Lecture #15. 인공지능(행동트리)

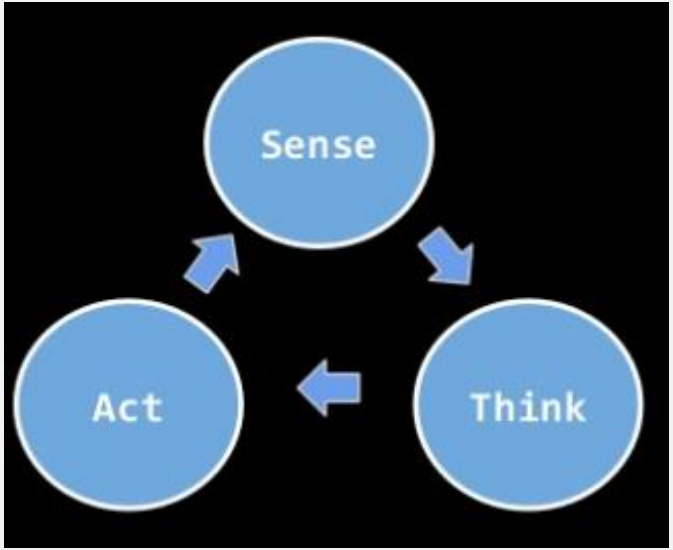
2D 게임 프로그래밍

이대현 교수

한국공학대학교
TECH UNIVERSITY OF KOREA

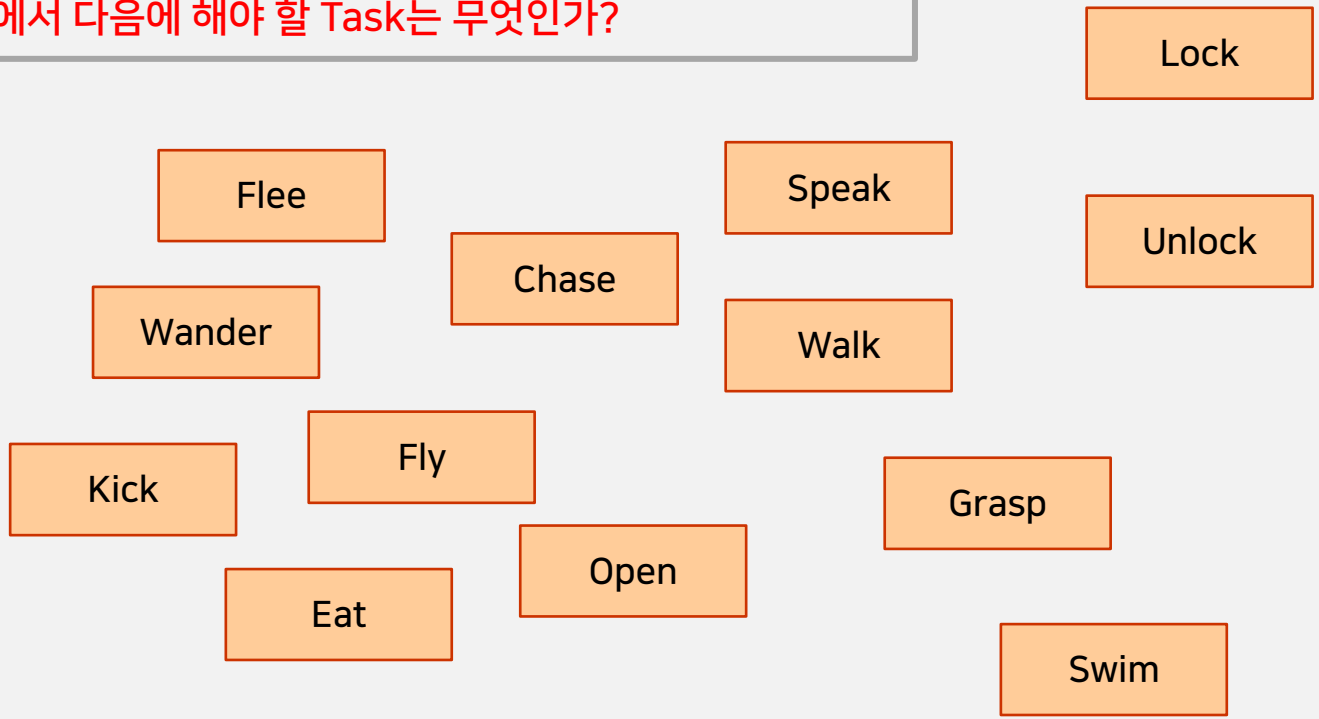
게임 인공 지능

- 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?



