Behavior Tree

smileeagle

휘날려 만들어 틀린부분도 있을 수 있으며
데코레이션 같은 건 없습니다.

우리는 뭐든 2시간이면 하잖아요?

왜 동시에 3가지 일을 못하죠?

목차

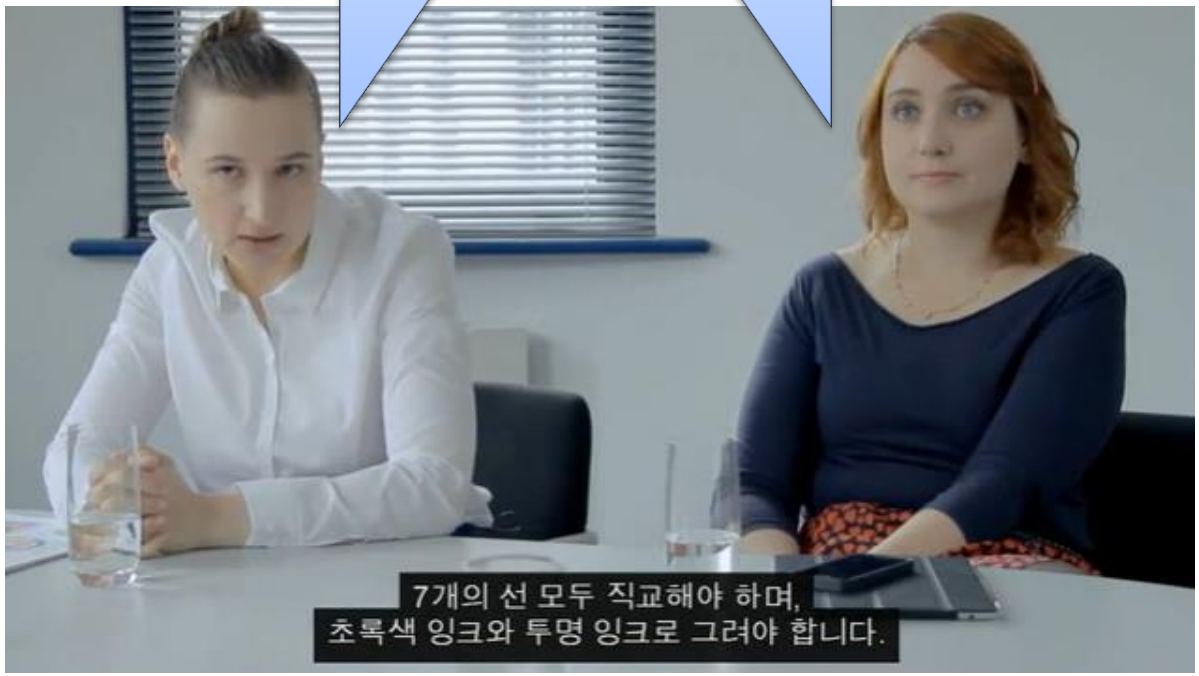
- Needs
- Scripting, FSM, HFSM
- BTs

Needs

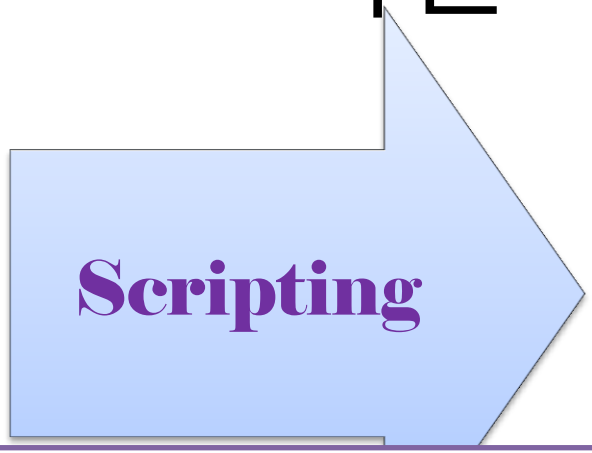
- 디자이너가 원하는 것

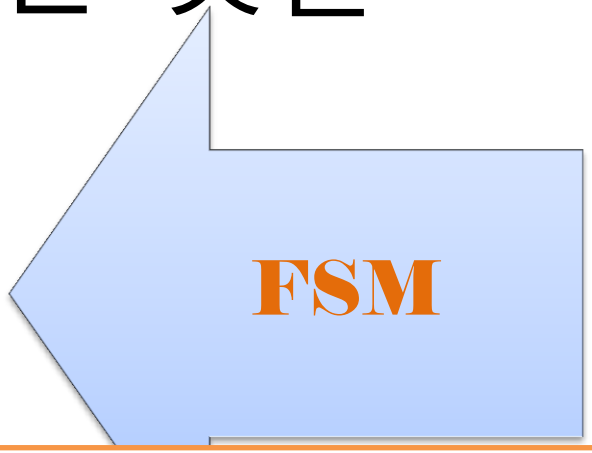
- NPC Full Control
- NPC act Goal Direct

- Autonomously AI Organization
- NPC React to Event



기존에 있는 것들





- NPC Full Control
- Widespread Experience
- Bad Maintenance
- Designers : Programmer?

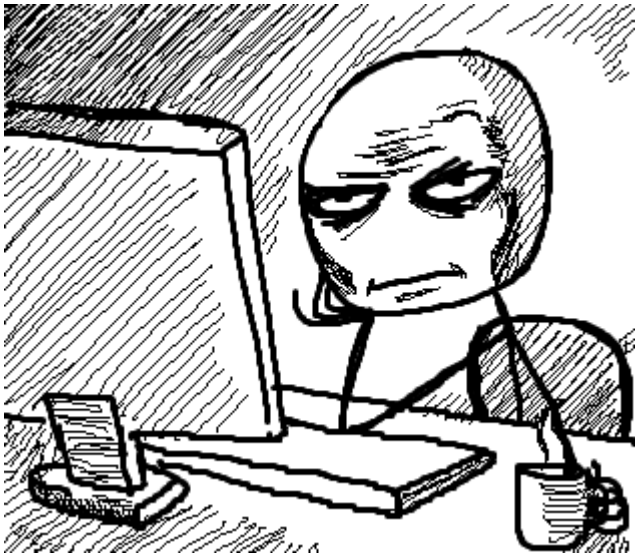
- Autonomously AI Organization
- Simple For Programmer
- Full Reactive
- Difficult to make goal directed

Scripting

- 대표적으로 c++ lua가 있음
  - WoW가 씀
  - 인터프리터 방식 (자바 처럼: 쫌 다르긴 함)
  - fast, portable, embeddable, powerful but simple, small, free 라고 공식사이트에 적힘
- NPC의 행위를 외부 스크립트로 작업
  - 사람에 따라 매우 파워풀한 AI 제작 가능
  - Reload 가능

Scripting But..

