강의노트 Decision Making 에서

Behavior Trees

모든 state-action pair 들로서 plan을 세움

Represents all actions that an agent tasks in a behavior tree

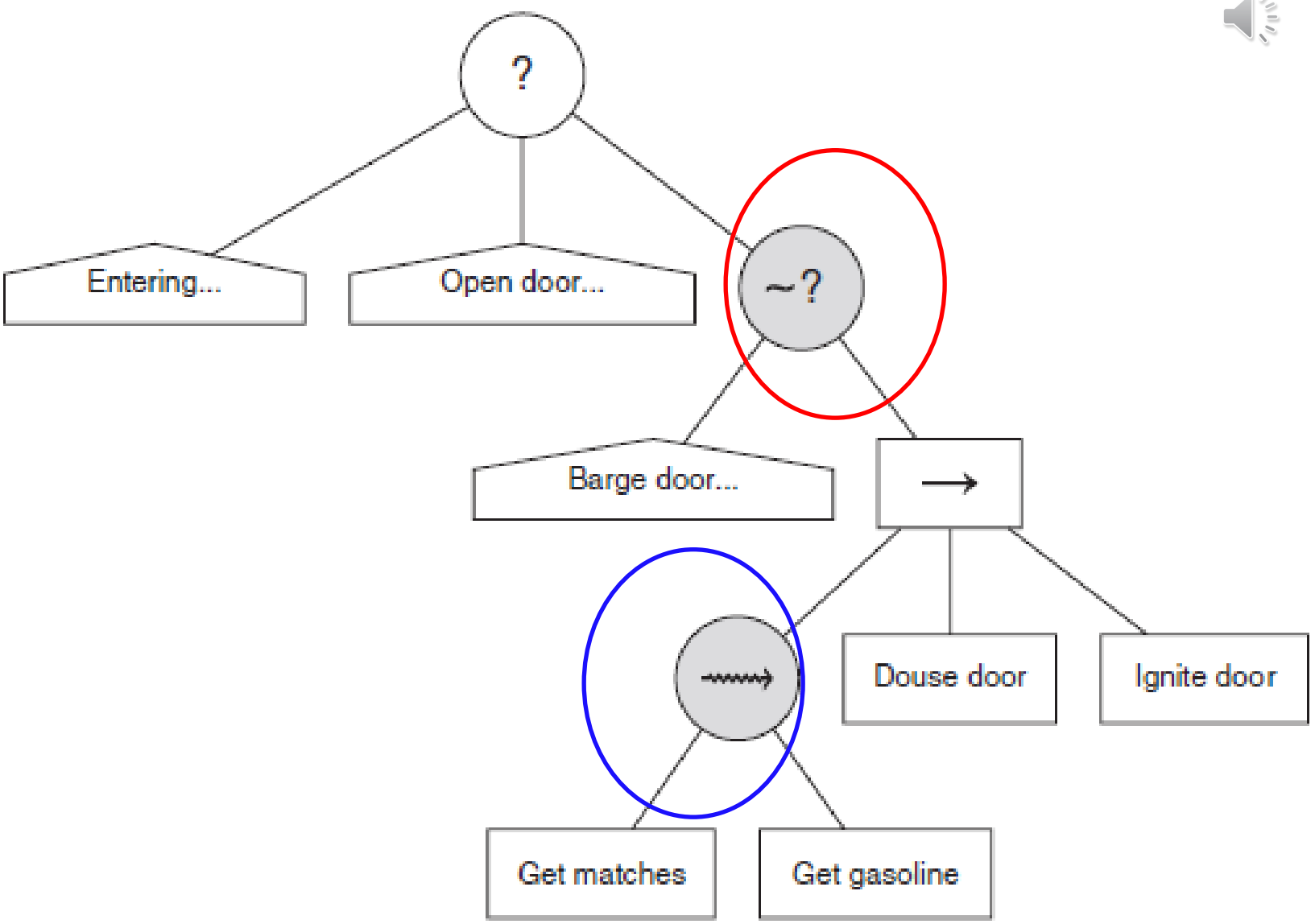
➔ Implements *reactive planning*

- Avatars check conditions (➔ to check the current game state) in order and base actions on those checks

Variation: Randomization

Do not always evaluate from left to right in a sequencer or selector

Selectors and sequences run their children nodes in a random order ➔ **Unpredictability 증가**

