Lecture #11. 캐릭터 컨트롤러

2D 게임 프로그래밍

이대현 교수

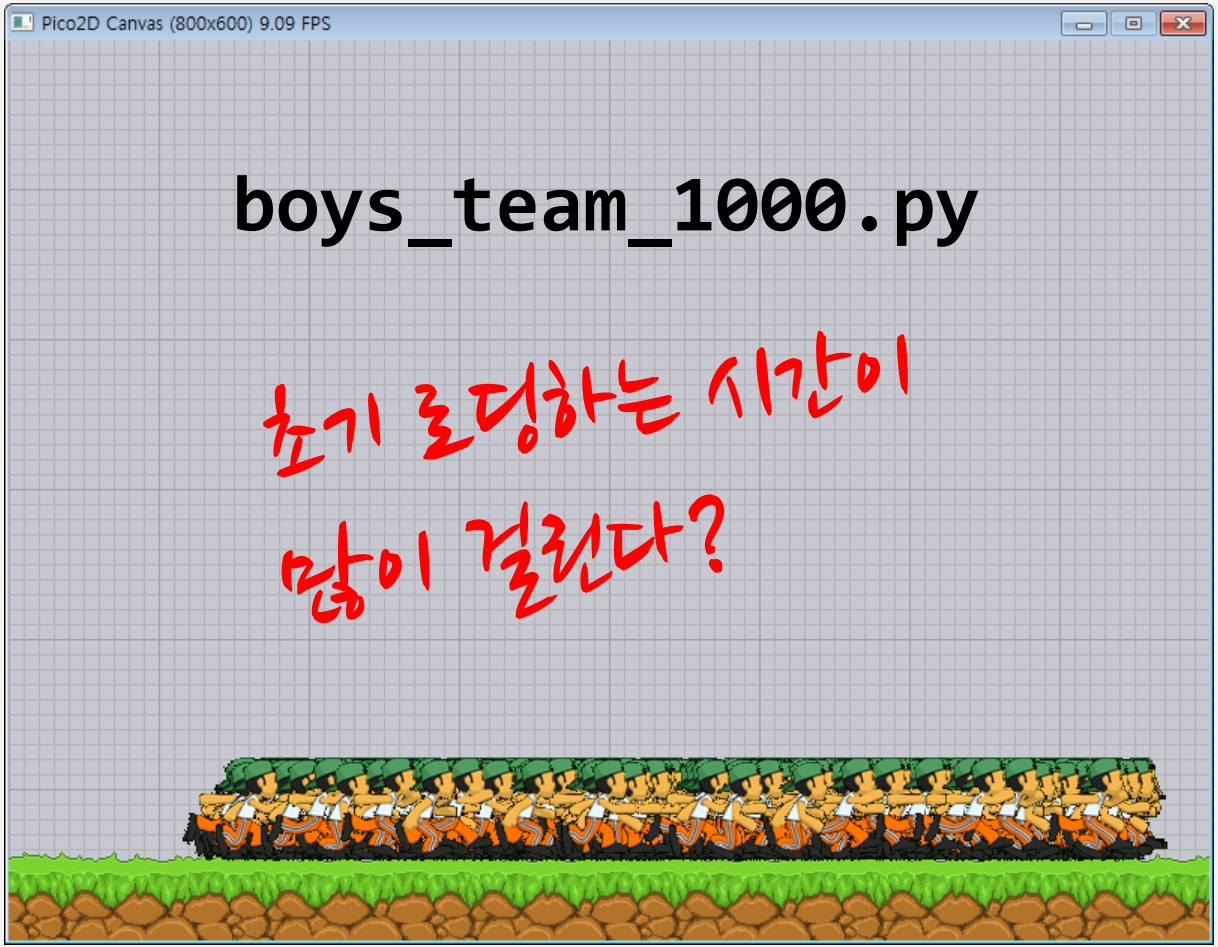
한국공학대학교
TECH UNIVERSITY OF KOREA

학습 내용

- 클래스 변수
- 캐릭터 컨트롤러
- 상태기계
- 이벤트 큐

1000명 선수

리소스 로딩 최적화



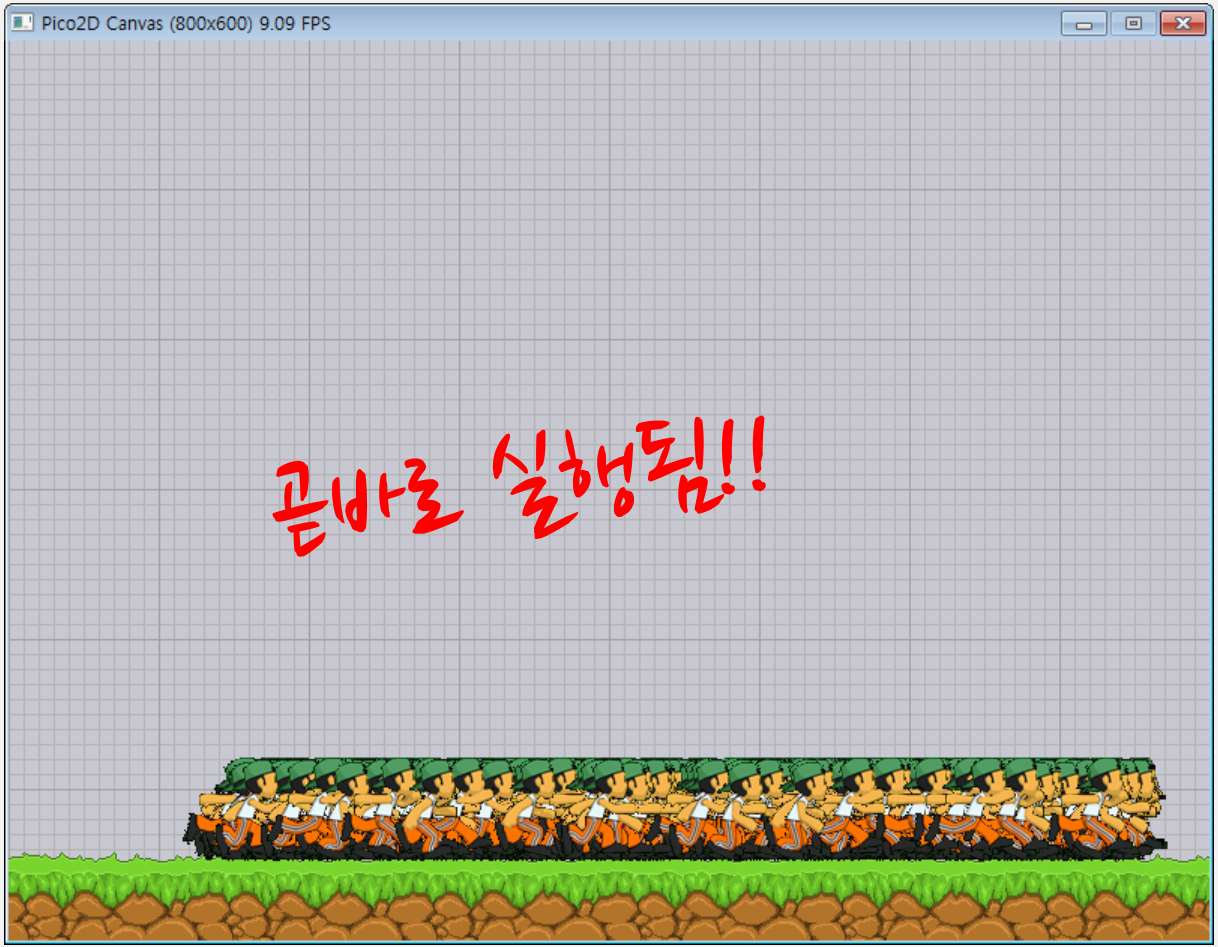
문제점은?

```
class Boy:

    def __init__(self):
        self.x, self.y = random.randint(100, 700), 90
        self.frame = random.randint(0, 7)
        self.image = load_image('run_animation.png')
```

객체의 멤버변수는 객체마다 따로 만들어진다!

1000번의 로딩이 반복


```
class Boy:
    image = None

    def __init__(self):
        self.x, self.y = random.randint(100, 700), 90
        self.frame = random.randint(0, 7)
        if Boy.image == None:
            Boy.image = load_image('run_animation.png')
```



클래스 변수

클래스 자체에 할당되는 변수.
객체들은 공유하는 동일한 변수를 갖게 됨.

```
class Boy:
    image = None
…
…   def __do_some():
…
        Boy.image = …
```

```
class Boy:
    image = None

    def __init__(self):
        self.x, self.y = random.randint(100, 700), 90
        self.frame = random.randint(0, 7)
        if Boy.image == None:
            Boy.image = load_image('run_animation.png')
```

단 한번의 이미지 로딩만 수행.

이미지 리소스를 모든 객체가 공유하게 됨.

self 없는 클래스

```
class Star:

    type = 'Star'
    x = 100

    def change():
        x = 200
        print('x is ', x)

print('x IS ', Star.x) # OK
Star.change()  # OK
print('x IS ', Star.x)

star = Star() # OK
print('x IS ', star.x) # OK
star.change()  # Error
```

self 의 의미

```
class Player:
    type = 'Player'

    def __init__(self):
        self.x = 100

    def where(self):
        print(self.x)

player = Player()
player.where()

# 클래스 변수 사용
print(Player.type)

# 클래스 함수 호출
Player.where() # error
Player.where(player) # OK, player.where() 과 같음.
```

캐릭터 컨트롤러(Character Controller)

- 게임 주인공의 행동을 구현한 것!
  - 키입력에 따른 액션
  - 주변 객체와의 인터랙션
- 게임 구현에서 가장 핵심적인 부분임.

