

2D 게임 프로그래밍

# 제7강 캐릭터컨트롤러

이대현  
한국산업기술대학교



# 학습 내용

- 캐릭터 컨트롤러
- JSON을 활용하는 객체 초기화 및 생성

## 캐릭터 컨트롤러(Character Controller)

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

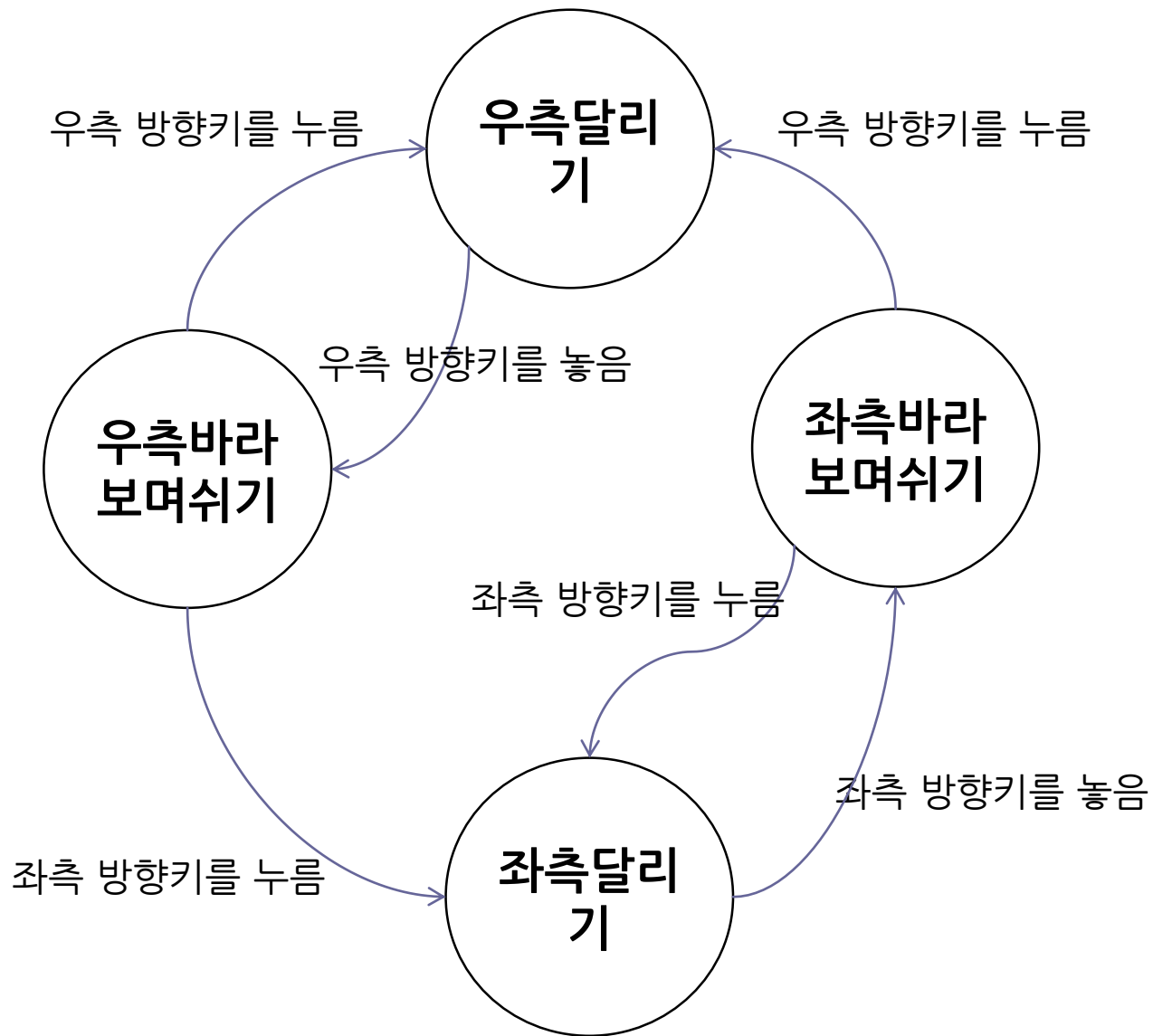


# 우리의 “주인공”은?

## ■ 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.
- 왼쪽 방향키 조작에 대해선 왼쪽으로 달리게 됩니다.
- 캔버스의 좌우측 가장자리에 도착하면 더 이상 달려나가지는 않습니다.

# 상태 다이어그램





# 캐릭터 컨트롤러 구현

# hero\_controller.py - class Hero



```
def handle_event(self, event):
    if (event.type, event.key) == (SDL_KEYDOWN, SDLK_LEFT):
        if self.state in (self.RIGHT_STAND, self.LEFT_STAND):
            self.state = self.LEFT_RUN
    elif (event.type, event.key) == (SDL_KEYDOWN, SDLK_RIGHT):
        if self.state in (self.RIGHT_STAND, self.LEFT_STAND):
            self.state = self.RIGHT_RUN
    elif (event.type, event.key) == (SDL_KEYUP, SDLK_LEFT):
        if self.state in (self.LEFT_RUN,):
            self.state = self.LEFT_STAND
    elif (event.type, event.key) == (SDL_KEYUP, SDLK_RIGHT):
        if self.state in (self.RIGHT_RUN,):
            self.state = self.RIGHT_STAND
```

