

Lecture #20. 정리

2D 게임 프로그래밍

이대현 교수



한국공학대학교
TECH UNIVERSITY OF KOREA

파이썬의 특징?

2D 게임?

- 게임이란?

- “가상 월드에 존재하는 여러 객체들의 상호작용”을 시뮬레이션하고 그 결과를 보여주는(렌더링) 것.

- 2D 게임?

- 현재 진행 중인 게임 가상 월드의 내용을 화면에 2D 그림으로 보여주는 것

Complex Data Type

▪ List – list

- 순서가 있는, 중복을 허용하는 데이터들의 집합.
- 원하는 데이터를 찾기 위해, 순서 index 를 이용.

[val1, val2, ...]

▪ Dictionary – dict

- 검색을 위한 키를 갖는 데이터들의 집합
- key – value 쌍 들의 집합

{ key1: val1, key2: val2, ... }

▪ Tuple – tuple

- 순서가 있는, 중복을 허용하는 데이터들의 집합
- 다만, 데이터값을 변경하는 것은 불가

(val1, val2, ...)

▪ Set – set

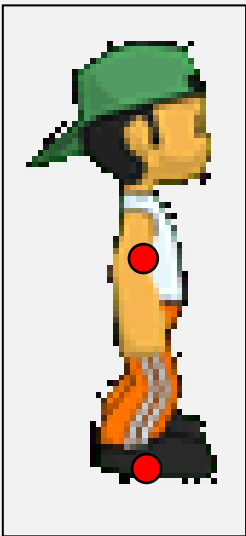
- 중복을 허용하지 않는, 순서에 상관없는 데이터들의 집합

{ val1, val2, ... }

2D 게임 개발 접근법

- 플랫폼 종속적 방법
 - Direct X
 - OpenGL
 - Simple Frame Buffer
- 플랫폼 독립적 방법, Cross Platform
 - Unity
 - Unreal
 - COCOS2D
 - SDL
 - 그 외의 범용 2D 렌더링 라이브러리

피봇(Pivot)



여기가 피봇입니다.

이 점을 피봇으로 삼기도 합니다

게임 루프

```
x = 0
```

```
while (x < 800):
```

```
    clear_canvas_now()  
    grass.draw_now(400, 30)  
    character.draw_now(x, 90)
```

