Lecture #15. 인공지능(행동트리)

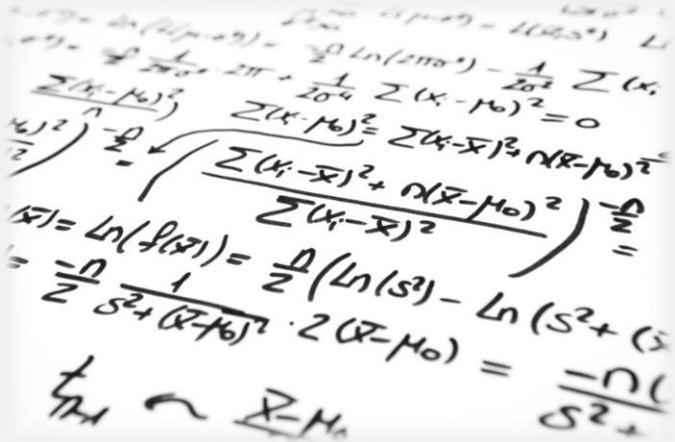
2D 게임 프로그래밍

이대현 교수

한국산업기술대학교
KOREA POLYTECHNIC UNIVERSITY

컴퓨터에게 쉬운 것과 어려운 것

Easy



VS

Hard

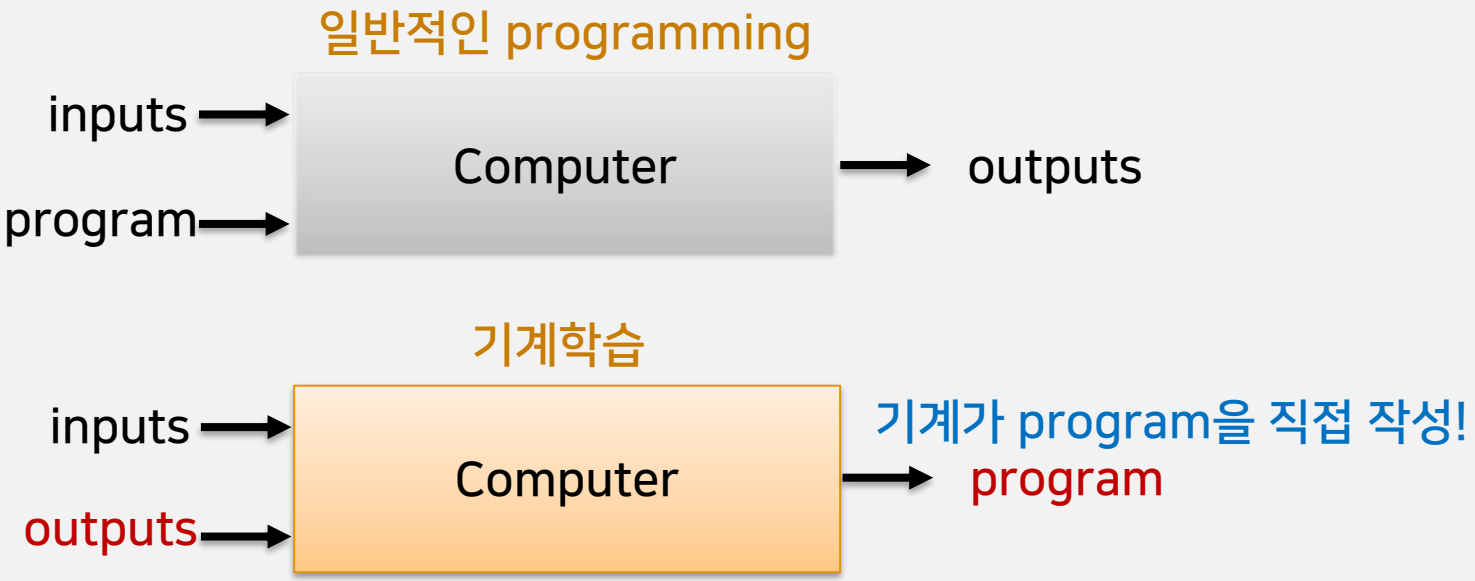
- 컴퓨터가 잘하는 것은 명확하게 정의된 일, 즉 알고리즘에 대한 수행이다.
- 사람이 진화과정에서 자연스럽게 터득한 것들이 컴퓨터에게는 어렵다.

규칙 기반 학습의 부작용(?)

