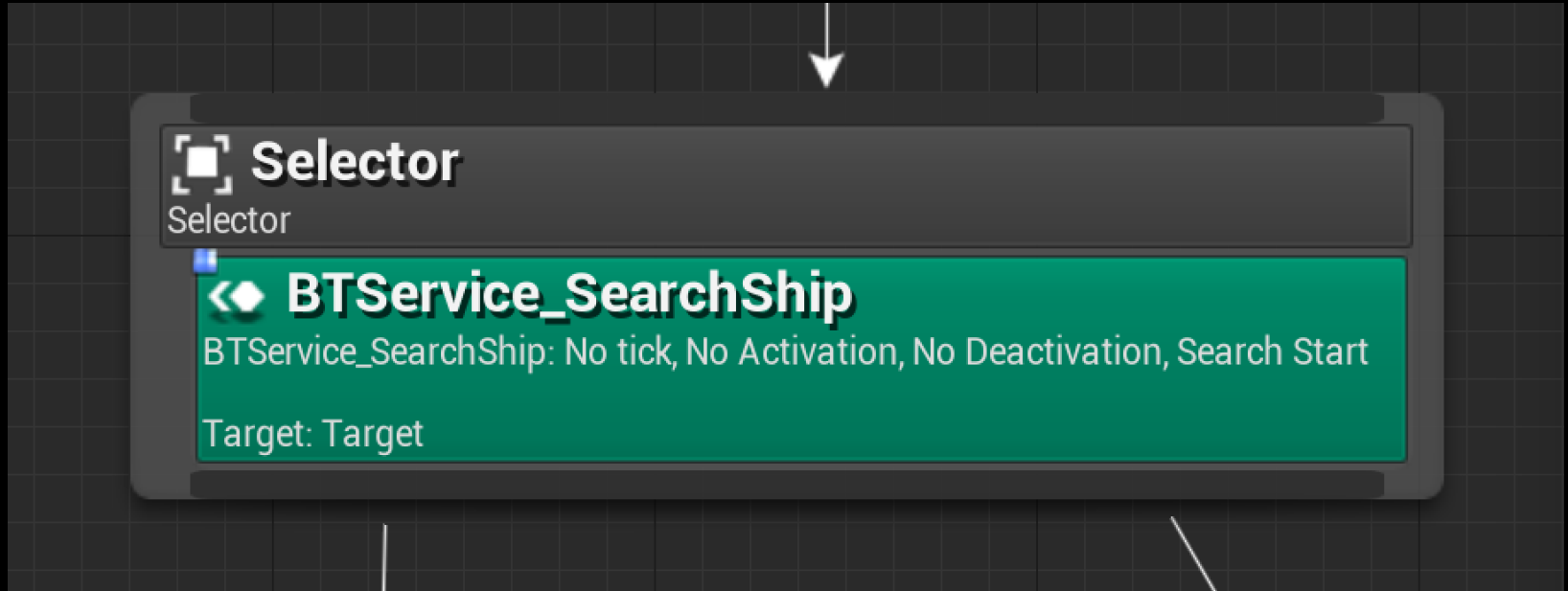
Unreal Engine 4 Behavior Tree

Composite : Simple Parallel

- The Simple Parallel node allows a single main task node to be executed along side of a full tree. When the main task finishes, the setting in **Finish Mode** dictates the secondary tree.

Unreal Engine 4 Behavior Tree

Service



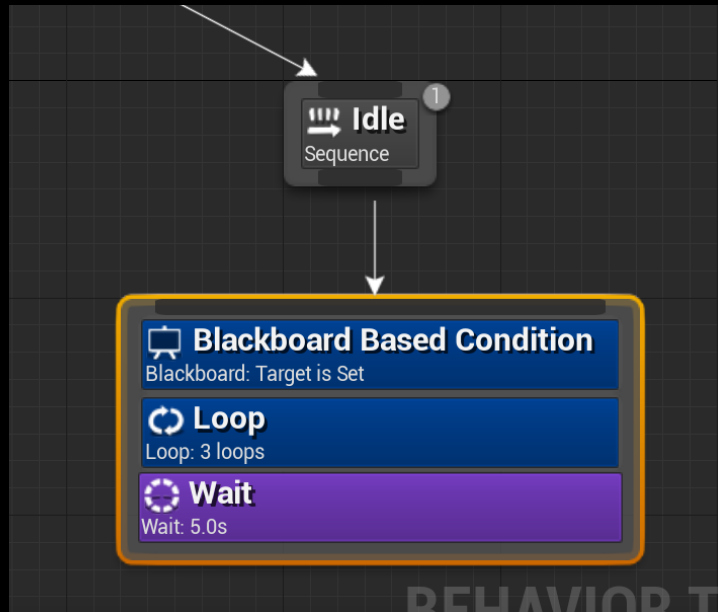
Unreal Engine 4 Behavior Tree

Service

- These attach to Composite nodes, and will execute at their defined frequency. These are often used to make checks and to update the Blackboard. These take the place of traditional Parallel nodes.