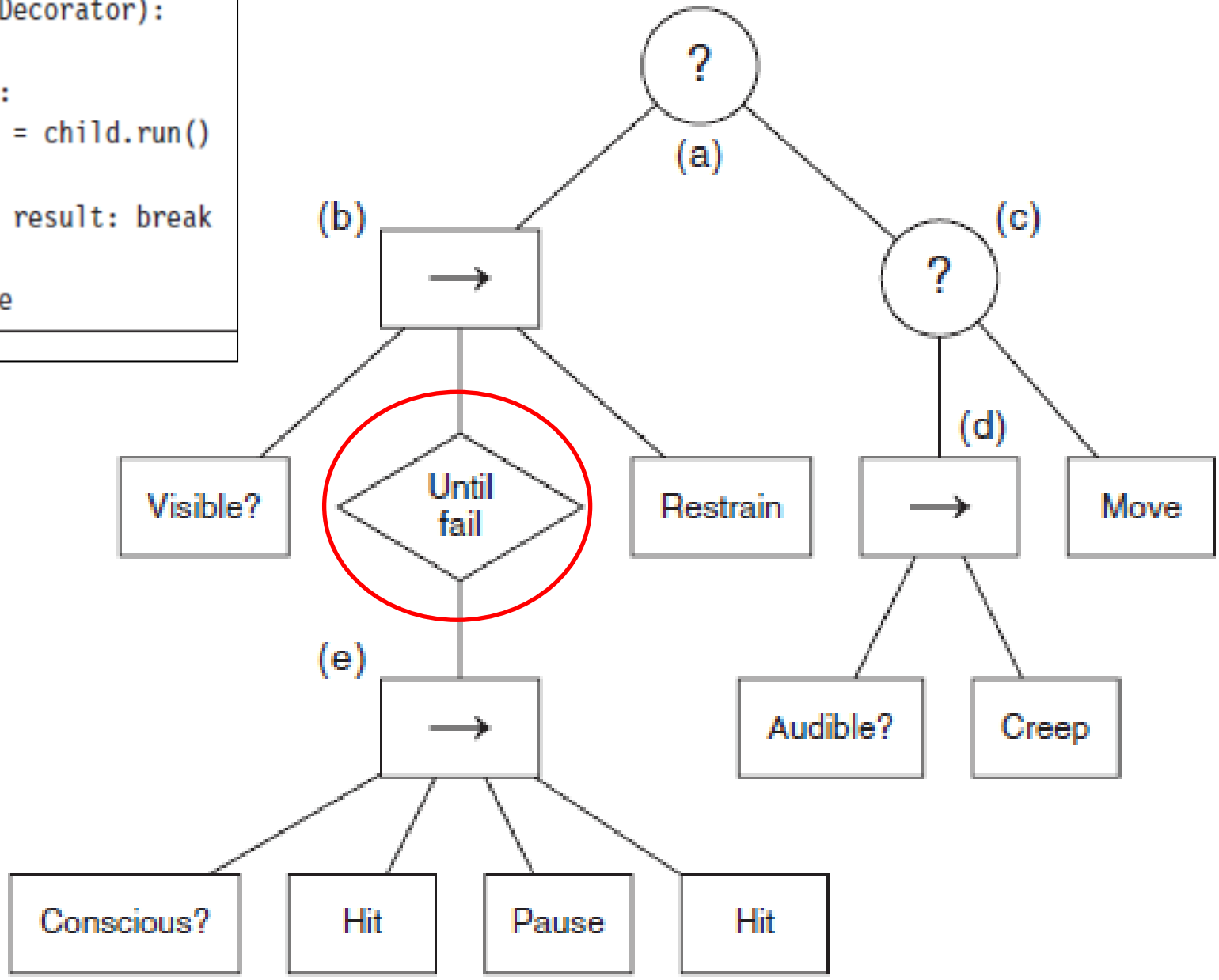

?
Entering...
Open door...
~?
Barge door...
→
⇝
Douse door
Ignite door
Get matches
Get gasoline