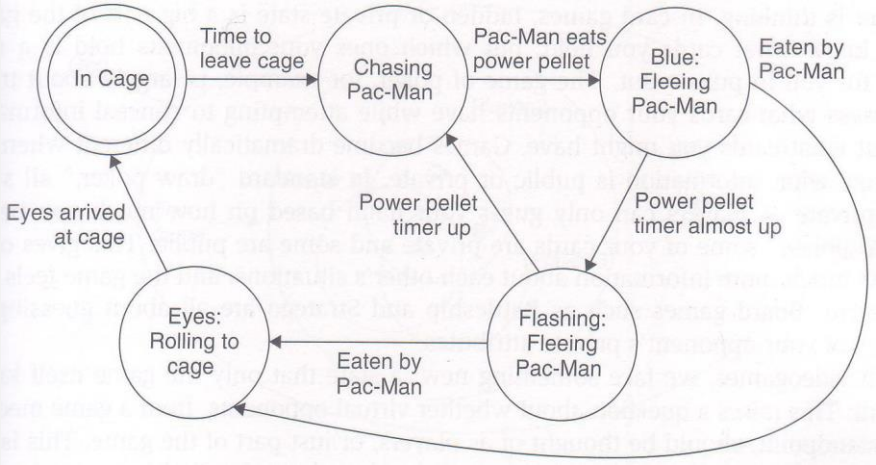
의사 결정의 구현을 어떻게?

- 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.



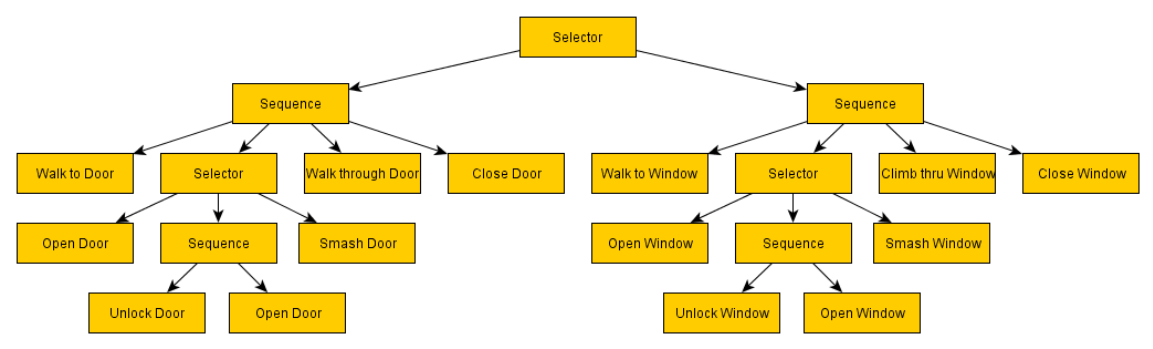
Behavior Tree

- 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

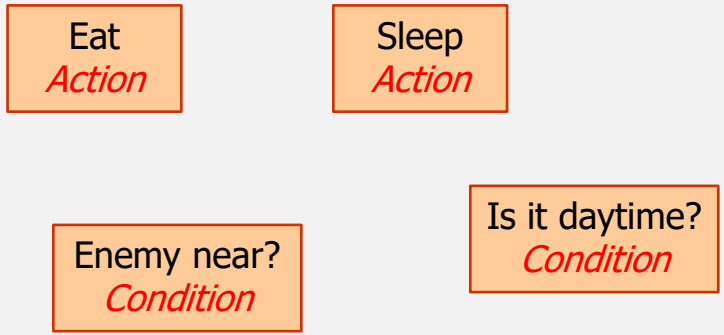
기본 구조

- 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.



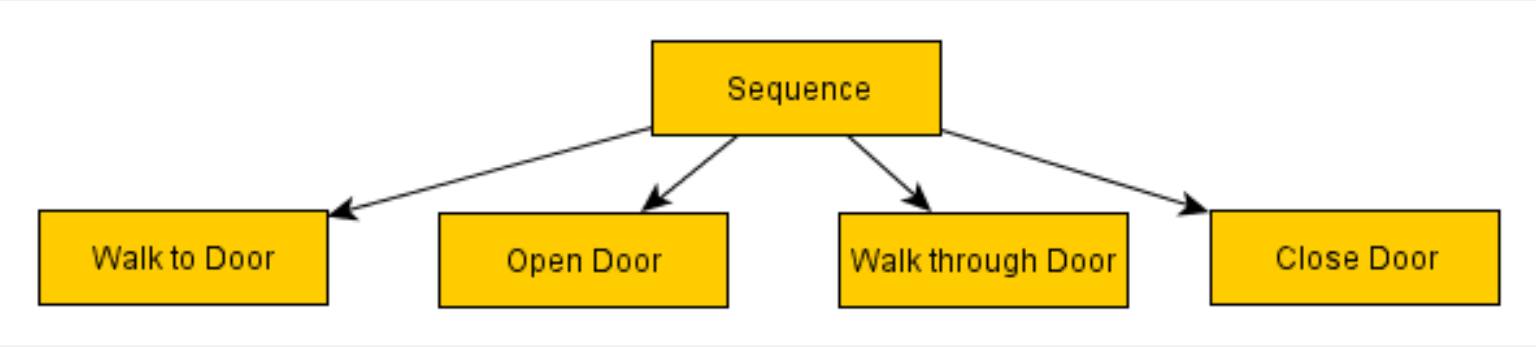
Leaf Node

- 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.