우리의 "주인공"은?

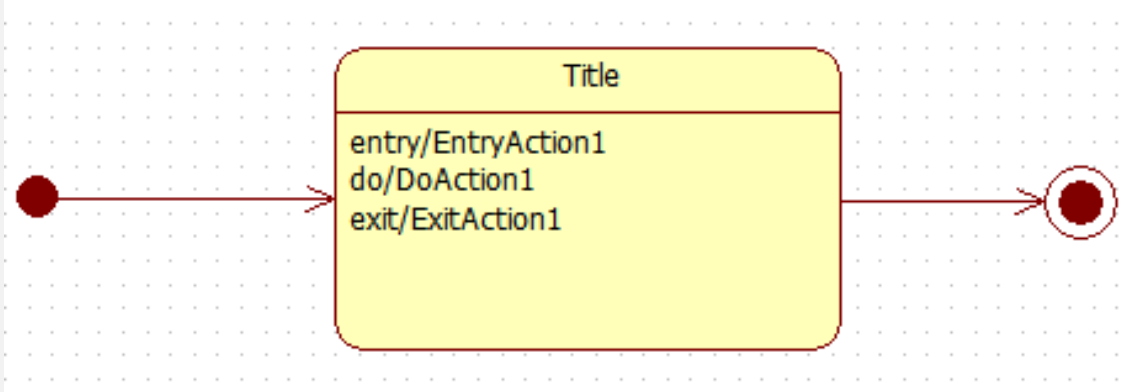
- **캐릭터 컨트롤러의 행위를 적으면…**
  - 처음 소년의 상태는 제자리에 서서 휴식을 하고 있습니다.
  - 이 상태에서 오른쪽 방향키를 누르면 소년은 오른쪽으로 달리게 됩니다.
  - 방향키를 계속 누르고 있으면, 소년도 계속 오른쪽으로 달리죠.
  - 방향키에서 손가락을 떼면 소년은 달리기를 멈추고 휴식상태에 들어갑니다.
  - 한참 지나도, 방향키 입력이 없으면 소년은 취침에 들어갑니다.
  - 달리는 중에, Dash 키를 누르면 빠르게 달립니다.
  - 왼쪽 방향키 조작에 대해선 왼쪽으로 달리게 됩니다.
  - 캔버스의 좌우측 가장자리에 도착하면 더 이상 달려나가지는 않습니다.

상태 다이어그램(State Diagram)

- 시스템의 변화를 모델링하는 다이어그램.
- 사건이나 시간에 따라 시스템 내의 객체들이 자신의 상태(state)를 바꾸는 과정을 모델링함.
- 모델링, 명세, 그리고 구현에 모두 사용되는 강력한 툴
- 상태(state)의 변화 예
  - 스위치를 누를 때마다 탁상 전등 상태는 “켜짐”에서 “꺼짐”으로 바뀐다.
  - 리모트 컨트롤의 버튼을 누르면 TV의 상태는 한 채널을 보여주다가 다른 상태를 보여주게 된다.
  - 얼마간의 시간이 흐르면 세탁기의 상태는 “세탁”에서 “헹굼”으로 바뀐다.

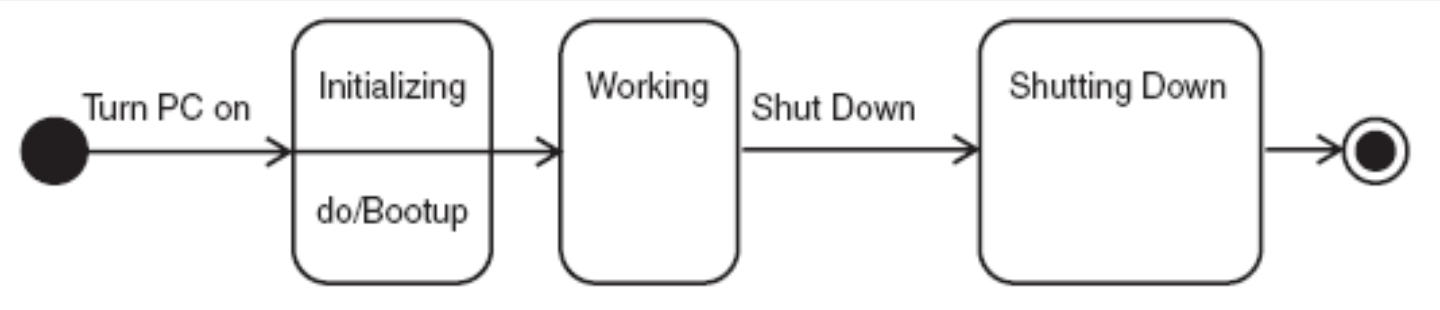
상태(State)