Decorators

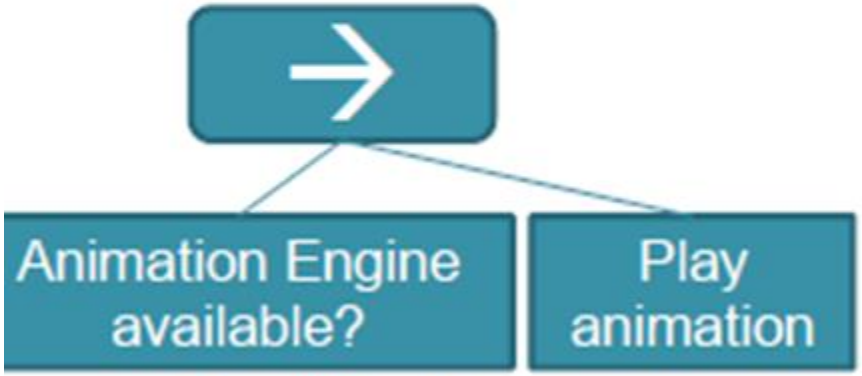
- Has only one child 자식을 갖고 있음.
- Class that wraps another class modifying behavior
- Has same interface as original class

Running “Until Fail”

```
class UntilFail (Decorator):
    def run():
       while True:
           result = child.run()

           if not result: break

       return True
```



- Guarding resources
  - Limited resources
    - Character can have only one animation



Condition Version

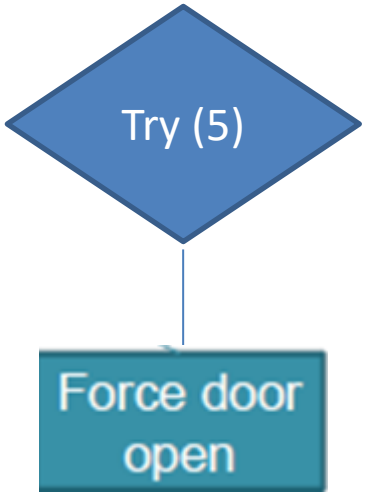
Decorator Version

Guarding a resource using a Condition and Selector

“semaphore 획득”과 같은 기능을 수행

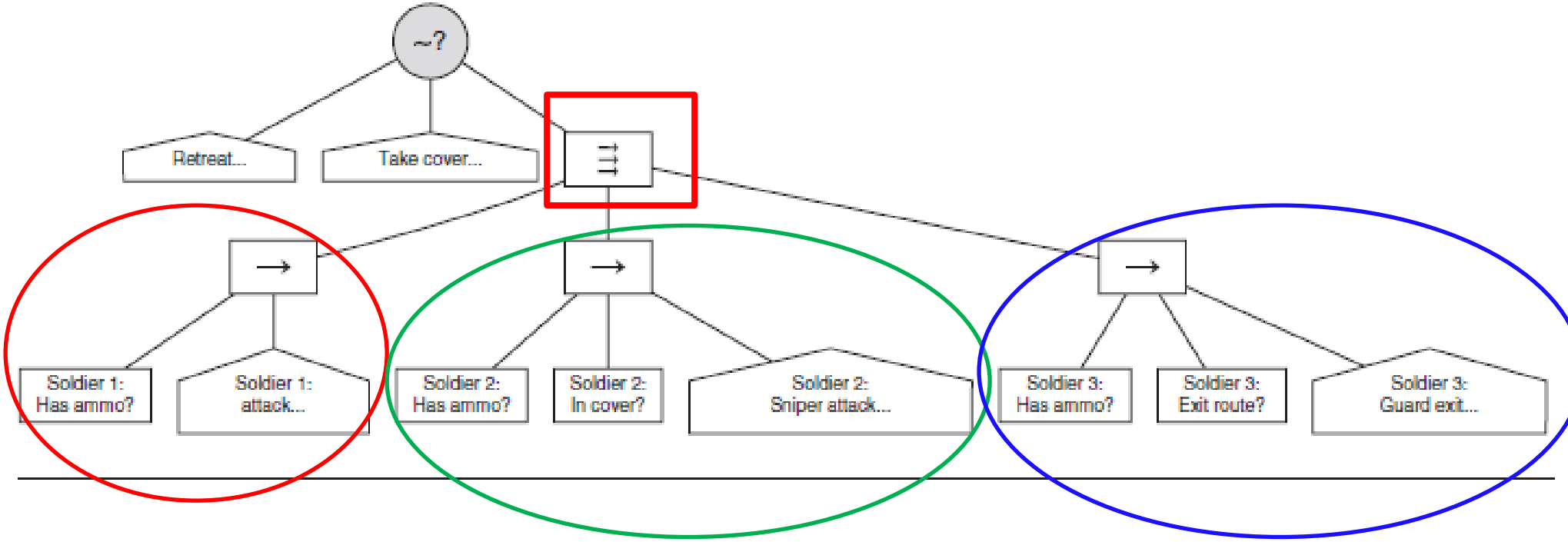
- Limit the number of tries
  - E.g.: Do not try to force open a door to often

```
class Limit (Decorator)
    runLimit
    runSoFar = 0

    def run():
        if runSoFar >= runLimit:
            return False

        runSoFar++
        return child.run()
```



- Inverter:
  Inverts the result of its child node and return it.

