

# Lecture #10. 캐릭터 컨트롤러

2D 게임 프로그래밍

이대현 교수

# 학습 내용

---

- 클래스 변수
- 캐릭터 컨트롤러
- 상태기계
- 이벤트 큐



1000명 선수  
리소스 로딩 최적화

# boys\_team\_1000.py

초기 로딩하는 시간이  
많이 걸린다?



# 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.

## 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.

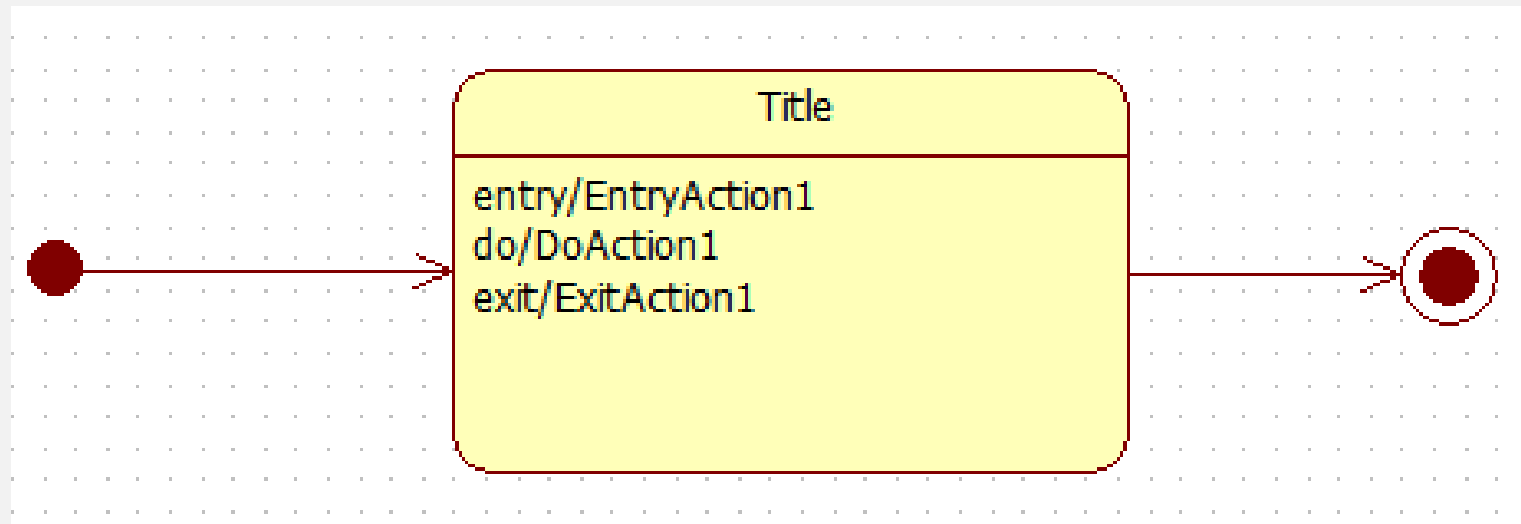
# 상태 다이어그램(State Diagram)

---

- 시스템의 변화를 모델링하는 다이어그램.
- 사건이나 시간에 따라 시스템 내의 객체들이 자신의 상태(state)를 바꾸는 과정을 모델링함.
- Modeling, specification 및 implementation 에 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.

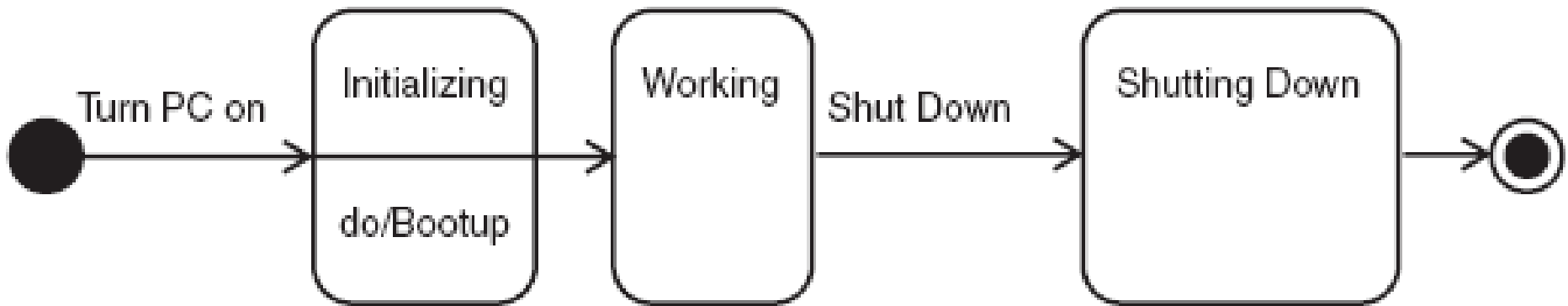
# 상태(State)