- 상태 : 어떤 조건을 만족하는 동안 머무르면서, 정해진 일을 수행하고 이벤트를 기다리는 “상황”
- Entry action : 특정한 상태로 들어갈 때마다 발생하는 일
- Exit action : 특정한 상태에서 나갈 때마다 발생하는 일
- Do activity : 특정 상태에 머무르는 동안 수행하는 일(반복될 수 있음)



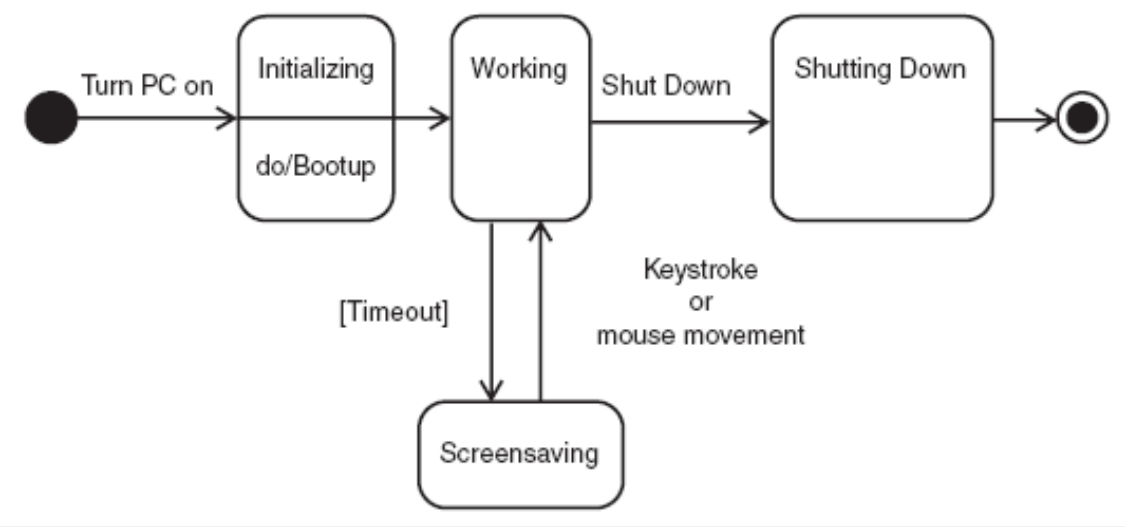
상태 변화(State Transition)

- A transition is a relationship between two states; it indicates that an object in the first state will perform certain actions, then enter the second state when a given event occurs.



이벤트(Event)

- 상태 변화(State Transition)을 일으키는 원인이 되는 일
  - 외부적인 이벤트 : 예) 키보드 입력
  - 내부적인 이벤트 : 예) 타이머
  - 경우에 따라서는 이벤트 없이도 상태 변화가 있을 수 있음.

