Unreal Engine 4 Behavior Tree

jaewan.huey.Park@gmail.com

Contents

What and Why?

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)

What and Why?

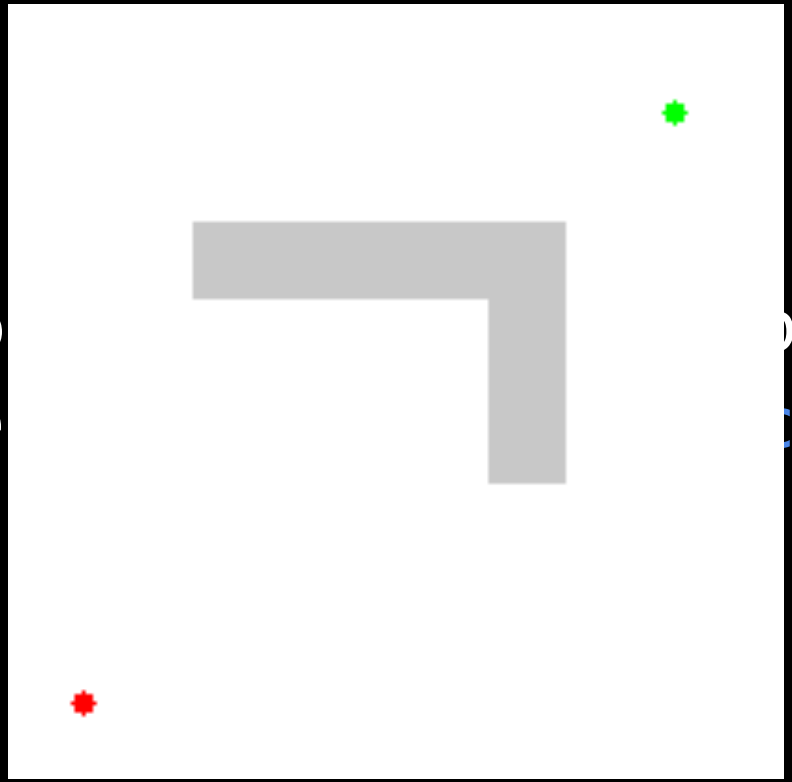
Why?

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

What and Why?

Why?

- Offer supporting goal-o behaviors and re king).



Wha

Why

- ing
  s

What and Why?

Why not use 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

- 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

10 Reasons the Age of Finite State Machines is Over

#2 They’re Low-Level

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

10 Reasons the Age of Finite State Machines is Over

#3 Their Logic is Limited

- Finite state machines, as they are defined formally, are computationally limited (a.k.a. Turing incomplete). This means you can’t do things like counting by default.

10 Reasons the Age of Finite State Machines is Over

#4 They Require Custom Extensions

- 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

10 Reasons the Age of Finite State Machines is Over

#5 They Are Hard to Standardize

- 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.

10 Reasons the Age of Finite State Machines is Over

#6 They Are Not Deliberative

- 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.

10 Reasons the Age of Finite State Machines is Over

#7 They Have Concurrency Nightmares

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

10 Reasons the Age of Finite State Machines is Over

#8 They Scale Poorly

- 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.

10 Reasons the Age of Finite State Machines is Over

#9 They Are Labor Intensive

- 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!

10 Reasons the Age of Finite State Machines is Over

#10 Industry is Moving On

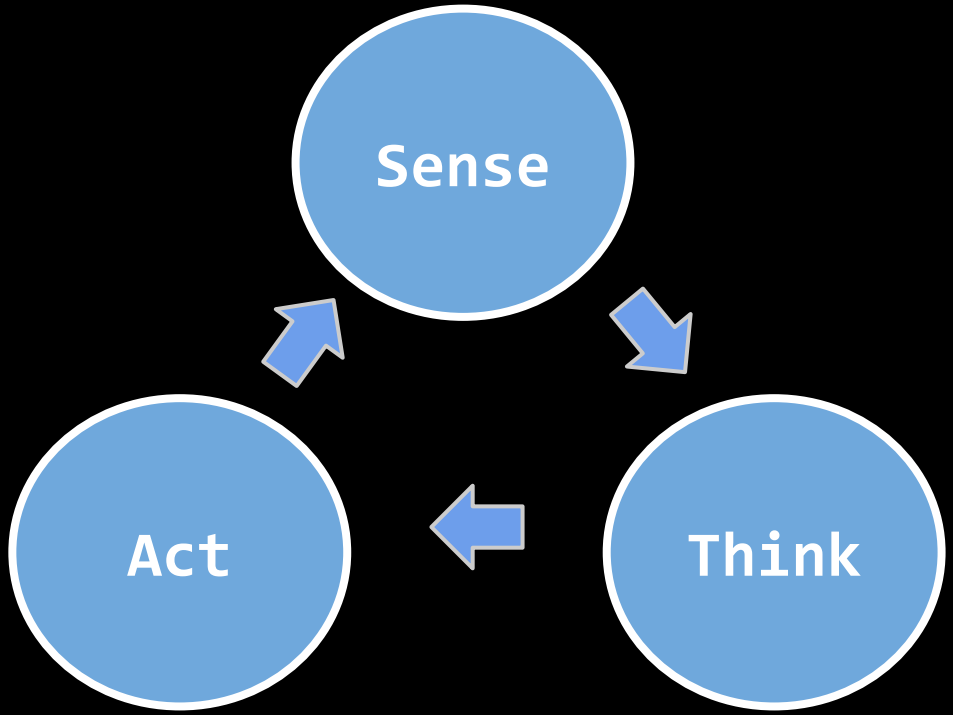
- Experienced game developers are using finite state machines less and less, switching to alternatives like behavior trees.