# hero\_controller.py - class Hero



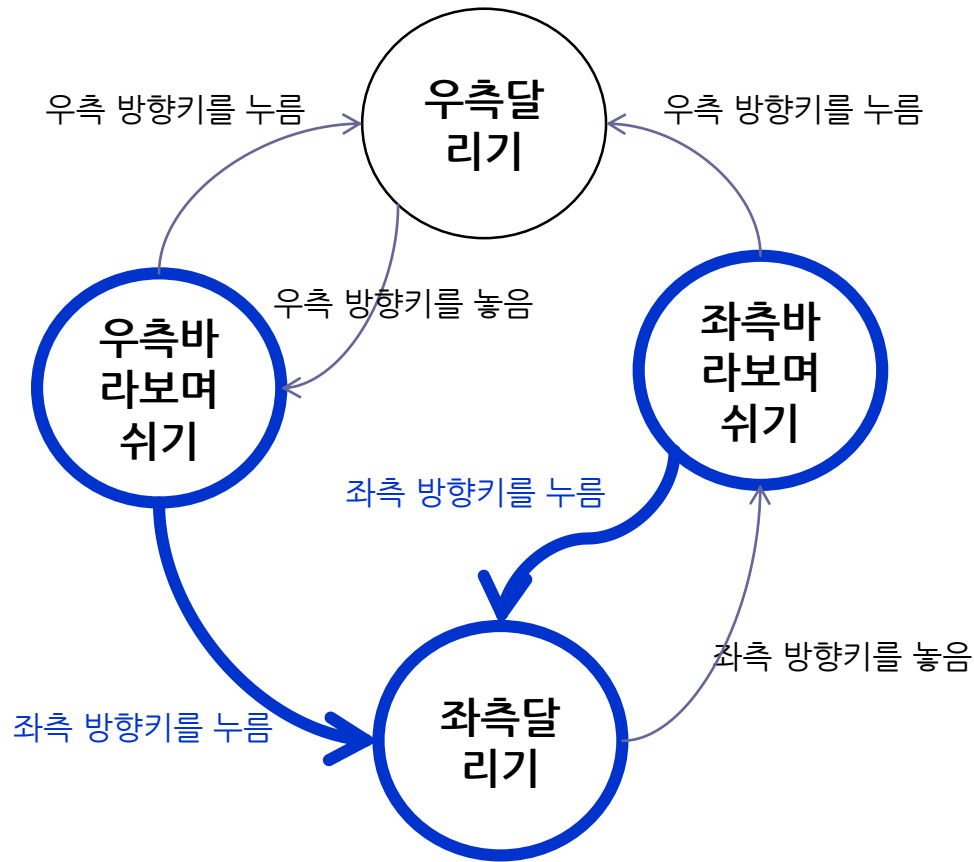
```
def update(self):  
    self.frame = (self.frame + 1) % 8  
    if self.state == self.RIGHT_RUN:  
        self.x = min(800, self.x + 5)  
    elif self.state == self.LEFT_RUN:  
        self.x = max(0, self.x - 5)
```





```
def handle_events():
    global running
    global hero
    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
        else:
            hero.handle_event(event)
```

# 상태 다이어그램의 구현



```
if (event.type, event.key) == (SDL_KEYDOWN, SDLK_LEFT):  
    if self.state in (self.RIGHT_STAND, self.LEFT_STAND):  
        self.state = self.LEFT_RUN
```

```
def update(self):  
    self.frame = (self.frame + 1) % 8  
    if self.state == self.RIGHT_RUN:  
        self.x = min(800, self.x + 5)  
    elif self.state == self.LEFT_RUN:  
        self.x = max(0, self.x - 5)
```

