Lecture #16. 인공지능(행동트리)

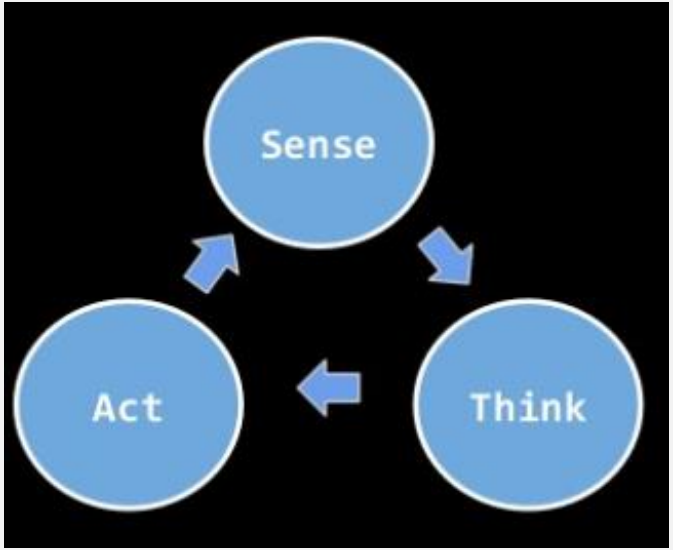
2D 게임 프로그래밍

이대현 교수

한국공학대학교
TECH UNIVERSITY OF KOREA

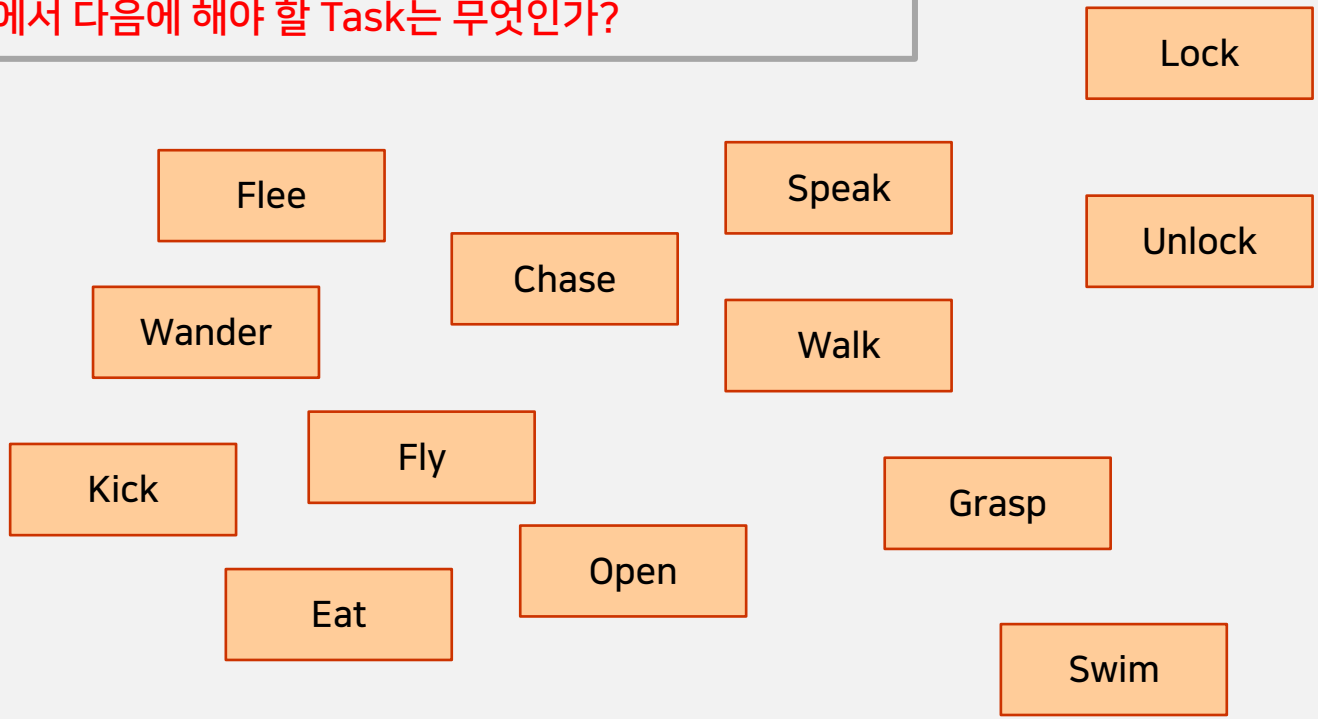
게임 인공 지능

- 게임 객체는 주변의 상황을 인식(Sens)
- 인식된 결과를 바탕으로 행동을 결정(Think)
- 실제로 행동을 수행함(Act)



Key Problems

Agent가 수행할 수 있는 수많을 Task들이 있을 때, Agent의 AI 를 구현하기 위해 Task들을 어떤 순서로 실행할 것인가?
현 상황에서 다음에 해야 할 Task는 무엇인가?