Concurrency



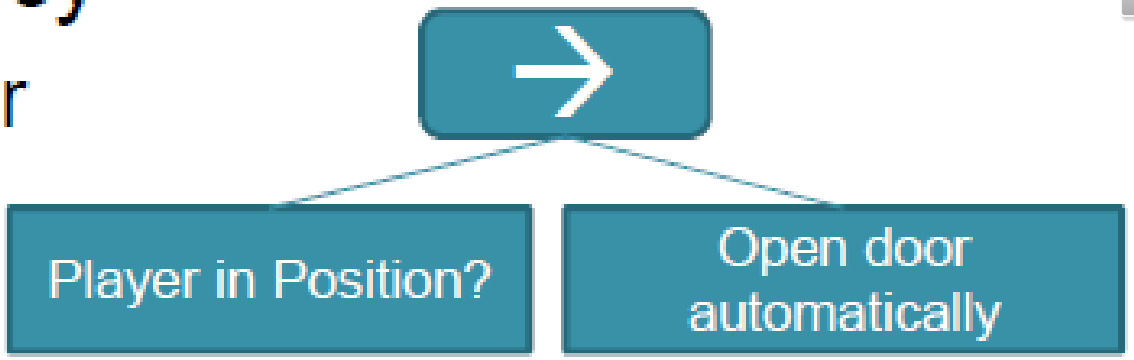
Using Parallel to implement group behavior

Concurrency

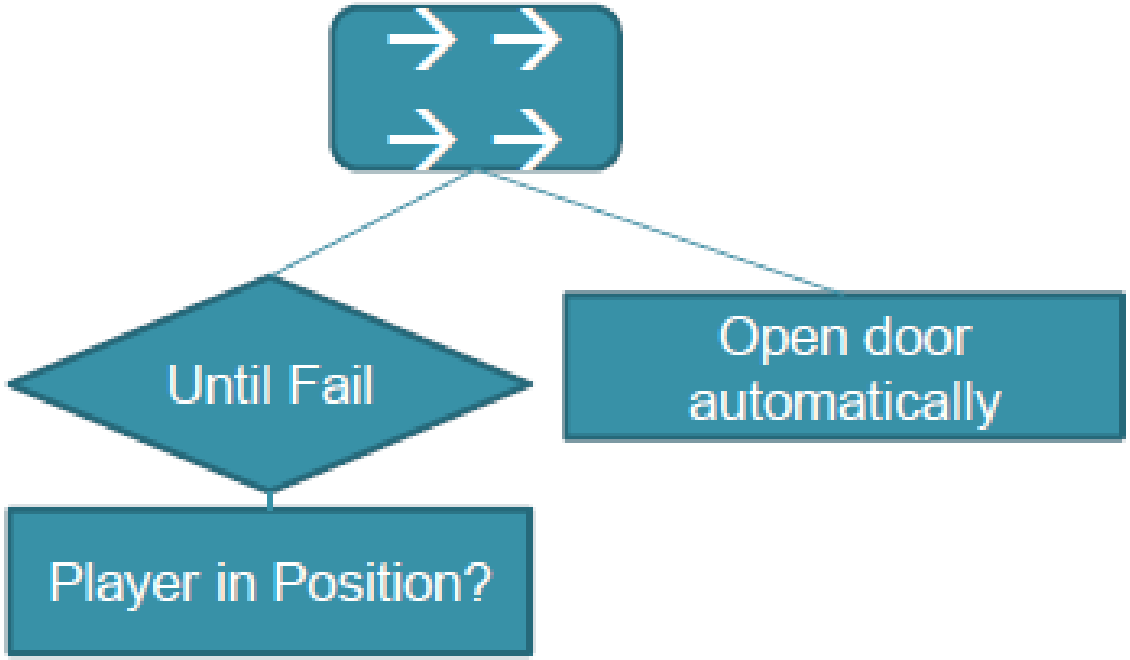
- Behavior trees need to run concurrently
  - Implementing with threads
  - Cooperative multi-tasking
    - Use a new composite task: Parallel
    - Need to generate a policy for parallel
      - E.g.: Return failure as soon as a child returns with failure (Sequencer Policy)

Parallel node가 언제 (when) 그리고 무엇을(what) return 하느냐?

