

# Lecture #10. 게임 프레임웍

2D 게임 프로그래밍

이대현 교수



한국공학대학교  
TECH UNIVERSITY OF KOREA

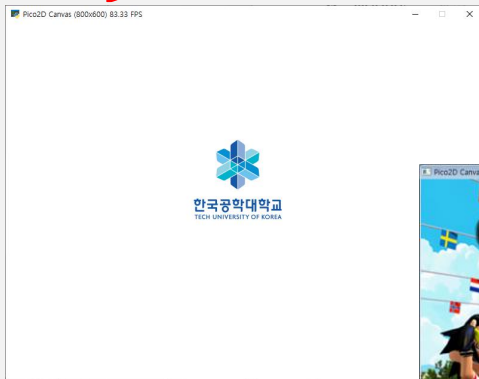
# 학습 내용

---

- 파이썬 모듈(module)
- 게임 상태
- 게임 프레임웍
- 로고 화면의 구현
- 타이틀 화면 상태의 구현
- 메인 게임 상태의 구현

# 오늘 만들어 볼 것

로고화면상태



타이틀화면상태



메인플레이상태



# Python Module

- 파이썬의 정의(definitions)와 문장(statements)을 담고 있는 파일
- 파일이름: 00000.py (확장자:py)

```
class Grass:
    pass

def update():
    global x
    x = x + 1

x = 0
update()
```

클래스 정의

함수 정의

문장

- 그 자체로도 실행 가능하며, 다른 모듈에서 импорт(import)해서 사용할 수도 있음. импорт되면, 그 자체가 하나의 객체가 됨.(싱글톤 객체가 됨)

# show\_files.py

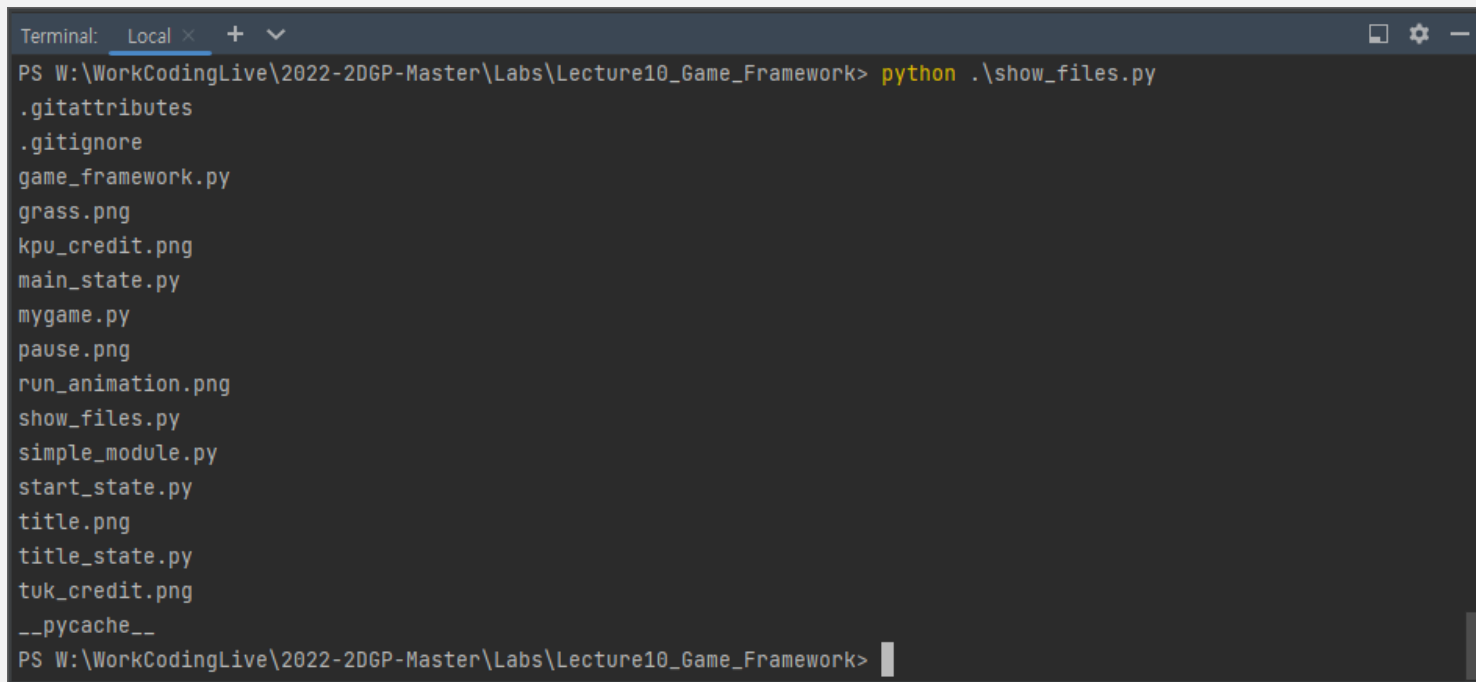
---

```
import os

file_name_list = os.listdir()
for name in file_name_list:
    print(name)
```