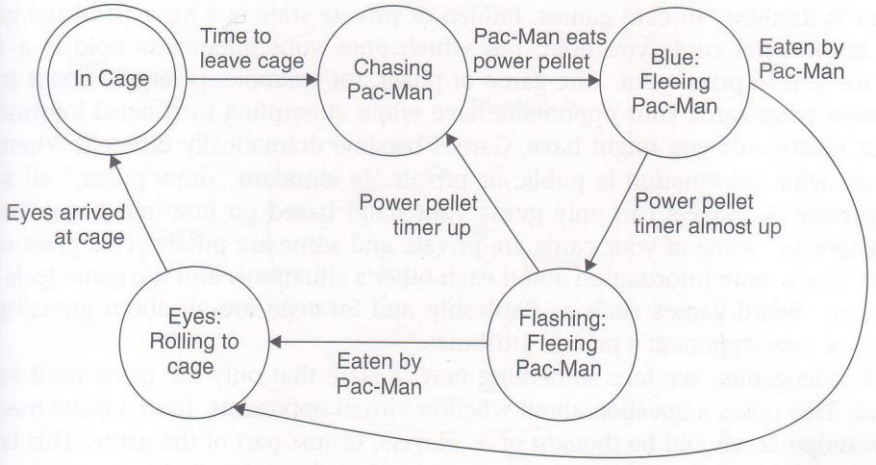
의사 결정의 구현을 어떻게?

- 로직을 하드 코딩할 수도 있음.
  - 게임에 종속됨.
  - 동일한 코드를 여기저기 복사해서 쓰게 됨.

- NPC의 의사 결정을 좀 더 구조적으로 할 수 있는 방법이 필요.

FSM – 가장 전통적인 게임 AI 구현 방식

- 시스템의 변화를 모델링하는 다이어그램.
- 사건이나 시간에 따라 시스템 내의 객체들이 자신의 상태(state)를 바꾸는 과정을 모델링함.
- 상태의 개수가 늘어남에 따라, 와이어링(이벤트의 변화 추적)이 복잡해짐.
- 정확히 상태를 분리해서, 추출하는 것이 어려움.
- HFSM(Hierachical FSM)이 실전에서는 사용됨.



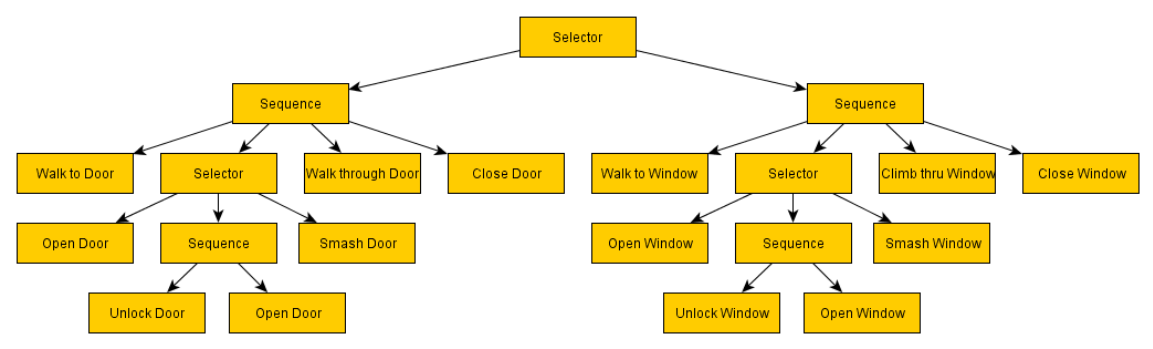
Behavior Tree

- 객체의 인공지능행동을 트리 구조로 구현한 것.
- FSM 방식 – 상태와 이벤트에 따라서, 다음 상태를 결정
- BT 방식 – Goal 을 달성하기 위한 Task들을 구성. 재사용이 쉽고 직관적임.
- HALO 에서 사용된 후, 기본 구조가 공개됨.
- GTA 등에서도 사용

https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

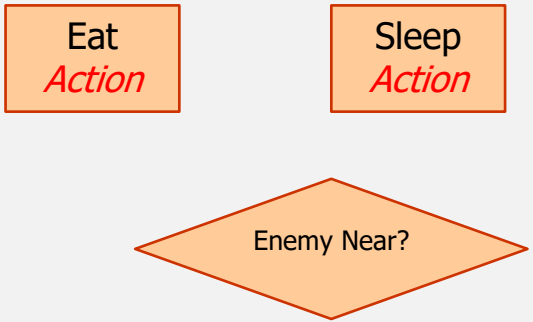
기본 구조

- 트리 구조
  - 말 그대로, 객체의 행위들을 tree 구조로 연결하여 나타냄.
- 매 프레임마다 tree 구조가 실행됨.
  - Root node 부터 시작해서, 아래로 실행되어 나감.
- node는 상태값을 반환함.
  - SUCCESS, FAIL, RUNNING
- Node가 자식 노드가 있으면, 자식 노드들을 실행하고, 그 결과를 종합하여 노드의 최종 상태 값을 결정함.