What and Why?

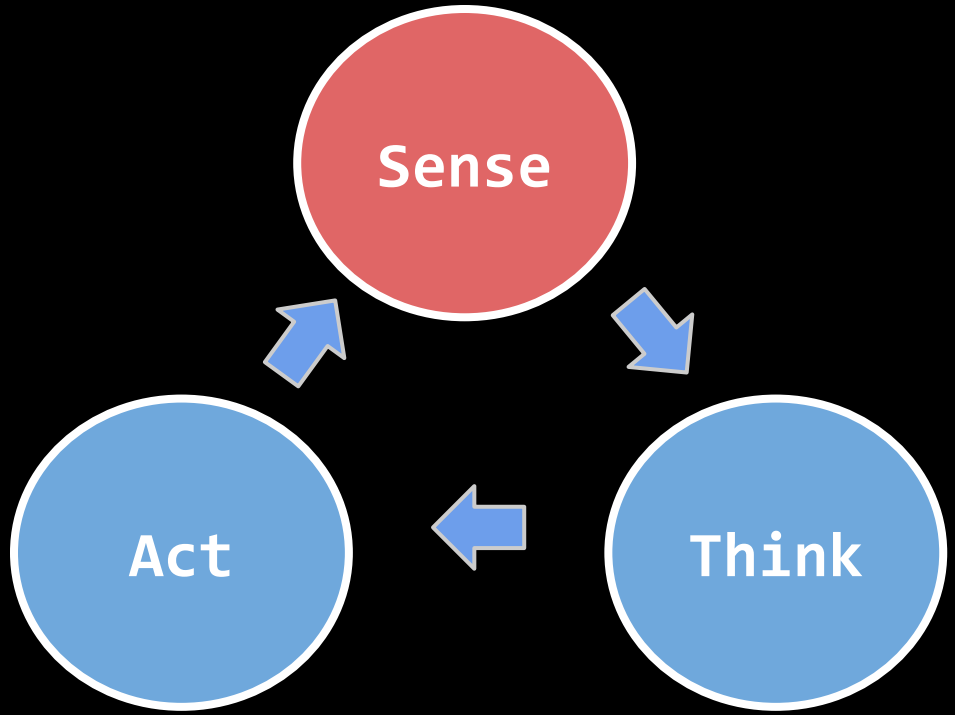
What about other solutions?

- 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, ...)