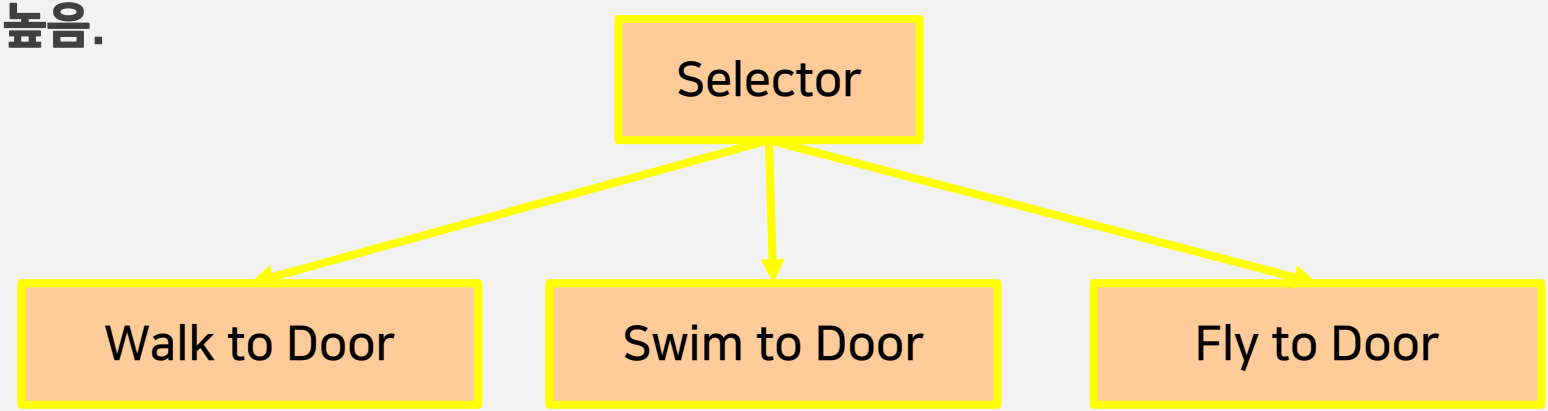
Sequence Node

- 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.