# show\_files.py 을 단독 실행하면?

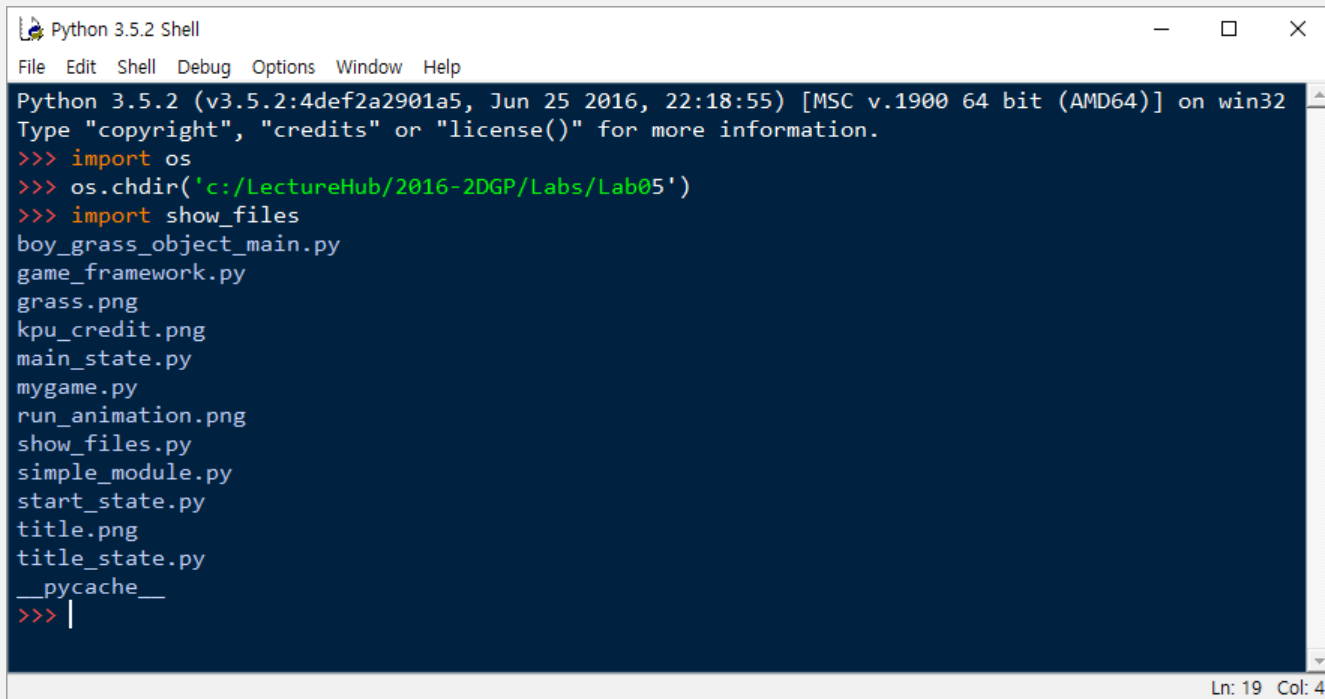
- Pycharm 의 Terminal 에서 직접 show\_files.py 를 치면, 실행이 됨. 아니면 python show\_files.py 해도 실행이 됨.



```
Terminal: Local x + v
PS W:\WorkCodingLive\2022-20GP-Master\Labs\Lecture10_Game_Framework> python .\show_files.py
.gitattributes
.gitignore
game_framework.py
grass.png
kpu_credit.png
main_state.py
mygame.py
pause.png
run_animation.png
show_files.py
simple_module.py
start_state.py
title.png
title_state.py
tuk_credit.png
__pycache__
PS W:\WorkCodingLive\2022-20GP-Master\Labs\Lecture10_Game_Framework>
```

# show\_files.py 을 import 하면?

- 모듈을 임포트하는 경우, 모듈이 실행됨.



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.chdir('c:/LectureHub/2016-2DGP/Labs/Lab05')
>>> import show_files
boy_grass_object_main.py
game_framework.py
grass.png
kpu_credit.png
main_state.py
mygame.py
run_animation.png
show_files.py
simple_module.py
start_state.py
title.png
title_state.py
__pycache__
>>> |
```

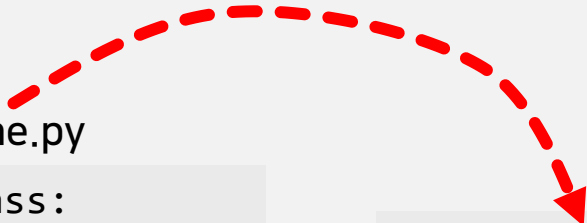
Ln: 19 Col: 4

# module 의 사용: 임포트한 후, 모듈이름.0000

---

game.py

```
class Grass:  
    pass  
  
def update():  
    global x  
    x = x + 1  
  
x = 0  
update()
```



```
import game  
  
grass = game.Grass()  
  
game.update()
```





리팩토링

# 리팩토링 (1) - 기존 코드를 play\_state.py 로 정리

## boy\_grass\_object.py

```
from pico2d import *

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

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

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

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

    def draw(self):
        self.image.clip_draw(self.frame*100, 0, 100, 100, self.x, self.y)

def handle_events():
    global running
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            running = False
        elif event.type == SDL_KEYDOWN and event.key == SDLK_ESCAPE:
            running = False

open_canvas()

boy = Boy()
grass = Grass()
running = True

# game main loop code
while running:
    handle_events()
    boy.update()
    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()
    delay(0.05)

# finalization code
close_canvas()
```

## play\_state.py

```
boy = None
grass = None
running = None

# 초기화
def enter():
    global boy, grass, running
    boy = Boy()
    grass = Grass()
    running = True

# 종료
def exit():
    global boy, grass
    del boy
    del grass

def update():
    boy.update()

def draw():
    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()

open_canvas()

enter()
while running:
    handle_events()
    update()
    draw()
exit()

close_canvas()
```

수정

# 리팩토링 (2) – play\_state.py 에서 메인 코드 분리하여 main.py 생성

## play\_state.py

```
boy = None
grass = None
running = None

# 초기화
def enter():
    global boy, grass, running
    boy = Boy()
    grass = Grass()
    running = True

# 종료
def exit():
    global boy, grass
    del boy
    del grass

def update():
    boy.update()

def draw():
    clear_canvas()
    grass.draw()
    boy.draw()
    update_canvas()

open_canvas()

enter()
while running:
    handle_events()
    update()
    draw()
exit()

close_canvas()
```

분리

## main.py

```
import pico2d
import play_state

pico2d.open_canvas()

play_state.enter()

while play_state.running:
    play_state.handle_events()
    play_state.update()
    play_state.draw()

play_state.exit()

pico2d.close_canvas()
```



로그 화면 구현

# 로고상태의 구현: logo\_state.py (1)