# Hero에 이벤트 처리를 일임: 효과는? 장점은??

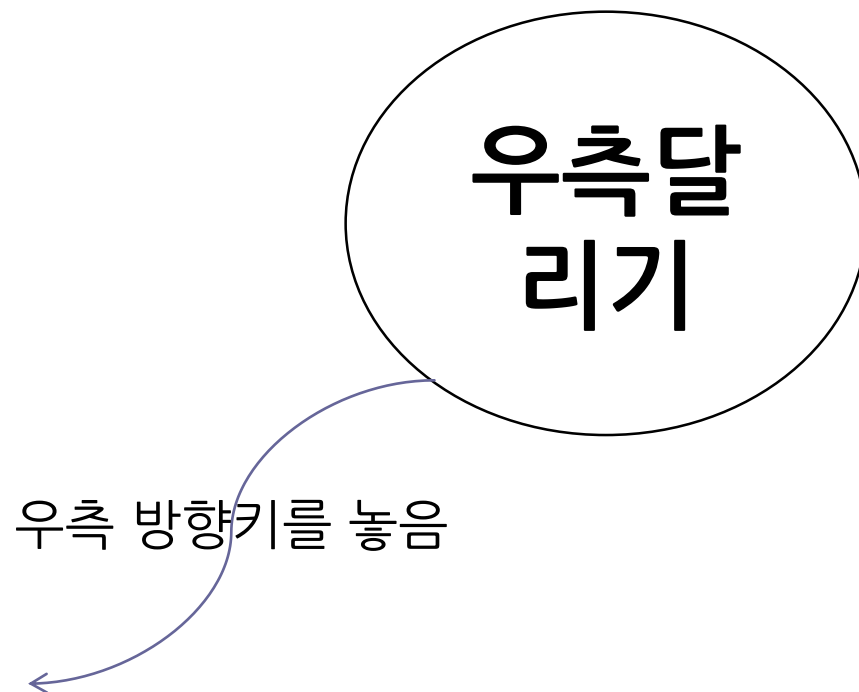
```
def handle_events():  
    global running  
    global hero  
    events = get_events()  
    for event in events:  
        if event.type == SDL_QUIT:  
            running = False  
        elif (event.type == SDL_KEYDOWN and  
              event.key == SDLK_ESCAPE):  
            running = False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

# 앞의 코드는 문제가 있다... 어떤?

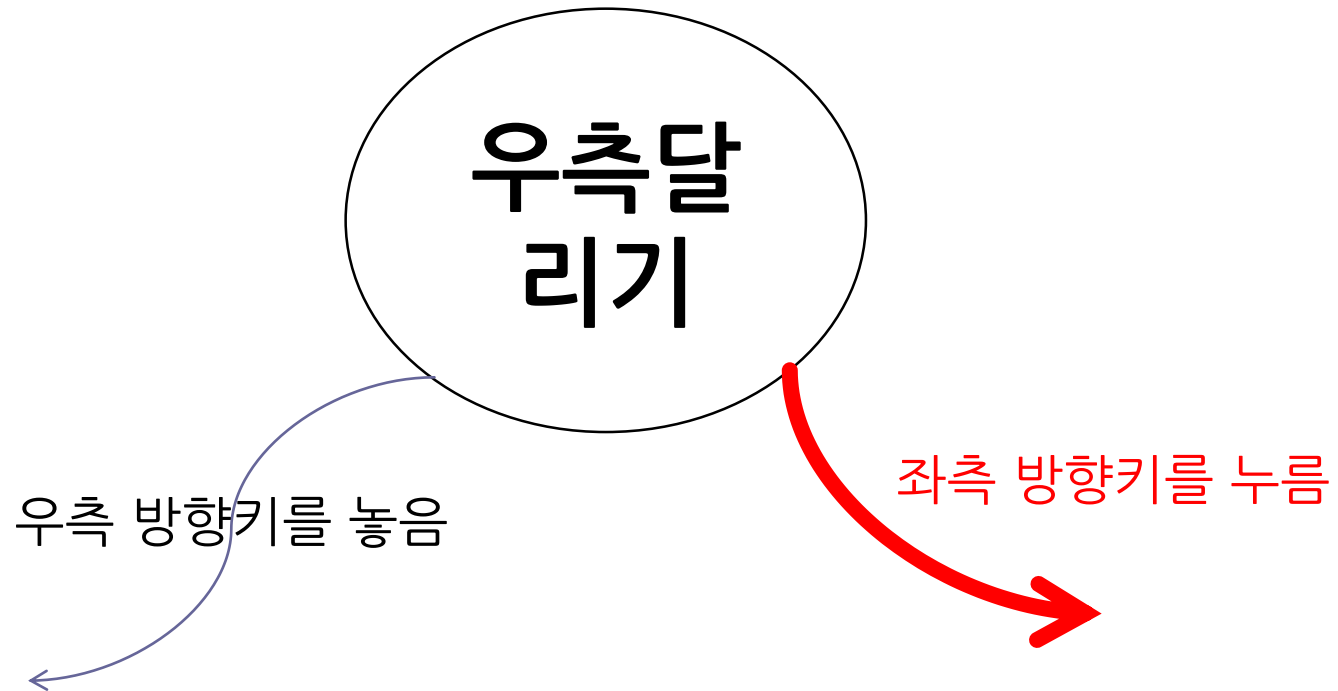