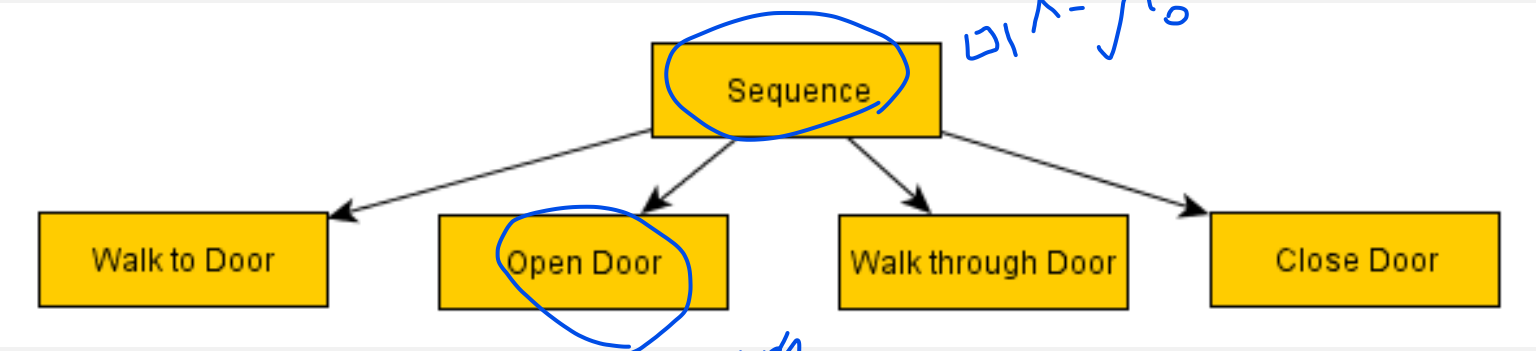
Leaf Node

- 단위 작업을 수행하는 노드로써, Action 또는 Condition 처리.
- Action
  - 어떤 일을 수행함.
  - 이동, 공격 등등
  - 목적을 달성하기 위해서 "매 프레임마다 해야 할 일”을 담음.
  - 수행 결과는 세 종류 : SUCCESS, FAIL, RUNNING(Task의 수행이 진행 중임. SUCCESS/FAIL 판단 유보)
- Condition
  - 여러가지 주변 상황, 상태등을 검사함.
  - 주인공과의 거리, 장애물 상태, 아이템 속성 등등
  - 조건 검사 결과, SUCCESS 또는 FAIL을 return함.



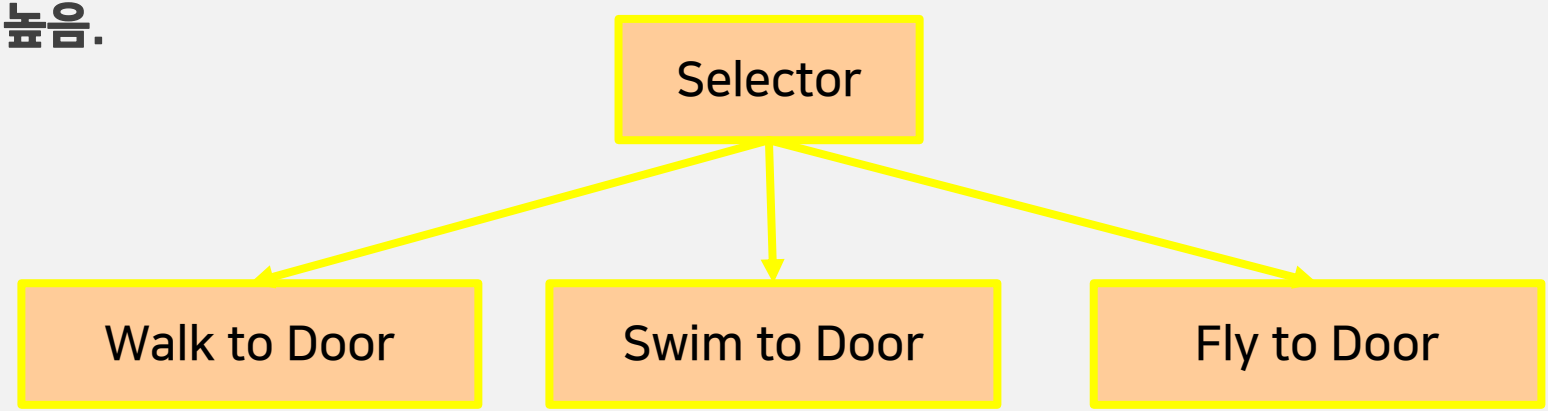
Sequence Node

- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하며서 실행됨.
- 모든 자식 노드가 다 SUCCESS 되면, 노드도 성공
- 여러 개의 작업이 모두 다 차근 차근 진행되어야 하는 경우 – AND 조건
- 하나라도 FAIL 되면, 실행 중단. Sequence Node 도 FAIL
- 실행 결과, 처음으로 RUNNING이 나오면, 자식 노드의 위치를 기록함. 결과는 RUNNING임.
- 어떤 목표를 달성하기 위해 수행해야 하는 Task 들을 차례로 모두 완수해야 하는 경우에 사용됨.