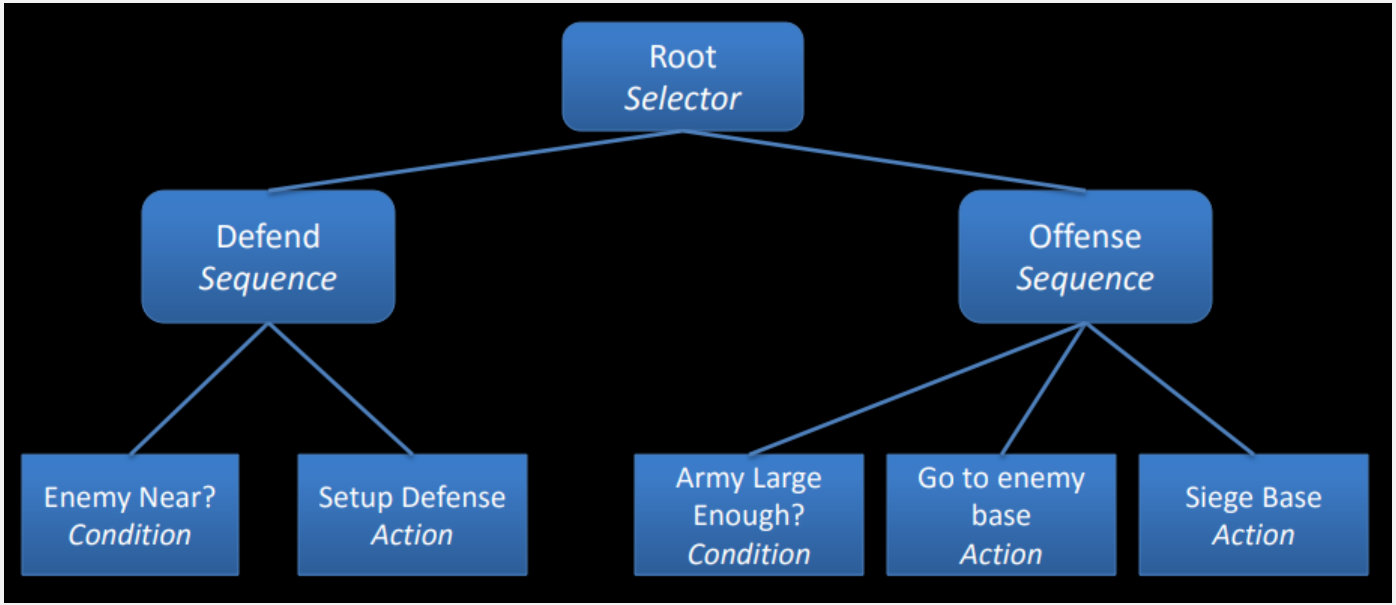


Selector Node

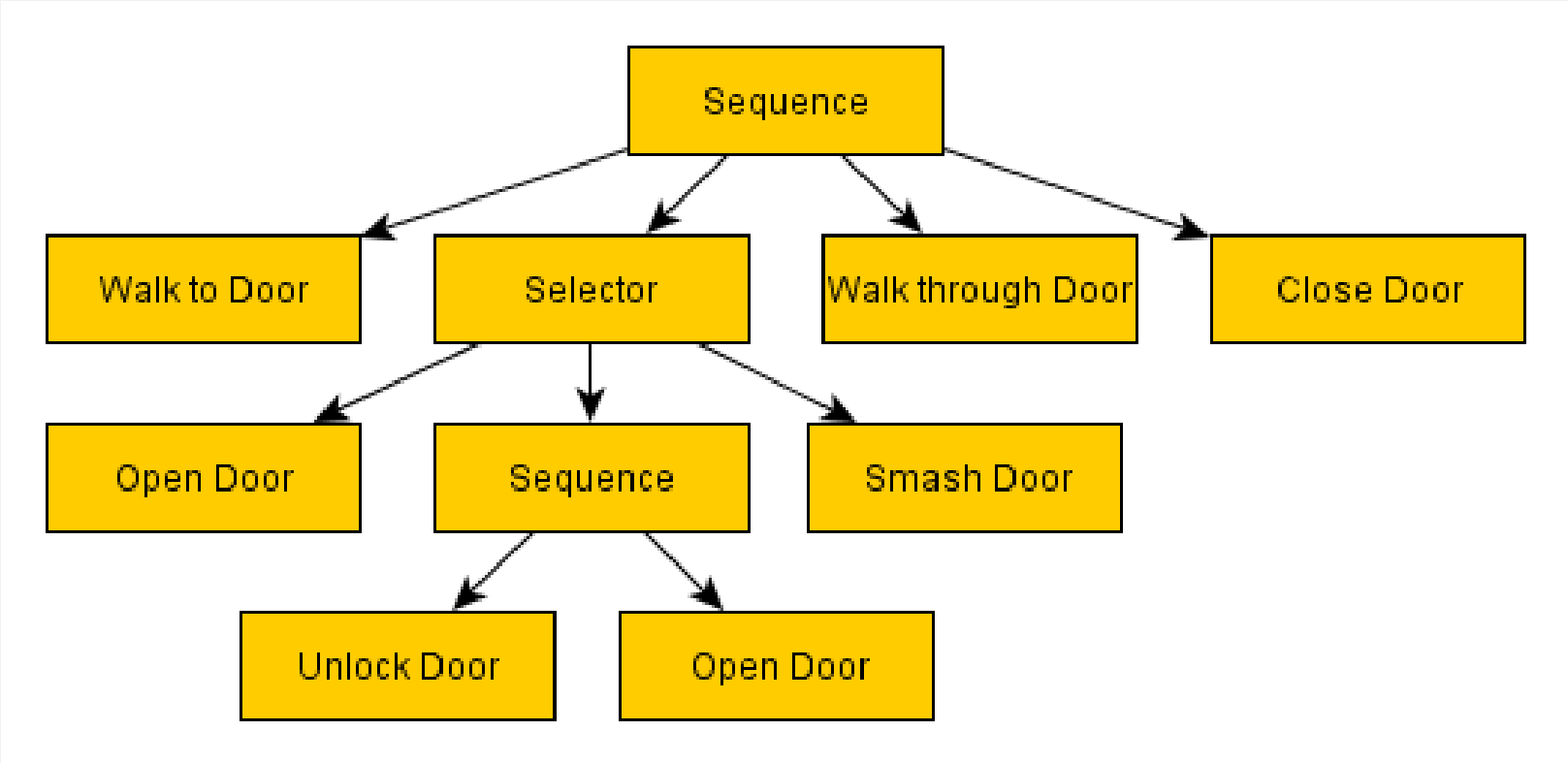
- 자식 노드 중, 하나만 성공하면 성공
- 여러 개의 작업 중, 하나를 선택하는 개념 – OR
- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하면서 실행됨.
- 실행 결과 처음으로 SUCCESS, 또는 RUNNING이 나오면 더 이상 진행되지 않으며, 노드의 결과는 SUCCESS 또는 RUNNING 이 됨.
- 모든 자식 노드가 다 FAIL이면, 노드의 결과도 FAIL임.
- 작업에 우선 순위를 부여할 때 사용됨. 즉, 왼쪽에 있는 노드가, 오른쪽에 있는 노드보다 우선 순위가 높음.



BT 예제 #1



BT 예제 #2



Move To Task

Root Node

zombie.py

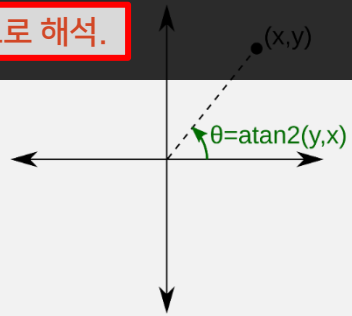

```
def move_to(self, radius = 0.5):
    distance = (self.tx - self.x)**2 + (self.ty - self.y)**2
    self.dir = math.atan2(self.ty - self.y, self.tx - self.x)
    if distance < (PIXEL_PER_METER * radius)**2:
        self.speed = 0
        return BehaviorTree.SUCCESS
    else:
        self.speed = RUN_SPEED_PPS
        return BehaviorTree.RUNNING
```

```
def build_behavior_tree(self):
    move_to_node = Leaf('Move To', self.move_to)
    self.bt = BehaviorTree(move_to_node)
```

```
def update(self):
    self.bt.run()
    self.calculate_current_position()
```

```python
def move_to(self, radius = 0.5):
    distance = (self.tx - self.x)**2 + (self.ty - self.y)**2
    self.dir = math.atan2(self.ty - self.y, self.tx - self.x)
    if distance < (PIXEL_PER_METER * radius)**2:
        self.speed = 0
        return BehaviorTree.SUCCESS
    else:
        self.speed = RUN_SPEED_PPS
        return BehaviorTree.RUNNING
```

dir 을 radian 으로 해석.