기계 학습(Machine Learning)

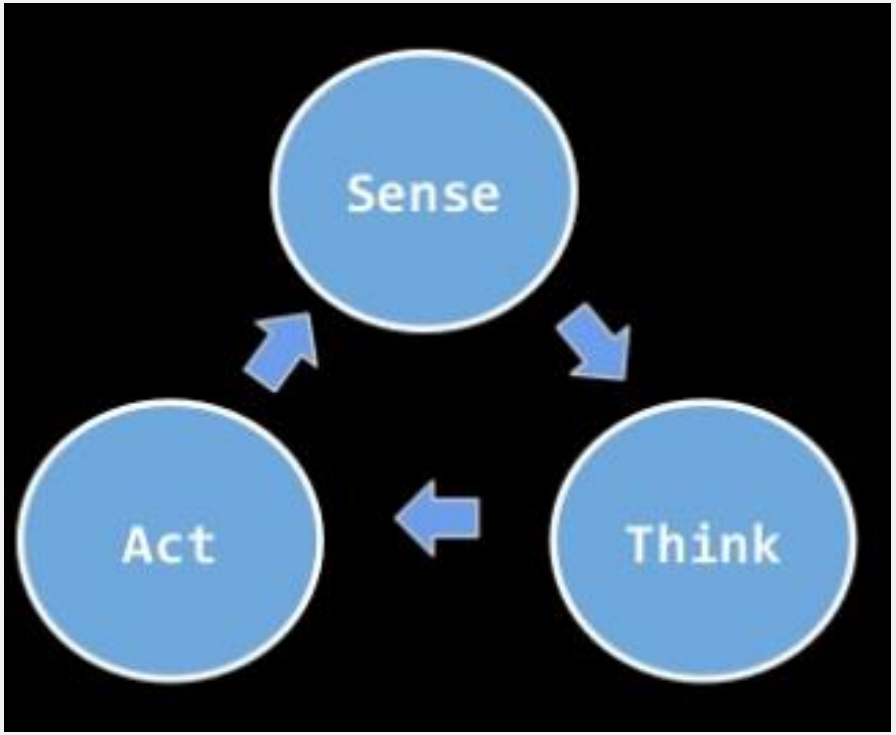
Machine learning is the subfield of computer science that "gives computers the ability to learn without being explicitly programmed"

기계 학습은 "컴퓨터에 명시적으로 프로그래밍하지 않고 학습 할 수 있는 능력을 부여하는" 컴퓨터 과학의 하위 분야.



게임 인공 지능

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



의사 결정

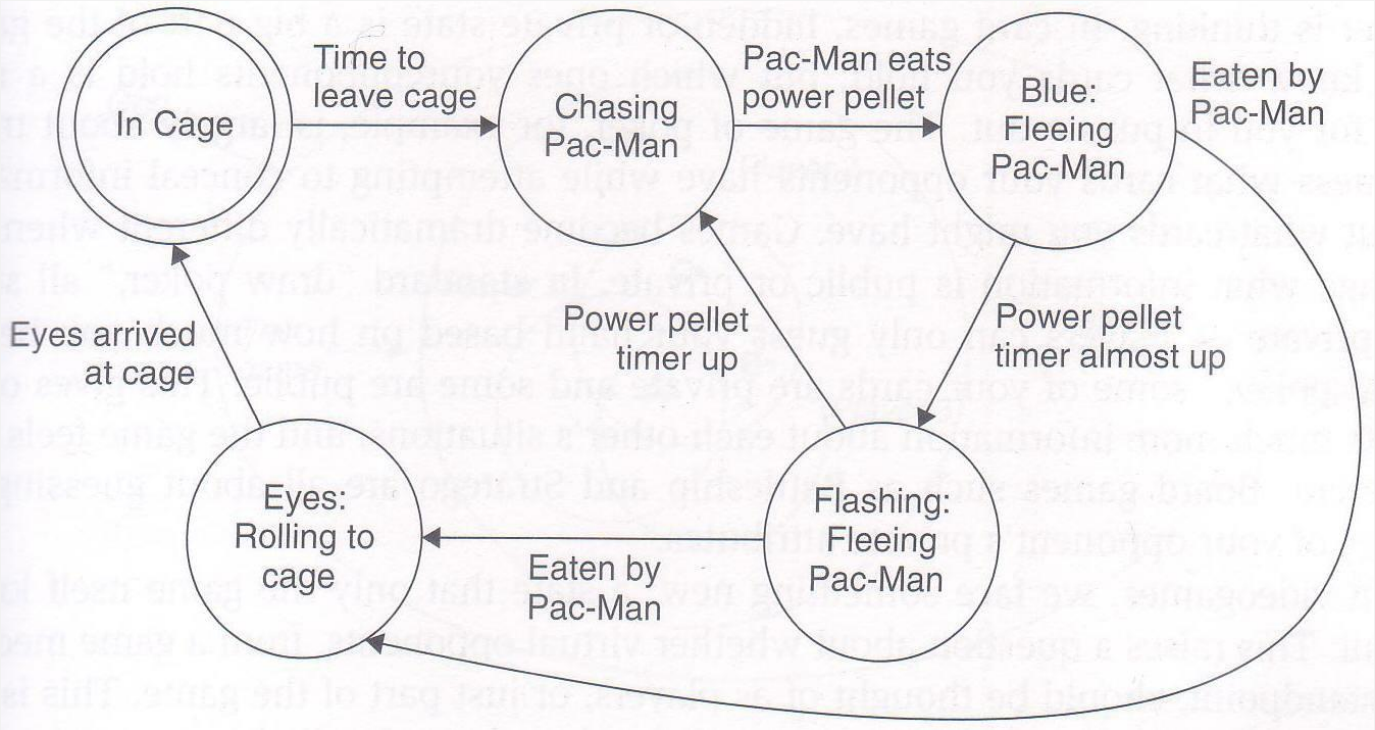
- NPC 들이 뭔가 해야 함, 근데 뭘 하지?