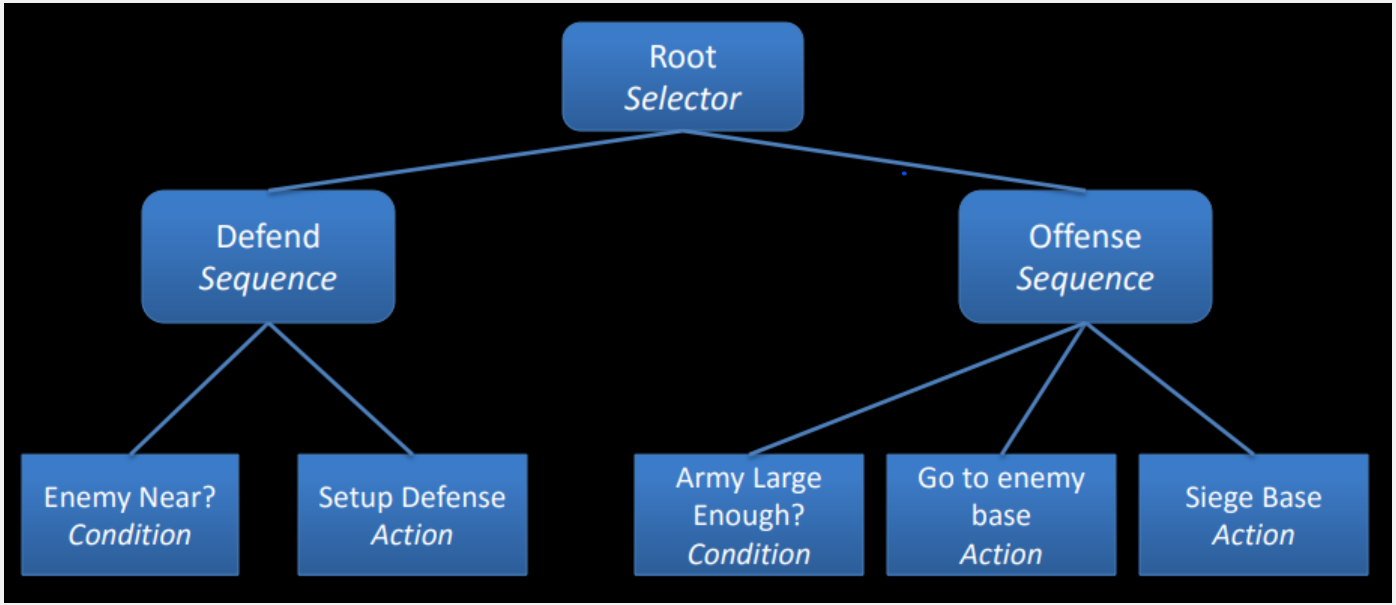


Selector Node

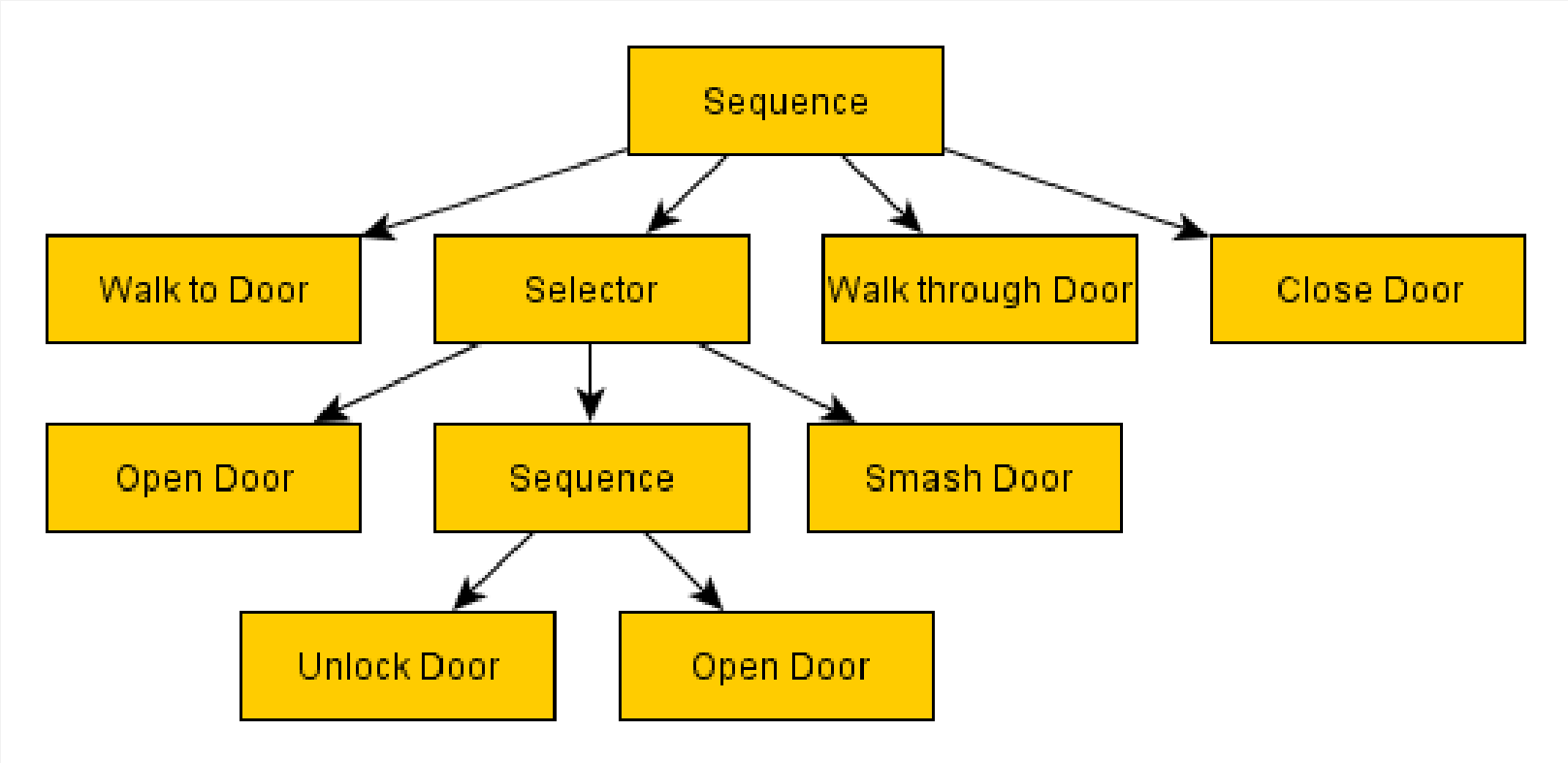
- 자식 노드 중, 하나만 성공하면 성공
- 여러 개의 작업 중, 하나를 선택하는 개념 – OR
- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하면서 실행됨.
- 실행 결과 처음으로 SUCCESS, 또는 RUNNING이 나오면 더 이상 진행되지 않으며, 노드의 결과는 SUCCESS 또는 RUNNING 이 됨.
- 모든 자식 노드가 다 FAIL이면, 노드의 결과도 FAIL임.
- 작업에 우선 순위를 부여할 때 사용됨. 즉, 왼쪽에 있는 노드가, 오른쪽에 있는 노드보다 우선 순위가 높음.