- 이 짓을 기획자가 해야 함



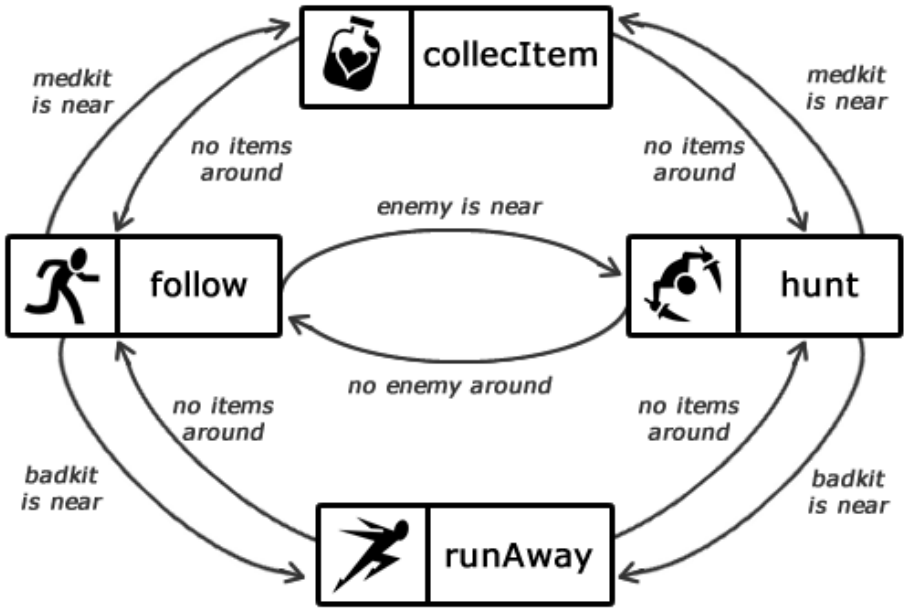
```
--make 3 layers of mountains (you can change the fre
a1 = mountLayer(width, length, f*width/512, f*length
a2 = mountLayer(width, length, 2*f*width/512, 2*f*le
heightmap = {}
for x = 0, width - 1 do
   heightmap[x + xpos2] = {}
   for z = 0, length - 1 do
      heightmap[x + xpos2][z + zpos2] = a1[x][z] + a
   end
end
shellheightmap = {}
for x = 0, width - 1 do
   shellheightmap[x + xpos2] = {}
   for z = 0, length - 1 do
      shellheightmap[x + xpos2][z + zpos2] = heightm
   end
end
gprogress = 0
gloadstatus.Text = "Generating Terrain Shape..."
k = 1 + zpos2
while k < length - 1 + zpos2 do
   for x = 1 + xpos2, width - 2 + xpos2 do
```

FSM

- State와 Logic의 명확한 1:1 대응 형태
- Action 과 Transition 두 가지로 구성
- 장단점
  - 단순하고 명확한 구조에서는 빠른 개발이 가능
  - 복잡한 구조의 비즈니스 로직에서는 비추
    - 점자 로직이 복잡해지면 컨디션 체크를 위해 Handle에 수많은 IF/Case 를 넣어야 한다

FSM

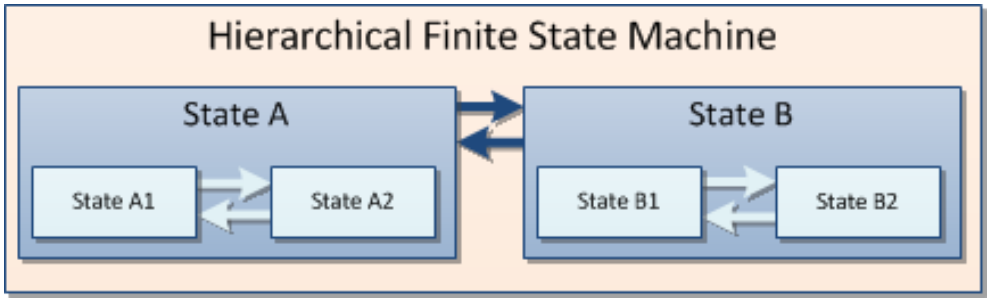
그림 어딘가에서 불펌.. 링크 까머금..



- AddState(STATE_FOLLOW, OnFollow);
- AddState(STATE_COLLECT_ITEM, OnCollectItem);
- AddState(STATE_HUNT, OnHunt);
- AddState(STATE_RUN_AWAY, OnRunAway);
- // STATE_FOLLOW
- AddTransition(STATE_FOLLOW, EVENT_ENEMY_NEAR, STATE_HUNT);
- AddTransition(STATE_FOLLOW, EVENT_MEDKIT_NEAR, STATE_COLLECT_ITEM);
- ….

HFSM

그림 어딘가에서 불펌.. 링크 까머금..

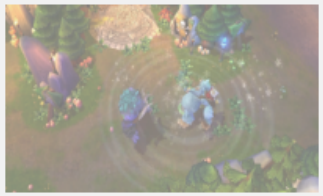


- 디자인 절차는 일반 FSM과 유사
  - State 그룹핑
  - FSM이
    void *func[MAX_STATE] 으로 만들어 진다면
    2Depth의 HFSM은
    *void *func[MAX_STATE][MAX_SUB_STATE] 로 구성된다는 느낌?
    (코드 구성에 따라 다중 Depth 가능)

- 장단점
  - State의 확장이 용이
  - Transition 에 대한 로직의 중복성을 어느 정도 피하는 것이 가능
  - 역시 State에 대한 Depth가 깊어질 수록 Hell

BTs

PREMIUM TEASER



Behavior Trees and Level Scripting in LEAGUE OF LEGENDS

Alex J. Champandard on August 1, 2012

How can BTs be implemented into MOBA-style games to support human-like bots? In this interview with Andrew Woo, AI Programmer at Riot Games, you'll find out more about the behavior tree implementation in LEAGUE OF LEGENDS. You'll discover how a visual tool was used for the scripting and design of levels, and more!

Access the rest of this feature by joining industry experts and other professionals as a PREMIUM member in the leading Game AI training program.

Subscribe Now...

Individual

(Non-Commercial)

From $59 per Quarter.

Sign-Up ... or click for more details.

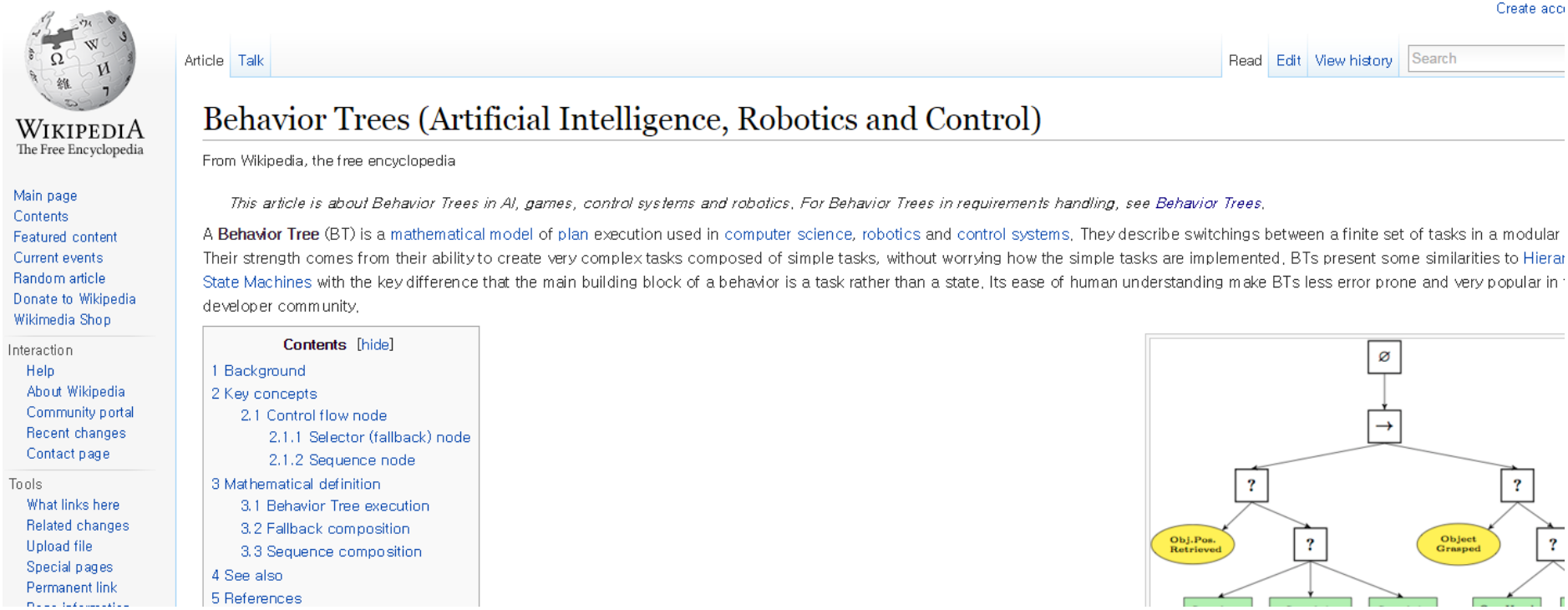
Studio

(Commercial)

$189 to $995 Quarterly.

Sign-Up ... or click for more details.