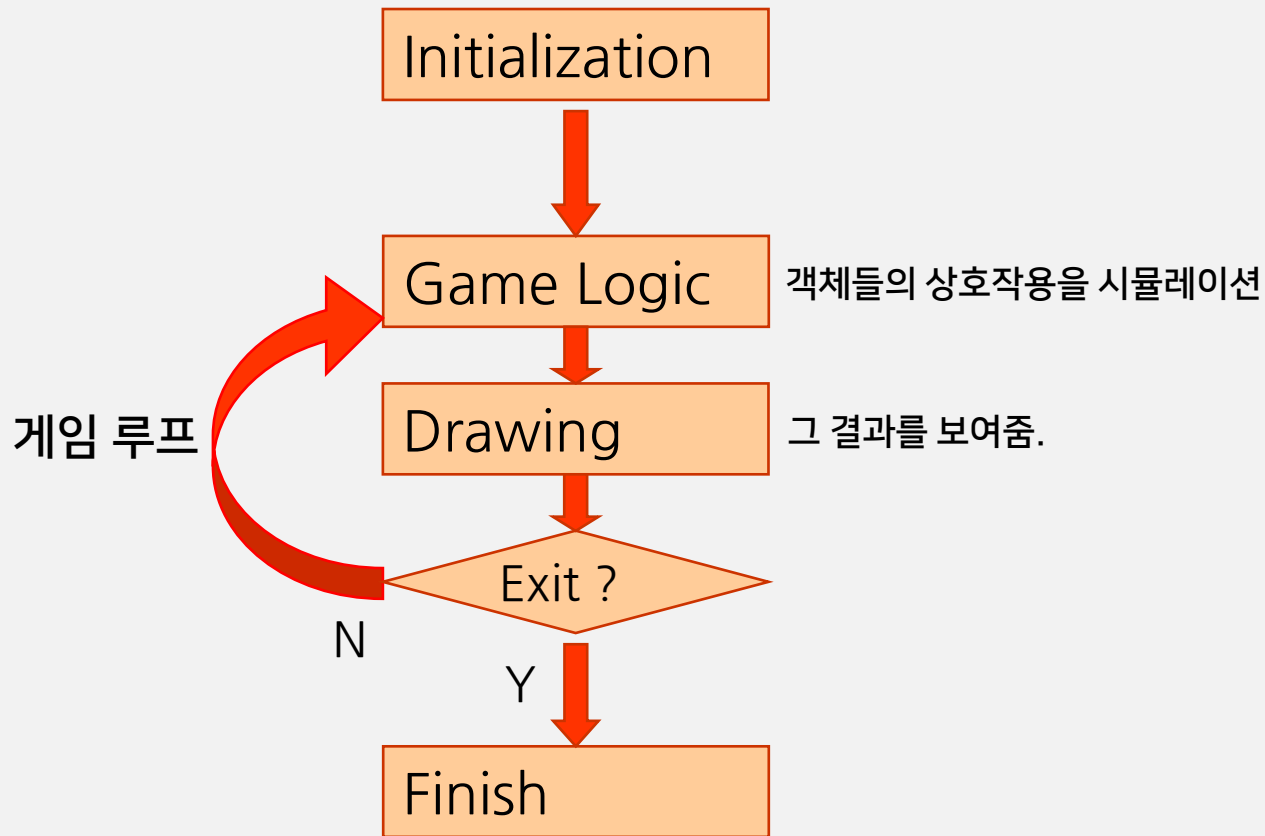
Game Rendering

```
    x = x + 2
```

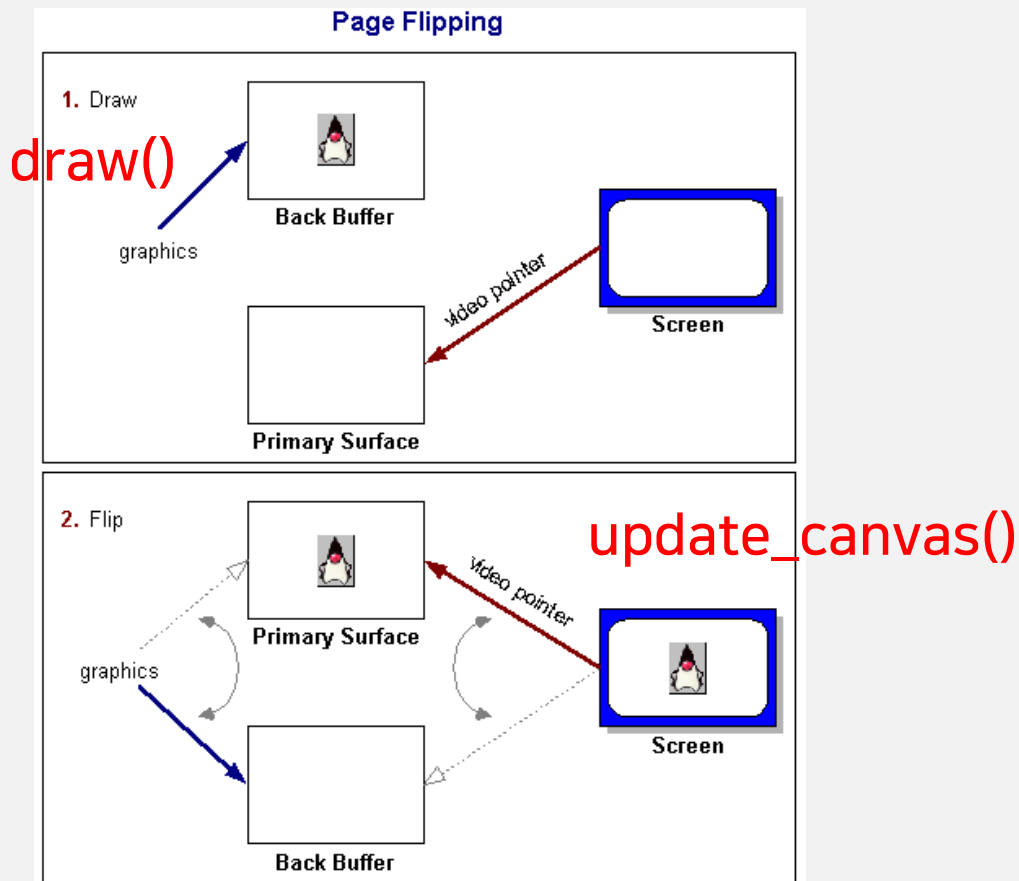
Game Logic

```
    delay(0.01)