```
from pico2d import *

running = True
image = None
logo_time = 0.0

def enter():
    global image
    image = load_image('tuk_credit.png')
    pass

def exit():
    global image
    del image
    pass
```

## 로고상태의 구현: logo\_state.py (2)



```
def update():
    global logo_time
    global running
    if logo_time > 1.0:
        logo_time = 0
        running = False
    delay(0.01)
    logo_time += 0.01

def draw():
    clear_canvas()
    image.draw(400, 300)
    update_canvas()

def handle_events():
    events = get_events()
```



```
import pico2d
import logo_state

start_state = logo_state      # 모듈을 변수로 취급할 수 있음.

pico2d.open_canvas()

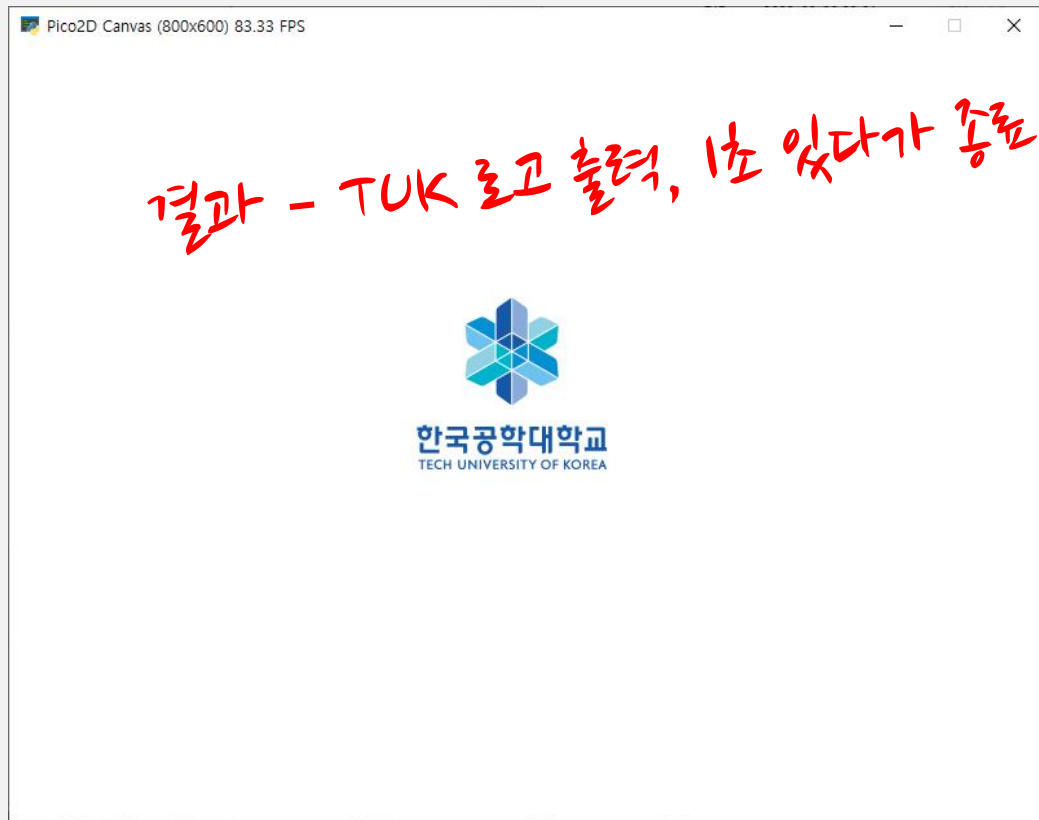
start_state.enter()

while start_state.running:
    start_state.handle_events()
    start_state.update()
    start_state.draw()

start_state.exit()