Theory



Theory



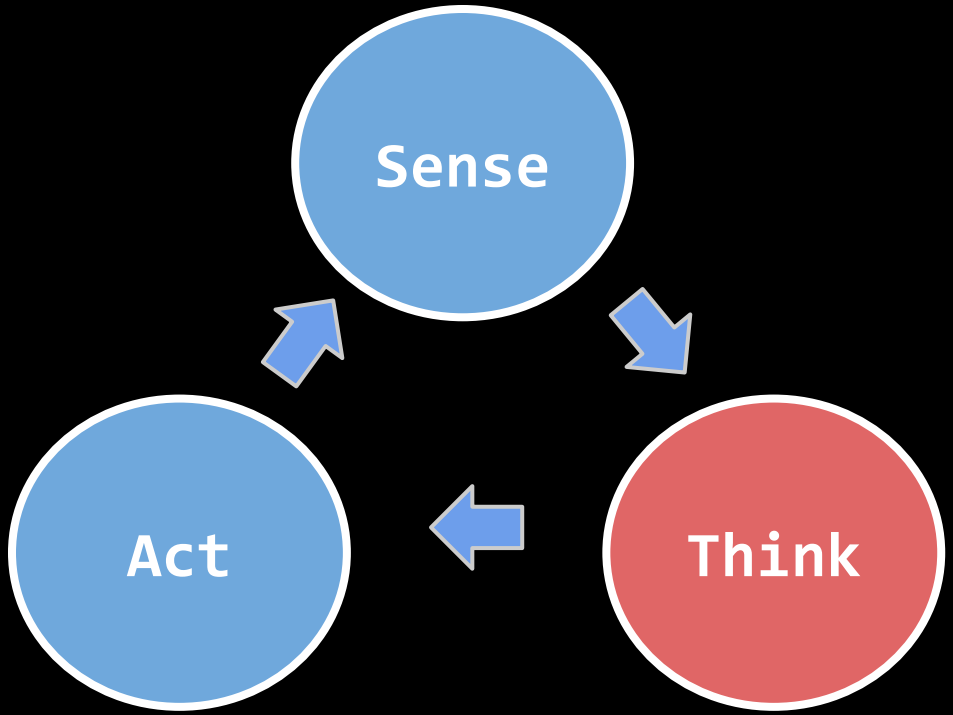
Generally rely on physics engine

Usually very expensive

Use infrequently

Services

Theory



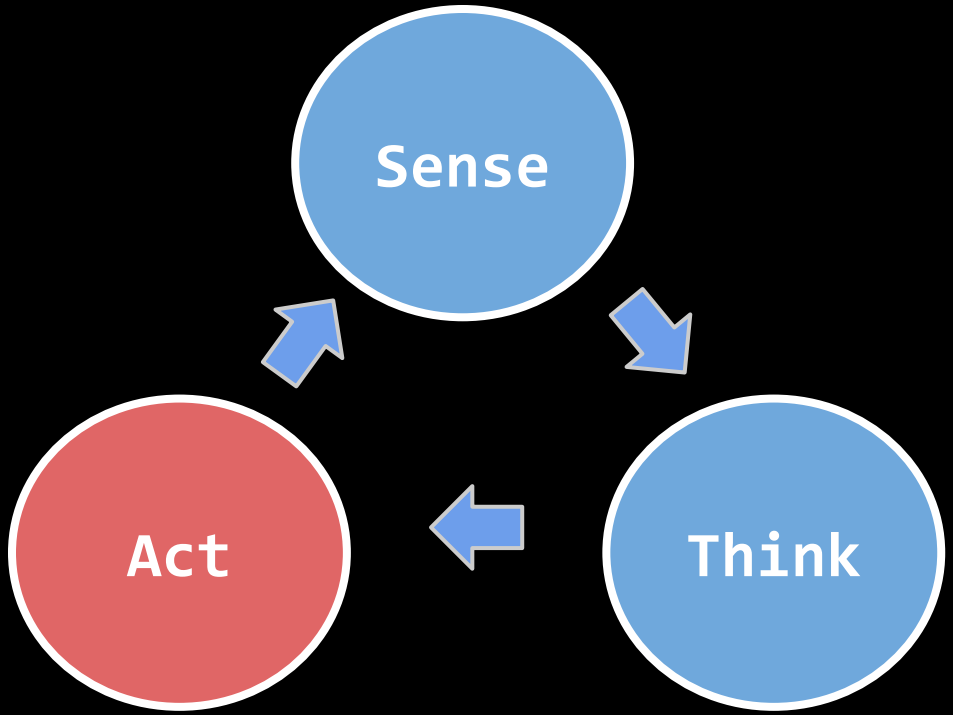
Decision Logic

Generally quite simple

Design intensive