BTs



Create acc

Article | Talk | Read | Edit | View history | Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link

Behavior Trees (Artificial Intelligence, Robotics and Control)

From Wikipedia, the free encyclopedia

This article is about Behavior Trees in AI, games, control systems and robotics. For Behavior Trees in requirements handling, see Behavior Trees.

A **Behavior Tree** (BT) is a mathematical model of plan execution used in computer science, robotics and control systems. They describe switchings between a finite set of tasks in a modular
Their strength comes from their ability to create very complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. BTs present some similarities to Hierar
State Machines with the key difference that the main building block of a behavior is a task rather than a state. Its ease of human understanding make BTs less error prone and very popular in
developer community.

Contents [hide]
1 Background
2 Key concepts
2.1 Control flow node
2.1.1 Selector (fallback) node
2.1.2 Sequence node
3 Mathematical definition
3.1 Behavior Tree execution
3.2 Fallback composition
3.3 Sequence composition
4 See also
5 References

- Selector와 Sequences의 조합을 통한 다양한 State 의 표현이 가능
- Modularity, Reusability
- Task
  - Condition, Action, Composite 3가지 타입으로 구성

BTs

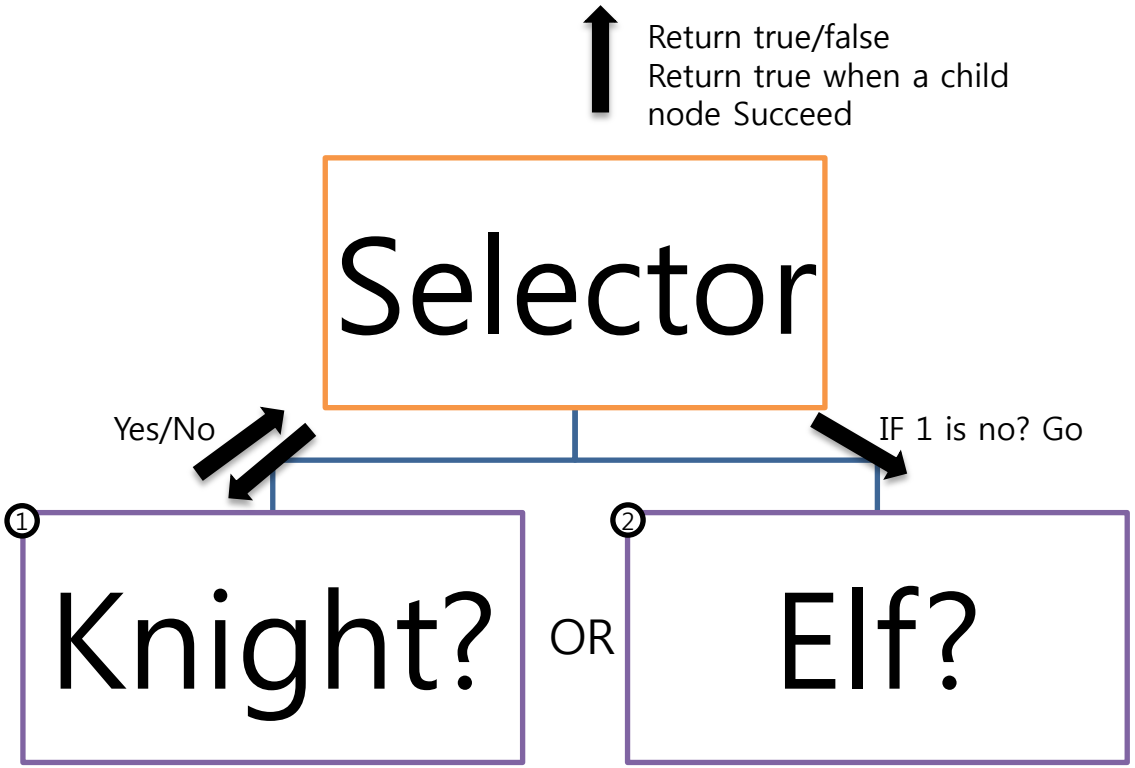
- Task
  - Condition: 조건에 대한 검사, 단말 노드에만 존재
  - Action: 로직, 단말 노드에만 존재
  - Composite: Condition, Action 그리고 다른 Composite간 결합, 줄기 노드
    - Selector
      - 자식노드가 수행 결과가 True 일 경우 바로 그 결과를 반환
      - 자식노드 수행 결과가 False일 경우 다음 자식 노드로 진입
    - Sequence
      - 자식노드가 수행 결과가 True 일 경우 계속 순차 실행
      - 자식노드가 수행 결과가 False일 경우 경우 다음 자식 실행 중지
    - Parallel: 여러 자식 노드가 동시에 실행되도록 함
  - Decorator
    - Decorator Design pattern
    - 하나의 자식 노드만 가짐
- BlackBoard
  - Task의 의사결정을 위한 정보 저장
  - 다수의 AI NPC가 하나의 Blackboard 공유 가능
    - 즉 다수의 AI NPC가 Blackboard를 통해 통신을 함
  - Task와 Data를 분리 : Data(각종 Task를 구성하는데 필요한 정보)

BTs에서 하나의 자식 노드에 대한 수행을 완료하면 그 수행 결과값(True or False)을 반환하는데, Composite Task는 그 결과값을 보고, 트리를 계속 진행할지 여부 결정