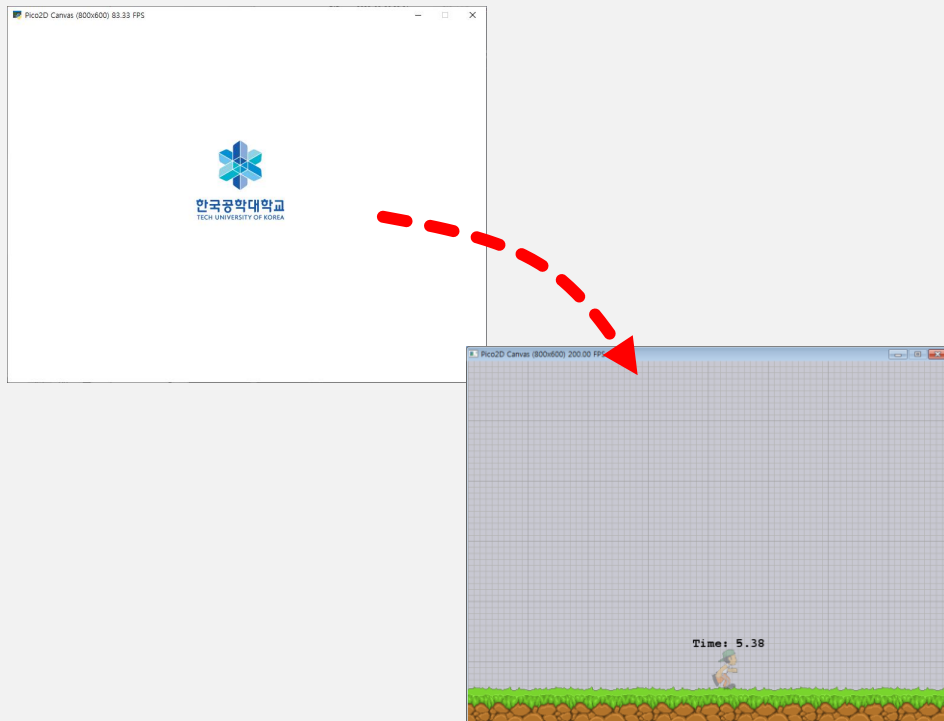
pico2d.close_canvas()
```

# 실행 - main.py 를 실행





# 로고 화면 후에 플레이 모드로 가려면?



# 게임 상태(Game State)의 이해 (1)

## ■ 게임 상태란?

- 게임 프로그램 실행 중의 어떤 특정 상황(또는 모드, 씬).
- 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임의 상태에 따라 달라짐.
- 작은 게임 루프로 볼 수 있음.

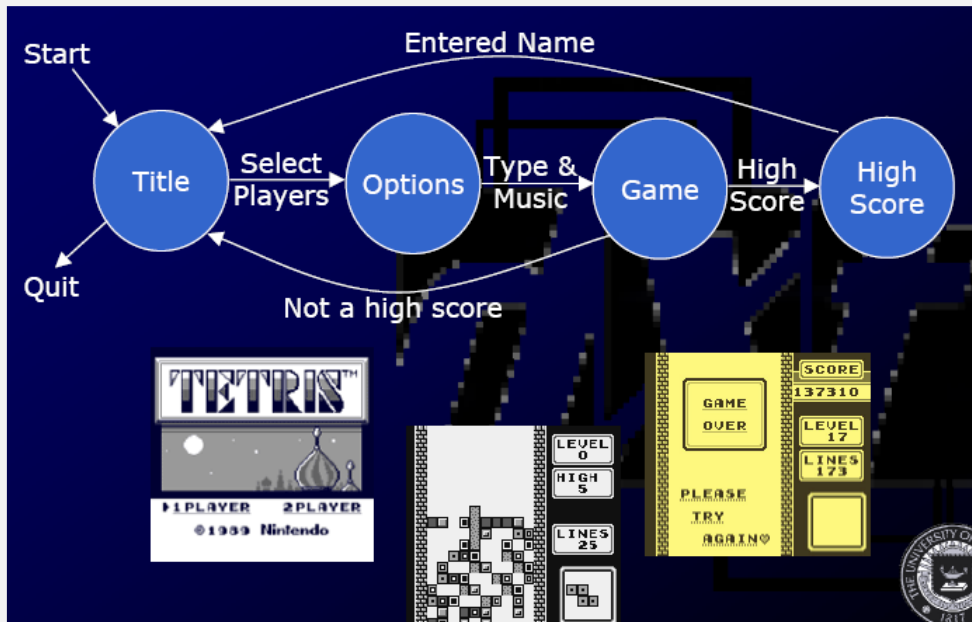


- 맵 선택 상태.
- 방향키는 맵 선택을 처리.

- 게임 메인 플레이 상태.
- 방향키는 캐릭터의 이동을 처리.

# 게임 상태(Game State)의 이해 (2)

- 게임 프로그램은 게임 상태(모드, 씬)의 집합으로 구현됨.
- 예) 테트리스 게임

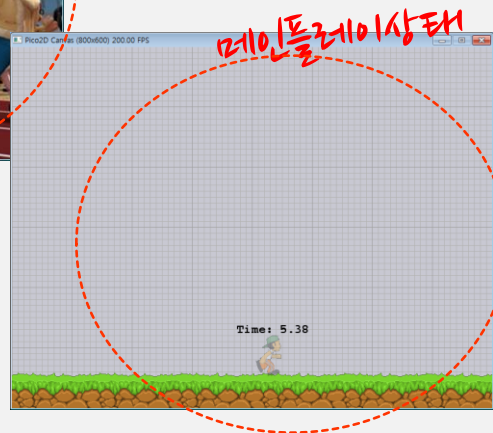


# 게임 프레임워크

---

- 게임 상태들을 효과적으로 연결하는 소프트웨어 구조.
- 일종의 Task Switching Software
- 디자인 패턴 중, State Pattern에 해당됨.

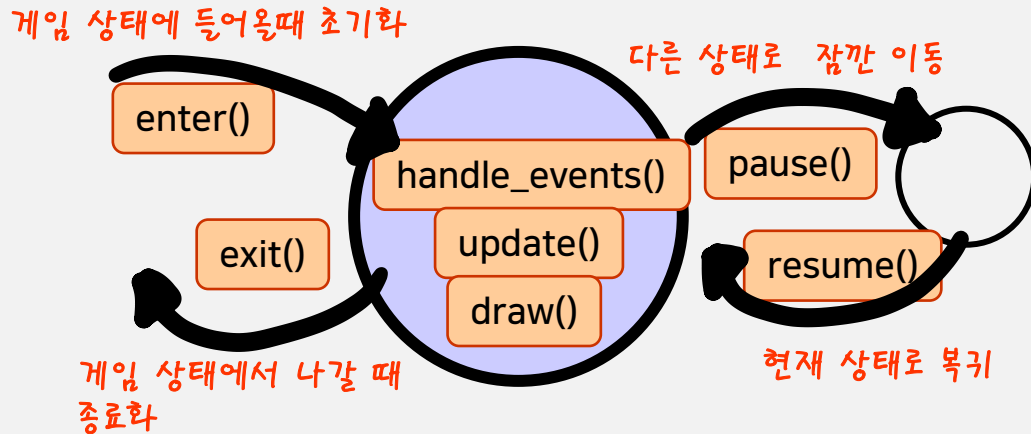
# 게임 프레임워크 활용 순서 #1. 각각의 상태를 구현



# 게임 프레임워크 활용 순서 #2. 상태간의 전환을 구현.



# 게임 상태의 구현



# 상태간의 전환: game\_framework을 이용

---

**run(state):**

state를 시작 게임 상태로 하여, 게임 실행을 시작함.

**quit():** 게임을 중단

**change\_state(state):**

게임 상태를 state로 변화. 이전 게임 상태를 완전히 나옴.

**push\_state(state):**

게임 상태를 state로 변화. 이전 게임 상태는 남아 있음.

**pop\_state():** 이전 게임 상태로 복귀





## 로그 화면 구현 (2)

# 로고상태의 구현: logo\_state.py 수정