Decorator

Theory



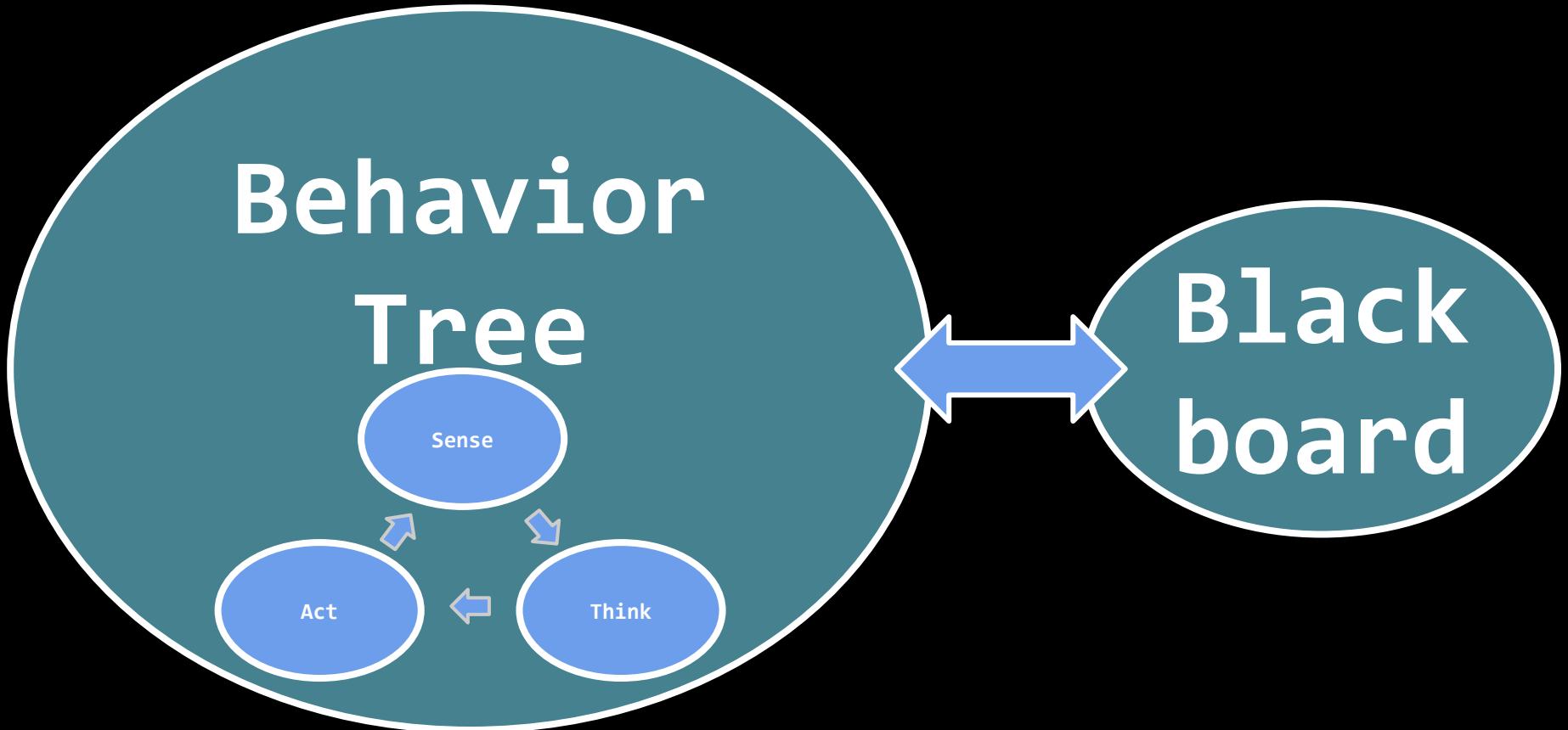
Action execution

Often long running

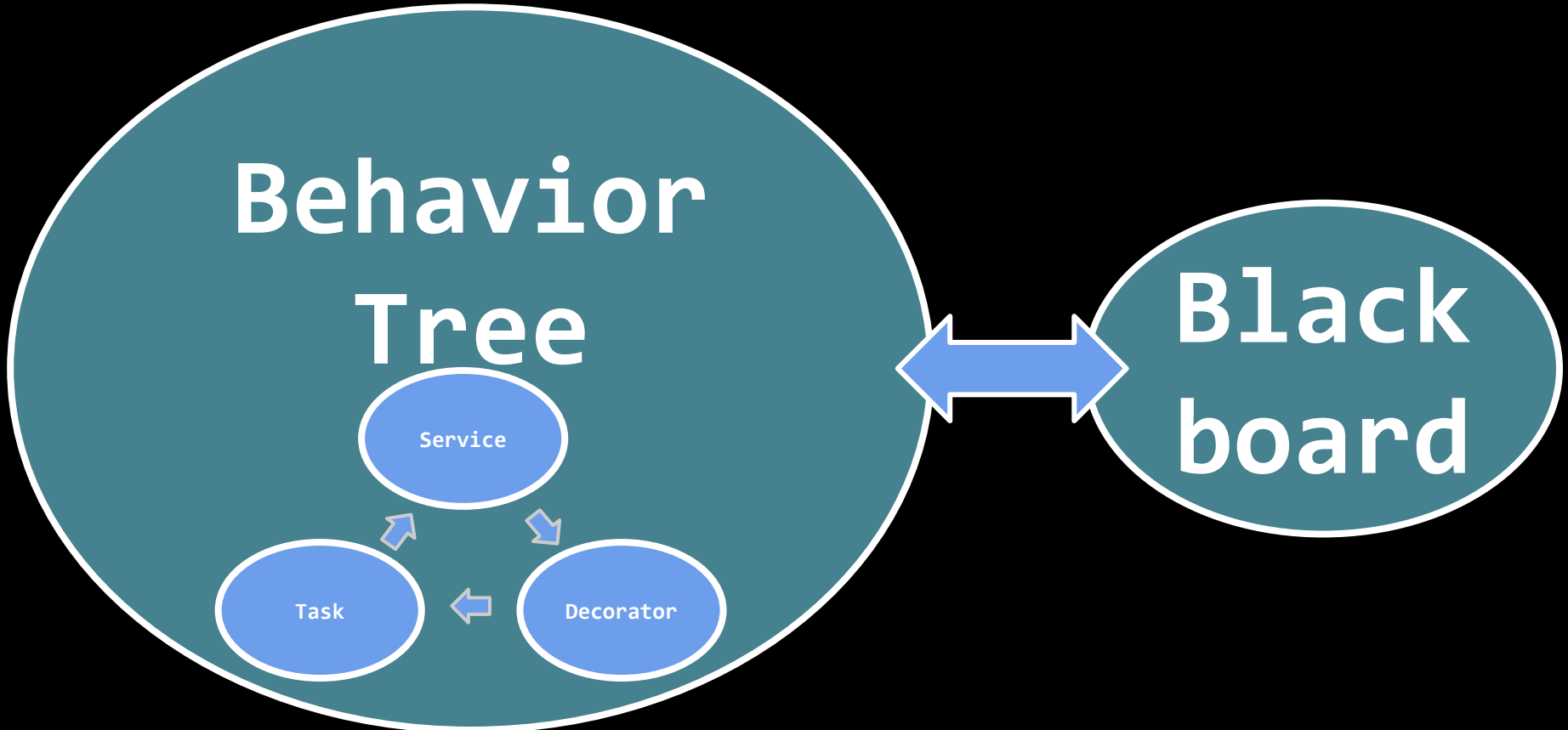
Can fail to complete

Task

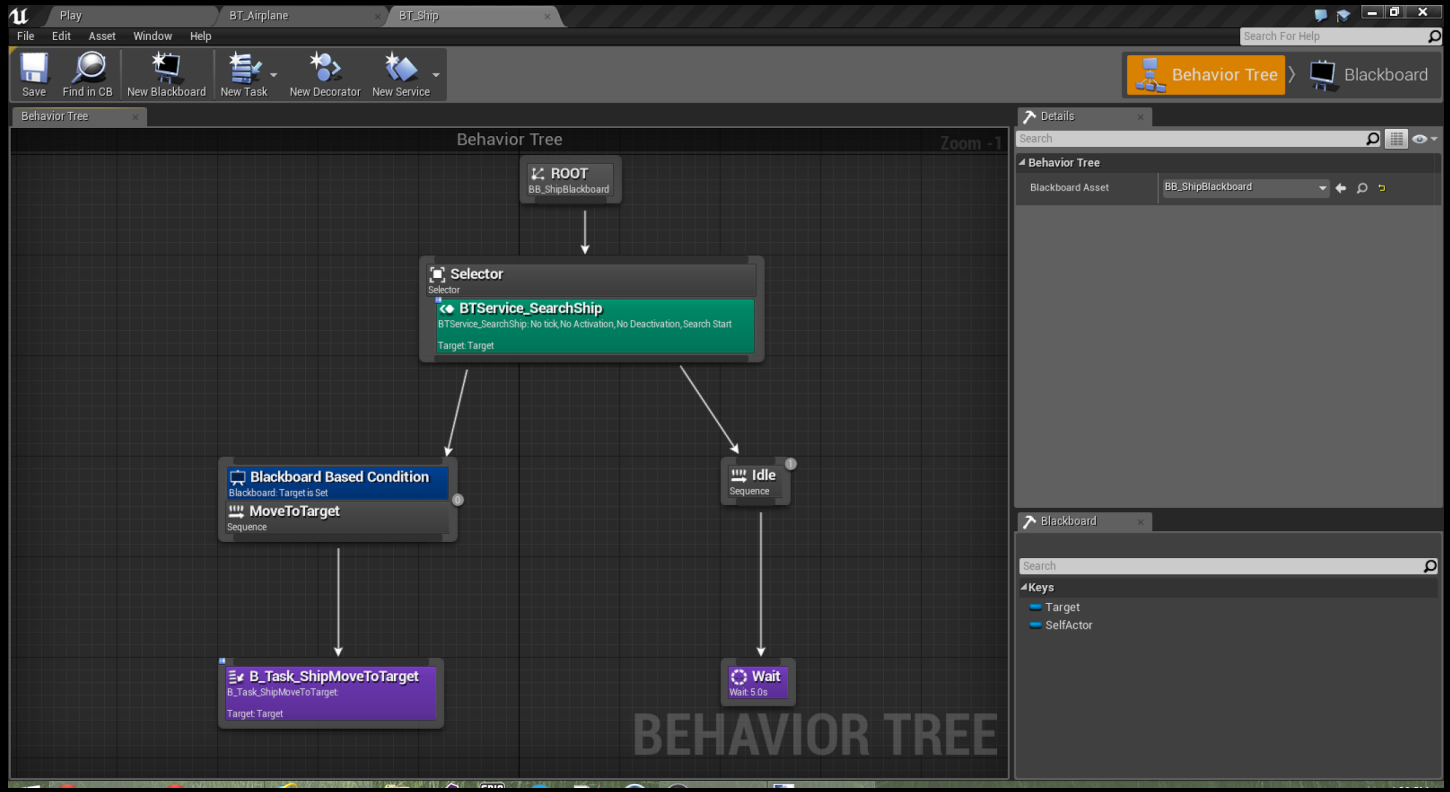
Theory



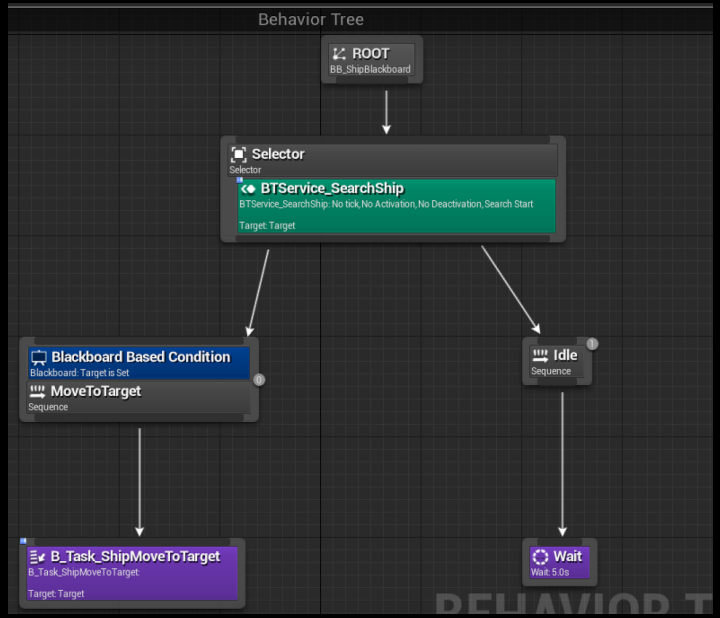