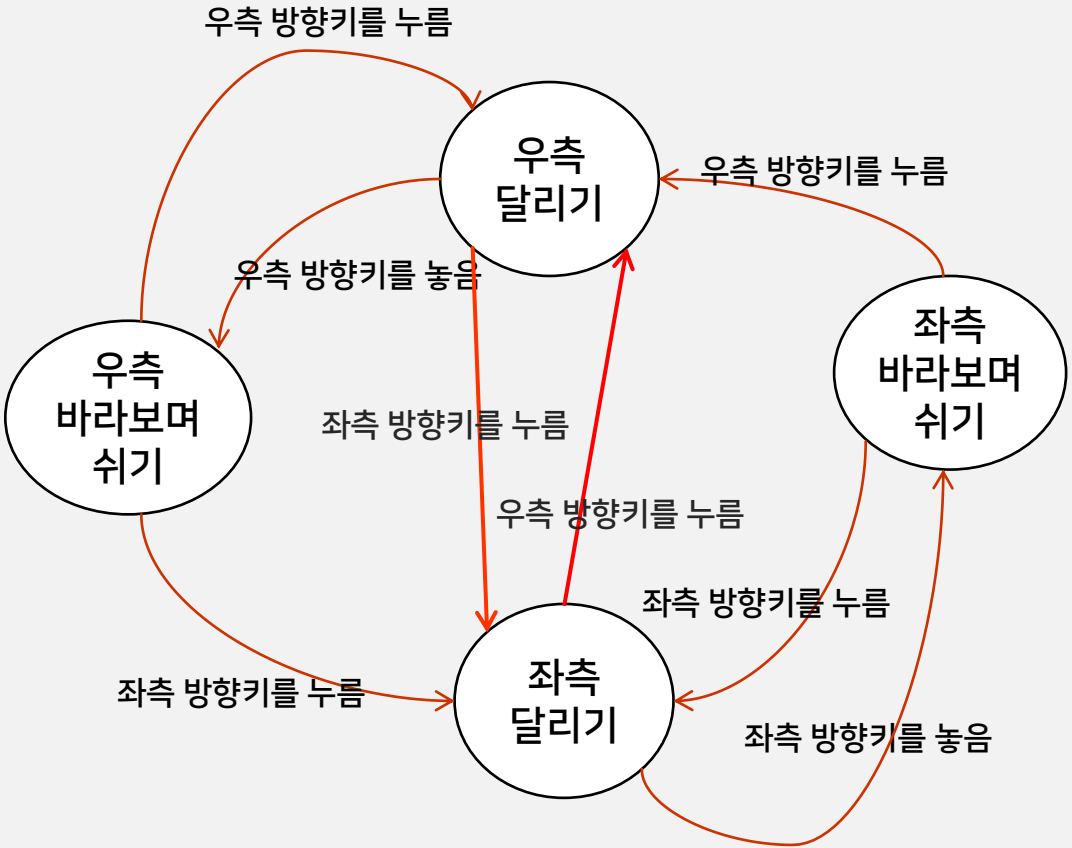


상태와 이벤트 찾기

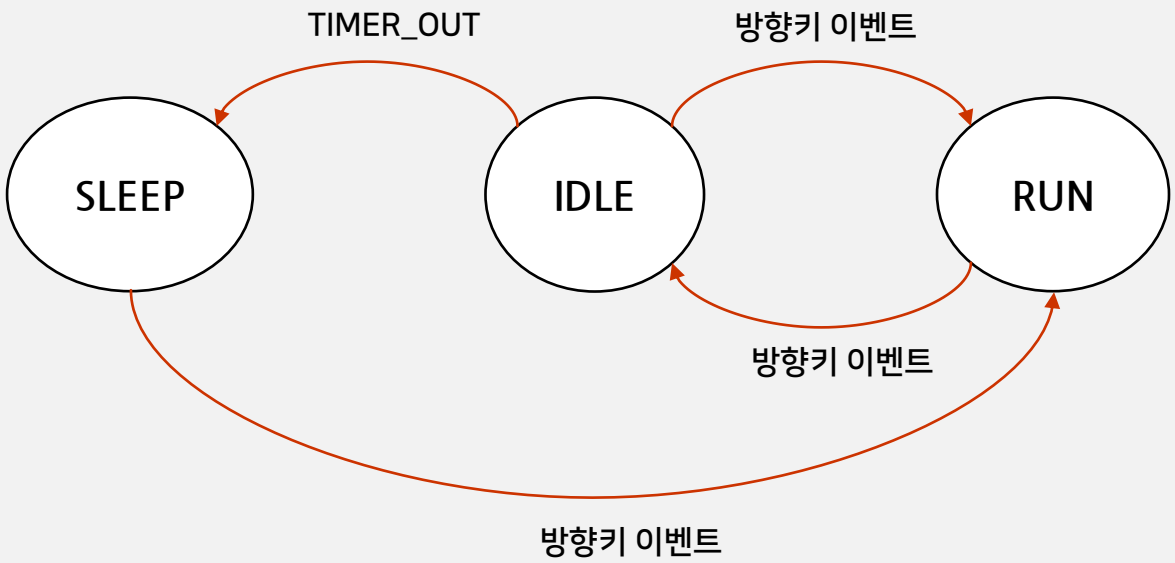
- 주인공의 움직임 상태를 찾아보자.

- 주인공의 상태에 변화를 일으킬 수 있는 이벤트를 찾아보자.

상태 다이어그램 #1



상태 다이어그램 #2



캐릭터 컨트롤러 구현(IDLE & RUN)

Source Code Files

- game_framework.py
- mygame.py – 실행 시작 파일
- play_state.py – 메인 게임상태

클래스 분리

- play_state.py 로부터 boy 클래스와 grass 클래스를 분리 리팩토링

```python
from pico2d import *

class Boy:
    def __init__(self):
        self.x, self.y = 0, 90
        self.frame = 0
        self.dir, self.face_dir = 0, 1
        self.image = load_image('animation_sheet.png')

    def update(self):
        self.frame = (self.frame + 1) % 8
        self.x += self.dir * 1
        self.x = clamp(0, self.x, 800)

    def draw(self):
        if self.dir == -1:
            self.image.clip_draw(self.frame*100, 0, 100, 100, self.x, self.y)
        elif self.dir == 1:
            self.image.clip_draw(self.frame*100, 100, 100, 100, self.x, self.y)
        else:
            if self.face_dir == 1:
                self.image.clip_draw(self.frame * 100, 300, 100, 100, self.x, self.y)
            else:
                self.image.clip_draw(self.frame * 100, 200, 100, 100, self.x, self.y)
```

```python
from pico2d import *

class Grass:
    def __init__(self):
        self.image = load_image('grass.png')

    def draw(self):
        self.image.draw(400, 30)
```