BT 예제 #1

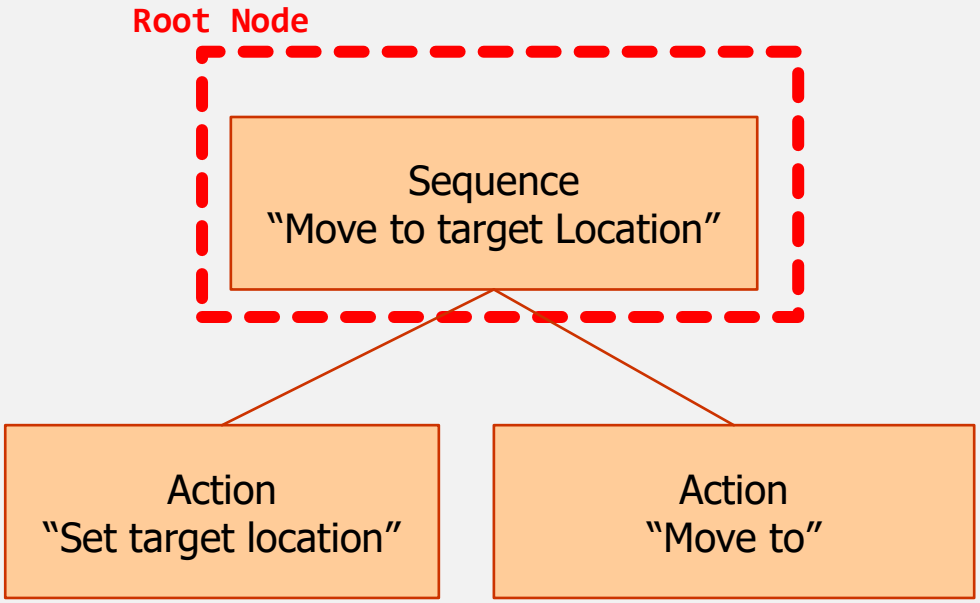


BT 예제 #2



Move To Target Location

목표 Behavior Tree



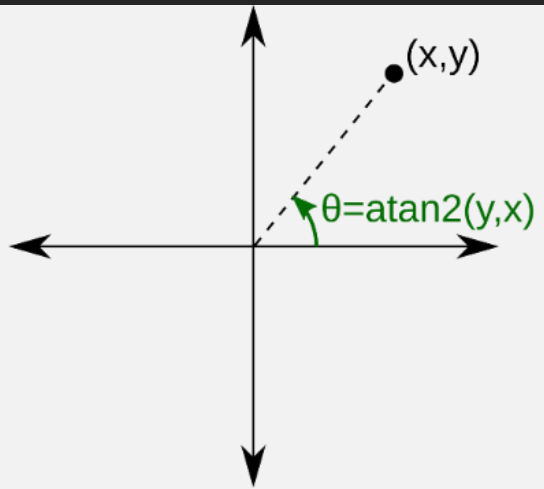
Action “Set Target Location” 구현

```
def set_target_location(self, x=None, y=None):
    if not x or not y:
        raise ValueError('Location should be given')
    self.tx, self.ty = x, y
    return BehaviorTree.SUCCESS
```

Action “Move To” 구현 (1)

```
def distance_less_than(self, x1, y1, x2, y2, r):
    distance2 = (x1 - x2) ** 2 + (y1 - y2) ** 2
    return distance2 < (PIXEL_PER_METER * r) ** 2

def move_slightly_to(self, tx, ty):
    self.dir = math.atan2(ty - self.y, tx - self.x)
    self.speed = RUN_SPEED_PPS
    self.x += self.speed * math.cos(self.dir) * game_framework.frame_time
    self.y += self.speed * math.sin(self.dir) * game_framework.frame_time