- Concurrency
  - Sequencer



  - Parallel

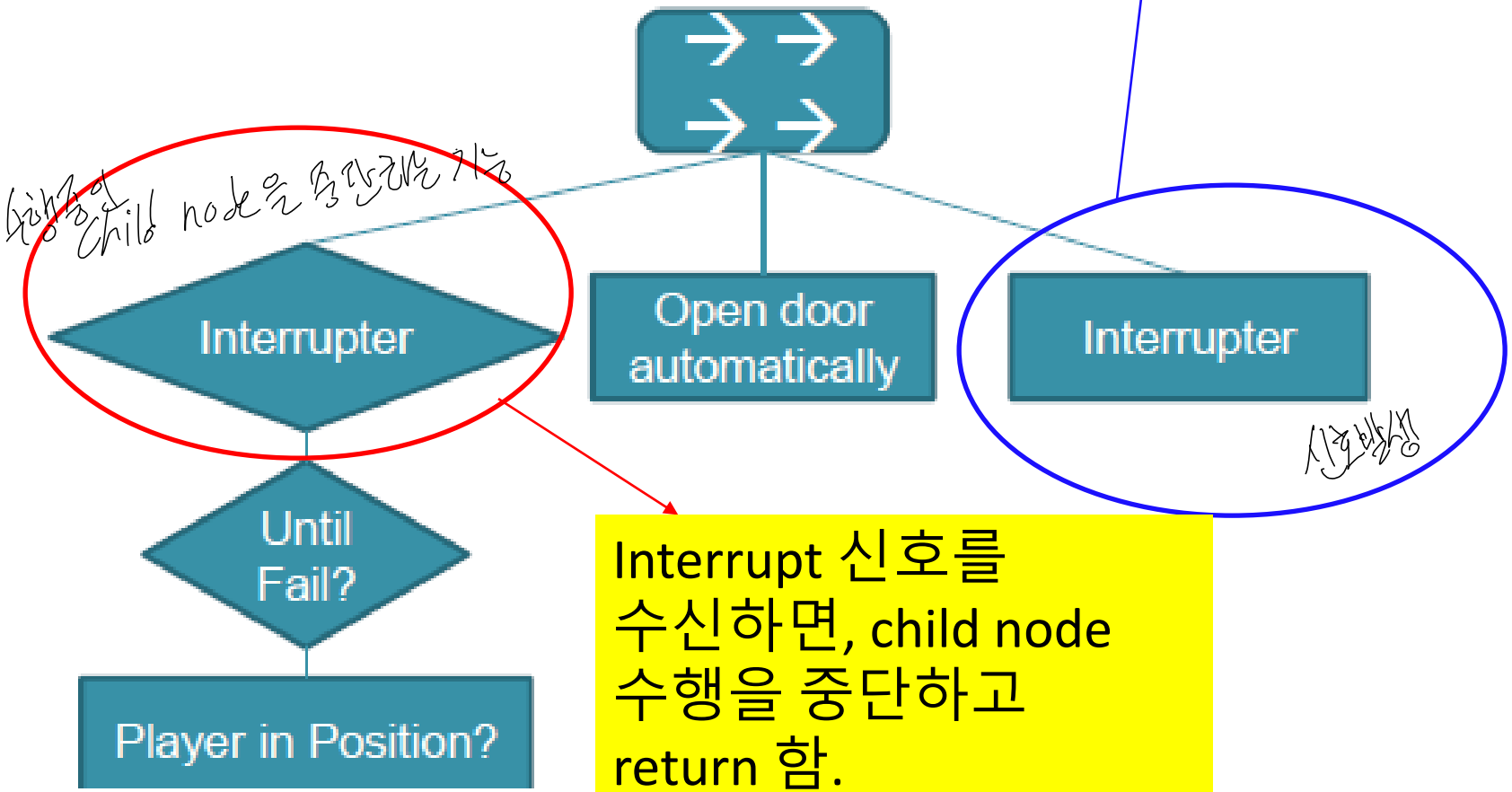


Using Parallel to keep track of Conditions: the Condition is repeatedly checked.