Unreal Engine 4 Behavior Tree

Decorator



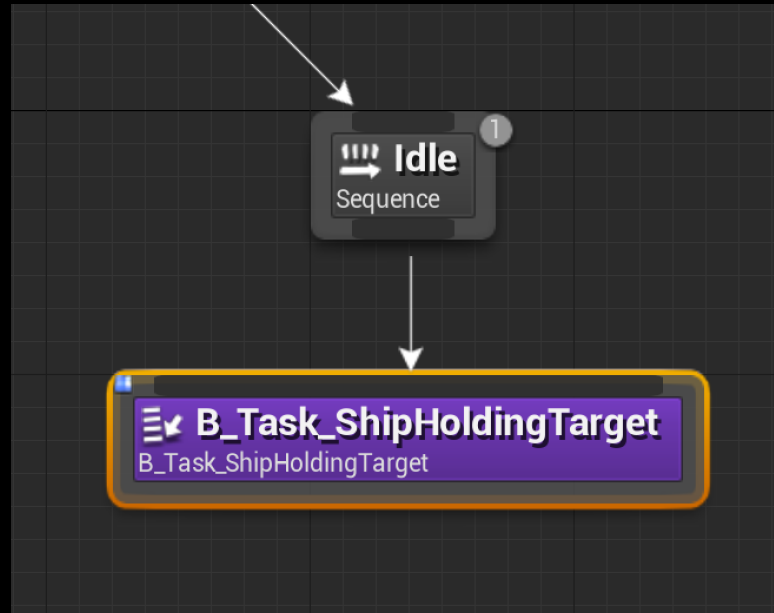
Unreal Engine 4 Behavior Tree

Decorator

- Also known as conditionals. These attach to another node and make decisions on whether or not a branch in the tree, or even single node, can be executed.

Unreal Engine 4 Behavior Tree

Task



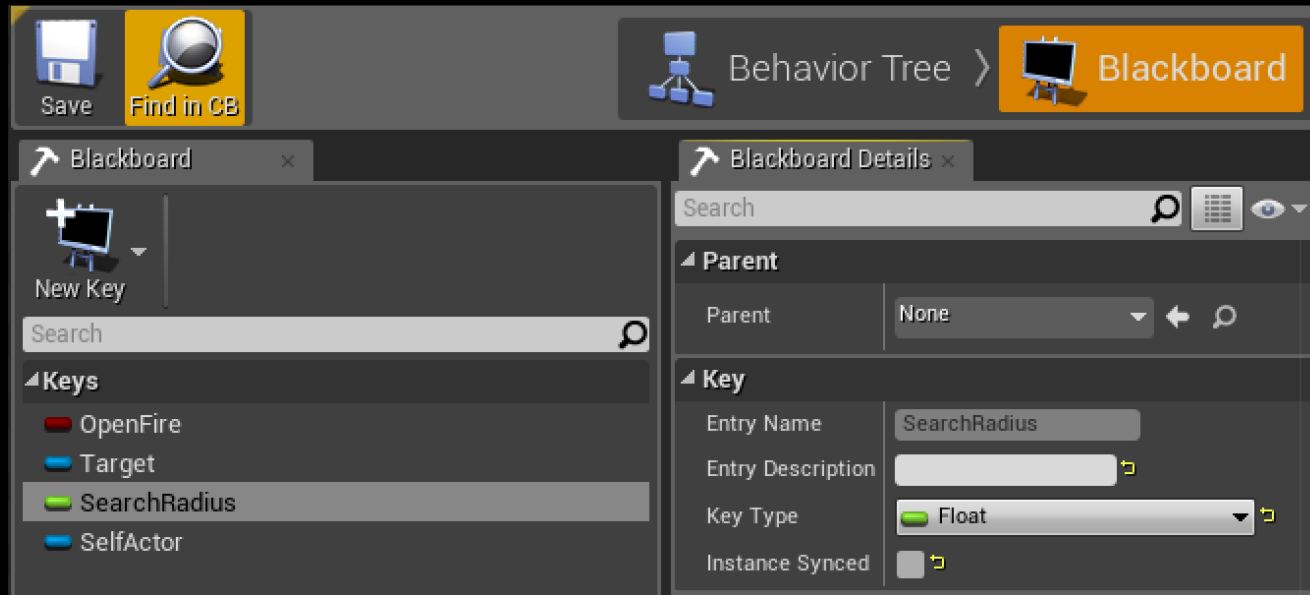
Unreal Engine 4 Behavior Tree

Task

- There are leaves of the tree, the nodes that “do” things.