```

dir 을 radian 으로 해석.



Action “Move To” 구현 (2)

```
def move_to(self, r=0.5):
    self.state = 'Walk'
    self.move_slightly_to(self.tx, self.ty)
    if self.distance_less_than(self.tx, self.ty, self.x, self.y, r):
        return BehaviorTree.SUCCESS
    else:
        return BehaviorTree.RUNNING
```

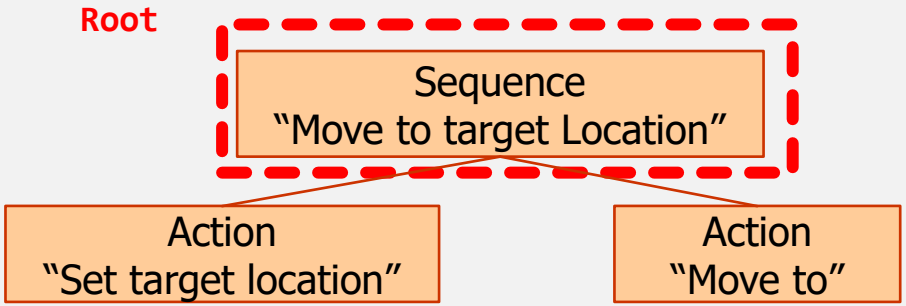
Behavior Tree 구성

```
def build_behavior_tree(self):
        a1 = Action('Set target location', self.set_target_location, 500, 50)

    a2 = Action('Move to', self.move_to)

    root = SEQ_move_to_target_location = Sequence('Move to target location', a1, a2)

    self.bt = BehaviorTree(root)
```

Action none 를 생성

self.set_target_location 함수를 연결

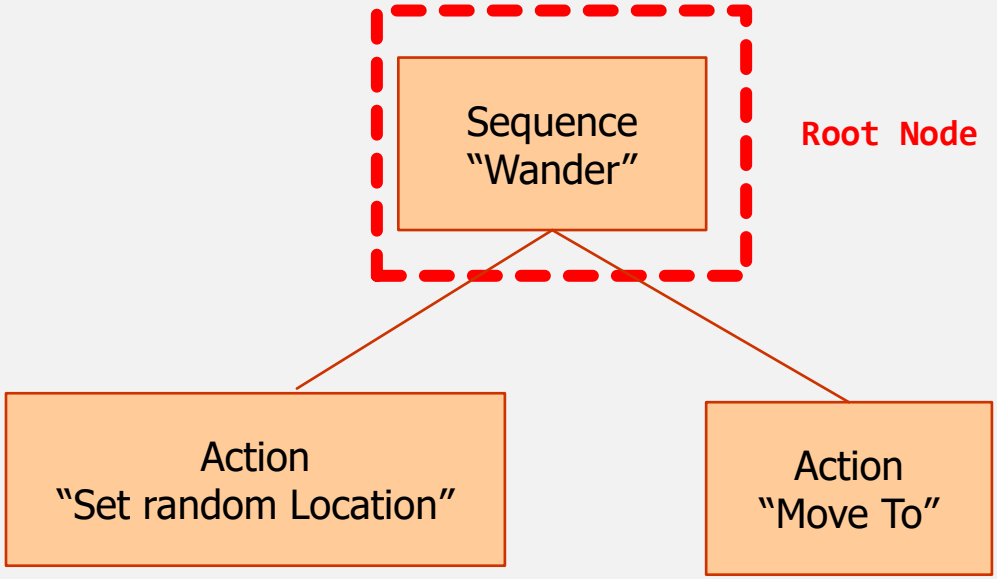
Sequence 노드를 BT의 root 로 지정.