Use two decorator

- An interrupter decorator
  - Passes on success
- An interrupt generator

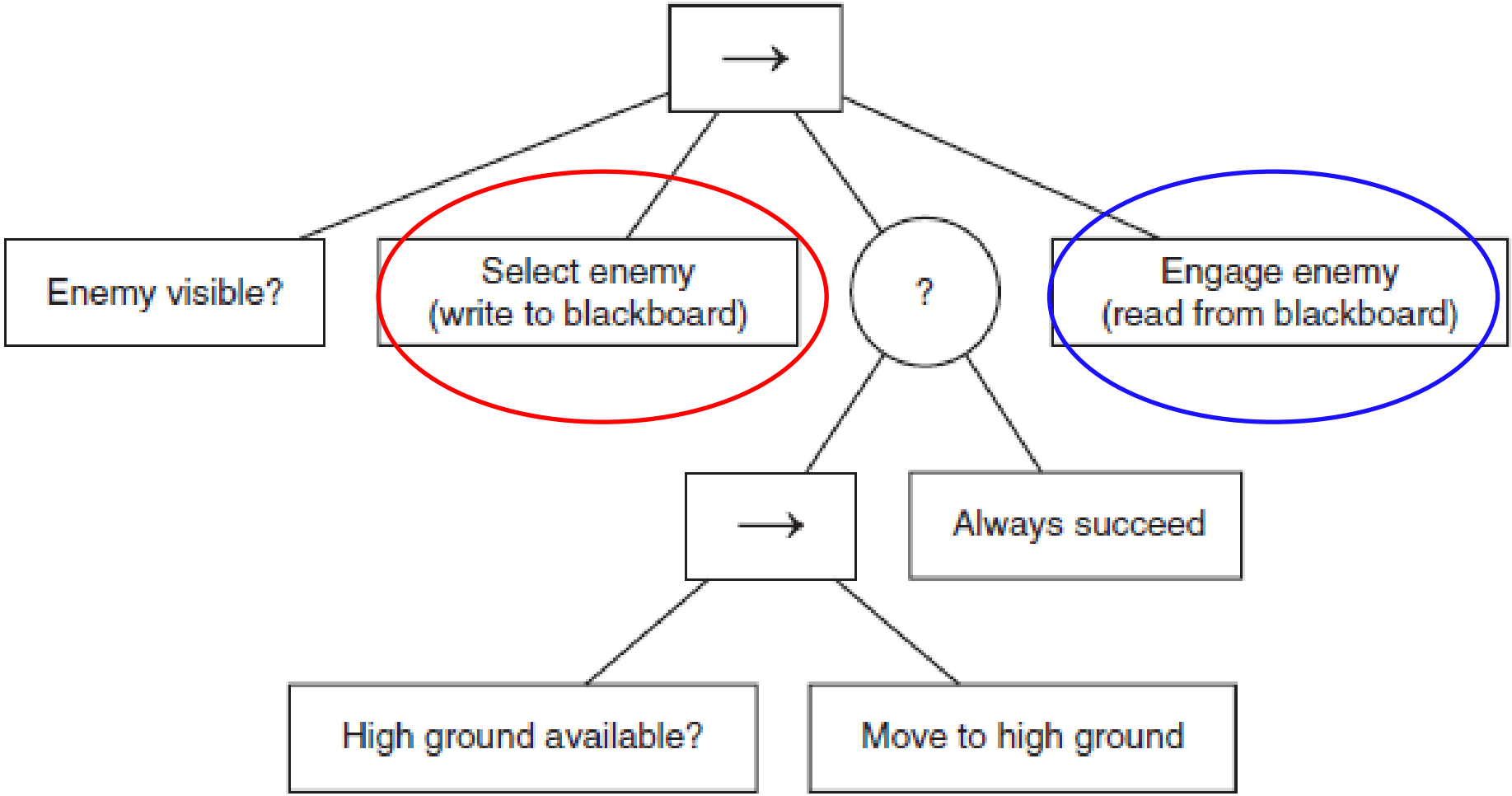
Interrupt
신호를 발생
시킴



Interrupt 신호를
수신하면, child node
수행을 중단하고
return 함.