https://wiki.unrealengine.com/Blueprint_Behavior_Tree_Tutorial
언리얼4의 BTs

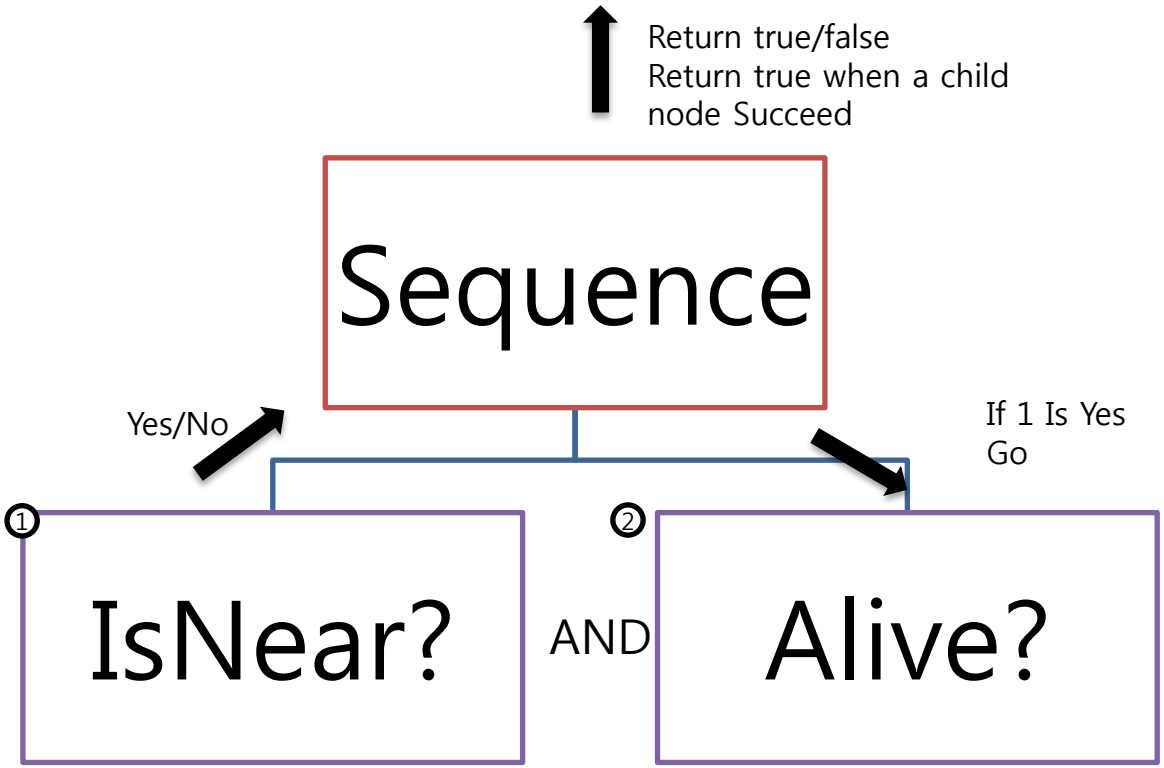
Selector

- 자신의 자식 노드 중 하나라도 성공하면 True를 상위 노드로 올려 줌

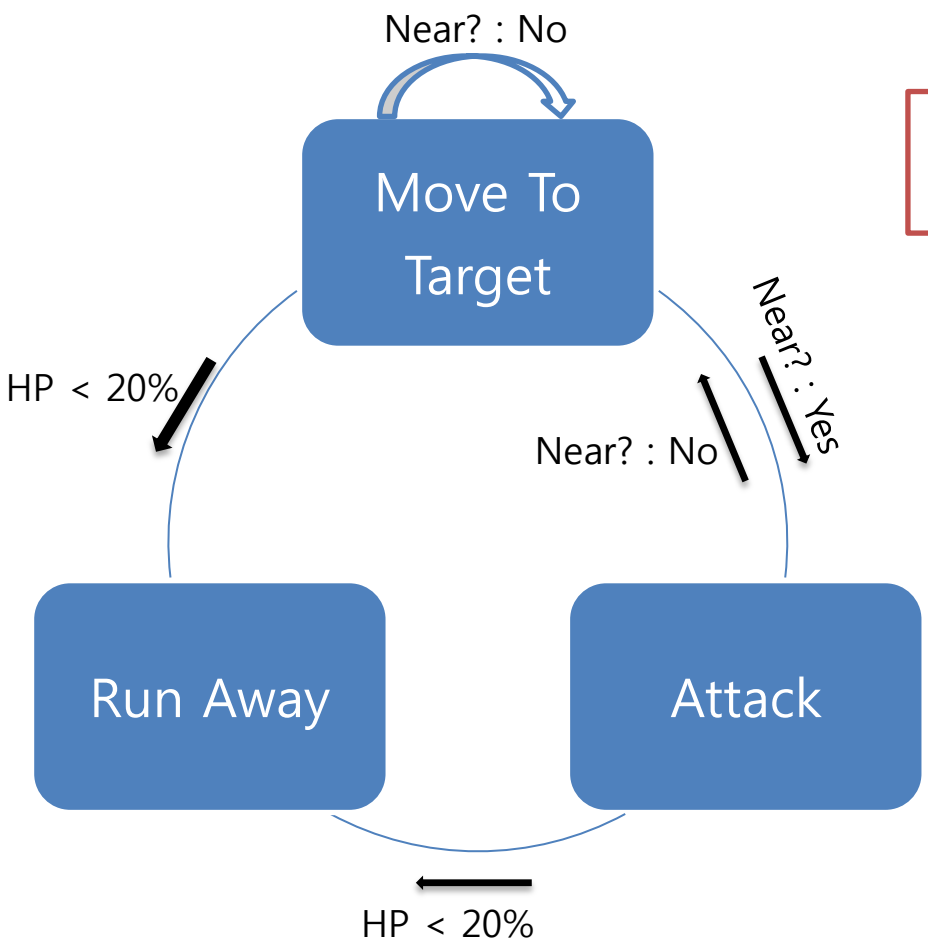


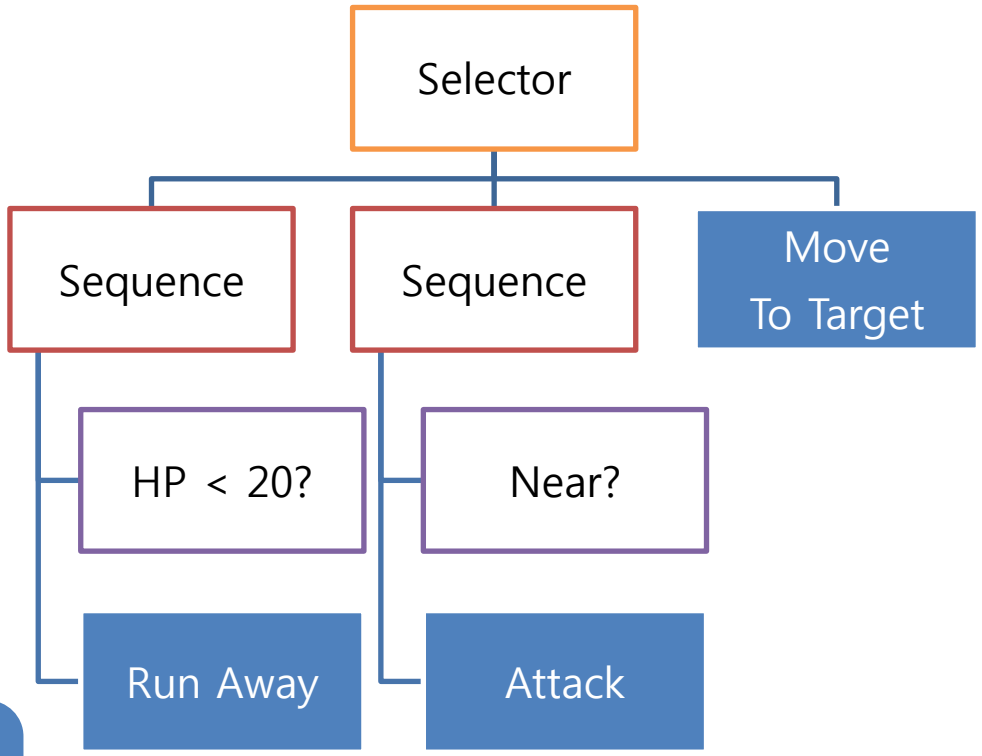
Sequence

- 자신의 자식 노드 모두 성공하면 True를 상위 노드로 올려 줌



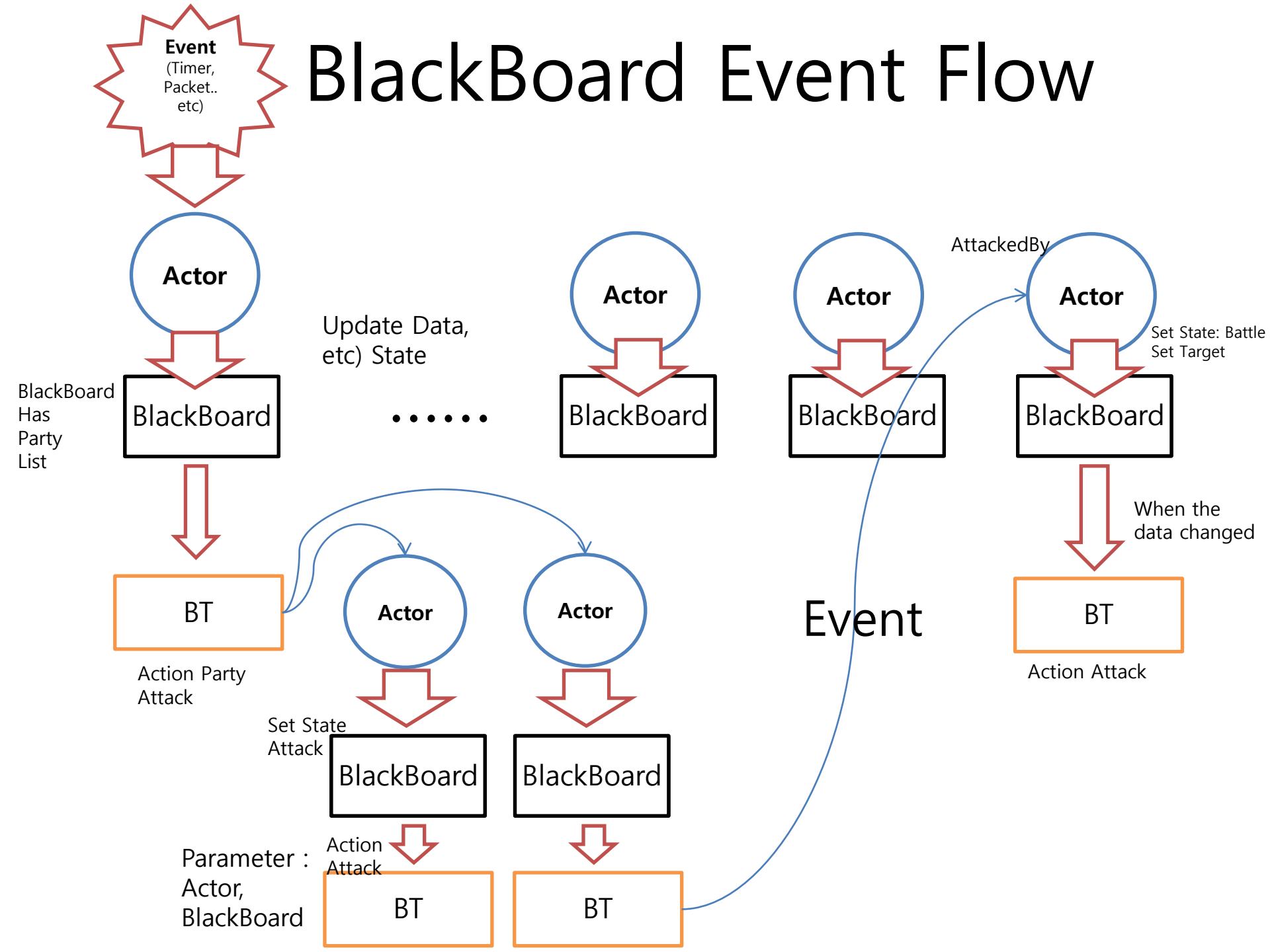
FSM 과의 비교


Near? : No
Move To Target
HP < 20%
Near? : Yes
Near? : No
Run Away
Attack
HP < 20%


Selector
Sequence
Sequence
Move To Target
HP < 20?
Near?
Run Away
Attack

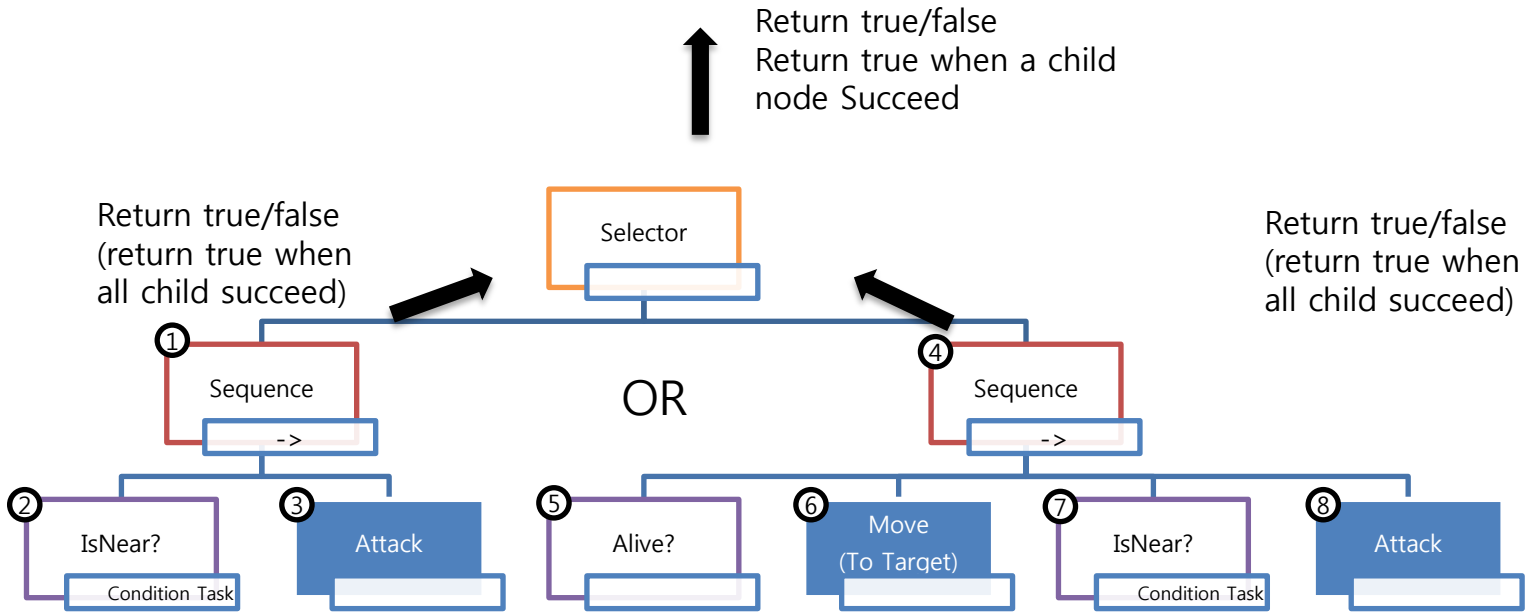
BTs

- BlackBoard를 사용하는 이유
  - **Event-Driven Behavior**
    - 인공지능 로직은 게임 매 프레임마다 업데이트하지 않음. (부하)
      인공지능 로직이 프레임마다 실행될 필요는 없기 때문에
      타이머의 일정 주기나 Event 발생 시 AI 로직이 실행
      즉, Blackboard에 Changed Event가 발생할 때 마다 이벤트를 BT에 전달, 이 시점에 BT의 실행이 이루어 짐
  - **Cache Calculations**
    - 동일한 데이터가 반복적으로 같은 연산을 수행 시 부하 발생
    - 이미 계산된 연산 결과를 캐시
  - **Centralize Data**
    - Blackboard를 통해 BT에서 사용하는 데이터를 구조적으로 한곳에 모아 둠
    - 각각의 클래스에 산재한 멤버에 참조하기 위해 이리저리 함수를 호출하는 구조로 개발되면, 스파게티 코드가 됨
    - 데이터의 캡슐화가 됨


Event
(Timer,
Packet..
etc)
BlackBoard Event Flow
Actor
BlackBoard
BlackBoard
Has
Party
List
Update Data,
etc) State
Actor
BlackBoard
Actor
BlackBoard
AttackedBy
Actor
BlackBoard
Set State: Battle
Set Target
When the
data changed
BT
Action Party
Attack
Actor
Actor
Event
BT
Action Attack
Set State
Attack
BlackBoard
BlackBoard
Parameter :
Actor,
BlackBoard
Action
Attack
BT
BT