- 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.



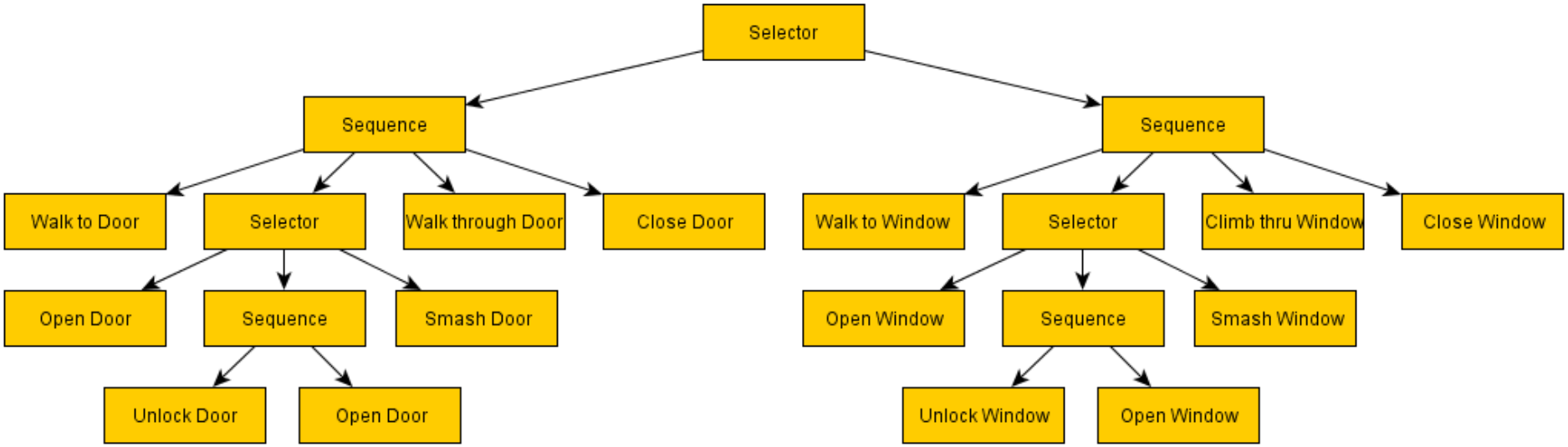
Behavior Tree

- 객체의 인공지능행동을 트리 구조로 구현한 것.
- FSM 방식 – 상태와 이벤트에 따라서, 다음 상태를 결정
- BT 방식 – Goal 을 달성하기 위한 Task들을 구성. 재사용이 쉽게 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

- Action 또는 Condition 임.
- Action
  - 어떤 일을 수행함.
  - 이동, 공격 등등
  - 목적을 달성하기 위해서 "매 프레임마다 해야 할 일"을 담음.
- Codition
  - 여러가지 주변 상황, 상태등을 검사함.
  - 주인공과의 거리, 장애물 상태, 아이템 속성 등등
  - 조건 검사 결과, SUCCESS 또는 FAIL을 return함.

Eat
Action

Sleep
Action

Enemy near?
Condition

Is it daytime?
Condition

Sequence Node

- 모든 자식 노드가 다 SUCCESS 되면, 노드도 성공
- 여러 개의 작업이 모두 다 차근 차근 진행되어야 하는 경우 – AND 조건
- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하며서 실행됨.
- 실행 결과, 처음으로 FAIL이 나오면, 노드가 FAIL됨.
- 실행 결과, 처음으로 RUNNING이 나오면, 자식 노드의 위치를 기록함. 결과는 RUNNING임.