- A **state** is a condition in which an object can reside during its lifetime while it satisfies some condition, performs an activity, or waits for an event.
- An **entry action** is the first thing that occurs each time an object enters a particular state.
- An **exit action** is the last thing that occurs each time an object leaves a particular state.
- A **do activity** is an interruptible sequence of actions that an object can perform while it resides in a given state. (Actions are not interruptible.)



# 상태 변화(State Transition)

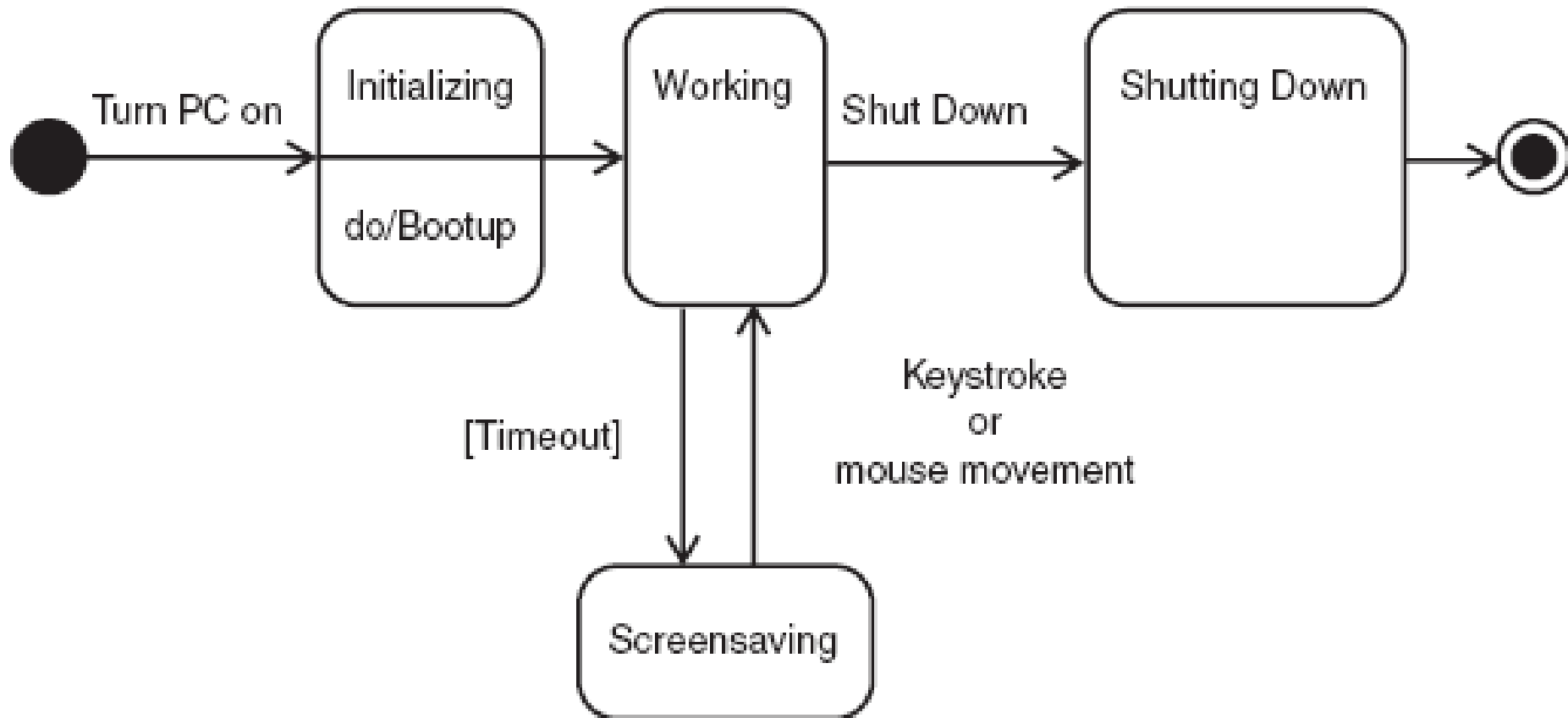
- A transition is a relationship between two states; it indicates that an object in the first state will perform certain actions, then enter the second state when a given event occurs.