Unreal Engine 4 Behavior Tree

Blackboard



Unreal Engine 4 Behavior Tree

Blackboard

- A blackboard is a simple place where data can be written and read for decision making purposes.
- A blackboard can be used by a single AI pawn, shared by squad.

Unreal Engine 4 Behavior Tree

Blackboard : Why use?

- To make efficient event-driven behaviors
- To cache calculations
- As a scratch-pad for behaviors
- To centralize data

Unreal Engine 4 Behavior Tree

Blackboard : Notice

- Don't clutter the blackboard with lots of `super-specific-case data`.
- If `only one node needs` to know something, it can potentially fetch the value itself rather than adding every bit of tree.

Unreal Engine 4 Behavior Tree

Blackboard : Notice

- If you fail to copy data to the blackboard properly, it may cause you some debugging nightmare!
- If very frequently updated and copied to the blackboard, that could be bad for performance.

Practice



Airplane Dogfighting

State

- When I track
- When I get tracking
- When I can't find target
- When I suddenly encountered an obstacle

Airplane Dogfighting

Task

- Move to target
- Evasion maneuver
- Cobra maneuver
- Emergency evasion
- Throttle control

Airplane Dogfighting

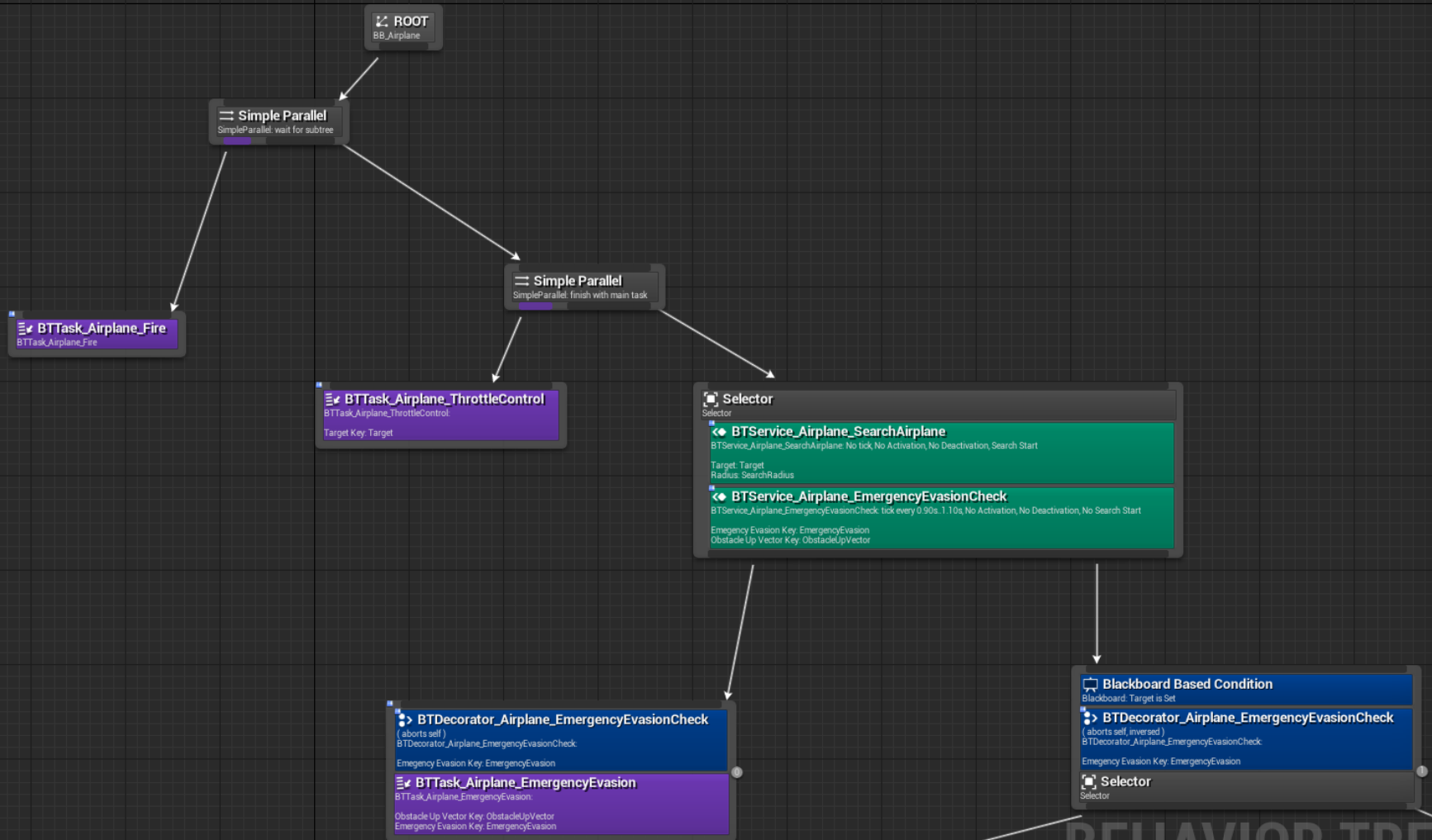
Service

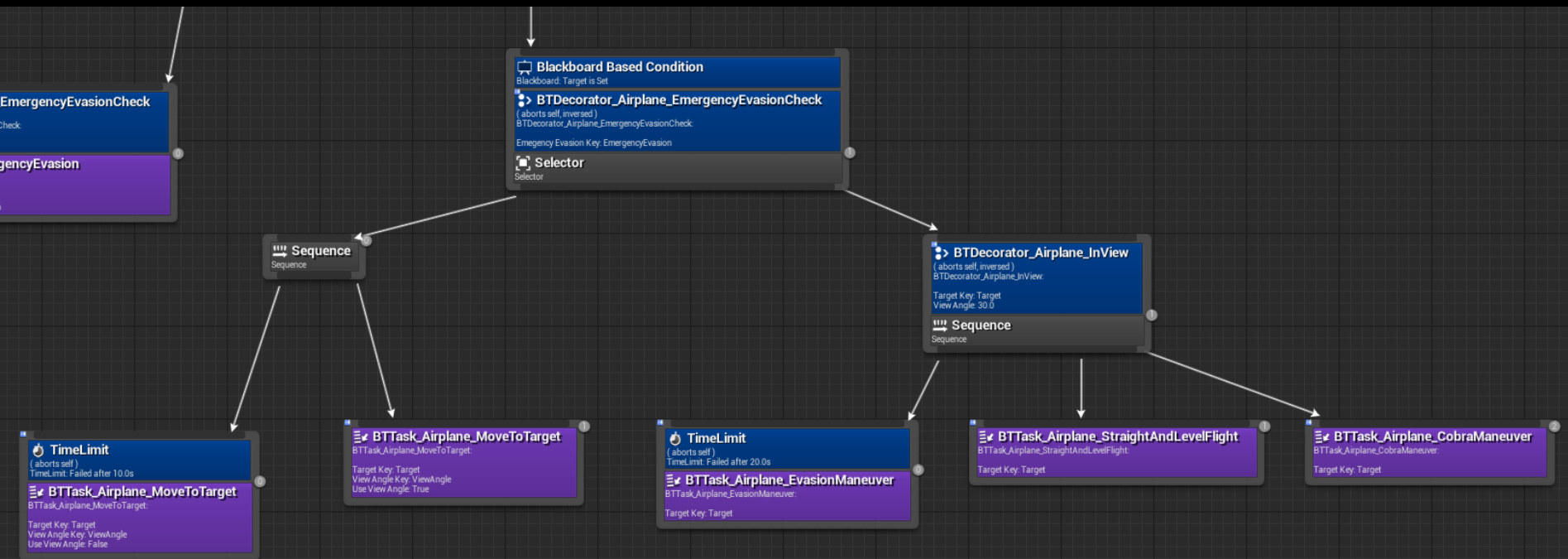
- Search airplane
- Emergency evasion check

Airplane Dogfighting

Decorator

- Blackboard Based Condition
- Time Limit
- Emergency Evasion Check
- In View





Q & A

References

- [AiGameDev.com](#)
- [Behavior Trees What and Why : Unreal Engine Forum](#)
- [10 Reasons the Age of Finite State Machines is Over](#)
- [Blackboard Documentation : Unreal Engine Forum](#)
- [zoombapup : AI Tutorial youtube playlist](#)
- [길안에서 묻다 : 네이버 블로그](#)