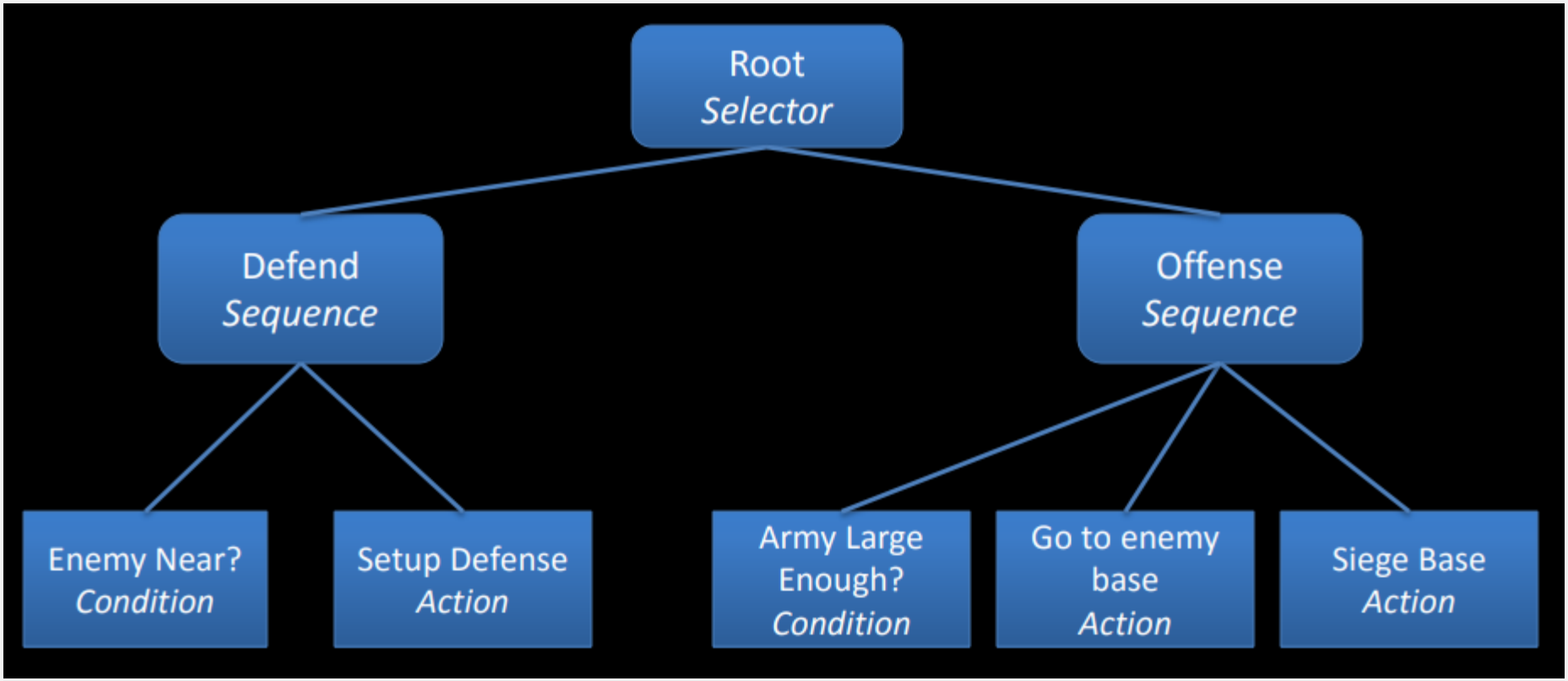
Selector Node

- 자식 노드 중, 하나만 성공하면 성공
- 여러 개의 작업 중, 하나를 선택하는 개념 – OR
- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하며서 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.
- 우선 순위가 높은 노드를 왼쪽에 배치

데이터 기록

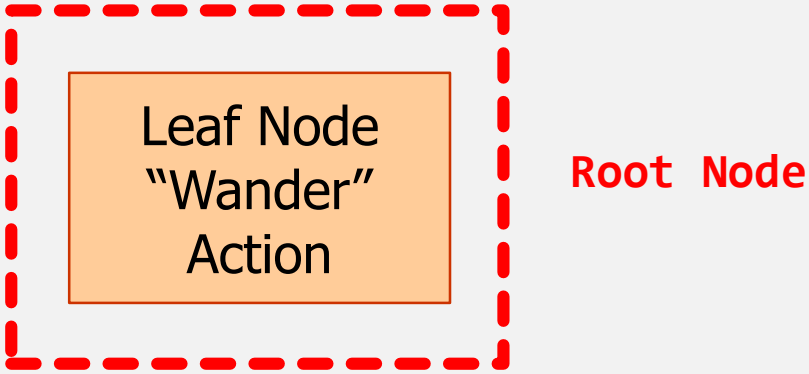
- 각 노드 간에 서로 정보를 교환할 필요가 발생.
- 어떻게 정보를 공유할 것인가?
- 객체 내의 변수를 이용할 수도 있지만,, 여러 개의 객체들이 같은 정보를 공유해야 하는 상황도 있기 때문에…
  - 예) NPC 들간에 주인공의 위치 정보를 공유
- **Blackboard**
  - 정보를 저장할 수 있는 공유 객체
  - Dictionary 등을 이용하여, 쉽게 구현할 수 있음.