```

게임 기본 구조



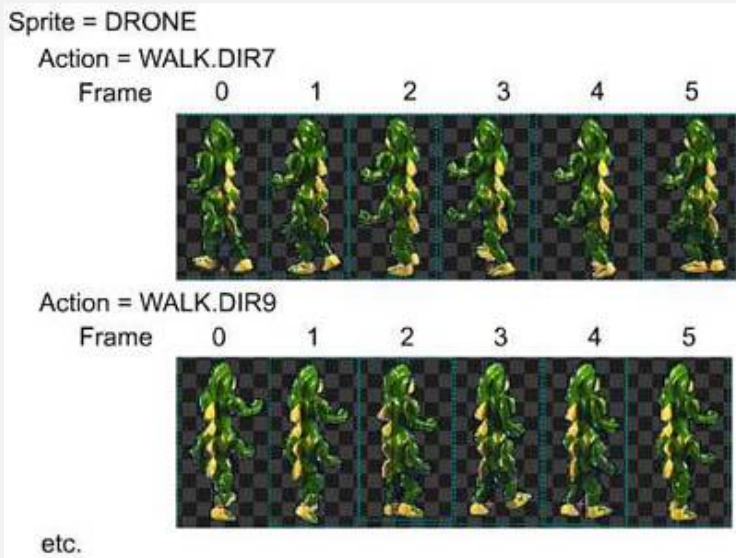
페이지 플리핑(Page Flipping)



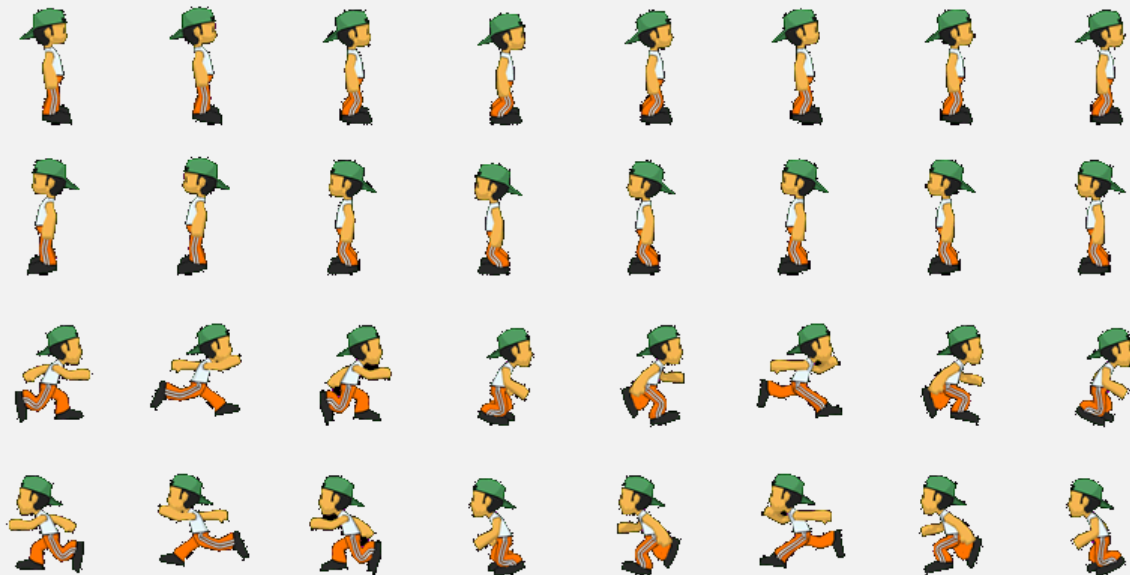
애니메이션(Animation)