먼저 오른쪽 방향키를 눌러서 캐릭터를 오른쪽으로 계속 움직입니다.  
계속 오른쪽 키를 누른 상태에서 동시에 왼쪽 방향키를 누릅니다.  
어떤가요? 소년은 여전히 오른쪽으로 움직입니다.  
이제 오른쪽 방향키를 땁니다.  
여전히 좌측방향키는 눌러진 상태입니다.  
소년은 정지합니다. 좌측방향키가 눌러있는데도 말입니다.

두개의 방향 키가  
동시에 눌려지는 것을  
다루지 못한다!

# 우측 달리기 상태에서 발생할 수 있는 이벤트?

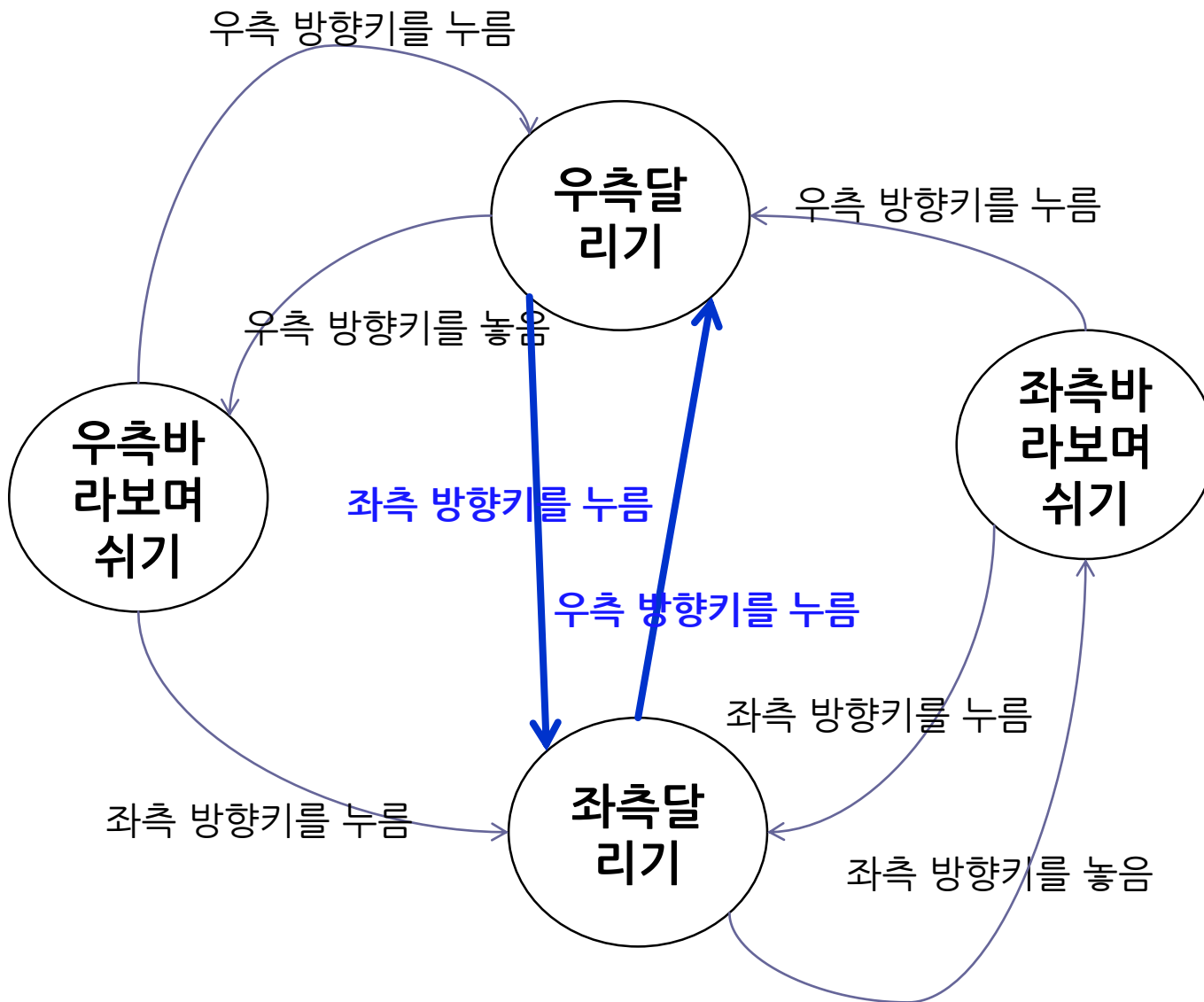


# 우측 달리기 상태에서 발생할 수 있는 이벤트?

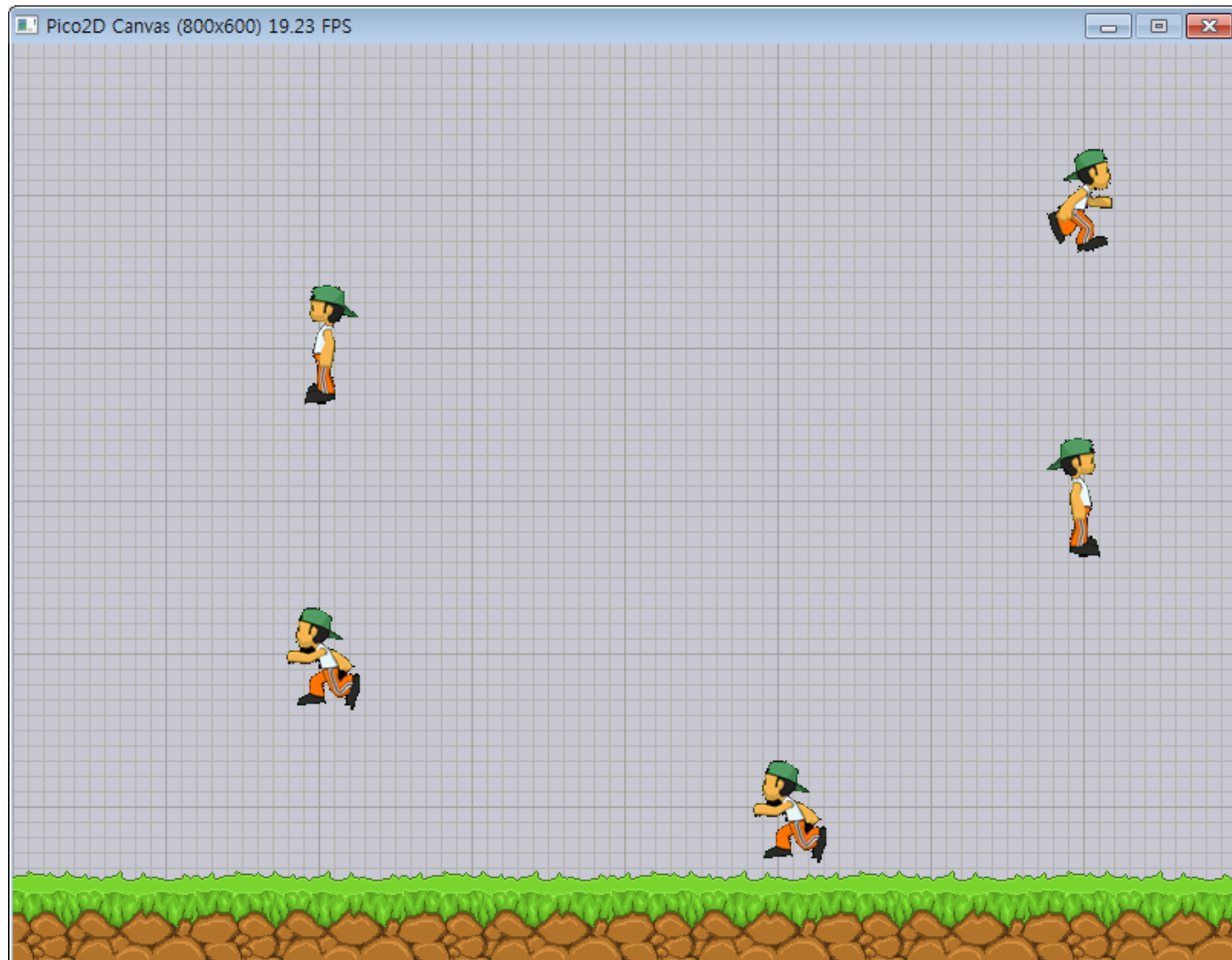




# 수정된 상태 다이어그램



# 객체들의 초기 상태를 어떻게 할까?



## 흔히 하는 “무식한 방법”: 하드코딩(Hard Coding)