BT 예제



Wander BT

버전

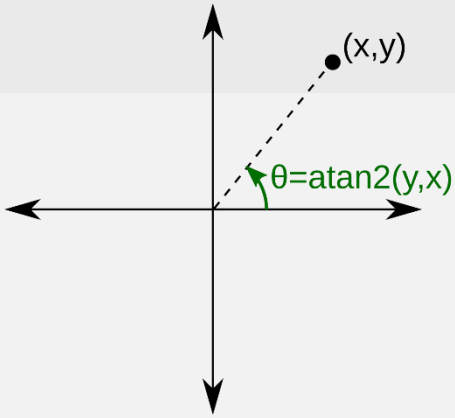


zombie.py

배회 액션에서 매 프레임 해야 할 일은?
캐릭터를 속도와 프레임을 타임을 고려해서 이동해야 함.
1초마다, 방향을 랜덤하게 바꿔야 함.

```
class Zombie:

    def wander(self):
        self.speed = RUN_SPEED_PPS
        self.calculate_current_position()
        self.timer -= game_framework.frame_time
        if self.timer < 0:
            self.timer += 1.0
            self.dir = random.random()*2*math.pi

        return BehaviorTree.SUCCESS
```

dir 을 radian 으로 해석.



zombie.py


```
def build_behavior_tree(self):
    wander_node = LeafNode("Wander", self.wander)

    self.bt = BehaviorTree(wander_node)
```

wander_node를 생성

self.wander 함수를 연결

wander_node를 BT의 root node로 지정.

zombie.py


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

좀비의 업데이트는 BT를 통해 진행됨.
매 프레임마다 BT를 실행함.

따라서, BT의 노드는 매 프레임마다 실행될 내용이 들어가야 함.

BehaviorTree

```
class BehaviorTree:
    FAIL, RUNNING, SUCCESS = -1, 0, 1

    def __init__(self, root_node):
        self.root = root_node

    def run(self):
        self.root.run()
```

BehaviorTree.py (1)

```
class Node:

    def add_child(self, child):
        self.children.append(child)

    def add_children(self, *children):
        for child in children:
            self.children.append(child)
```

BehaviorTree.py (2)

```
class LeafNode(Node):

    def __init__(self, name, func):
        self.name = name
        self.func = func

    def run(self):
        return self.func()
```

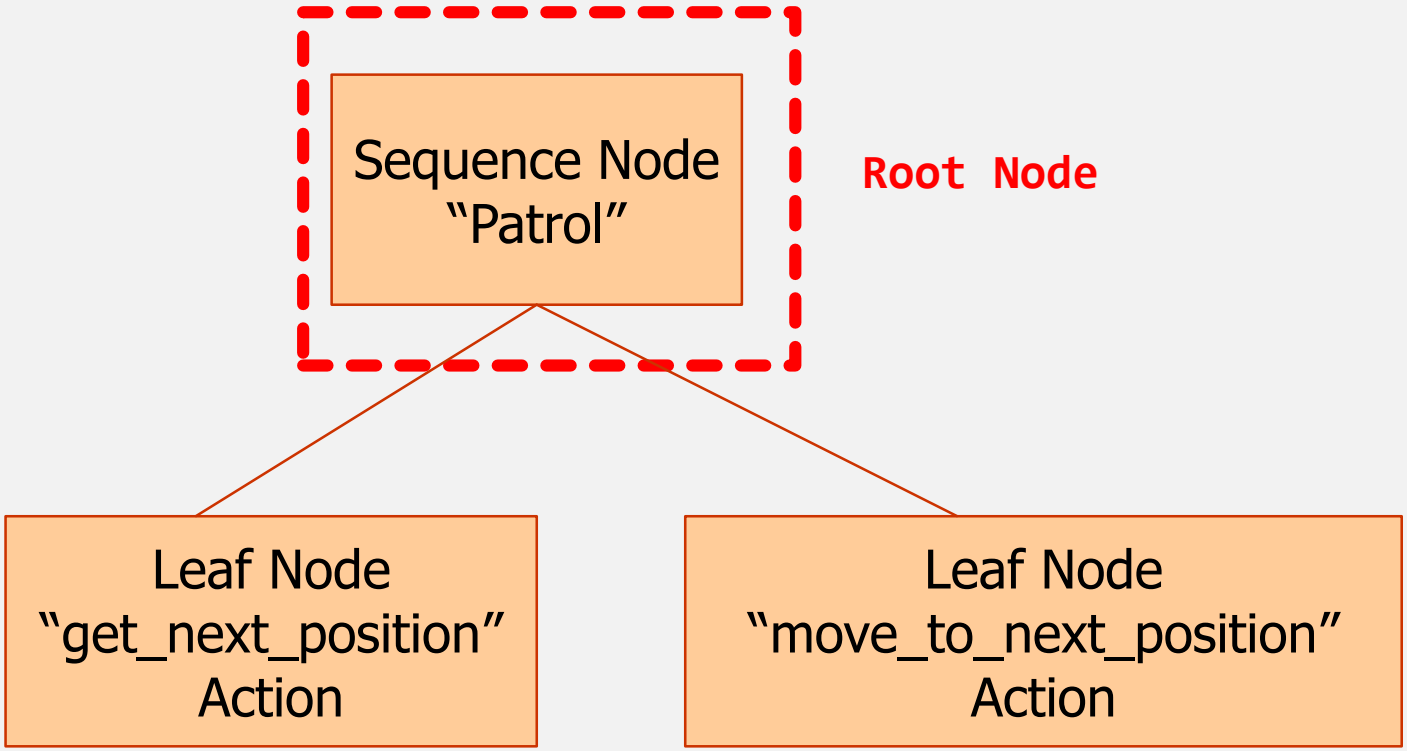
BehaviorTree.py (3)

```
class SequenceNode(Node):
    def run(self):
        for pos in range(self.prev_running_pos, len(self.children)):
            result = self.children[pos].run()
            if BehaviorTree.RUNNING == result:
                self.prev_running_pos = pos
                return BehaviorTree.RUNNING
            elif BehaviorTree.FAIL == result:
                self.prev_running_pos = 0
                return BehaviorTree.FAIL
        self.prev_running_pos = 0
        return BehaviorTree.SUCCESS
```