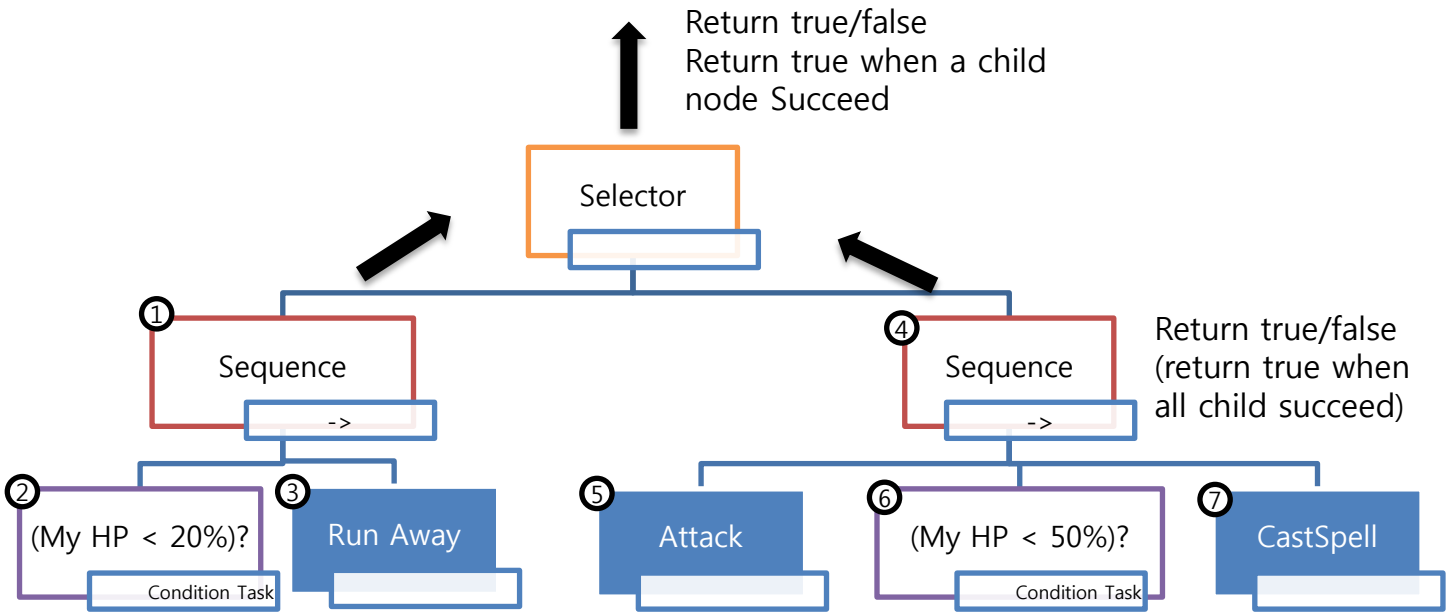
간단한 이해1

- NPC가 PC와 싸움
  - 타겟만 보이면 무조건 붙어서 때리려고만 하는 단순한 NPC



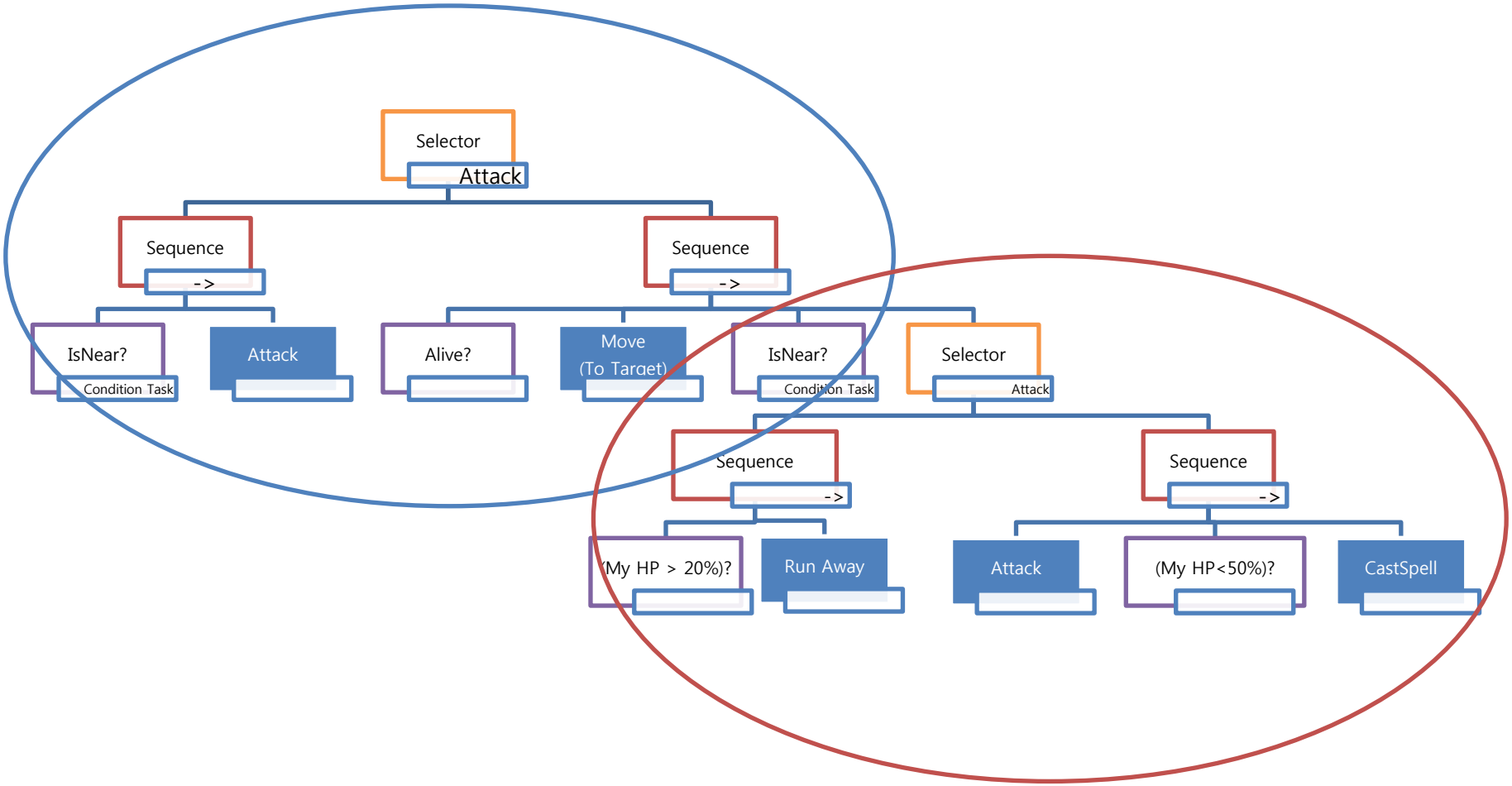
간단한 이해2

- 공격 모듈을 구성해 보자

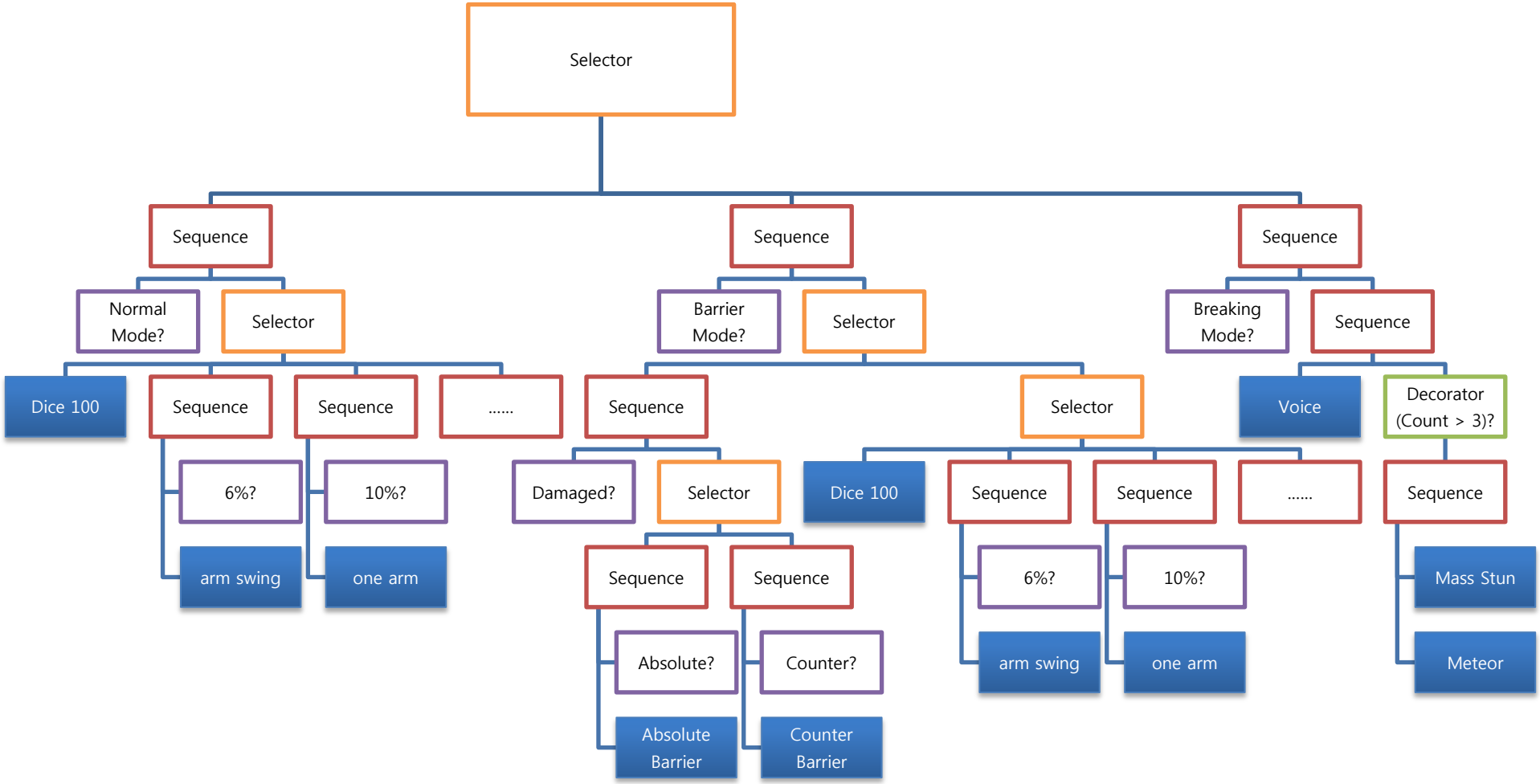


앞의 1 과 2를 합치면?