```
player = Boy()  
player.x = 400  
player.y = 300  
player.state = Boy.RIGHT_RUN
```

문제점은?  
왜 무식한가?

# 소프트 코딩(Soft Coding)

x : 400  
y : 400  
state : RIGHT\_RUN



```
data_struct = {"total_spam":0,"total_ham":0,"total_spam_words":0,"total_ham_words":0,"bag_of_words":{}}

def learn(message, messagetype):
    bag_of_words = data_struct['bag_of_words']
    words = message.split(" ")
    for word in words:
        try:
            bag_of_words[word][messagetype]+=1
        except KeyError:
            bag_of_words[word] = [1-messagetype,messagetype]
            data_struct["total_spam_words"]+=messagetype
            data_struct["total_ham_words"]+=(1-messagetype)

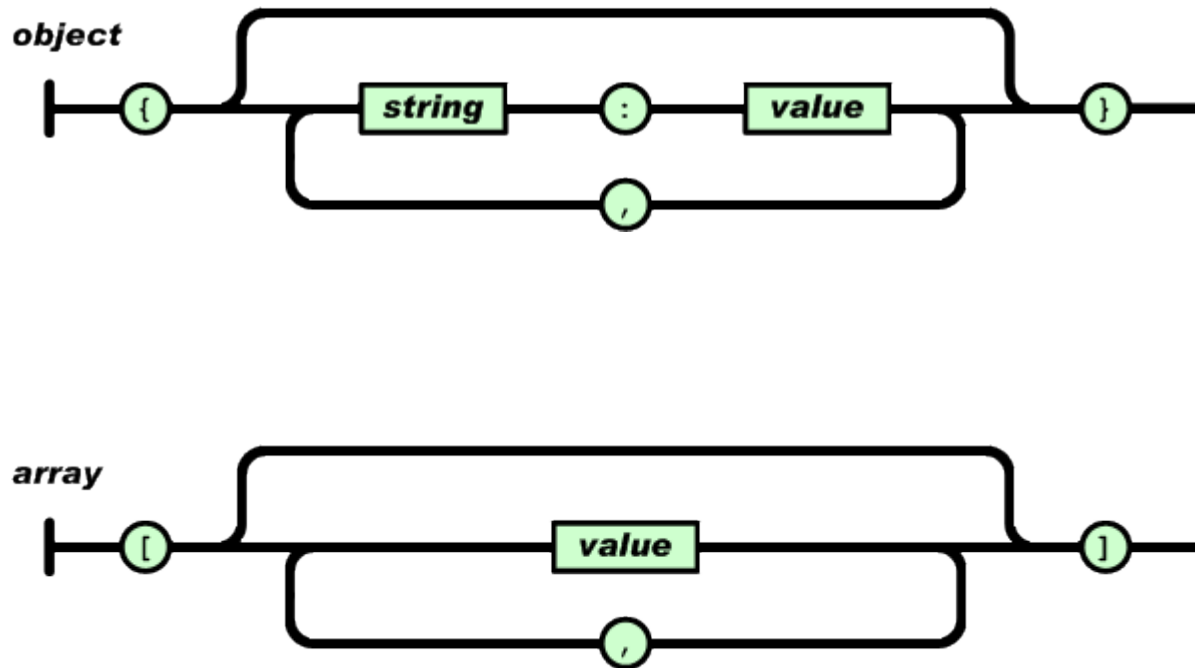
    data_struct["total_spam"]+=messagetype
    data_struct["total_ham"]+=(1-messagetype)
#K is laplacian smoother
def predict(message,K):
    bag_of_words = data_struct['bag_of_words']
    k=K
    #total messages
    total_messages=data_struct["total_spam"]+data_struct["total_ham"]
    #prior probability of spam
    p_s=(data_struct["total_spam"]+k)/(total_messages*1.0+k*2)
    #prior probability of ham
    p_h=(data_struct["total_ham"]+k)/(total_messages*1.0+k*2)
    words=message.split(" ")
    #p_m_s of message given its spam && p_m_h probability of message given its ham
    p_m_s=1
    p_m_h=1
    for word in words:
        p_m_s*=((bag_of_words[word][1]*1.0+k)/(data_struct["total_spam_words"]+k*(len(bag_of_words))))
        p_m_h*=((bag_of_words[word][0]*1.0+k)/(data_struct["total_ham_words"]+k*(len(bag_of_words))))
    #bayes rule && p_s_m probability of spam given a particluar message
    p_s_m = p_s*p_m_s/(p_m_s*p_s+p_m_h*p_h)
    return p_s_m

#1 corresponds to message is spam and 0 that it is ham
learn("offer is secret",1)
learn("click secret link",1)
learn("secret sports link",1)
learn("play sports today",0)
learn("went play sports",0)
learn("secret sports event",0)
learn("sports is today",0)
learn("sports cost money",0)

print predict("today is secret",1)
```

# JSON(Java Script Object Notation)

- 객체를 교환(저장 및 전송 등)하기 위한 텍스트 형식 표준
- 파이썬의 리스트와 딕셔너리와 거의 동일



# JSON 으로 나타낸 객체 사례

The screenshot displays the Notepad++ application window with the file `W:\work\Lecture\2014 02\2D Game Programming\Labs\master\Lab06\team_data.txt` open. The interface is split into two panes. The left pane, titled 'JSON Viewer', shows a hierarchical tree of the JSON data. The right pane shows the raw JSON text from the file `team_data.txt`.

**JSON Viewer Tree Structure:**

- JSON
  - Object
    - Tiffany
      - Object
        - Start State : Left
        - x : 100
        - y : 100
    - Yuna
      - Object
        - Start State : Right
        - x : 200
        - y : 200
    - Sunny
      - Object
        - Start State : Left
        - x : 300
        - y : 300
    - Yuri
      - Object
        - Start State : Right
        - x : 400
        - y : 400
    - Jessica
      - Object
        - Start State : Right
        - x : 400
        - y : 400

**Raw JSON Text:**

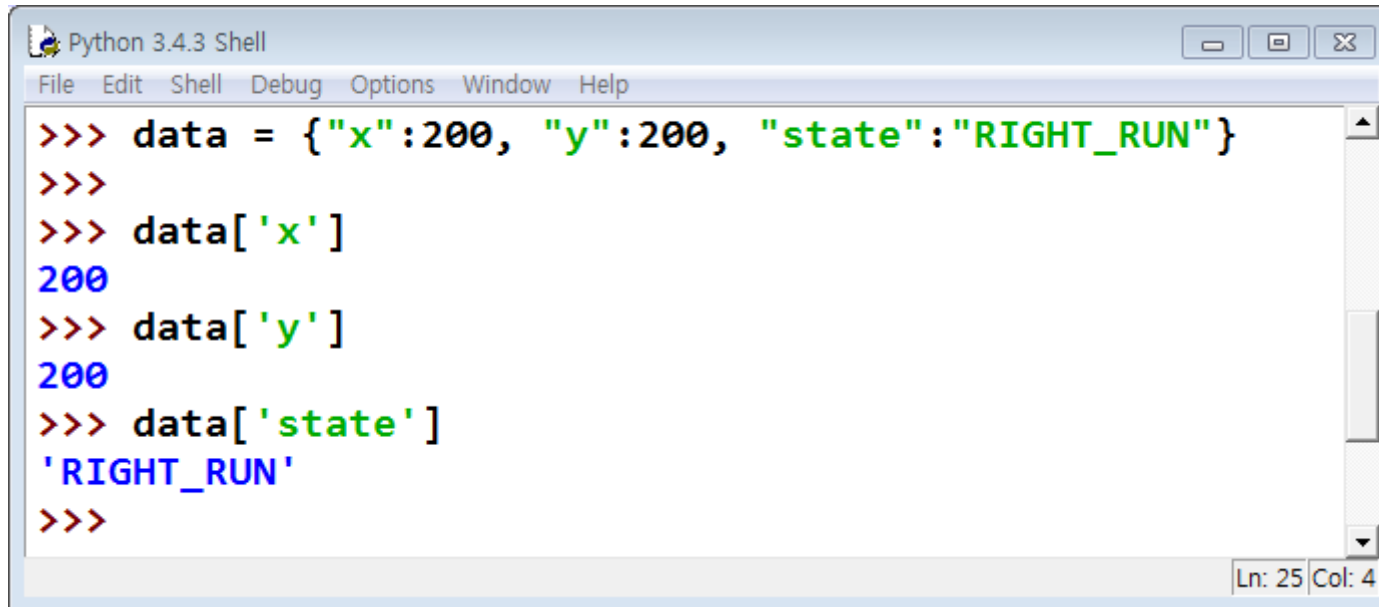
```
1 {  
2     "Tiffany" : {"StartState":"LEFT_STAND", "x":100, "y":100},  
3     "Yuna"    : {"StartState":"RIGHT_RUN", "x":200, "y":200},  
4     "Sunny"   : {"StartState":"LEFT_RUN", "x":300, "y":300},  
5     "Yuri"    : {"StartState":"RIGHT_STAND", "x":400, "y":400},  
6     "Jessica" : {"StartState":"LEFT_STAND", "x":500, "y":500}  
7 }  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23
```

The status bar at the bottom indicates: Normal text | length : 347 | lines : 26 | Ln : 22 | Col : 1 | Sel : 0 | 0 | Dos#Windows | ANSI as UTF-8 | INS



# JSON를 활용한 객체 초기화

# Dictionary 를 이용한

A screenshot of a Python 3.4.3 Shell window. The window has a title bar with the text 'Python 3.4.3 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains a Python interactive session. The user has entered several lines of code: a dictionary definition, and three lookups for the 'x', 'y', and 'state' keys. The interpreter has responded with the corresponding values. The status bar at the bottom right shows 'Ln: 25 | Col: 4'.