BehaviorTree.py (4)

```
class SelectorNode(Node):
    def run(self):
        for pos in range(self.prev_running_pos, len(self.children)):
            result = self.children[pos].run()
            if BehaviorTree.RUNNING == result:
                self.prev_running_pos = pos
                return BehaviorTree.RUNNING
            elif BehaviorTree.SUCCESS == result:
                self.prev_running_pos = 0
                return BehaviorTree.SUCCESS
        self.prev_running_pos = 0
        return BehaviorTree.FAIL
```

Patrol BT
순찰 좀비



zombie.py

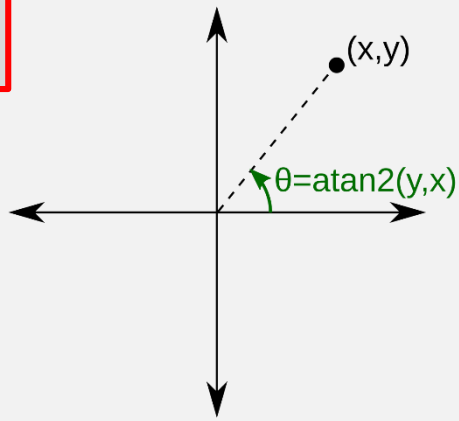

```
def __init__(self):
    # positions for origin at top, left
    positions = [(43, 750),(1118, 750),(1050, 530),(575, 220),(235, 33), (575,220),(1050, 530), (1118,750)]
    self.patrol_positions = []
    for p in positions:
        self.patrol_positions.append((p[0], 1024-p[1])) # convert for origin at bottom, left
    self.patrol_order = 1
    self.target_x, self.target_y = None, None
    self.x, self.y = self.patrol_positions[0]


    self.load_images()
    self.dir = random.random()*2*math.pi # random moving direction
    self.speed = 0
    self.timer = 1.0 # change direction every 1 sec when wandering
    self.frame = 0
    self.build_behavior_tree()
```

순찰 위치 계산 및 초기화.

zombie.py


```
def get_next_position(self):
    self.target_x, self.target_y =
        self.patrol_positions[self.patrol_order % len(self.patrol_positions)]
    self.patrol_order += 1
    self.dir = math.atan2(self.target_y - self.y, self.target_x - self.x)
    return BehaviorTree.SUCCESS
```

atan2를 이용하여,
각도 계산



zombie.py


```
def move_to_target(self):
    self.speed = RUN_SPEED_PPS
    self.calculate_current_position()

    distance = (self.target_x - self.x)**2 + (self.target_y - self.y)**2


    if distance < PIXEL_PER_METER**2:
        return BehaviorTree.SUCCESS
    else:
        return BehaviorTree.RUNNING
```

남은 거리 계산.