# 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.



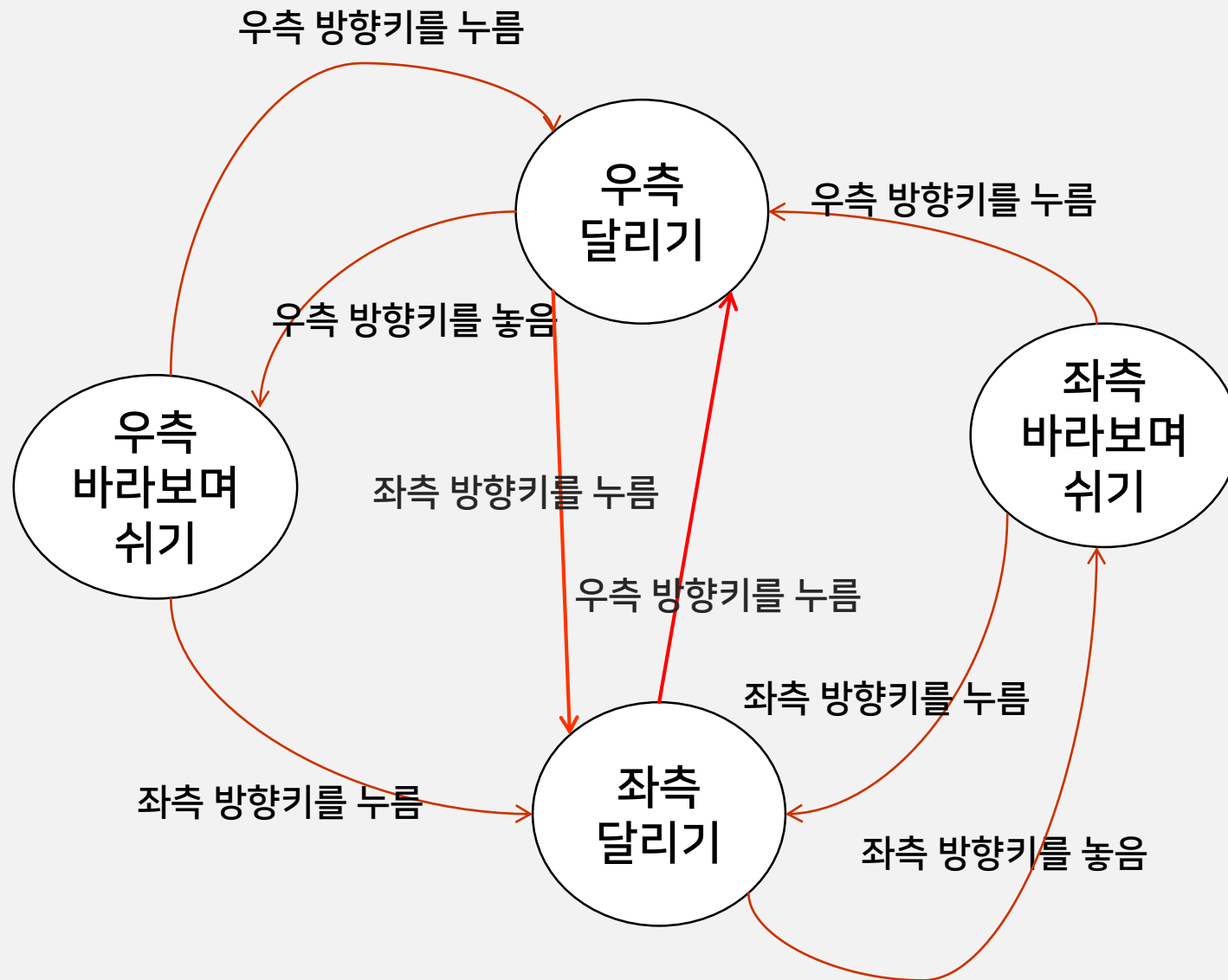
# 상태와 이벤트 찾기

---

- 주인공의 움직임 상태를 찾아보자.
- 주인공의 상태에 변화를 일으킬 수 있는 이벤트를 찾아보자.