■ 애니메이션이란?

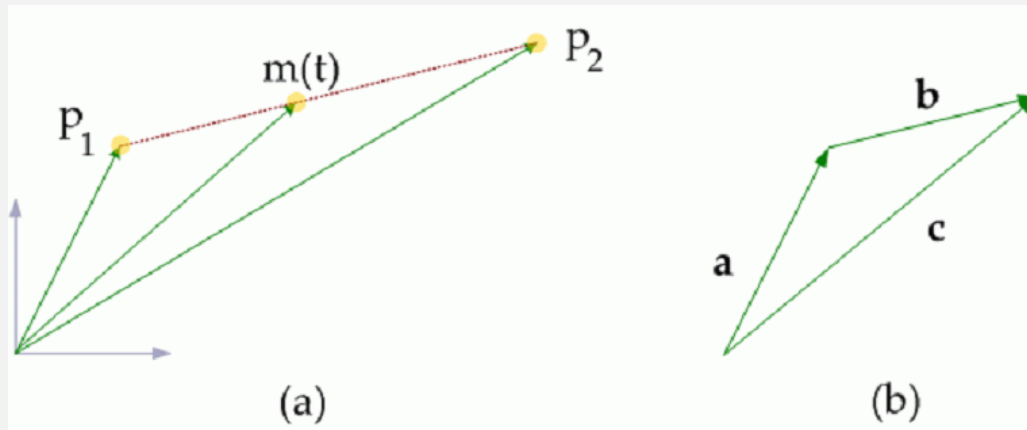
- 여러 개의 이미지를 일정한 시간 간격을 통해서 화면에 뿌림으로써, 물체가 움직이는 효과를 주는 것.
- 스프라이트는 여러 개의 action으로 구성됨.
 - Action: 달리기, 걷기, 제자리 동작 등과 같이 캐릭터의 움직임을 나타냄.
 - Action은 여러 개의 Frame으로 구성됨.
 - Frame은 한 개의 이미지