Unreal Engine 4 Behavior Tree



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

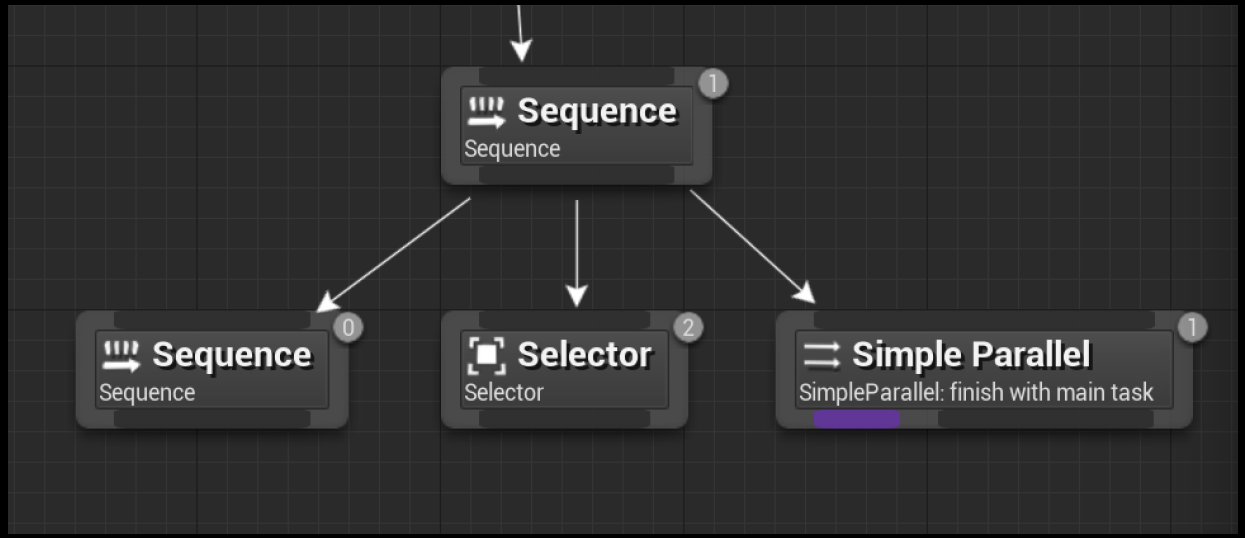
Unreal Engine 4 Behavior Tree

Composite : Sequence

- 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. If all the Sequence’s children succeed, then