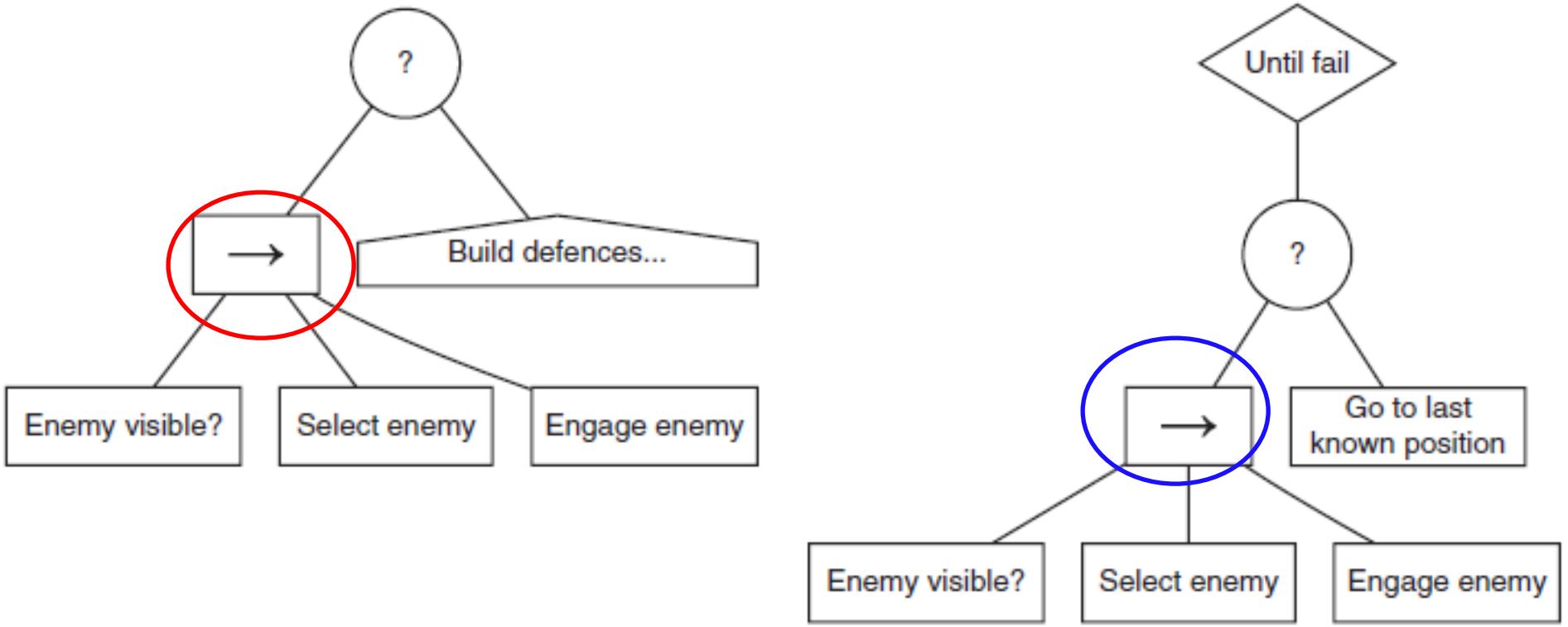
Behavior Trees

- Tasks need to have access to global data
- Passing as parameters generates a difficult API
- Use **blackboard** data structure
  - Write data and messages into a common, global storage structure



A behavior tree communicating via blackboard

Reusing Sub-trees



Common sub-trees across characters:

Use the behavior tree library.

Behavior Trees

Instantiation

- Probably need several copies of a behavior tree for different characters
- Possibilities:

- (1) Define a behavior tree, and instantiate trees for characters
- (2) Create a specification for a behavior tree, and build a tree when needed.
- (3) Use behavior tree tasks that don't keep local state and use separate state objects

MARKOV SYSTEMS

Markov System

- State of system is given as an array(vector) of values.
- Values change by multiplying with a transition matrix vector

➔

- State Vector: event prob., priority, some attribute score, etc. ➔ the sum of all entries can be 1 (of prob.) or not
- Transition Matrix: how to change state vector

Markov System

(Example)

Vector represents safety of 5 sniping positions (sniping:저격)

Taking a shot from the first position alerts enemy to

- Presence of a sniper
- Approximate location

Safety points will change.