Behavior Tree 실행

```
def update(self):
    self.frame = (self.frame + FRAMES_PER_ACTION * ACTION_PER_TIME * game_framework.frame_time) % FRAMES_PER_ACTION
    self.bt.run()
```

Wander BT 버전

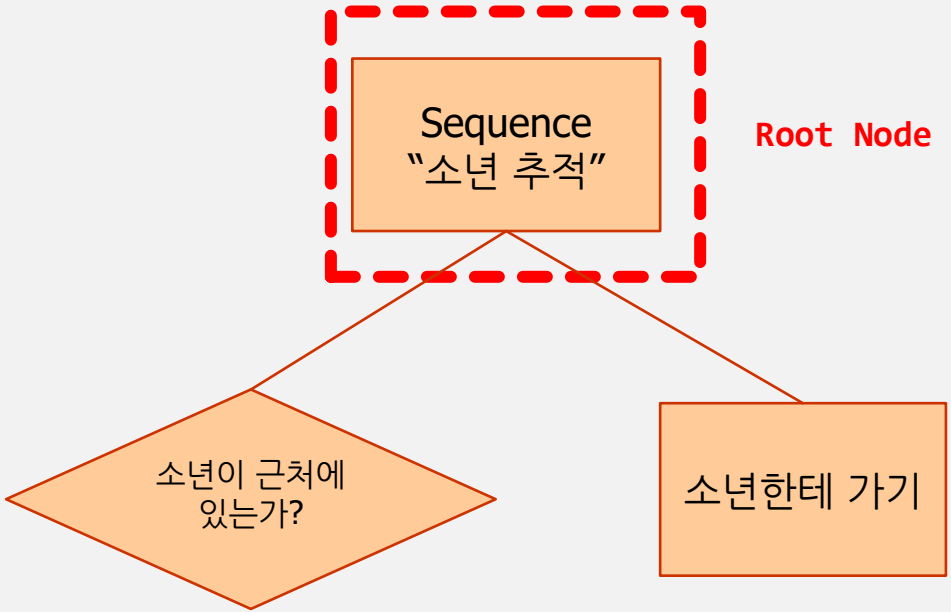


Wander(배회) BT 구성


```
def set_random_location(self):
    self.tx, self.ty = random.randint(100, 1280 - 100), random.randint(100, 1024 - 100)
    return BehaviorTree.SUCCESS


def build_behavior_tree(self):
    a1 = Action('Set target location', self.set_target_location, 500, 50)
    a2 = Action('Move to', self.move_to)
    root = self.SEQ_move_to_target_location = Sequence('Move to target location', a1, a2)

    a3 = Action('Set random location', self.set_random_location)
    root = SEQ_wander = Sequence('Wander', a3, a2)

    self.bt = BehaviorTree(root)
```

소년 추적 좀비 BT

소년 추적 BT



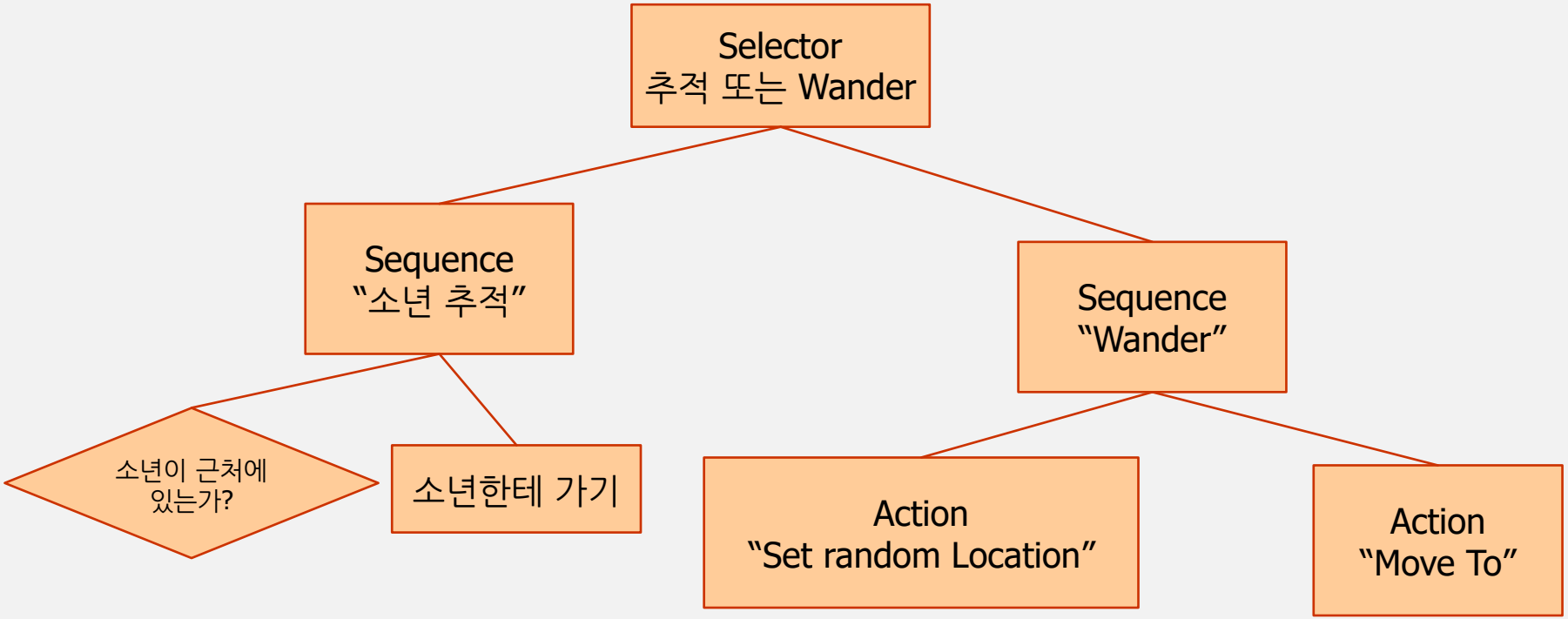
Condtion 과 Action 구현

```
def is_boy_nearby(self, r):
    if self.distance_less_than(play_mode.boy.x, play_mode.boy.y, self.x, self.y, r):
        return BehaviorTree.SUCCESS
    else:
        return BehaviorTree.FAIL

def move_to_boy(self, r=0.5):
    self.state = 'Walk'
    self.move_slightly_to(play_mode.boy.x, play_mode.boy.y)
    if self.distance_less_than(play_mode.boy.x, play_mode.boy.y, self.x, self.y, r):
        return BehaviorTree.SUCCESS
    else:
        return BehaviorTree.RUNNING
```