- NPC가 조금 똑똑해 짐 : AI의 모듈화

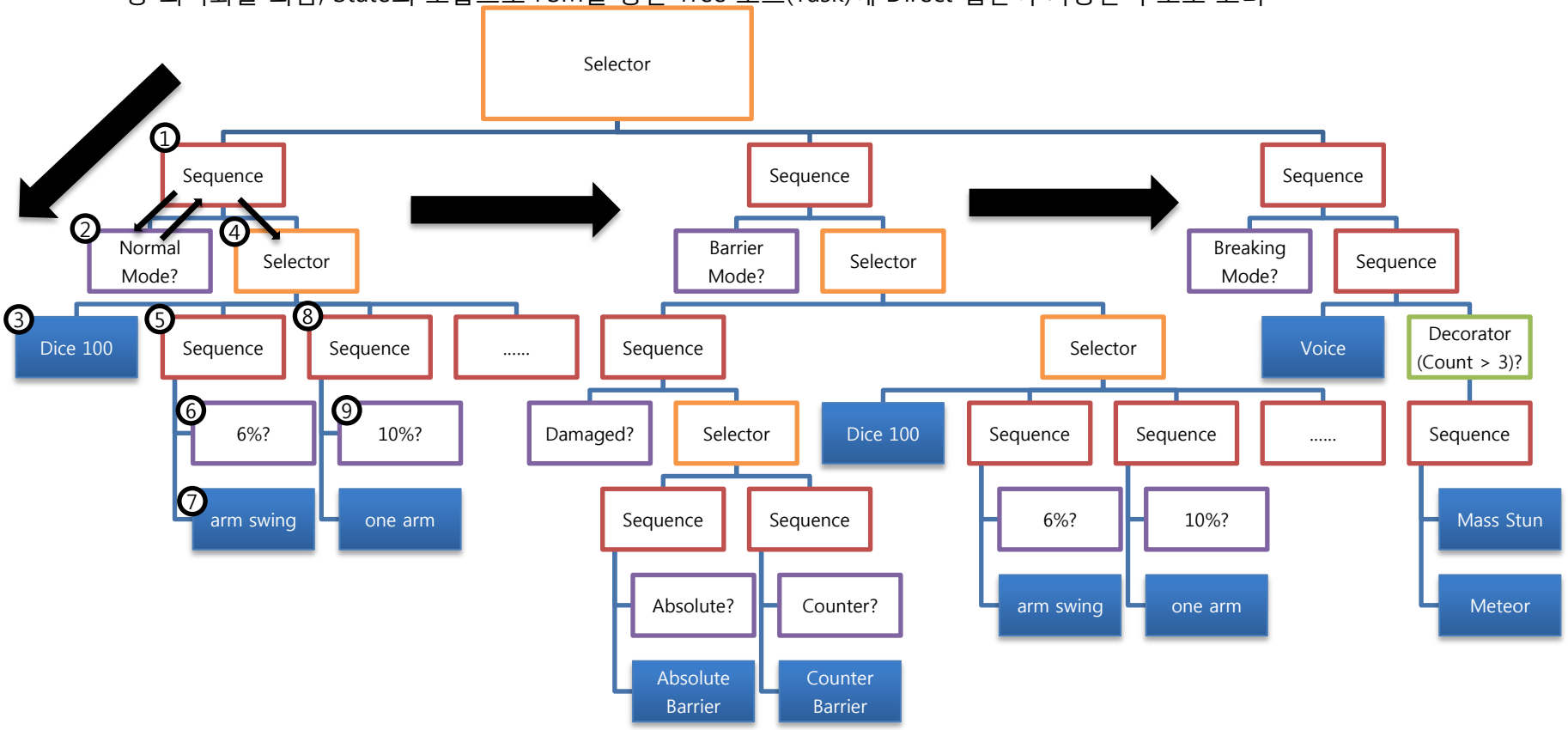


실제 라이브 구성


Selector
Sequence
Normal Mode?
Selector
Dice 100
Sequence
6%?
arm swing
Sequence
10%?
one arm
.....
Sequence
Barrier Mode?
Selector
Sequence
Damaged?
Selector
Sequence
Absolute?
Absolute Barrier
Sequence
Counter?
Counter Barrier
Selector
Dice 100
Sequence
6%?
arm swing
Sequence
10%?
one arm
.....
Sequence
Breaking Mode?
Sequence
Voice
Decorator (Count > 3)?
Sequence
Mass Stun
Meteor

BTs 트리 순회

- depth-first order - 기본적으로 전위 순회
- 알고리즘
  b : 평균 자식 노드 수 d : depth
  - **O** : 1 + b + b^2 + … + b^d = (b^(d+1) – 1) / (b – 1)
  - **Θ** : d + 1
  - **Ω : d+1과** (b^(d+1) – 1) / (b – 1) 의 평균 = (b^d+1 + db + b – d -2) / 2(b-1)
- 기본 컨셉은 Big O Notation 기준 b^d 의 시간 복잡도를 가지나, Condition Task 의 배치 방식에 따라 Breadth First Search 를 섞은 구성이 가능
- 실제 개발 시 Heuristic Search, Solution Path Search, Problem Reduction, Constraint Satisfaction 등의 방식을 조합하여, 성능 최적화를 꾀함, State의 조합으로 FSM을 통한 Tree 노드(Task)에 Direct 접근이 가능한 구조도 고려