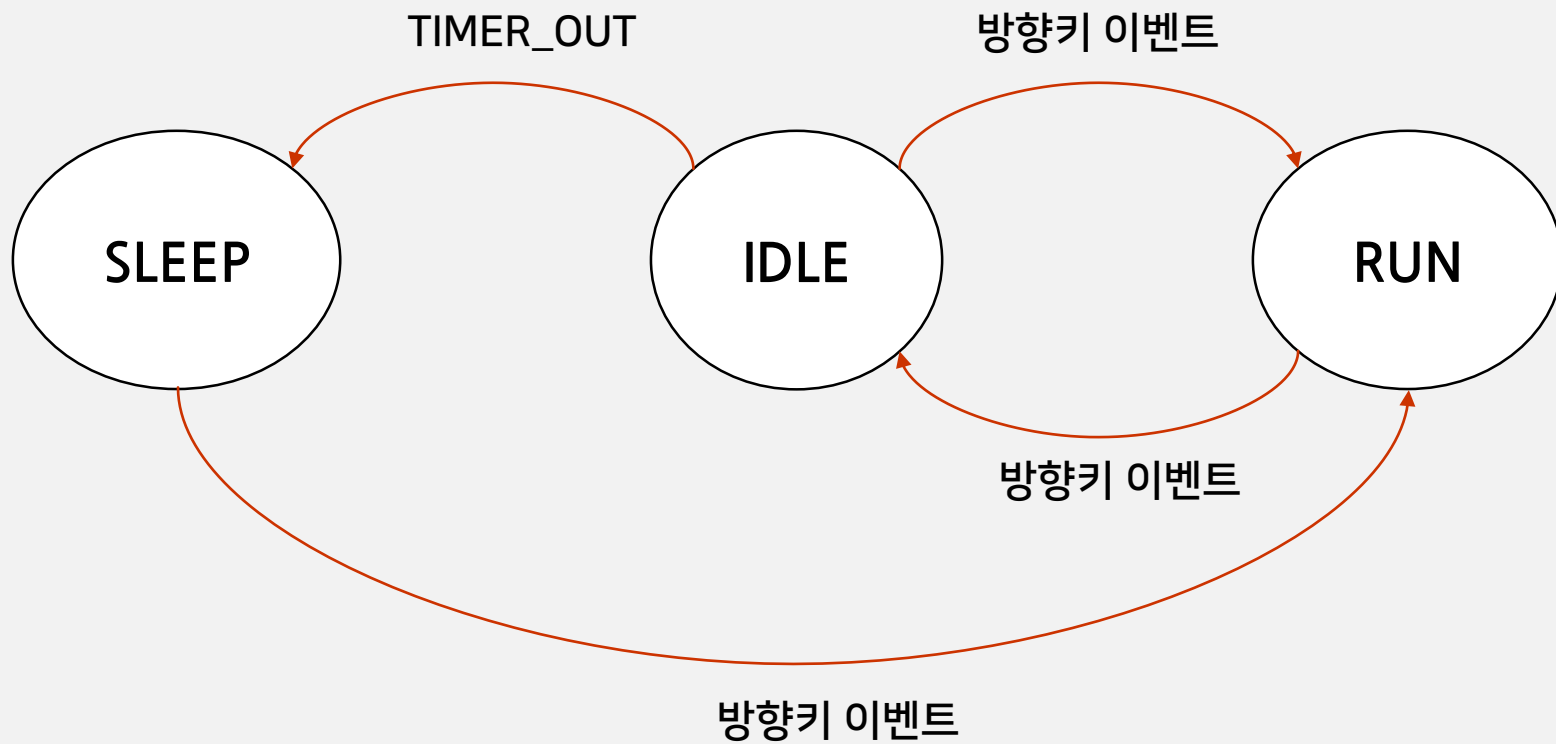


# 상태 다이어그램 #1



# 상태 다이어그램 #2

---





# 캐릭터 컨트롤러 구현(IDLE & RUN)

# Source Code Files

---

- `game_framework.py`
- `mygame.py` – 실행 시작 파일
- `main_state.py` – 메인 게임상태
- `Grass.py` – 잔디 클래스
- `Boy.py` – 소년 클래스 (실습 코딩)



```
# Boy State
```

```
IDLE, RUN = range(2)
```

```
# Boy Event
```

```
RIGHT_DOWN, LEFT_DOWN, RIGHT_UP, LEFT_UP = range(4)
```

```
key_event_table = {  
    (SDL_KEYDOWN, SDLK_RIGHT): RIGHT_DOWN,  
    (SDL_KEYDOWN, SDLK_LEFT): LEFT_DOWN,  
    (SDL_KEYUP, SDLK_RIGHT): RIGHT_UP,  
    (SDL_KEYUP, SDLK_LEFT): LEFT_UP  
}
```

```
next_state_table = {  
    IDLE: {RIGHT_UP: RUN, LEFT_UP: RUN, RIGHT_DOWN: RUN, LEFT_DOWN: RUN},  
    RUN: {RIGHT_UP: IDLE, LEFT_UP: IDLE, LEFT_DOWN: IDLE, RIGHT_DOWN: IDLE}  
}
```