Unreal Engine 4 Behavior Tree

Composite : Sequence



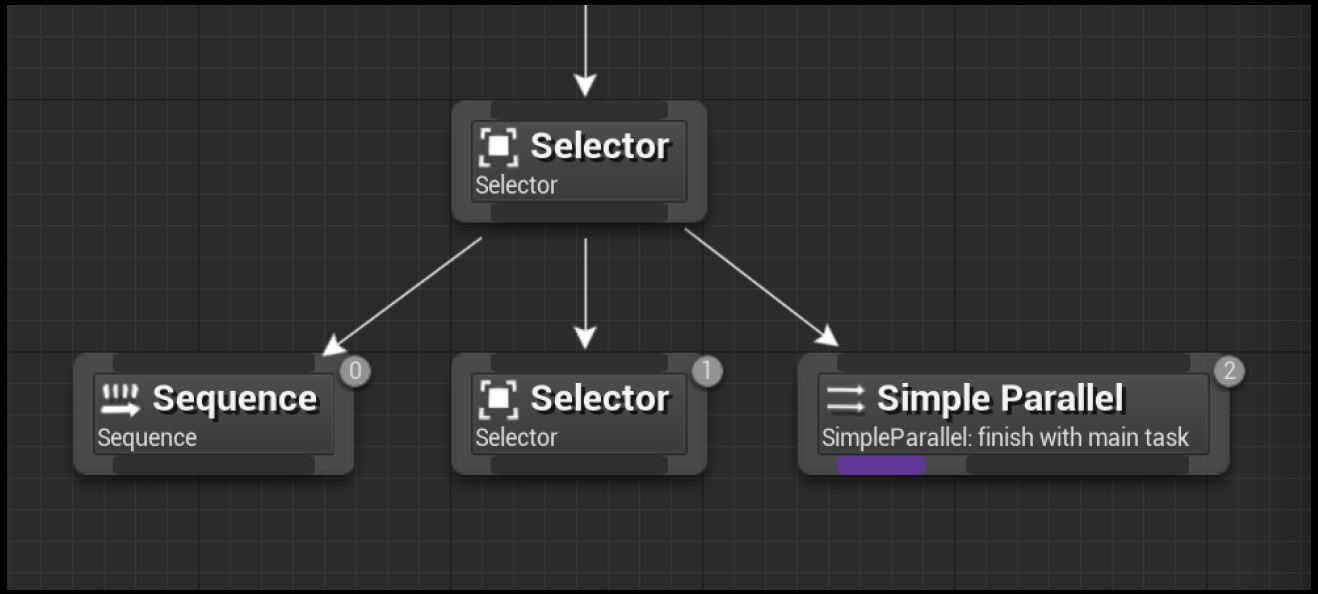
Unreal Engine 4 Behavior Tree

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. If all the Selector’s children fail,