```
import game_framework
from pico2d import *

image = None
logo_time = 0.0

def enter():
    global image
    image = load_image('tuk_credit.png')

def exit():
    global image
    del image

def update():
    global logo_time
    # global running
    if logo_time > 1.0:
        logo_time = 0
        game_framework.quit()
    delay(0.01)
    logo_time += 0.01

def draw():
    clear_canvas()
    image.draw(400, 300)
    update_canvas()

def handle_events():
    events = get_events()
```

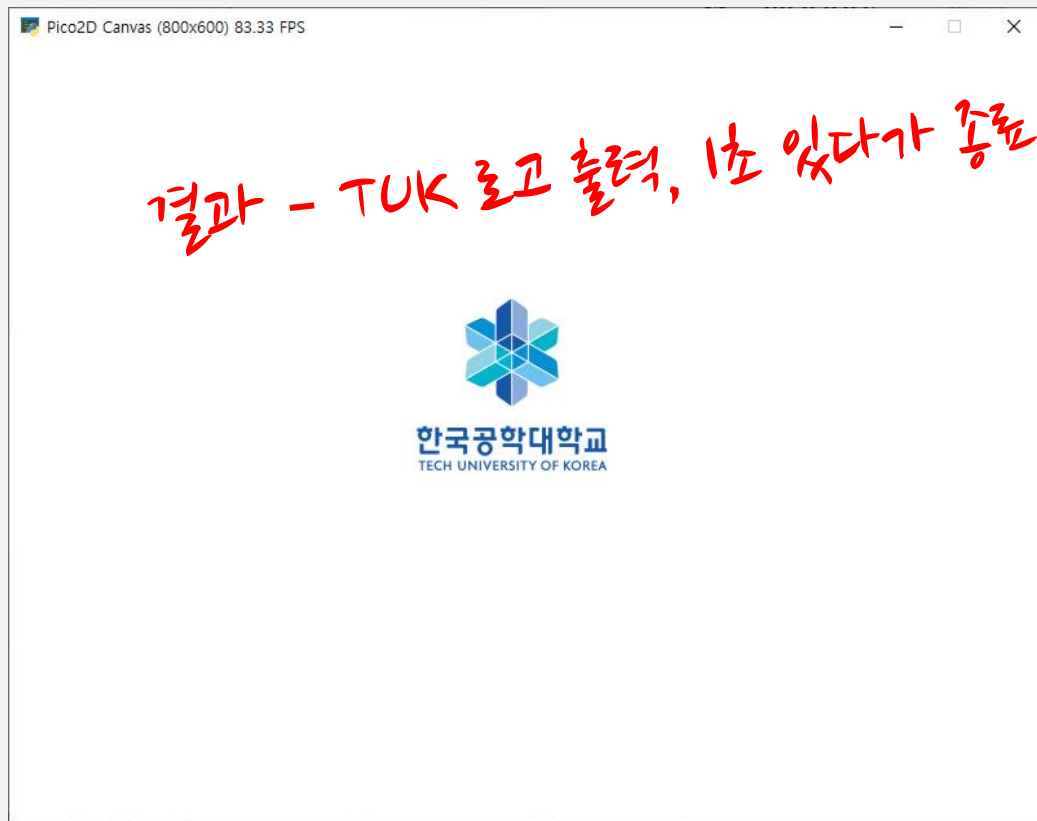


```
import pico2d
import game_framework

import logo_state

pico2d.open_canvas()
game_framework.run(logo_state)
pico2d.close_canvas()
```

# 실행 - mygame.py 를 실행



# 게임 상태의 뼈대

---

```
def enter(): pass

def exit(): pass

def update(): pass

def draw(): pass

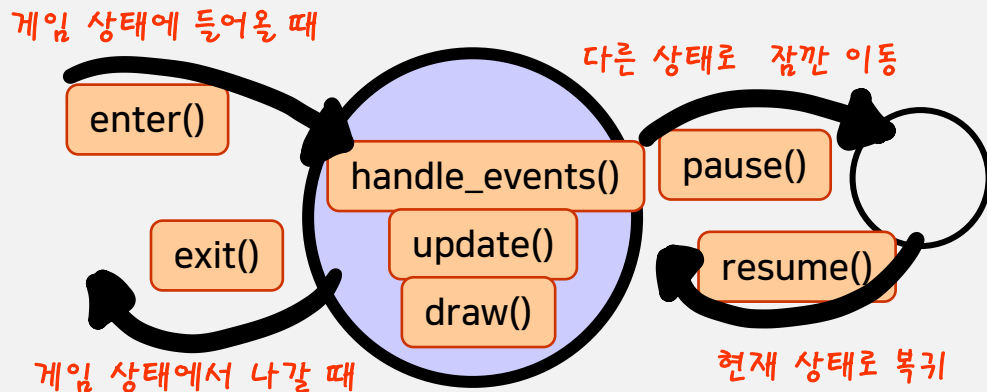
def handle_events(): pass

def pause(): pass

def resume(): pass
```

# 0000\_state 의 구현과 활용

1. 0000\_state.py 를 만든다
2. 0000\_state.py의 내부 함수들을 작성한다.
3. 다른 소스에서 import 0000\_state 를 해서 활용한다.



# 게임의 구성과 시작 - 게임프레임워크 활용

---

- `game_framework` 를 import 한다.
- 시작 게임 상태를 import 한다.
- 시작 게임 상태를 지정한 후, `game_framework` 를 시작한다.