```
>>> data = {"x":200, "y":200, "state":"RIGHT_RUN"}
>>>
>>> data['x']
200
>>> data['y']
200
>>> data['state']
'RIGHT_RUN'
>>>
```

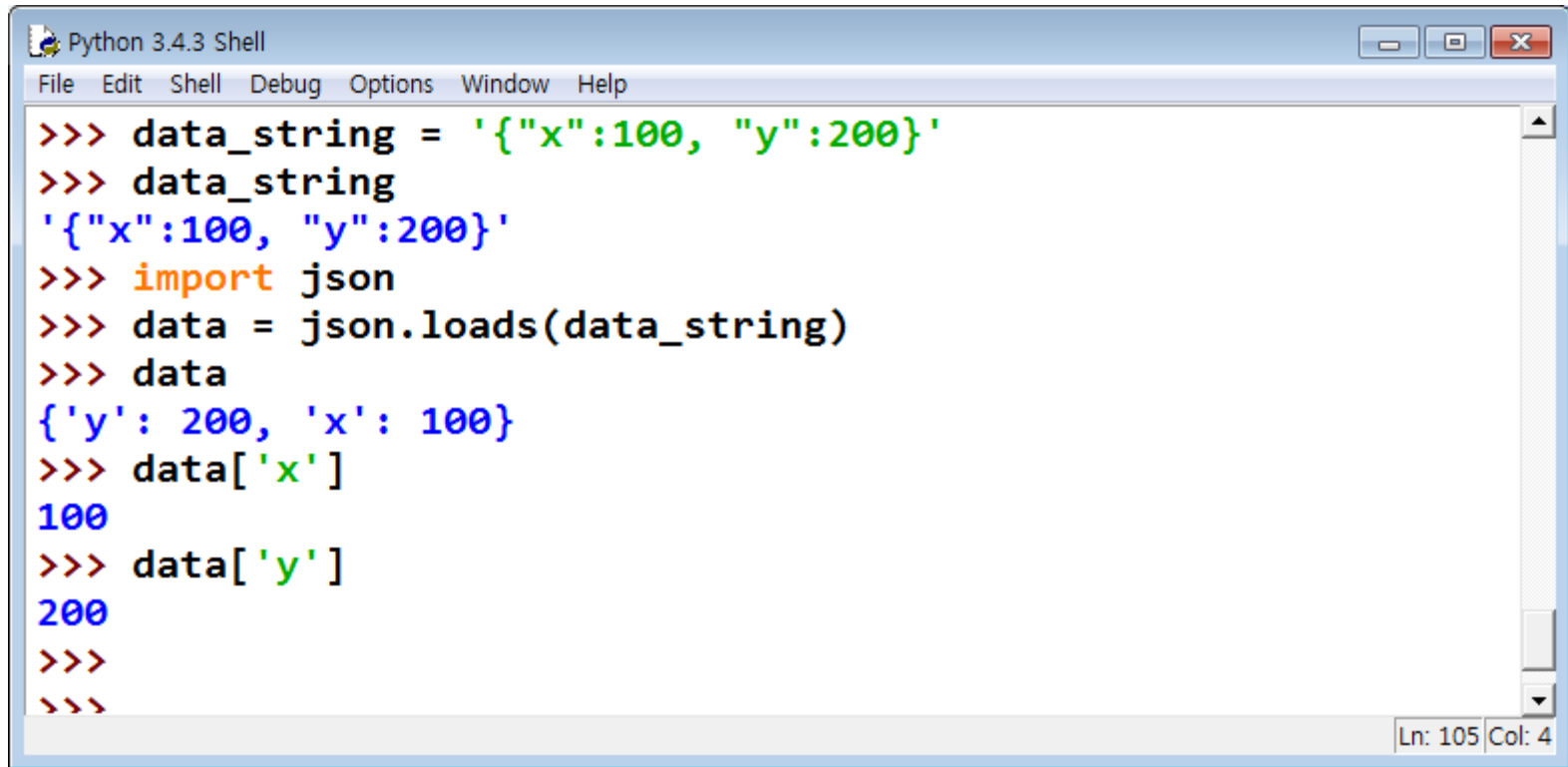


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

>>> data = {"yuna":{"x":100, 'y':100, 'StartState':'RIGHT_RUN'}}
>>>
>>> data['yuna']
{'StartState': 'RIGHT_RUN', 'y': 100, 'x': 100}
>>> data['yuna']['x']
100
>>> data['yuna']['y']
100
>>> data['yuna']['StartState']
'RIGHT_RUN'
>>>
>>>
```

Ln: 37 Col: 4

# JSON을 활용한 data\_string의 변환

A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area contains a series of Python commands and their outputs. The commands are: 1. data\_string = '{"x":100, "y":200}' (green text), 2. data\_string (blue text), 3. import json (orange text), 4. data = json.loads(data\_string) (black text), 5. data (blue text), 6. data['x'] (green text), 7. data['y'] (green text). The outputs are: 1. '{"x":100, "y":200}' (blue text), 2. {'y': 200, 'x': 100} (blue text), 3. 100 (blue text), 4. 200 (blue text). The status bar at the bottom right shows 'Ln: 105 Col: 4'.