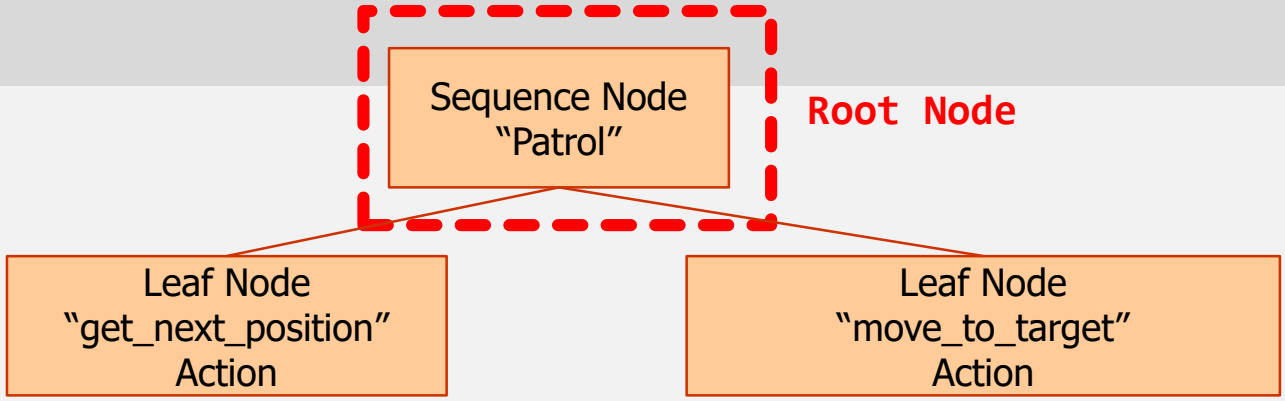
거의 근접했으면, SUCCESS,

아직 남았으면, 다음 프레임에서 계속 진행하도록 하기 위해서 RUNNING을 리턴

zombie.py

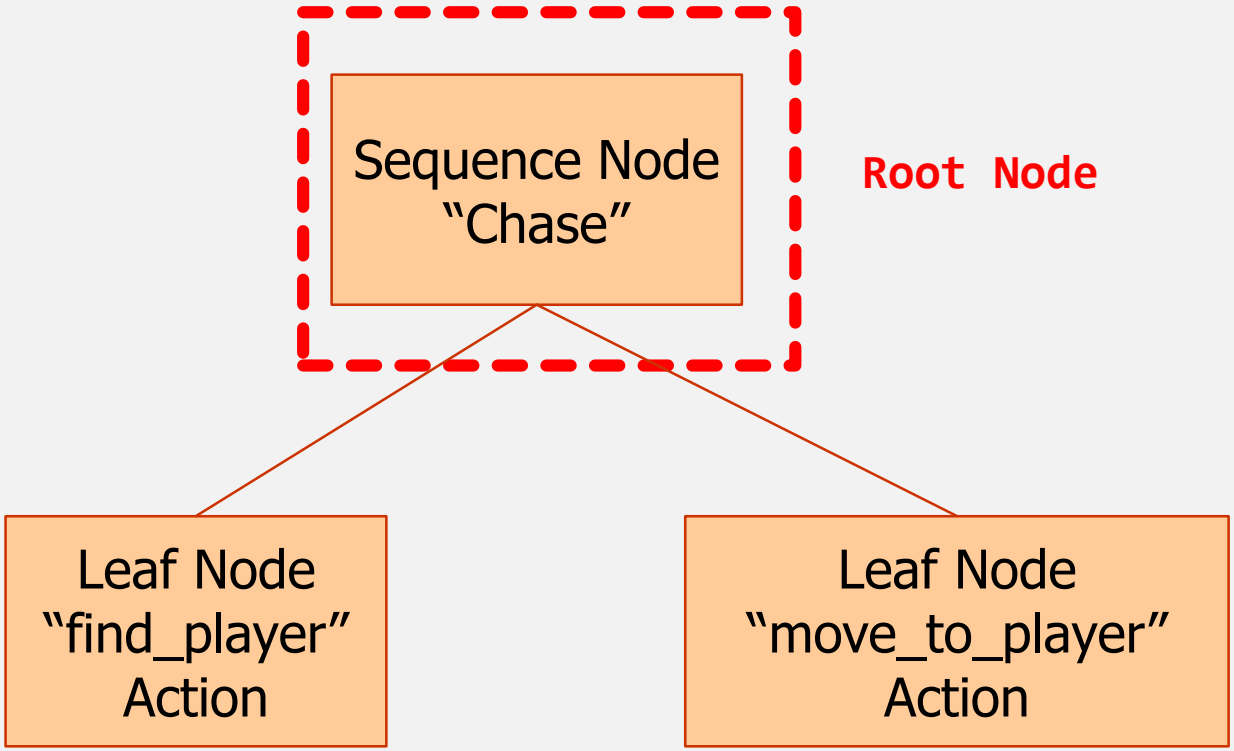
BT 구성

```
def build_behavior_tree(self):
    get_next_position_node = LeafNode("Get Next Position", self.get_next_position)
    move_to_target_node = LeafNode("Move to Target", self.move_to_target)
    patrol_node = SequenceNode("Patrol")
    patrol_node.add_children(get_next_position_node, move_to_target_node)
    self.bt = BehaviorTree(patrol_node)
```



실습

플레이어를 추적하는 좀비 AI



zombie.py – find_player


```
def find_player(self):
    boy = main_state.get_boy()
    distance = (boy.x - self.x)**2 + (boy.y - self.y)**2
    if distance < (PIXEL_PER_METER * 10)**2:
        self.dir = math.atan2(boy.y - self.y, boy.x - self.x)
        return BehaviorTree.SUCCESS
    else:
        self.speed = 0
        return BehaviorTree.FAIL
```