```
import game_framework
import pico2d

import start_state

pico2d.open_canvas()
game_framework.run(start_state)
pico2d.close_canvas()
```

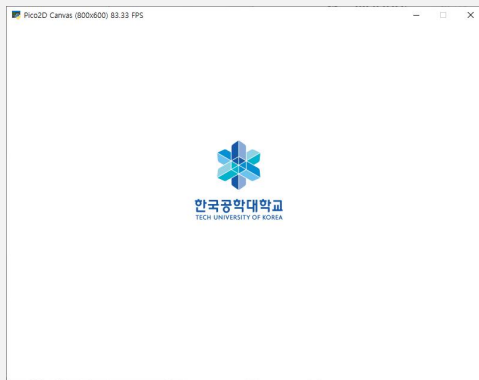


타이틀 상태 추가



# 로고 화면에 이어지는 타이틀 화면

logo\_state.py



title\_state.py



# title\_state.py



```
import game_framework
from pico2d import *

image = None

def enter():
    global image
    image = load_image('title.png')

def exit():
    global image
    del image

def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN and event.key == SDLK_ESCAPE:
            game_framework.quit()

def draw():
    clear_canvas()
    image.draw(400, 300)
    update_canvas()
```

# logo\_state.py 의 수정



```
import title_state

def update():
    global logo_time

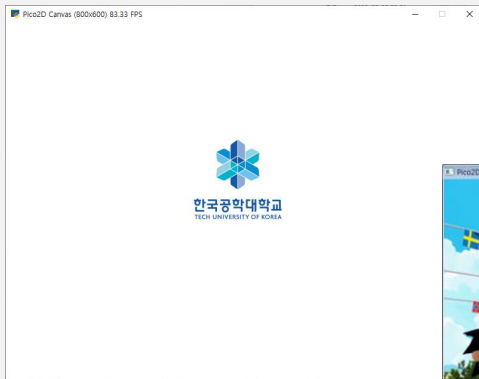
    if (logo_time > 1.0):
        logo_time = 0
        # game_framework.quit()
        game_framework.change_state(title_state)
    delay(0.01)
    logo_time += 0.01
```



게임 메인 상태 추가

# 로고 화면에 이어지는 타이틀 화면

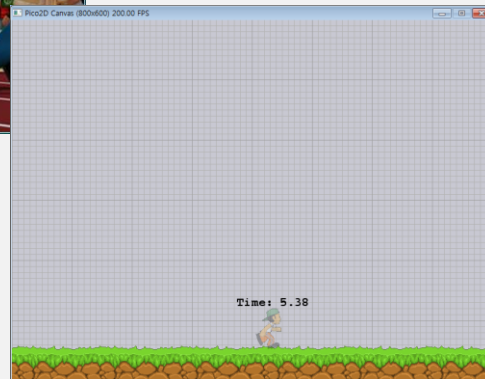
logo\_state.py



title\_state.py



play\_state.py



# play\_state.py 의 수정



```
import game_framework
import title_state

def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN and event.key == SDLK_ESCAPE:
            game_framework.change_state(title_state)
    delay(0.01)
```

# title\_state.py 의 수정



```
import play_state

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



게임 상태의 self test



# \_\_name\_\_ 속성

- 파이썬 내부에서 사전에 정의된 속성.
- 일반적으로 그 값은 모듈의 이름임.
- 하지만, module 을 임포트하지 않고, 직접 실행하는 경우, "\_\_main\_\_" 이라는 문자열값을 갖게 되어, 현재 모듈이 단독적으로 실행되는 상황을 구분함.

simple\_module.py

```
print("The value of __name__ is '" + __name__ + "'.")
```

import 하는 경우

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>>
>>> import simple_module
The value of __name__ is 'simple_module'.
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

실행하는 경우

```
명령 프롬프트
c:\Temp\TestPython>simple_module.py
The value of __name__ is '__main__'.
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
c:\Temp\TestPython>
```

# \_\_name\_\_ 속성의 활용

- 어떤 모듈이 임포트되는 경우와 직접 실행이 되는 경우를 구분할 수 있음.
- 따라서 모듈이 단독으로 직접 실행되는 상황에서만 필요한 일들을 구분해서 처리할 수 있음.
- 메인함수를 흉내내거나, 모듈 자체를 self test 하는 용도로 활용 가능

```
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..  
# 어쩌구저쩌구..
```

```
def main():  
    print("이 함수는 단독으로 모듈을 실행할 경우만, 실행됩니다.")
```

```
if __name__ == '__main__':  
    main()
```

# play\_state.py 의 self test - 마지막 부분에 추가



```
def test_self():  
    import play_state  
    pico2d.open_canvas()  
    game_framework.run(play_state)  
    pico2d.close_canvas()  
  
if __name__ == '__main__':  
    test_self()
```

# chage\_state

다음 상태로 가기 전에  
title\_state.exit()가 호출됨.  
즉, 모든 객체가 소멸됨.

title\_state.py

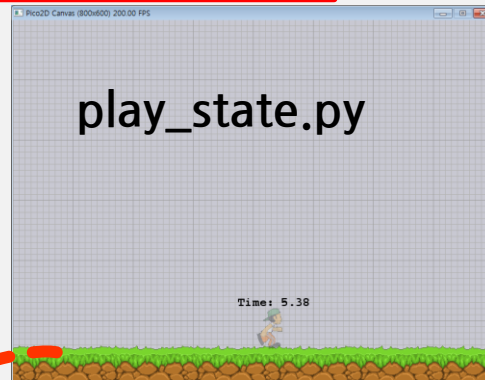


title\_state.enter() 실행

change\_state(play\_state)

play\_state.enter() 실행

change\_state(title\_state)



play\_state.exit() 실행

# push\_state, pop\_state

다음 상태로 가기 전에  
play\_state.pause()가 호출됨.  
play\_state의 모든 객체는 유지

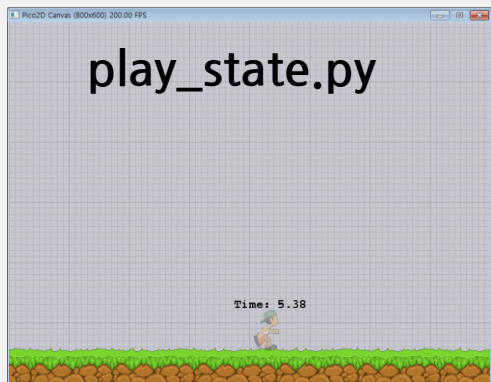
Item\_state.enter() 실행

push\_state(item\_state)

pop\_state()

play\_state.resume()이 호출됨.