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> data_string = '{"x":100, "y":200}'
>>> data_string
'{"x":100, "y":200}'
>>> import json
>>> data = json.loads(data_string)
>>> data
{'y': 200, 'x': 100}
>>> data['x']
100
>>> data['y']
200
>>>
>>>
Ln: 105 Col: 4
```

# json\_player.py - create\_team()



```
def create_team():
    team_data_text = '
        {
            "Tiffany" : {"StartState":"LEFT_RUN", "x":100, "y":100},
            "Yuna"     : {"StartState":"RIGHT_RUN", "x":200, "y":200},
            "Sunny"    : {"StartState":"LEFT_STAND", "x":300, "y":300},
            "Yuri"     : {"StartState":"RIGHT_STAND", "x":400, "y":400},
            "Jessica" : {"StartState":"LEFT_RUN", "x":500, "y":500}
        }
    '

    player_state_table = {
        "LEFT_RUN" : Boy.LEFT_RUN,
        "RIGHT_RUN" : Boy.RIGHT_RUN,
        "LEFT_STAND" : Boy.LEFT_STAND,
        "RIGHT_STAND" : Boy.RIGHT_STAND
    }

    team_data = json.loads(team_data_text)
```

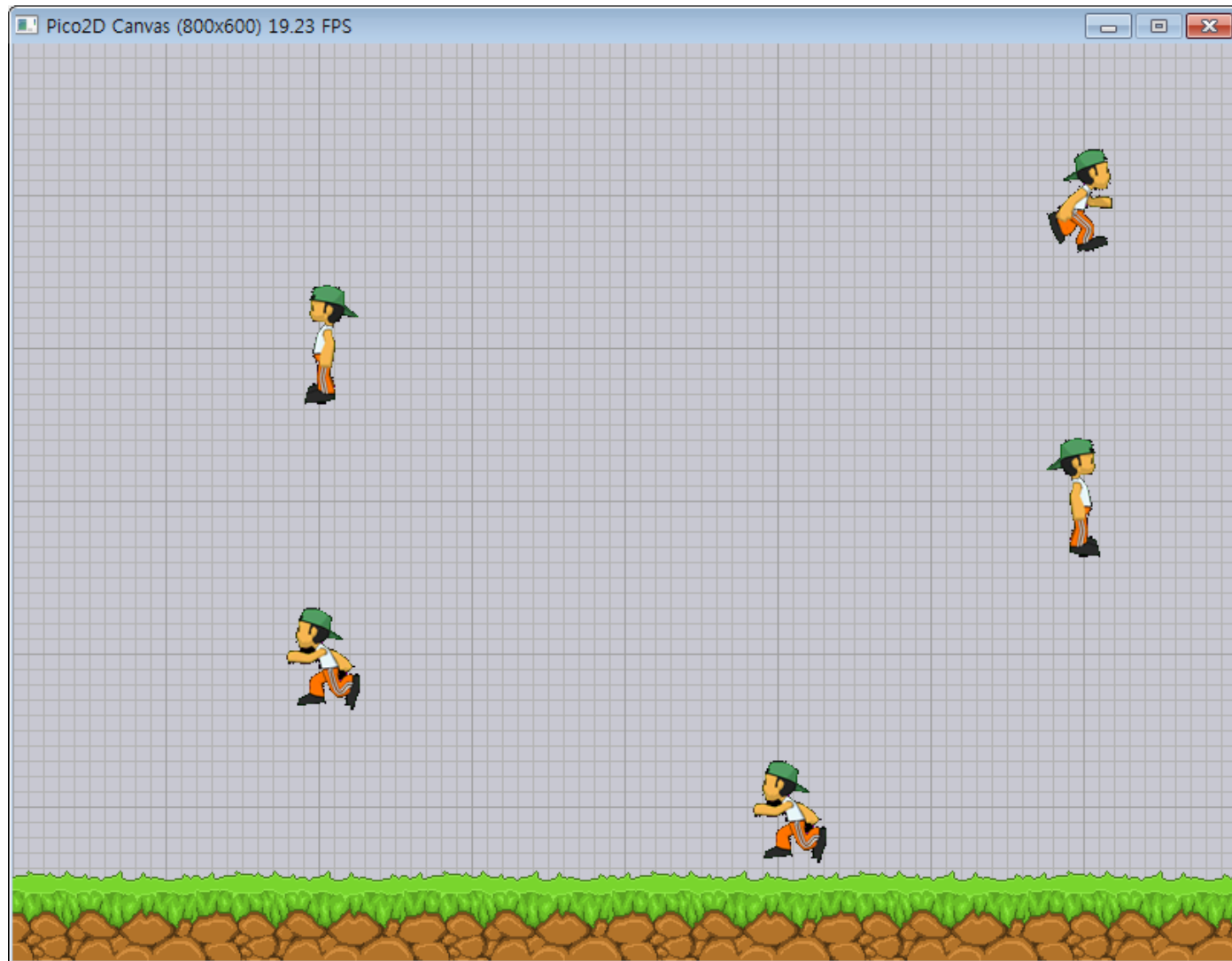
# json\_player.py - create\_team()



```
team = []
for name in team_data:
    player = Boy()
    player.name = name
    player.x = team_data[name]['x']
    player.y = team_data[name]['y']
    player.state = player_state_table[team_data[name]['StartState']]
    team.append(player)

return team
```

# 실행 결과



# 초기 객체들의 상태 설정 데이터

```
team_data_text = '
{
    "Tiffany" : {"StartState":"LEFT_RUN", "x":100, "y":100},
    "Yuna"     : {"StartState":"RIGHT_RUN", "x":200, "y":200},
    "Sunny"    : {"StartState":"LEFT_STAND", "x":300, "y":300},
    "Yuri"     : {"StartState":"RIGHT_STAND", "x":400, "y":400},
    "Jessica"  : {"StartState":"LEFT_RUN", "x":500, "y":500}
},
'
```

텍스트 자료를 파이썬 내부 객체로 변환!