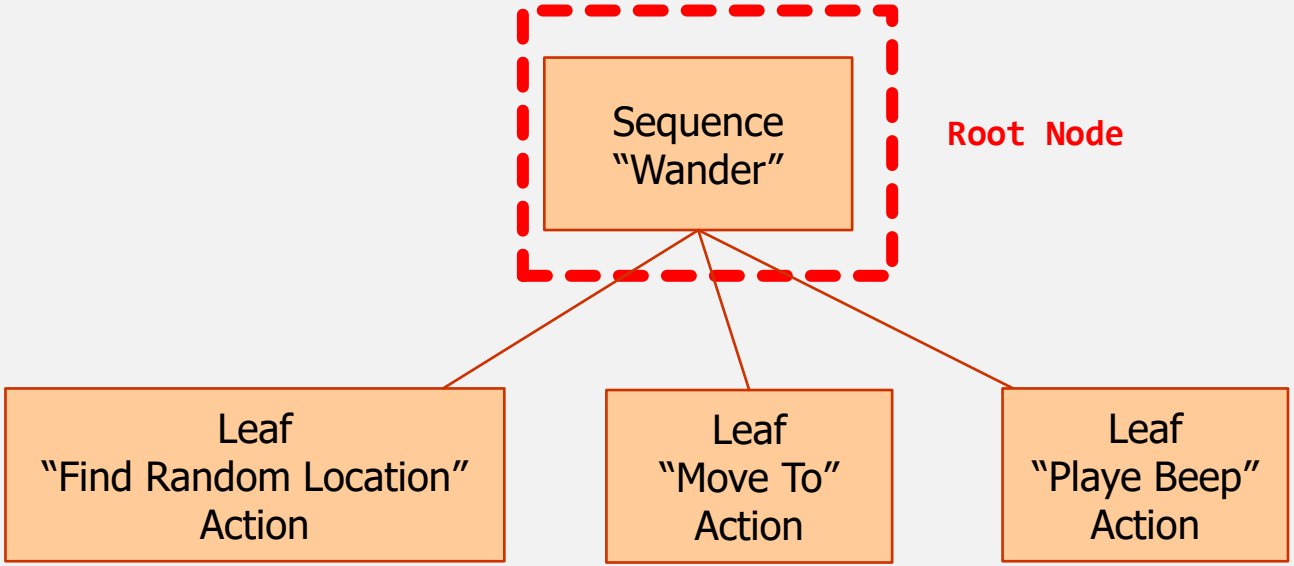
move_to_node를 생성

self.move_to 함수를 연결

```
def build_behavior_tree(self):
    move_to_node = Leaf('Move To', self.move_to)
    self.bt = BehaviorTree(move_to_node)
```

move_to_node를 BT의 root node로 지정.

Wander BT 버전

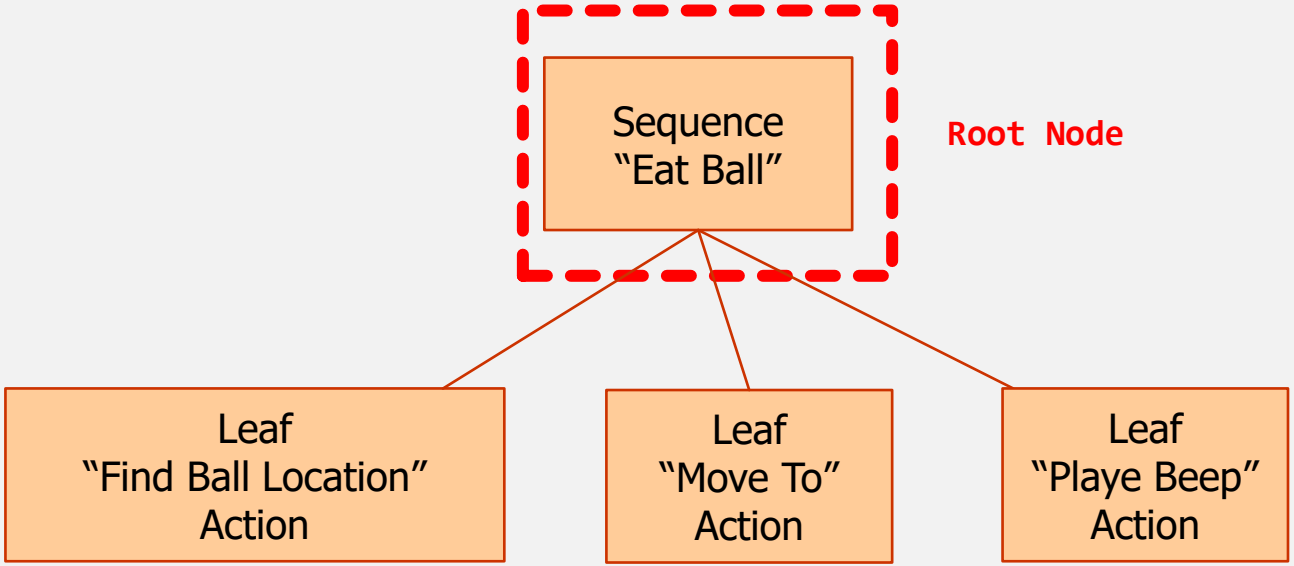


zombie.py


```
def find_random_location(self):
    self.tx, self.ty = random.randint(50, 1230), random.randint(50, 974)
    self.target_marker.x, self.target_marker.y = self.tx, self.ty
    return BehaviorTree.SUCCESS
```

```
def build_behavior_tree(self):
    find_random_location_node = Leaf('Find Random Location', self.find_random_location)
    move_to_node = Leaf('Move To', self.move_to)
    play_beep_node = Leaf('Play Beep', self.play_beep)
    wander_sequence = Sequence('Wander', find_random_location_node, move_to_node, play_beep_node)
    self.bt = BehaviorTree(wander_sequence)
```