boy.handle_event 도입

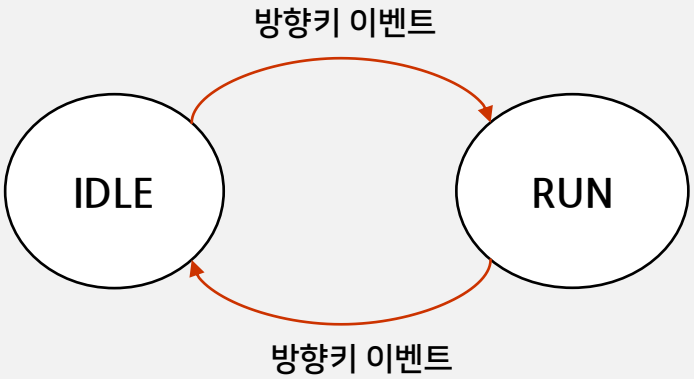
boy.py

```
def handle_event(self, event):
    if event.type == SDL_KEYDOWN:
        match event.key:
            case pico2d.SDLK_LEFT:
                self.dir -= 1
            case pico2d.SDLK_RIGHT:
                self.dir += 1
    elif event.type == SDL_KEYUP:
        match event.key:
            case pico2d.SDLK_LEFT:
                self.dir += 1
                self.face_dir = -1
            case pico2d.SDLK_RIGHT:
                self.dir -= 1
                self.face_dir = 1
```

play_state.py

```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif (event.type, event.key) == (SDL_KEYDOWN, SDLK_ESCAPE):
            game_framework.quit()
        else:
            boy.handle_event(event)
```

상태 변환 구현 목표



어떤 상태가 있는가?

어떤 이벤트가 있는가?

#1. 상태의 정의 – boy.py


```python
class IDLE:
    @staticmethod
    def enter():
        pass

    @staticmethod
    def exit():
        pass

    @staticmethod
    def do():
        pass

    @staticmethod
    def draw():
        pass
```

```python
class RUN:
    def enter():
        pass

    def exit():
        pass

    def do():
        pass

    def draw():
        pass
```

여기서 class 의 역할은 특정함수를 모아서 그루핑하는 역할. 객체 생성이 아님!

#2. 이벤트의 정의 – boy.py


```
RD, LD, RU, LU = range(4)

key_event_table = {
    (SDL_KEYDOWN, SDLK_RIGHT): RD,
    (SDL_KEYDOWN, SDLK_LEFT): LD,
    (SDL_KEYUP, SDLK_RIGHT): RU,
    (SDL_KEYUP, SDLK_LEFT): LU
}
```

dictionary 를 이용한 키매핑

```
RD, LD, RU, LU = range(4)


key_event_table = {
    (SDL_KEYDOWN, SDLK_RIGHT): RD,
    (SDL_KEYDOWN, SDLK_LEFT): LD,
    (SDL_KEYUP, SDLK_RIGHT): RU,
    (SDL_KEYUP, SDLK_LEFT): LU
}
```

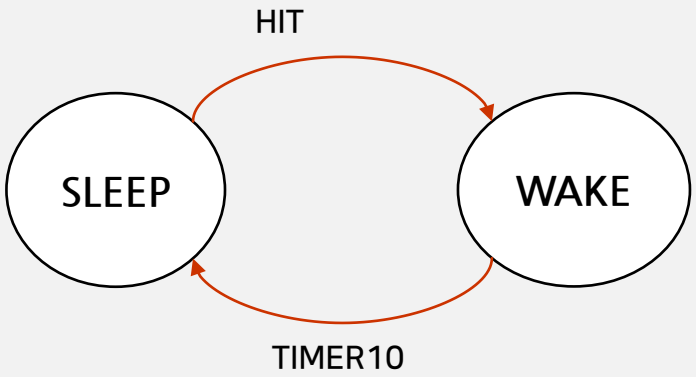
0부터 3까지의 정수값이 차례로 할당됨.

Key는 (정수, 정수) tuple

Value 는 정수

입력 키값 해석을 단순화시키고, 키입력을 단일이벤트로 만들기 위한 매핑

#3. 상태 변환 구현 – 상태 변환 테이블 활용



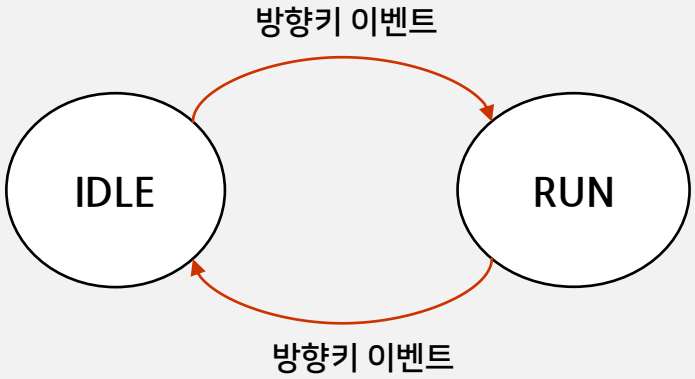
현재 상태	이벤트	다음 상태
SLEEP	HIT	WAKE
WAKE	TIMER10	SLEEP