Unreal Engine 4 Behavior Tree

Composite : Selector



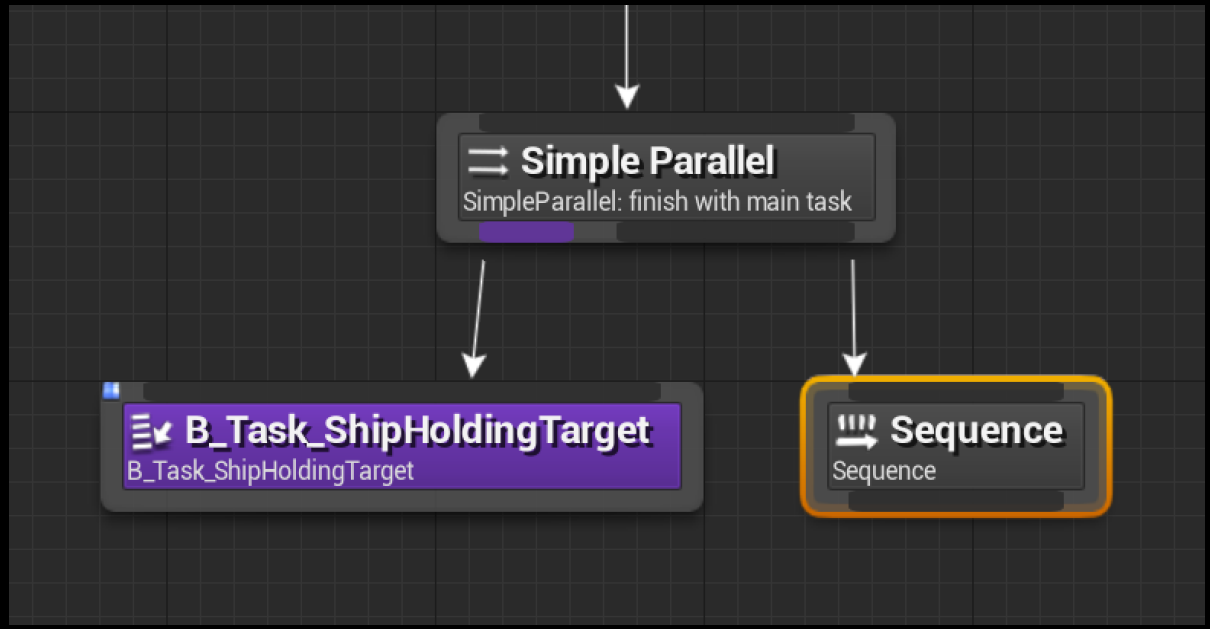
Unreal Engine 4 Behavior Tree

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 if the node should finish **Immediately**, aborting the