Eat Ball



Find Ball Location

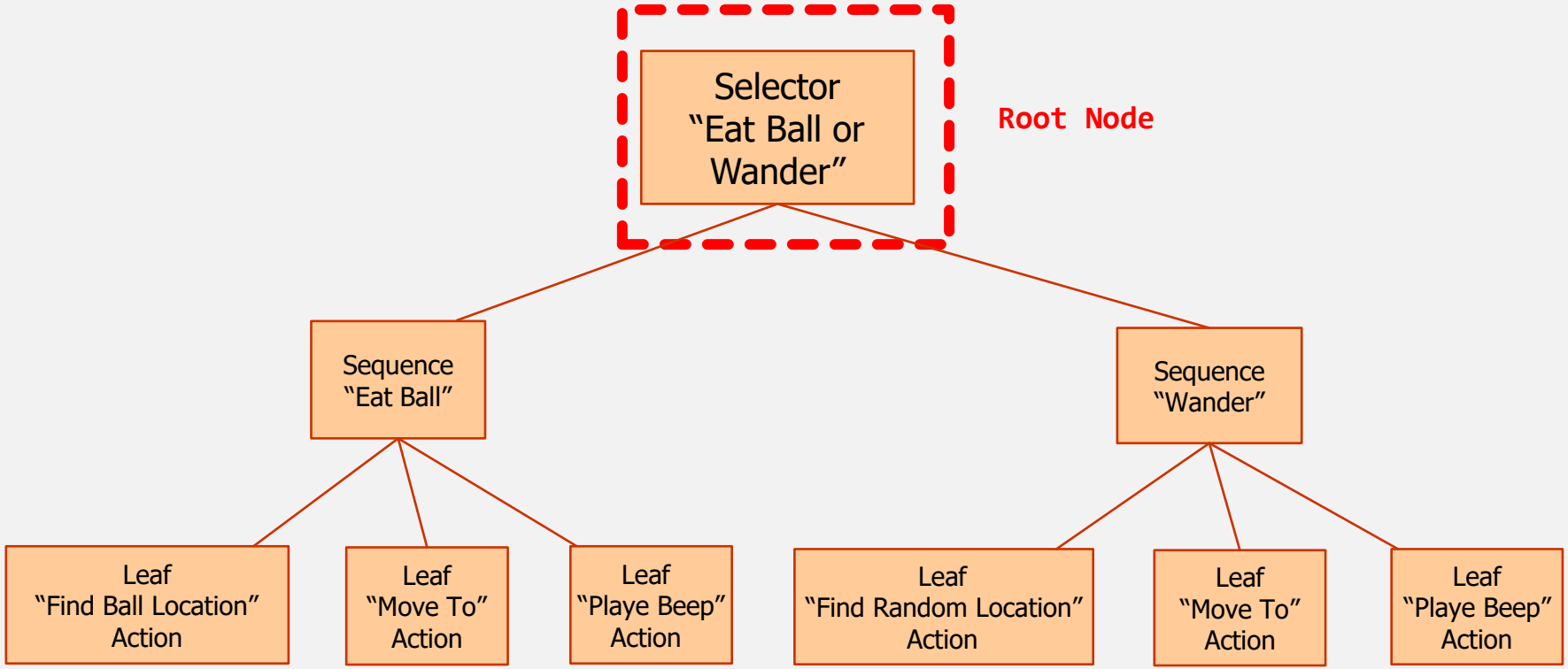

```
def find_ball_location(self):
    self.target_ball = None
    shortest_distance = 1280 ** 2
    # find in-sight(5meters) and nearest ball
    for o in game_world.all_objects():
        if type(o) is Ball:
            ball = o
            distance = (ball.x - self.x)**2 + (ball.y - self.y)**2
            if distance < (PIXEL_PER_METER * 7)**2 and distance < shortest_distance:
                self.target_ball = ball
                shortest_distance = distance
    if self.target_ball is not None:
        self.tx, self.ty = self.target_ball.x, self.target_ball.y
        return BehaviorTree.SUCCESS
    else:
        return BehaviorTree.FAIL
```

BT 구성


```
def build_behavior_tree(self):
    find_random_location_node = Leaf('Find Random Location', self.find_random_location)
    move_to_node = Leaf('Move To', self.move_to)
    play_beep_node = Leaf('Play Beep', self.play_beep)
    wander_sequence = Sequence('Wander', find_random_location_node, move_to_node, play_beep_node)

    find_ball_location_node = Leaf('Find Ball Location', self.find_ball_location)
    eat_ball_sequence = Sequence('Eat Ball', find_ball_location_node, move_to_node, play_beep_node)
    self.bt = BehaviorTree(eat_ball_sequence)
```