상태 변환 테이블

```
table = {
    "SLEEP": {"HIT": "WAKE"},
    "WAKE": {"TIMER10": "SLEEP"}
}
```

현재상태와 이벤트로부터 다음 상태를 계산

소년의 상태 변환 구현




```
next_state = {
    IDLE: {RU: RUN,  LU: RUN,  RD: RUN,  LD: RUN},
    RUN:  {RU: IDLE, LU: IDLE, RD: IDLE, LD: IDLE}
}
```

#4. 상태 변환 실행 구현 – 큐를 활용

- 큐(Queue)
  - 표를 사기 위한 줄서기
  - 먼저 집어 넣은 데이터가 먼저 나오는 구조(FIFO: First In First Out)

```
>>>event_queue = []
>>> event_queue.insert(0, 'cat')
>>> event_queue.insert(0, 'dog')
>>> event_queue.insert(0, 'pig')
>>> event_queue
['pig', 'dog', 'cat']
>>> event_queue.pop()
'cat'
>>> event_queue
['pig', 'dog']
>>> event_queue.pop()
'dog'
>>> event_queue
['pig']
```

boy.py – 상태 변환 실행 추가


```python
def __init__(self):
    self.x, self.y = 0, 90
    self.frame = 0
    self.dir, self.face_dir = 0, 1
    self.image = load_image('animation_sheet.png')

    self.event_que = []
    self.cur_state = IDLE
    self.cur_state.enter()
```

```python
def update(self):
    self.cur_state.do(self)

    if self.event_que:
        event = self.event_que.pop()
        self.cur_state.exit(self)
        self.cur_state = next_state[self.cur_state][event]
        self.cur_state.enter(self, event)

def draw(self):
    self.cur_state.draw(self)

def add_event(self, event):
    self.event_que.insert(0, event)

def handle_event(self, event):
    if (event.type, event.key) in key_event_table:
        key_event = key_event_table[(event.type, event.key)]
        self.add_event(key_event)
```

```
def __init__(self):
    self.x, self.y = 0, 90
    self.frame = 0
    self.dir, self.face_dir = 0, 1
    self.image = load_image('animation_sheet.png')

    self.event_que = []
    self.cur_state = IDLE
    self.cur_state.enter()
```

이벤트 큐 초기화

현재 상태를 IDLE로 설정하고, entry action 을 실행.

```python
def update(self):
    self.cur_state.do(self)

    if self.event_que:
        event = self.event_que.pop()
        self.cur_state.exit(self)
        self.cur_state = next_state[self.cur_state][event]
        self.cur_state.enter(self, event)

def draw(self):
    self.cur_state.draw(self)

def add_event(self, event):
    self.event_que.insert(0, event)

def handle_event(self, event):
    if (event.type, event.key) in key_event_table:
        key_event = key_event_table[(event.type, event.key)]
        self.add_event(key_event)
```

현재 상태의 Do Activity 수행.

큐에 이벤트가 있으면,

현재 상태의 Exit action 수행.

현재 상태에서 이벤트에 따른 다음 상태를 계산.

다음 상태의 Entry action 수행.

입력된 키와 눌린 상태를 해석해서 이벤트를 만들고 이벤트 큐에 추가

상태 변환 테스트 – 간단한 PRINT 를 통해


```python
class IDLE:
    @staticmethod
    def enter():
        print('ENTER IDLE')

    @staticmethod
    def exit():
        print('EXIT IDLE')

    @staticmethod
    def do():
        pass

    @staticmethod
    def draw():
        pass
```

```python
class RUN:
    def enter():
        print('ENTER RUN')

    def exit():
        print('EXIT RUN')

    def do():
        pass

    def draw():
        pass
```

Draw 실행

IDLE class

```
@staticmethod
def draw(self):
    if self.face_dir == 1:
        self.image.clip_draw(self.frame * 100, 300, 100, 100, self.x, self.y)
    else:
        self.image.clip_draw(self.frame * 100, 200, 100, 100, self.x, self.y)
    pass
```

Run class

```
def draw(self):
    if self.dir == -1:
        self.image.clip_draw(self.frame*100, 0, 100, 100, self.x, self.y)
    elif self.dir == 1:
        self.image.clip_draw(self.frame*100, 100, 100, 100, self.x, self.y)
```

Boy class

```
def draw(self):
    self.cur_state.draw(self)
```

boy.py - 최종

event 에 따른 처리가 필요할 수 있기 때문에,
event를 전달받음.

```
class IDLE:
    @staticmethod
    def enter(self,event):
        print('ENTER IDLE')
        self.dir = 0

    @staticmethod
    def exit(self):
        print('EXIT IDLE')

    @staticmethod
    def do(self):
        self.frame = (self.frame + 1) % 8

    @staticmethod
    def draw(self):
        if self.face_dir == 1:
            self.image.clip_draw(self.frame * 100, 300, 100, 100, self.x, self.y)
        else:
            self.image.clip_draw(self.frame * 100, 200, 100, 100, self.x, self.y)
```