Item\_state.exit() 실행





## 아이템 스테이트 구현

# 상태간의 전환: game\_framework을 이용

---

**run(state):**

state를 시작 게임 상태로 하여, 게임 실행을 시작함.

**quit():** 게임을 중단

**change\_state(state):**

게임 상태를 state로 변화. 이전 게임 상태를 완전히 나옴.

**push\_state(state):**

게임 상태를 state로 변화. 기존 게임 상태는 남아 있음.

**pop\_state():** 이전 게임 상태로 복귀





# play\_state.py 수정 (1)



```
class Boy:
    def __init__(self):
        self.x, self.y = 0, 90
        self.frame = 0
        self.image = load_image('run_animation.png')
        self.ball_image = load_image('ball21x21.png')
        self.big_ball_image = load_image('ball41x41.png')
        self.item = None

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

    def draw(self):
        self.image.clip_draw(self.frame*100, 0, 100, 100, self.x, self.y)
        if self.item == 'BigBall':
            self.big_ball_image.draw(self.x+10, self.y+50)
        elif self.item == 'Ball':
            self.ball_image.draw(self.x+10, self.y+50)
```

## play\_state.py 수정 (2)



```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN:
            if event.key == SDLK_ESCAPE:
                game_framework.change_state(title_state)
            elif event.key == SDLK_i:
                game_framework.push_state(item_state)
```

# pause 와 resume 추가



- `push_state` 와 `pop_state` 를 호출하면, `pause`와 `resume` 이 call back 되므로 실제 내용은 없더라도 뼈대는 만들어줘야 함.

```
def pause():  
    pass  
  
def resume():  
    pass
```



```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN:
            match event.key:
                case pico2d.SDLK_ESCAPE:
                    game_framework.pop_state()
                case pico2d.SDLK_0:
                    play_state.boy.item = None
                    game_framework.pop_state()
                case pico2d.SDLK_1:
                    play_state.boy.item = 'Ball'
                    game_framework.pop_state()
                case pico2d.SDLK_2:
                    play_state.boy.item = 'BigBall'
                    game_framework.pop_state()
```

# item\_state.py 의 self test



```
def test_self():
    import item_state
    pico2d.open_canvas()
    game_framework.fill_states(play_state)
    game_framework.run(item_state)
    pico2d.close_canvas()

if __name__ == '__main__':
    test_self()
```

# 테스트를 위한 게임 상태 미리 채우기

```
def test_self():
    import item_state
    pico2d.open_canvas()
    game_framework.fill_states(play_state)
    game_framework.run(item_state)
    pico2d.close_canvas()

if __name__ == '__main__':
    test_self()
```

play\_state로 앞의 상태를 지정함.

fill\_states(state1, state2, state3, ...)

# 다른 모듈 내의 객체를 직접 액세스

---

```
case pico2d.SDLK_1:
    play_state.boy.item = 'Ball'
    game_framework.pop_state()
case pico2d.SDLK_2:
    play_state.boy.item = 'BigBall'
    game_framework.pop_state()
```

객체의 속성을 직접 액세스 가능함.  
좀 더 좋은 방법은 메소드 추가.  
예) `boy.set_item('Ball')`

# 플레이 배경을 유지하려면?

## item\_state.py

```
def draw():  
    clear_canvas()  
    play_state.draw_world()  
    image.draw(400, 300)  
    update_canvas()
```

play\_state의 배경을 그리고 나서,  
이후에 item\_state를 그림.

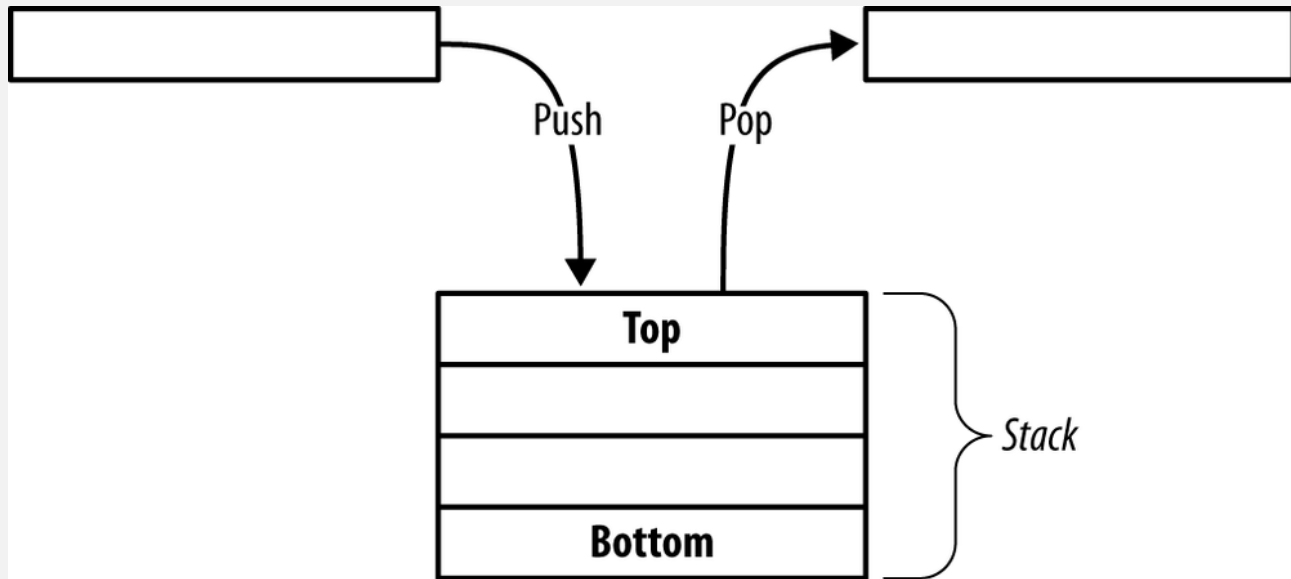
## play\_state.py

```
def draw_world():  
    grass.draw()  
    boy.draw()  
  
def draw():  
    clear_canvas()  
    draw_world()  
    update_canvas()
```