Wander or Eat Ball



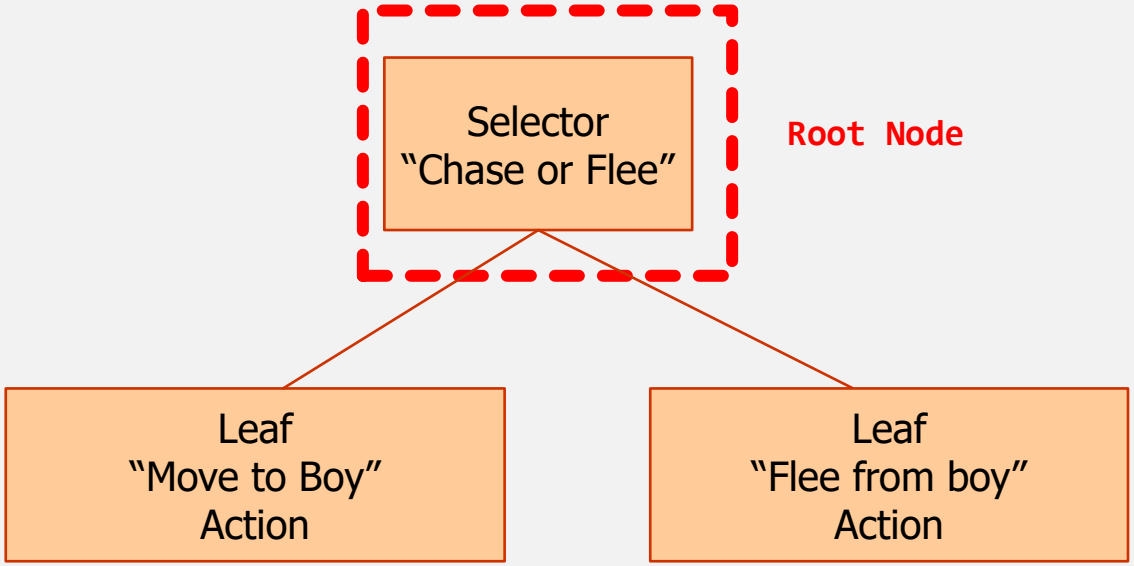
BT 구성


```
def build_behavior_tree(self):
    find_random_location_node = Leaf('Find Random Location', self.find_random_location)
    move_to_node = Leaf('Move To', self.move_to)
    play_beep_node = Leaf('Play Beep', self.play_beep)
    wander_sequence = Sequence('Wander', find_random_location_node, move_to_node, play_beep_node)

    find_ball_location_node = Leaf('Find Ball Location', self.find_ball_location)
    eat_ball_sequence = Sequence('Eat Ball', find_ball_location_node, move_to_node, play_beep_node)

    wander_or_eat_ball_selector = Selector('Wander or Eat Ball', eat_ball_sequence, wander_sequence)
    self.bt = BehaviorTree(wander_or_eat_ball_selector)
```

좀비와 소년의 대결




```
def move_to_boy(self):
    distance = self.calculate_squared_distance(self, server.boy)
    if distance > (PIXEL_PER_METER * 10)**2:
        self.speed = 0
        return BehaviorTree.FAIL

    if self.hp > server.boy.hp:
        self.dir = math.atan2(server.boy.y - self.y, server.boy.x - self.x)
        if distance < (PIXEL_PER_METER * 0.5) ** 2:
            self.speed = 0
            return BehaviorTree.SUCCESS
        else:
            self.speed = RUN_SPEED_PPS
            return BehaviorTree.RUNNING
    else:
        self.speed = 0
        return BehaviorTree.FAIL
```


```python
def flee_from_boy(self):
    distance = self.calculate_squared_distance(self, server.boy)
    if distance > (PIXEL_PER_METER * 10)**2:
        self.speed = 0
        return BehaviorTree.FAIL
    if self.hp <= server.boy.hp:
        self.dir = math.atan2(self.y - server.boy.y, self.x - server.boy.x)
        self.speed = RUN_SPEED_PPS
        return BehaviorTree.RUNNING
    else:
        self.speed = 0
        return BehaviorTree.FAIL
```