```
team_data =  
json.loads(team_data_text)
```

# 객체로 변환된 데이터

```
[Dbg]>>> team_data
{'Jessica': {'StartState': 'LEFT_STAND', 'x': 500, 'y': 500},
 'Sunny': {'StartState': 'LEFT_RUN', 'x': 300, 'y': 300},
 'Tiffany': {'StartState': 'LEFT_STAND', 'x': 100, 'y': 100},
 'Yuna': {'StartState': 'RIGHT_RUN', 'x': 200, 'y': 200},
 'Yuri': {'StartState': 'RIGHT_STAND', 'x': 400, 'y': 400}}
[Dbg]>>> type(team_data)
<class 'dict'>
[Dbg]>>>
```



# 팀 창단

```
team = []

for name in team_data:

    player = Boy()

    player.name = name

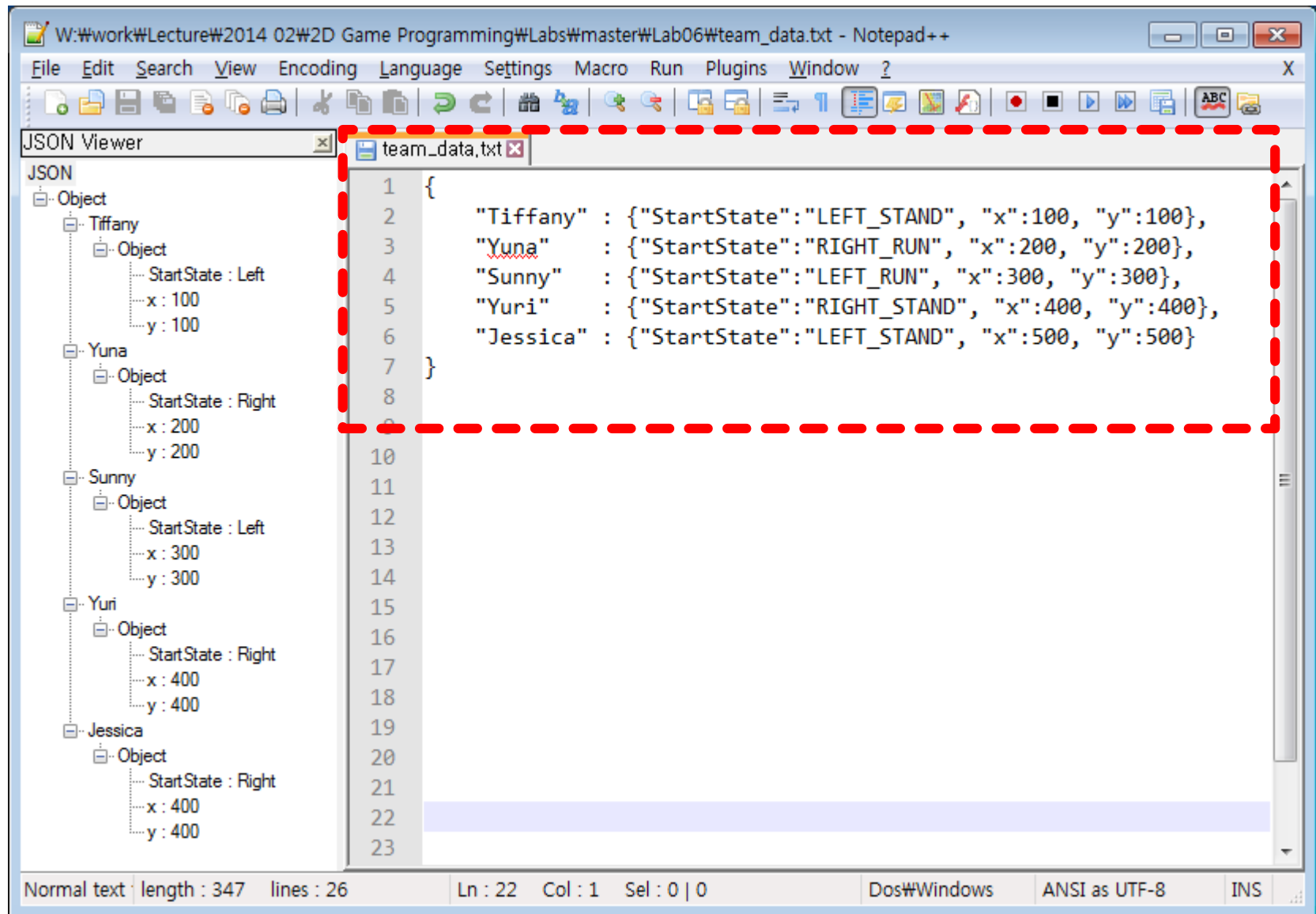
    player.x = team_data[name]['x']
    player.y = team_data[name]['y']

    player.state = player_state_table[team_data[name]['StartState']]

    team.append(player)

return team
```

# 객체 초기 설정 데이터를 텍스트 파일로... team\_data.txt



W:\work\Lecture\2014 02\2D Game Programming\Labs\master\Lab06\team\_data.txt - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

JSON Viewer

JSON

- Object
  - Tiffany
    - Object
      - Start State : Left
      - x : 100
      - y : 100
  - Yuna
    - Object
      - Start State : Right
      - x : 200
      - y : 200
  - Sunny
    - Object
      - Start State : Left
      - x : 300
      - y : 300
  - Yuri
    - Object
      - Start State : Right
      - x : 400
      - y : 400
  - Jessica
    - Object
      - Start State : Right
      - x : 400
      - y : 400

```
1 {
2     "Tiffany" : {"StartState":"LEFT_STAND", "x":100, "y":100},
3     "Yuna"    : {"StartState":"RIGHT_RUN", "x":200, "y":200},
4     "Sunny"   : {"StartState":"LEFT_RUN", "x":300, "y":300},
5     "Yuri"    : {"StartState":"RIGHT_STAND", "x":400, "y":400},
6     "Jessica" : {"StartState":"LEFT_STAND", "x":500, "y":500}
7 }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

Normal text | length : 347 | lines : 26 | Ln : 22 | Col : 1 | Sel : 0 | 0 | Dos\Windows | ANSI as UTF-8 | INS



```
#team_data = json.loads(team_data_text)

team_data_file = open('team_data.txt', 'r')
team_data = json.load(team_data_file)
team_data_file.close()
```