Transpose(M) x V ➔ V'

a state vector V represents the safety of four sniping positions:

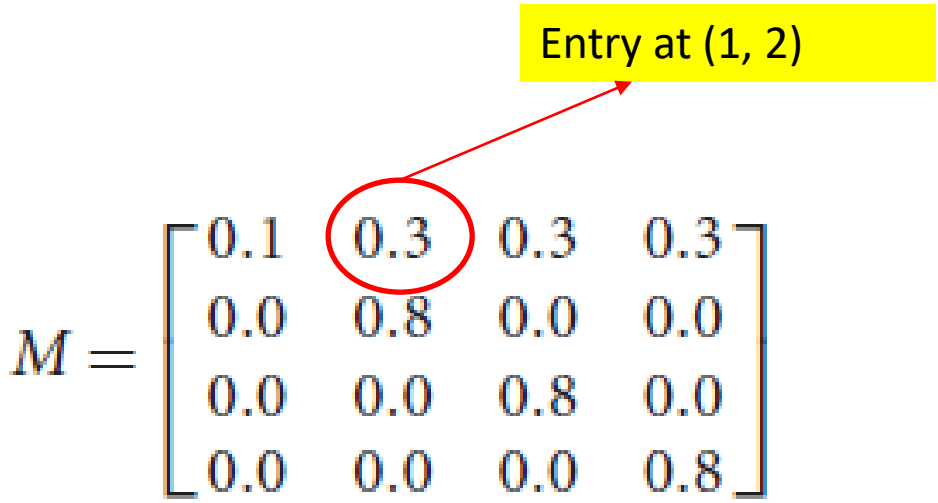
$$V = \begin{bmatrix} 1.0 \\ 0.5 \\ 1.0 \\ 1.5 \end{bmatrix}, \quad M = \begin{bmatrix} 0.1 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{bmatrix}$$

$$\mathbf{V'} = \begin{bmatrix} 0.1 \\ 0.7 \\ 1.1 \\ 1.5 \end{bmatrix},$$

Taking a shot from the first position will alert the enemy to its existence.

The safety of that position will diminish. But, while the enemy is focusing on the direction of the attack, the other positions will be correspondingly safer.

첫번째 entry의 값은 줄고, 그 외 emtry 들의 값은 증가 또는 변동 없음

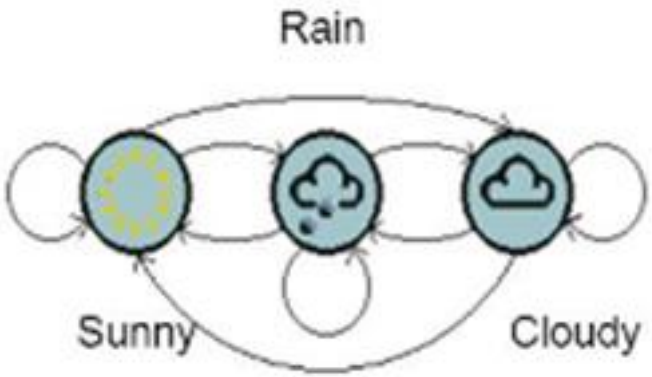


Transition matrices are always square. The element at *(i, j)* in the matrix represents the proportion of element *i* in the old state vector that is added to element *j* in the new vector.

Markov Chain in Math and Science

The values in the **state vector** are **probabilities for a set of events**, and the **transition matrix**. determines **what probability each event will have at the next trial** given their probabilities at the last trial.

ex



weather yesterday \ weather today	Sun	Cloud	Rain
Sun	0.5	0.25	0.25
Cloud	0.375	0.125	0.375
Rain	0.125	0.625	0.375

Sun	Cloud	Rain
1.0	0.0	0.0

대일 Sun could Rain 확률?
벡터와 matrix를 곱하면 됨
10일 이후 확률을 10번 matrix 곱함

state vector 예제

Markov State Machine as A Decision Making Tool

- Implementation
  - Transitions
    - belong to the whole state machine
    - triggered by conditions
    - apply transition matrix to state vector
  - Default Transition
    - Triggered by inaction for a certain time
      - Run a timer whenever no transaction fires
  - Actions
    - Not controlled by state, but possibly by transitions
    - Indirect through interpretation of state vector

Each transition = condition + transition matrix.

(We are not a specific state, unlike a normal FSM)

음성 강의 종료