BT 구성

```
def build_behavior_tree(self):
    a1 = Action('Set target location', self.set_target_location, 500, 50)
    a2 = Action('Move to', self.move_to)
    root = SEQ_move_to_target_location = Sequence('Move to target location', a1, a2)

    a3 = Action('Set random location', self.set_random_location)
    root = SEQ_wander = Sequence('Wander', a3, a2)

    c1 = Condition('소년이 근처에 있는가?', self.is_boy_nearby, 7)
    a4 = Action('소년한테 접근', self.move_to_boy)
    root = SEQ_chase_boy = Sequence('소년을 추적', c1, a4)

    self.bt = BehaviorTree(root)
```

배회하다가 소년 발견하면 추적



BT 구성

```
def build_behavior_tree(self):
    a1 = Action('Set target location', self.set_target_location, 500, 50)
    a2 = Action('Move to', self.move_to)
    root = SEQ_move_to_target_location = Sequence('Move to target location', a1, a2)

    a3 = Action('Set random location', self.set_random_location)
    root = SEQ_wander = Sequence('Wander', a3, a2)

    c1 = Condition('소년이 근처에 있는가?', self.is_boy_nearby, 7)
    a4 = Action('접근', self.move_to_boy)
    root = SEQ_chase_boy = Sequence('소년을 추적', c1, a4)

    root = SEL_chase_or_flee = Selector('추적 또는 배회', SEQ_chase_boy, SEQ_wander)

    self.bt = BehaviorTree(root)
```

Patrol BT
순찰 좀비

zombie.py


```
def __init__(self, x=None, y=None):
    self.x = x if x else random.randint(100, 1180)
    self.y = y if y else random.randint(100, 924)
    self.load_images()
    self.dir = 0.0      # radian 값으로 방향을 표시
    self.speed = 0.0
    self.frame = random.randint(0, 9)
    self.state = 'Idle'
    self.ball_count = 0

    self.tx, self.ty = 0, 0
    self.build_behavior_tree()

    self.patrol_locations = [(43, 274), (1118, 274), (1050, 494), (575, 804), (235, 991), (575, 804), (1050, 494),
(1118, 274)]
    self.loc_no = 0
```

순찰 위치 획득

```
def get_patrol_location(self):
    self.tx, self.ty = self.patrol_locations[self.loc_no]
    self.loc_no = (self.loc_no+1) % len(self.patrol_locations)
    return BehaviorTree.SUCCESS
```

BT 구성

```
def build_behavior_tree(self):
    a1 = Action('Set target location', self.set_target_location, 500, 50)
    a2 = Action('Move to', self.move_to)
    root = SEQ_move_to_target_location = Sequence('Move to target location', a1, a2)

    a3 = Action('Set random location', self.set_random_location)
    root = SEQ_wander = Sequence('Wander', a3, a2)

    c1 = Condition('소년이 근처에 있는가?', self.is_boy_nearby, 7)
    a4 = Action('접근', self.move_to_boy)
    root = SEQ_chase_boy = Sequence('소년을 추적', c1, a4)

    root = SEL_chase_or_flee = Selector('추적 또는 배회', SEQ_chase_boy, SEQ_wander)

    a5 = Action('순찰 위치 가져오기', self.get_patrol_location)
    root = SEQ_patrol = Sequence('순찰', a5, a2)
```