# boy.py – IDLE state



```
def enter_IDLE(self):
    self.timer = 1000
    self.frame = 0

def exit_IDLE(self):
    pass

def do_IDLE(self):
    self.frame = (self.frame + 1) % 8
    self.timer -= 1

def draw_IDLE(self):
    if self.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)
```

# boy.py – RUN state



```
def enter_RUN(self):
    self.frame = 0
    self.dir = self.velocity

def exit_RUN(self):
    pass

def do_RUN(self):
    self.frame = (self.frame + 1) % 8
    self.x += self.velocity
    self.x = clamp(25, self.x, 800-25)

def draw_RUN(self):
    if self.velocity == 1:
        self.image.clip_draw(self.frame * 100, 100, 100, 100, self.x, self.y)
    else:
        self.image.clip_draw(self.frame * 100, 0, 100, 100, self.x, self.y)
```



```
def change_state(self, state):  
    self.exit_state[self.cur_state](self)  
    self.enter_state[state](self)  
    self.cur_state = state
```

```
enter_state = {IDLE: enter_IDLE, RUN: enter_RUN}  
exit_state  = {IDLE: exit_IDLE,  RUN: exit_RUN}  
do_state    = {IDLE: do_IDLE,    RUN: do_RUN}  
draw_state  = {IDLE: draw_IDLE,  RUN: draw_RUN}
```





```
def update(self):
    self.do_state[self.cur_state](self)
    if len(self.event_que) > 0:
        event = self.event_que.pop()
        self.change_state(next_state_table[self.cur_state][event])

def draw(self):
    self.draw_state[self.cur_state](self)

def handle_event(self, event):
    if (event.type, event.key) in key_event_table:
        key_event = key_event_table[(event.type, event.key)]
        if key_event == RIGHT_DOWN:
            self.velocity += 1
        elif key_event == LEFT_DOWN:
            self.velocity -= 1
        elif key_event == RIGHT_UP:
            self.velocity -= 1
        elif key_event == LEFT_UP:
            self.velocity += 1
        self.add_event(key_event)
```

시스템



캐릭터 컨트롤러 구현

(IDLE & RUN & SLEEP)

# boy.py – SLEEP state와 TIME\_OUT 이벤트 추가



```
# Boy State
```

```
IDLE, RUN, SLEEP = range(3)
```

```
# Boy Event
```

```
RIGHT_DOWN, LEFT_DOWN, RIGHT_UP, LEFT_UP, TIME_OUT = range(5)
```

```
key_event_table = {  
    (SDL_KEYDOWN, SDLK_RIGHT): RIGHT_DOWN,  
    (SDL_KEYDOWN, SDLK_LEFT): LEFT_DOWN,  
    (SDL_KEYUP, SDLK_RIGHT): RIGHT_UP,  
    (SDL_KEYUP, SDLK_LEFT): LEFT_UP  
}
```

```
next_state_table = {  
    IDLE: {RIGHT_UP: RUN, LEFT_UP: RUN, RIGHT_DOWN: RUN, LEFT_DOWN: RUN,  
TIME_OUT: SLEEP},  
    RUN: {RIGHT_UP: IDLE, LEFT_UP: IDLE, LEFT_DOWN: IDLE, RIGHT_DOWN: IDLE},  
    SLEEP: {LEFT_DOWN: RUN, RIGHT_DOWN: RUN}  
}
```

# boy.py – SLEEP 상태 함수 추가



```
def enter_SLEEP(self):
    self.frame = 0

def exit_SLEEP(self):
    pass

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

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

# boy.py – SLEEP\_TIMER 이벤트 처리



```
def do_IDLE(self):  
    self.frame = (self.frame + 1) % 8  
    self.timer -= 1  
    if self.timer == 0:  
        self.add_event(SLEEP_TIMER)
```

# boy.py – SLEEP 함수 연결



```
enter_state = {IDLE: enter_IDLE, RUN: enter_RUN, SLEEP: enter_SLEEP}  
exit_state = {IDLE: exit_IDLE, RUN: exit_RUN, SLEEP: exit_SLEEP}  
do_state = {IDLE: do_IDLE, RUN: do_RUN, SLEEP: do_SLEEP}  
draw_state = {IDLE: draw_IDLE, RUN: draw_RUN, SLEEP: draw_SLEEP}
```