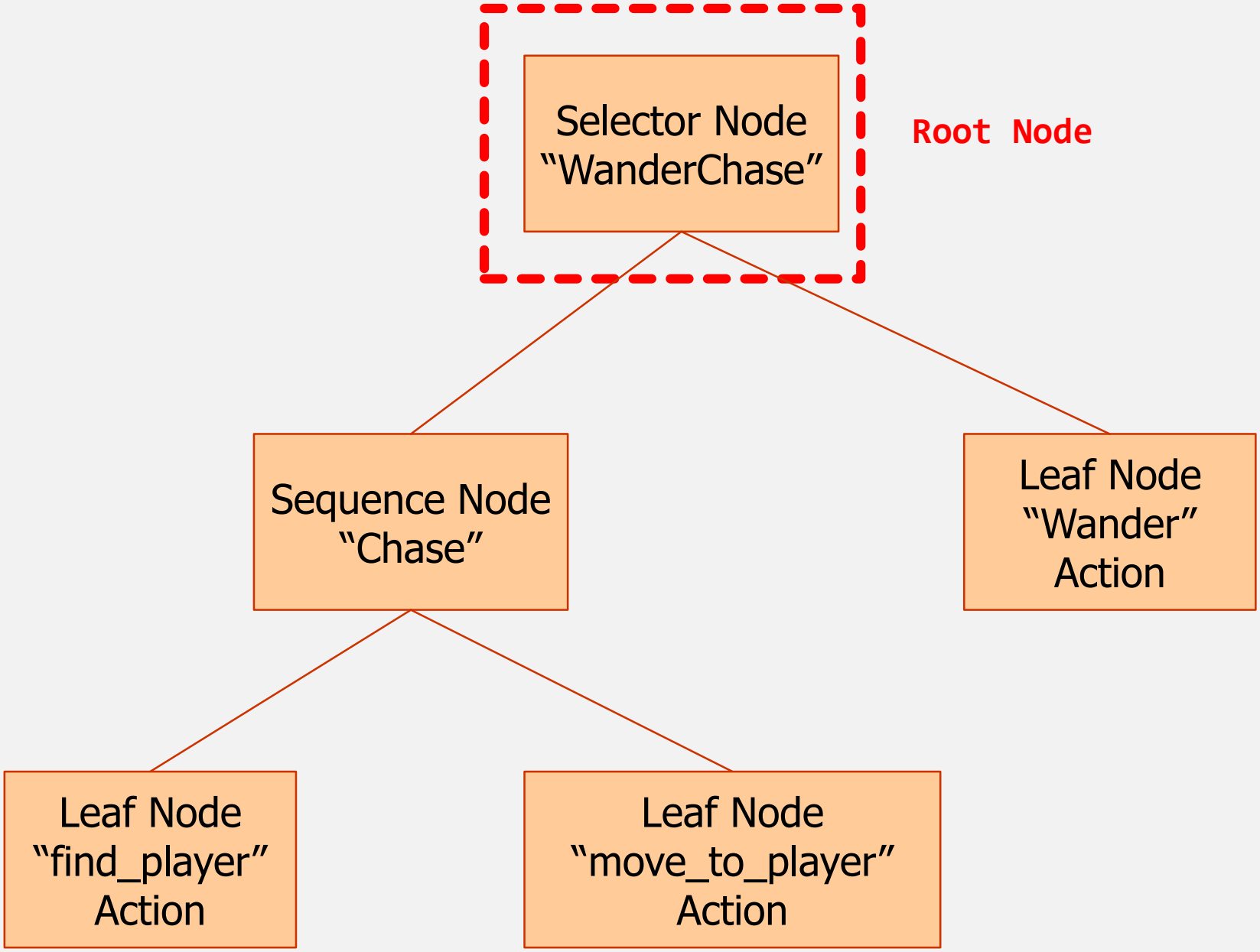
zombie.py – move_to_player


```
def move_to_player(self):
    self.speed = RUN_SPEED_PPS
    self.calculate_current_position()
    return BehaviorTree.SUCCESS
```

zombie.py – behavior tree 구성


```
def build_behavior_tree(self):
    find_player_node = LeafNode("Find Player", self.find_player)
    move_to_player_node = LeafNode("Move to Player", self.move_to_player)
    chase_node = SequenceNode("Chase")
    chase_node.add_children(find_player_node, move_to_player_node)
    self.bt = BehaviorTree(chase_node)
```

배회 및 추적하는 좀비 AI



zombie.py – behavior tree 구성


```
def build_behavior_tree(self):
    wander_node = LeafNode("Wander", self.wander)
    find_player_node = LeafNode("Find Player", self.find_player)
    move_to_player_node = LeafNode("Move to Player", self.move_to_player)
    chase_node = SequenceNode("Chase")
    chase_node.add_children(find_player_node, move_to_player_node)
    wander_chase_node = SelectorNode("WanderChase")
    wander_chase_node.add_children(chase_node, wander_node)
    self.bt = BehaviorTree(wander_chase_node)
```