Entry Action

Exit Action

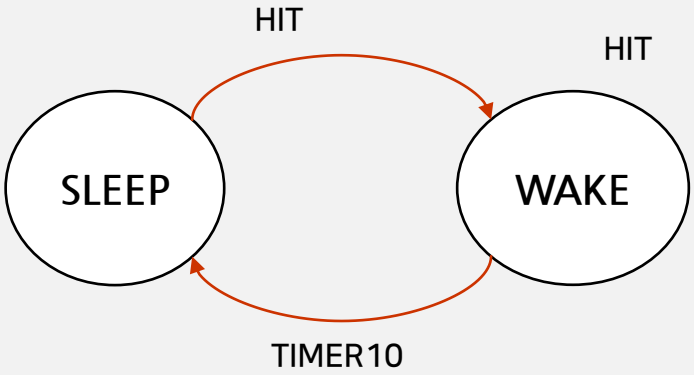
Do Activity

```python
class RUN:
    def enter(self, event):
        print('ENTER RUN')
        if event == RD:
            self.dir += 1
        elif event == LD:
            self.dir -= 1
        elif event == RU:
            self.dir -= 1
        elif event == LU:
            self.dir += 1

    def exit(self):
        print('EXIT RUN')
        self.face_dir = self.dir

    def do(self):
        self.frame = (self.frame + 1) % 8
        self.x += self.dir
        self.x = clamp(0, self.x, 800)

    def draw(self):
        print('DRAW RUN')
        if self.dir == -1:
            self.image.clip_draw(self.frame*100, 0, 100, 100, self.x, self.y)
        elif self.dir == 1:
            self.image.clip_draw(self.frame*100, 100, 100, 100, self.x, self.y)
```

```python
class Boy:

    def __init__(self):
        self.x, self.y = 800 // 2, 90
        self.frame = 0
        self.dir, self.face_dir = 0, 1
        self.image = load_image('animation_sheet.png')

        self.timer = 100

        self.event_que = []
        self.cur_state = IDLE
        self.cur_state.enter(self, None)

    def update(self):
        self.cur_state.do(self)

        if self.event_que:
            event = self.event_que.pop()
            self.cur_state.exit(self)
            self.cur_state = next_state[self.cur_state][event]
            self.cur_state.enter(self, event)

    def draw(self):
        self.cur_state.draw(self)

    def add_event(self, event):
        self.event_que.insert(0, event)

    def handle_event(self, event):
        if (event.type, event.key) in key_event_table:
            key_event = key_event_table[(event.type, event.key)]
            self.add_event(key_event)
```

처리되지 않는 이벤트는 어떻게?

???



캐릭터 컨트롤러 구현 (IDLE & RUN & SLEEP)

boy.py – TIMER 이벤트 추가


```
RD, LD, RU, LU, TIMER = range(5)

key_event_table = {
    (SDL_KEYDOWN, SDLK_RIGHT): RD,
    (SDL_KEYDOWN, SDLK_LEFT): LD,
    (SDL_KEYUP, SDLK_RIGHT): RU,
    (SDL_KEYUP, SDLK_LEFT): LU
}
```

boy.py – SLEEP 상태 함수 추가


```python
class SLEEP:

    def enter(self, event):
        print('ENTER SLEEP')
        self.frame = 0

    def exit(self):
        pass

    def do(self):
        self.frame = (self.frame + 1) % 8

    def draw(self):
        print('DRAW SLEEP')
        if self.face_dir == -1:
            self.image.clip_composite_draw(self.frame * 100, 200, 100, 100,
                                           -3.141592 / 2, '', self.x + 25, self.y - 25, 100, 100)
        else:
            self.image.clip_composite_draw(self.frame * 100, 300, 100, 100,
                                           3.141592 / 2, '', self.x - 25, self.y - 25, 100, 100)
```

boy.py – SLEEP 상태 변화 추가


```
next_state = {
    IDLE:  {RU: RUN,  LU: RUN,  RD: RUN,  LD: RUN, TIMER: SLEEP},
    RUN:   {RU: IDLE, LU: IDLE, RD: IDLE, LD: IDLE},
    SLEEP: {RU: RUN, LU: RUN, RD: RUN, LD: RUN}
}
```

boy.py – IDLE 상태의 TIMER 이벤트 처리


```
class IDLE:
    @staticmethod
    def enter(self,event):
        print('ENTER IDLE'
        self.timer = 1000

    @staticmethod
    def exit(self):
        print('EXIT IDLE')

    @staticmethod
    def do(self):
        self.frame = (self.frame + 1) % 8
        self.timer -= 1
        if self.timer == 0:
            self.add_event(TIMER)
```

```
clip_composite_draw(left, bottom, width, height, rad, flip, x, y, w,h)
```

rad: 회전각도(라디안값)

flip: 반전여부('h': 상하반전, 'v':좌우반전, 'hv': 상하좌우반전