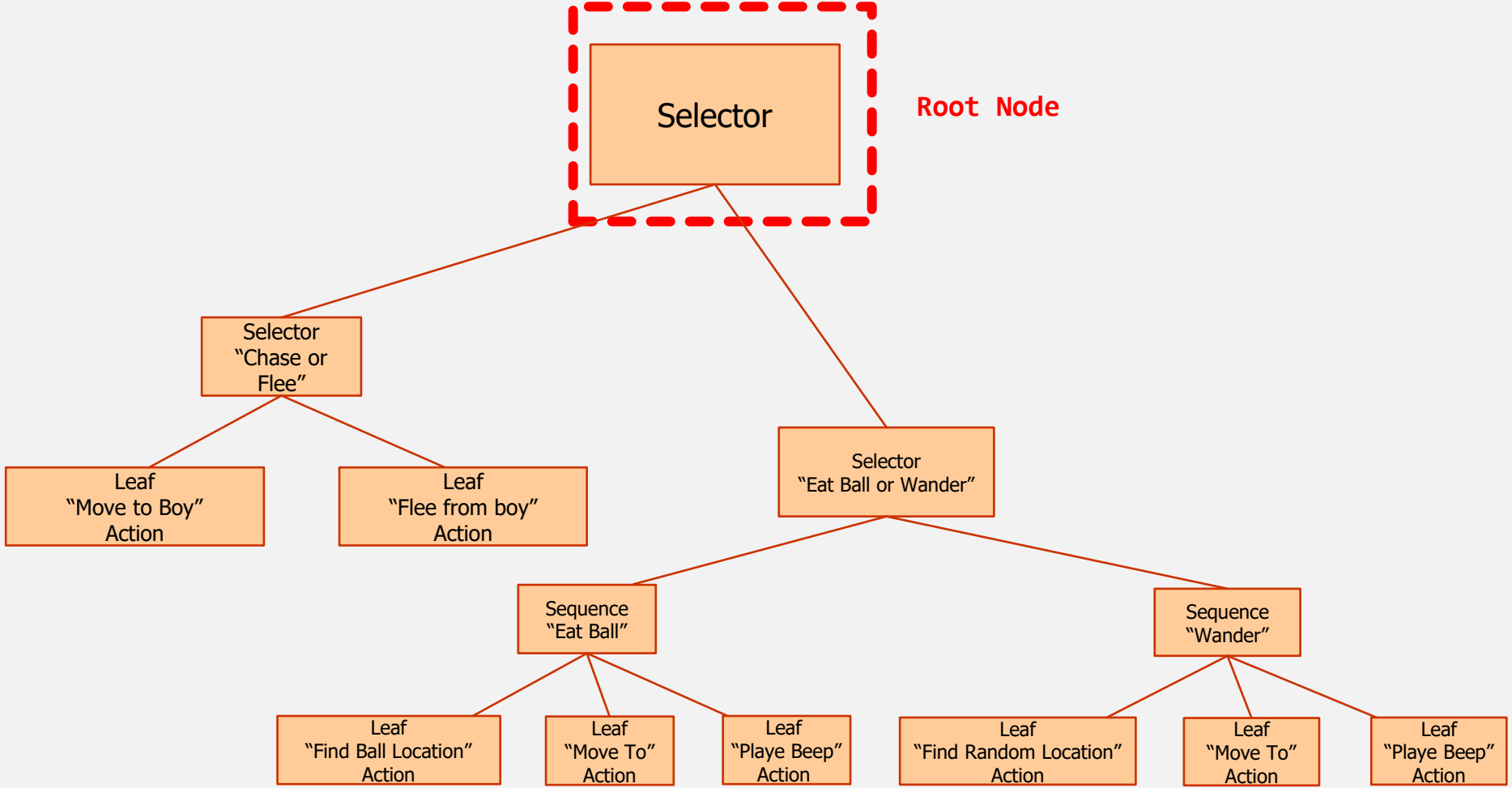

```
def build_behavior_tree(self):
    find_random_location_node = Leaf('Find Random Location', self.find_random_location)
    move_to_node = Leaf('Move To', self.move_to)
    play_beep_node = Leaf('Play Beep', self.play_beep)
    wander_sequence = Sequence('Wander', find_random_location_node, move_to_node, play_beep_node)

    find_ball_location_node = Leaf('Find Ball Location', self.find_ball_location)
    eat_ball_sequence = Sequence('Eat Ball', find_ball_location_node, move_to_node, play_beep_node)

    wander_or_eat_ball_selector = Selector('Wander & Eat Ball', eat_ball_sequence, wander_sequence)

    move_to_boy_node = Leaf('Move to Boy', self.move_to_boy)
    flee_from_boy_node = Leaf('Flee from Boy', self.flee_from_boy)
    chase_or_flee_selector = Selector('Chase or Flee Boy', move_to_boy_node, flee_from_boy_node)
    self.bt = BehaviorTree(chase_or_flee_node_selector)
```

실습 종합



```
def build_behavior_tree(self):
    find_random_location_node = Leaf('Find Random Location', self.find_random_location)
    move_to_node = Leaf('Move To', self.move_to)
    play_beep_node = Leaf('Play Beep', self.play_beep)
    wander_sequence = Sequence('Wander', find_random_location_node, move_to_node, play_beep_node)

    find_ball_location_node = Leaf('Find Ball Location', self.find_ball_location)
    eat_ball_sequence = Sequence('Eat Ball', find_ball_location_node, move_to_node, play_beep_node)

    wander_or_eat_ball_selector = Selector('Wander & Eat Ball', eat_ball_sequence, wander_sequence)

    move_to_boy_node = Leaf('Move to Boy', self.move_to_boy)
    flee_from_boy_node = Leaf('Flee from Boy', self.flee_from_boy)
    chase_or_flee_selector = Selector('Chase or Flee Boy', move_to_boy_node, flee_from_boy_node)

    final_selector = Selector('Final', chase_or_flee_selector, wander_or_eat_ball_selector)
    self.bt = BehaviorTree(final_selector)
```

Patrol BT

순찰 좀비

zombie.py


```
def prepare_patrol_points(self):
    # positions for origin at top, left
    positions = [(43, 750), (1118, 750), (1050, 530), (575, 220), (235, 33), (575,220), (1050, 530), (1118,750)]
    self.patrol_positions = []
    for p in positions:
        self.patrol_positions.append((p[0], 1024-p[1])) # convert for origin at bottom, left
```

순찰 위치 계산 및 초기화.