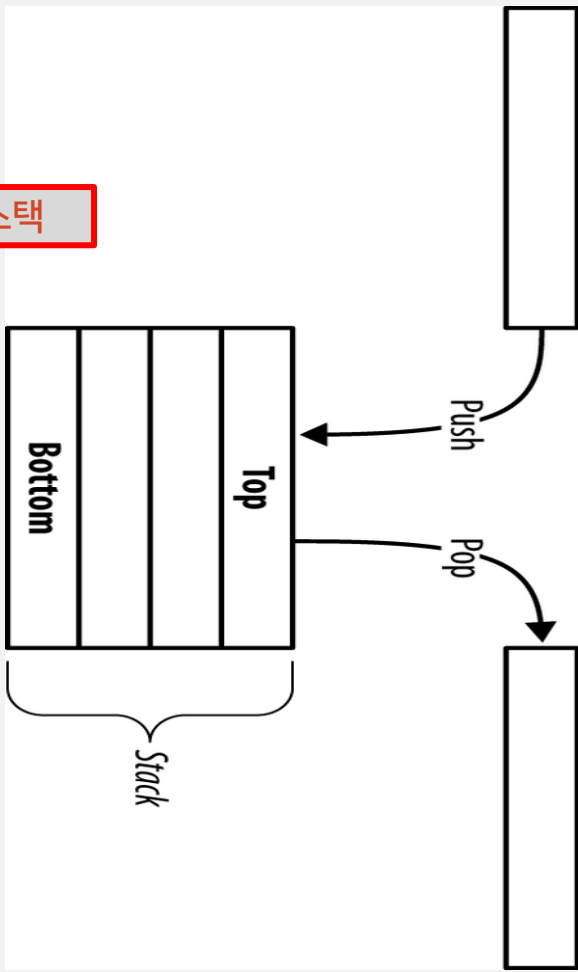




# Stack 자료 구조



list 를 이용한 스택



# game\_framework.py 분석(1)

```
def run(start_state):  
    global running, stack  
    running = True
```

start\_state 를 담고 있는 스택을 생성

```
    stack = [start_state]  
    start_state.enter()
```

```
    while (running):  
        stack[-1].handle_events()  
        stack[-1].update()  
        stack[-1].draw()
```

현재 게임 상태(다시 말하면, stack top에 있는 게임 상태)에 대한 게임 루프를 진행

```
    # repeatedly delete the top of the stack  
    while (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()
```

스택에 남아있는 모든 게임 상태들을 차례로 제거

## game\_framework.py 분석 (2)

```
def change_state(state):  
    global stack  
    if (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()  
    stack.append(state)  
    state.enter()
```

현재 상태를 삭제한 후,  
새로운 상태를 추가하고, enter로 들어간다.

```
def pop_state():  
    global stack  
    if (len(stack) > 0):  
        stack[-1].exit()  
        stack.pop()  
    if (len(stack) > 0):  
        stack[-1].resume()
```

Stack Top의 상태를 exit() 한 후, 상태를 제거.  
이제 Stack Top에는 이전 상태가 있으므로, 이  
내용을 다시 가져옴(resume)

## game\_framework.py 분석 (3)

```
def push_state(state):  
    global stack  
    if (len(stack) > 0):  
        stack[-1].pause()  
    stack.append(state)  
    state.enter()
```

현재 상태를 저장하고(Pause),  
새로운 상태로 들어감.

```
def quit():  
    global running  
    running = False
```

# 간단한 게임 상태의 연결 Ver 1.

---

```
pico2d.open_canvas()

start_state = logo_state
start_state.enter()
while start_state.running:
    start_state.handle_events()
    start_state.update()
    start_state.draw()
start_state.exit()

start_state = play_state
start_state.enter()
while start_state.running:
    start_state.handle_events()
    start_state.update()
    start_state.draw()
start_state.exit()

pico2d.close_canvas()
```

## 간단한 게임 상태의 연결 Ver 2.

---

```
pico2d.open_canvas()

states = [logo_state, play_state]
for state in states:
    state.enter()
    while state.running:
        state.handle_events()
        state.update()
        state.draw()
    state.exit()

pico2d.close_canvas()
```



# Boy 좌우 이동 구현 추가 (1)

```
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')
        self.ball_image = load_image('ball21x21.png')
        self.big_ball_image = load_image('ball41x41.png')
        self.item = None

    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)
        if self.item == 'BigBall':
            self.big_ball_image.draw(self.x+10, self.y+50)
        elif self.item == 'Ball':
            self.ball_image.draw(self.x+10, self.y+50)
```

# Boy 좌우 이동 추가 구현 (2)

```
def handle_events():
    events = get_events()
    for event in events:
        if event.type == SDL_QUIT:
            game_framework.quit()
        elif event.type == SDL_KEYDOWN:
            match event.key:
                case pico2d.SDLK_ESCAPE:
                    game_framework.change_state(title_state)
                case pico2d.SDLK_i:
                    game_framework.push_state(item_state)
                case pico2d.SDLK_LEFT:
                    boy.dir -= 1
                case pico2d.SDLK_RIGHT:
                    boy.dir += 1
        elif event.type == SDL_KEYUP:
            match event.key:
                case pico2d.SDLK_LEFT:
                    boy.dir += 1
                    boy.face_dir = -1
                case pico2d.SDLK_RIGHT:
                    boy.dir -= 1
                    boy.face_dir = 1
```