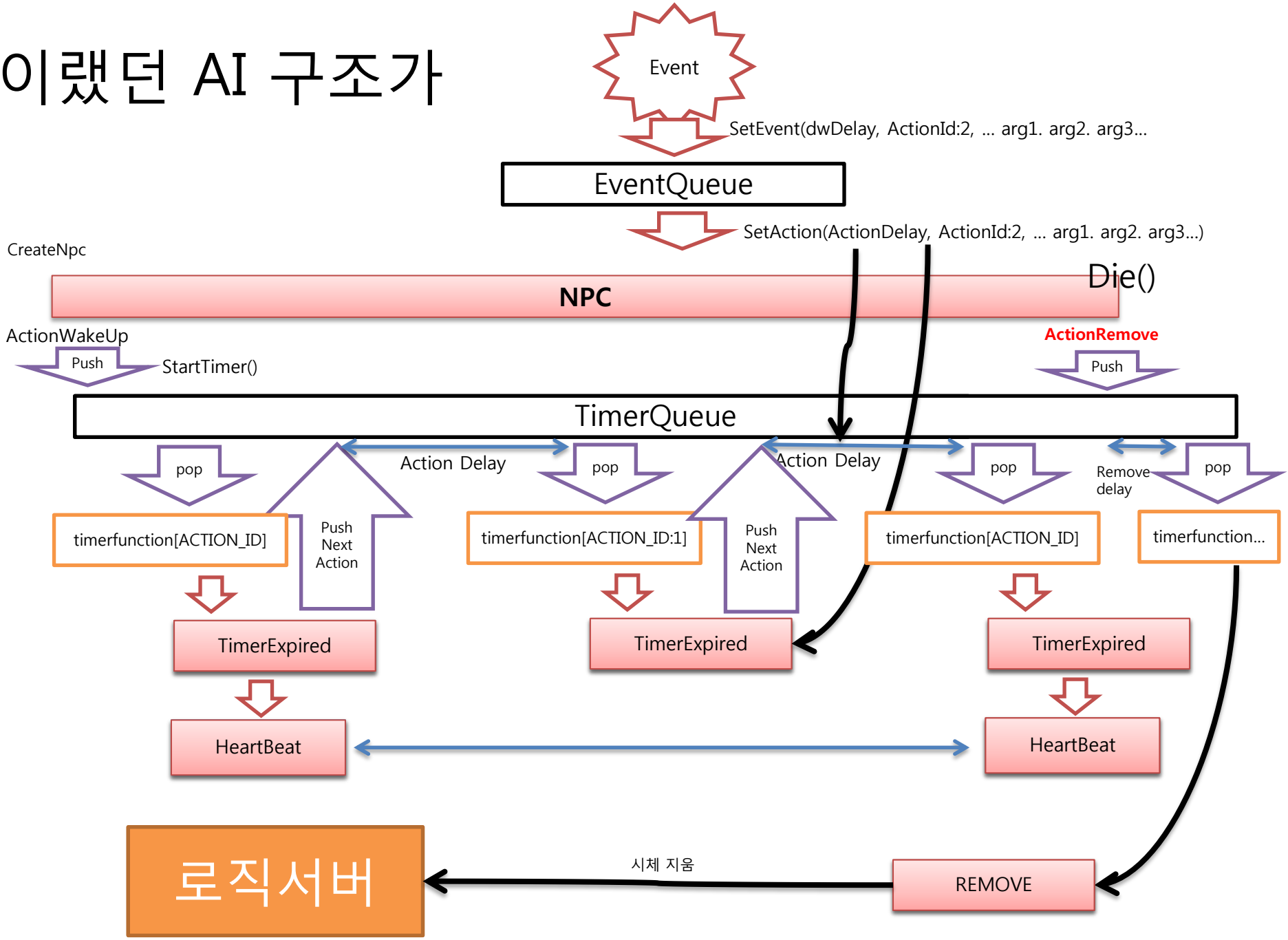


대충 개발은

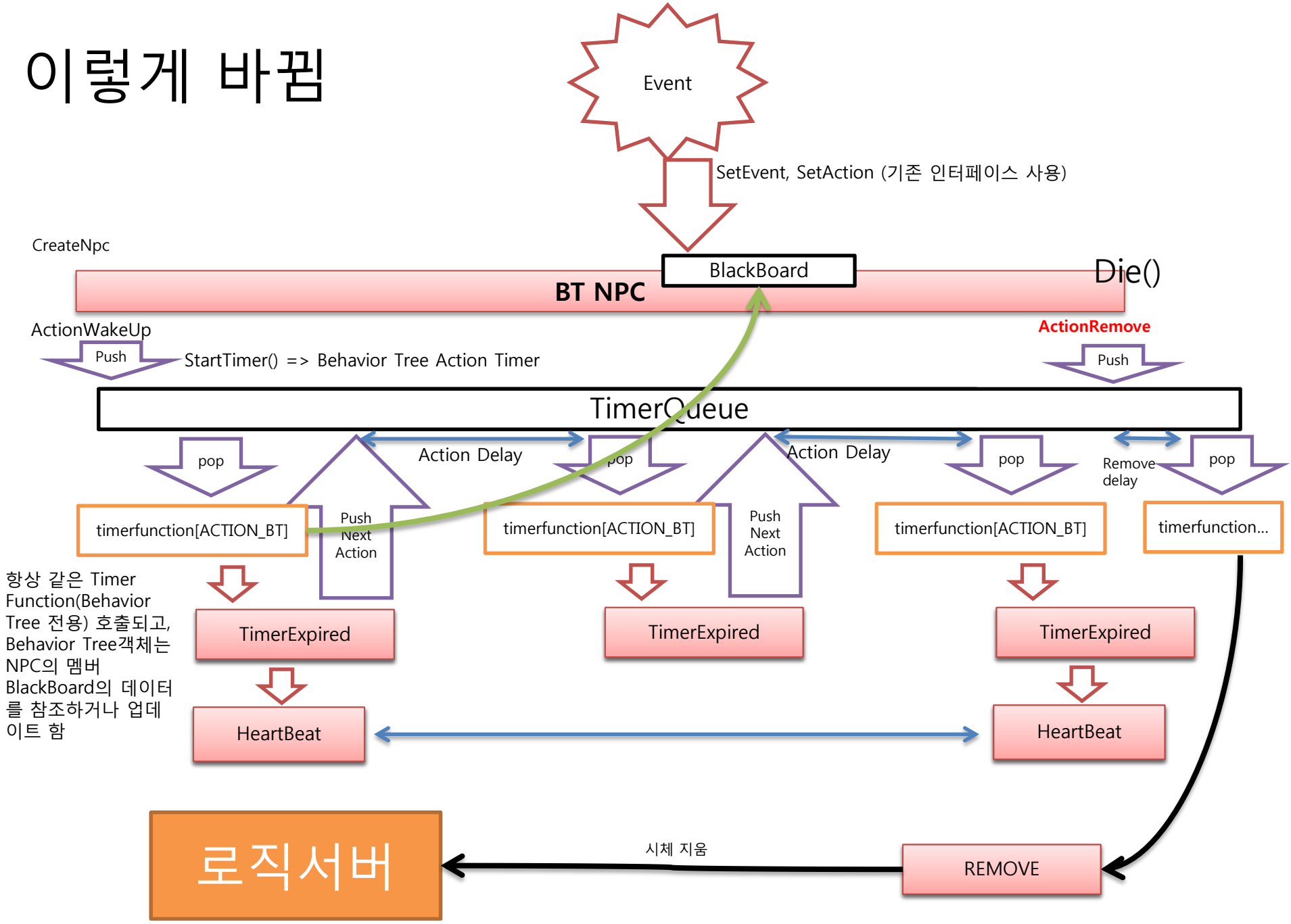
- 각 Task 노드를 Functor 형 상속으로 구성
- Functor는 CreateItem 처럼 카테고리화 하여 스크립팅 가능한 구조로 구성
- NPC의 멤버로 트리형 자료구조 삽입

- **BTs를 넣으면 할 수 있는 것들**
  - NPC AI에 대한 Tree 재배치가 가능한 구조를 만들 수 있음
    - NPC 생성 시 Tree Preset을 통해 선택적 AI 사용 가능
      - NPC가 Random AI 를 가지도록 구성이 가능함
      - NPC가 지능이 높으면 더 좋은 AI를 가지도록 구성 가능
  - User Bot 도 쉽게 개발 ➔ 필요한가??

이랬던 AI 구조가


Event
SetEvent(dwDelay, ActionId:2, ... arg1. arg2. arg3...
EventQueue
SetAction(ActionDelay, ActionId:2, ... arg1. arg2. arg3...)
CreateNpc
NPC
Die()
ActionWakeUp
Push
StartTimer()
ActionRemove
Push
TimerQueue
pop
Action Delay
pop
Action Delay
pop
Remove delay
pop
timerfunction[ACTION_ID]
Push Next Action
timerfunction[ACTION_ID:1]
Push Next Action
timerfunction[ACTION_ID]
timerfunction...
TimerExpired
TimerExpired
TimerExpired
HeartBeat
HeartBeat
로직서버
시체 지움
REMOVE

이렇게 바뀜


Event
SetEvent, SetAction (기존 인터페이스 사용)
CreateNpc
BlackBoard
Die()
BT NPC
ActionWakeUp
Push
StartTimer() => Behavior Tree Action Timer
ActionRemove
Push
TimerQueue
pop
Action Delay
pop
Action Delay
pop
Remove delay
pop
timerfunction[ACTION_BT]
Push Next Action
timerfunction[ACTION_BT]
Push Next Action
timerfunction[ACTION_BT]
timerfunction...
항상 같은 Timer Function(Behavior Tree 전용) 호출되고, Behavior Tree객체는 NPC의 멤버 BlackBoard의 데이터를 참조하거나 업데이트 함
TimerExpired
TimerExpired
TimerExpired
HeartBeat
HeartBeat
로직서버
시체 지움
REMOVE

BTs Class Hierarchy