Unreal Engine 4 Behavior Tree

Composite : Simple Parallel



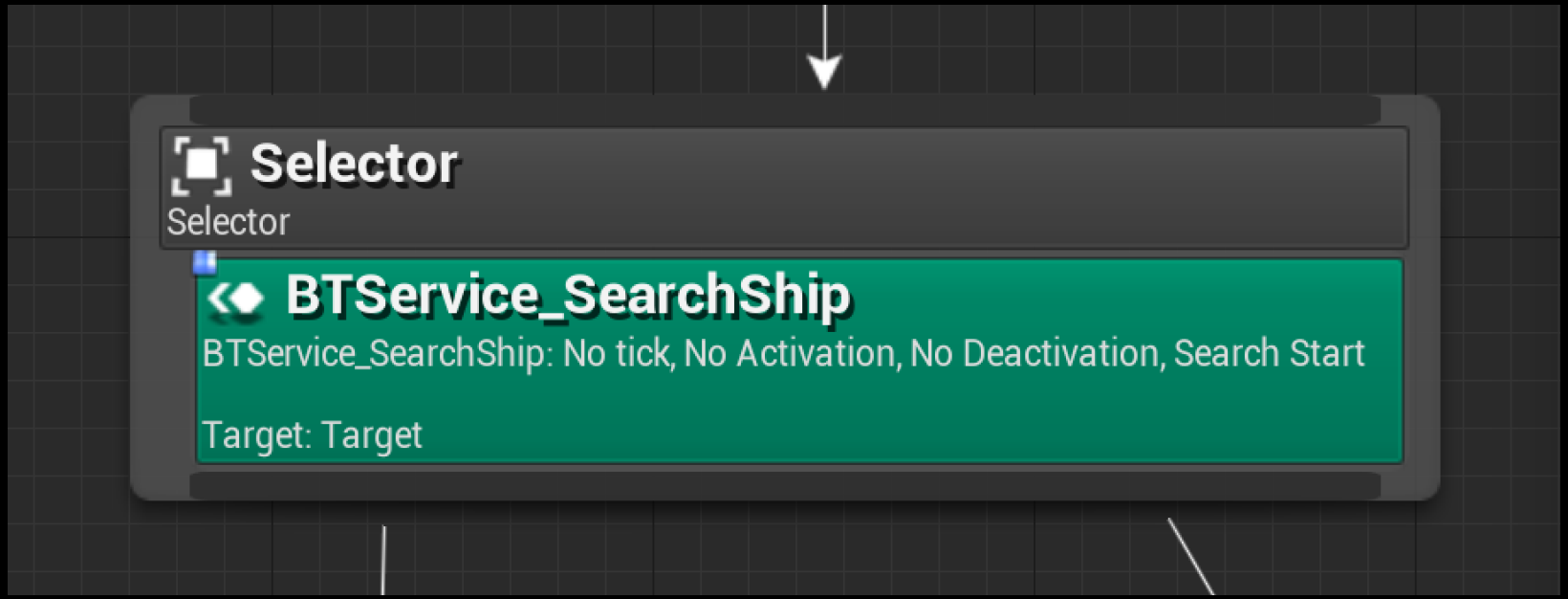
Unreal Engine 4 Behavior Tree

Service

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

Unreal Engine 4 Behavior Tree

Service



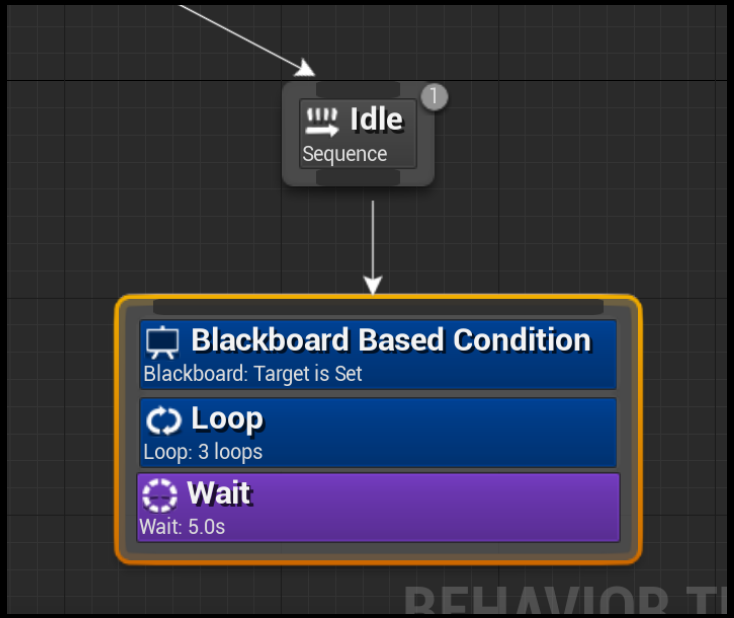
Unreal Engine 4 Behavior Tree

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

Decorator



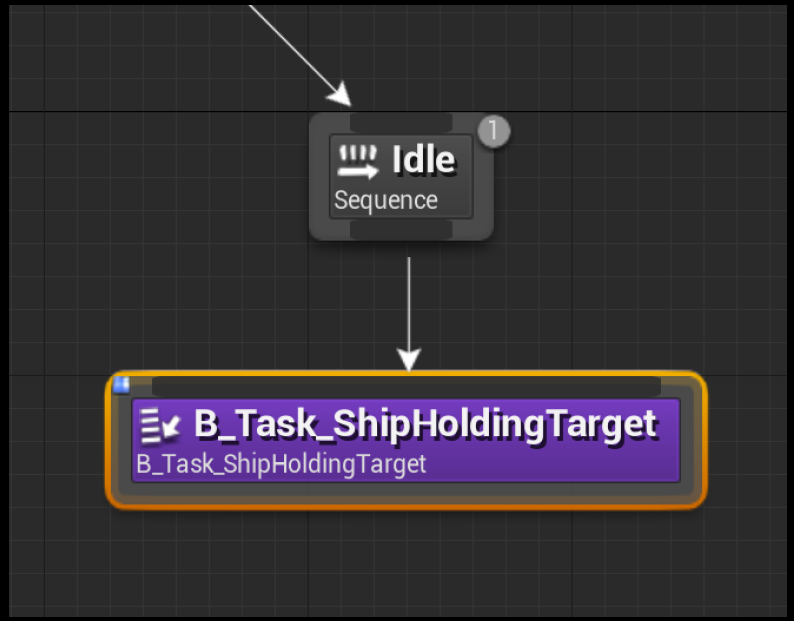
Unreal Engine 4 Behavior Tree

Task

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

Unreal Engine 4 Behavior Tree

Task



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, or used for any other purpose where it’s convenient to have a central place

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 : When do not use?

- 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 one more value to look through while studying every bit

Unreal Engine 4 Behavior Tree

Blackboard : When do not use?

- If you fail to copy data to the blackboard properly, it may cause you some debugging nightmare! If you looking at a value in ins source only, or in the blackboard only, you may not realize instantly that the two values

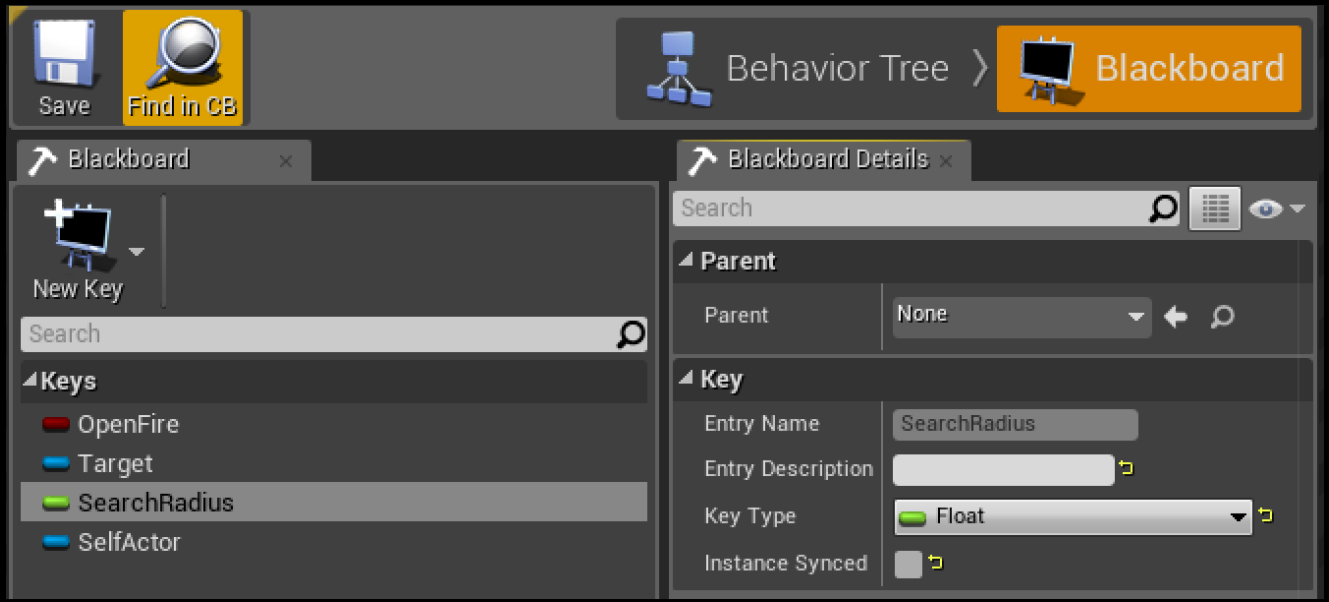
Unreal Engine 4 Behavior Tree

Blackboard : When do not use?

- If enough values are very frequently updated and so need to be copied to the blackboard constantly, that could be bad for performance(though it’s not likely in most cases; if you’re not sure, I wouldn’t worry about this issue,

Unreal Engine 4 Behavior Tree

Blackboard



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