스프라이트 시트



Parametric Representation of Lines

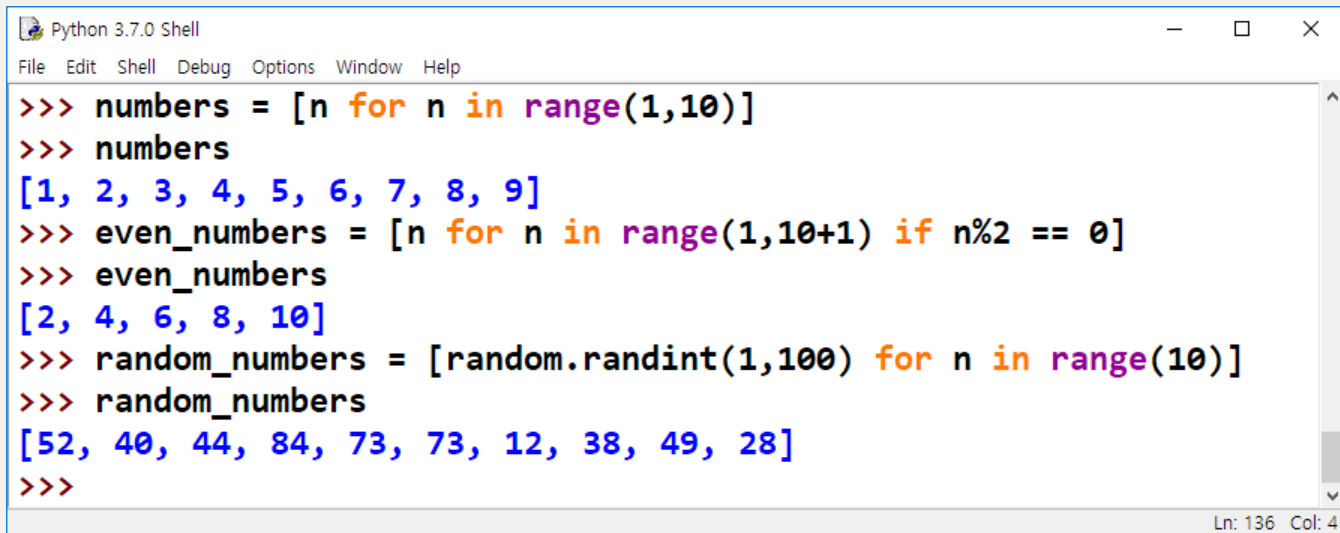


$$m(t) = p_1 + t (p_2 - p_1) = (1 - t) p_1 + t p_2 \quad (0 \leq t \leq 1)$$

$$\begin{aligned} m(t) &= p_1, \text{ at } t = 0 \\ &= p_2, \text{ at } t = 1 \end{aligned}$$

Python List Comprehension

- 리스트를 빠르게 만들기 위한 독특한 문법 구조
- 리스트 안에 있는 데이터들을 일정한 규칙을 가지고 생성해냄.

A screenshot of a Python 3.7.0 Shell window. The window has a title bar with the text 'Python 3.7.0 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 REPL session. The prompt '>>>' is followed by the code 'numbers = [n for n in range(1,10)]'. The next prompt '>>>' is followed by 'numbers', which returns the list '[1, 2, 3, 4, 5, 6, 7, 8, 9]'. The next prompt '>>>' is followed by 'even_numbers = [n for n in range(1,10+1) if n%2 == 0]'. The next prompt '>>>' is followed by 'even_numbers', which returns the list '[2, 4, 6, 8, 10]'. The next prompt '>>>' is followed by 'random_numbers = [random.randint(1,100) for n in range(10)]'. The next prompt '>>>' is followed by 'random_numbers', which returns the list '[52, 40, 44, 84, 73, 73, 12, 38, 49, 28]'. The final prompt '>>>' is followed by an empty line. At the bottom right of the window, the status bar shows 'Ln: 136 Col: 4'.

```
>>> numbers = [n for n in range(1,10)]
>>> numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> even_numbers = [n for n in range(1,10+1) if n%2 == 0]
>>> even_numbers
[2, 4, 6, 8, 10]
>>> random_numbers = [random.randint(1,100) for n in range(10)]
>>> random_numbers
[52, 40, 44, 84, 73, 73, 12, 38, 49, 28]
>>>
```

<https://docs.python.org/3.3/tutorial/datastructures.html#list-comprehensions>

추상화(Abstraction)





속성 + 행위 = 소년 객체

클래스(Class)

■ 클래스란?

- 유사한 여러 객체들에게 공통적으로 필요로 하는 데이터와 이 데이터 위에서 수행되는 함수들을 정의하는 소프트웨어 단위.
- 객체를 찍어내는 “도장”

클래스에 비유할 수 있는 것들		
	붕어빵 틀	제품 설계도
객체에 비유할 수 있는 것들		
	붕어빵	제품

클래스 변수

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

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

캐릭터 컨트롤러(Character Controller)

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

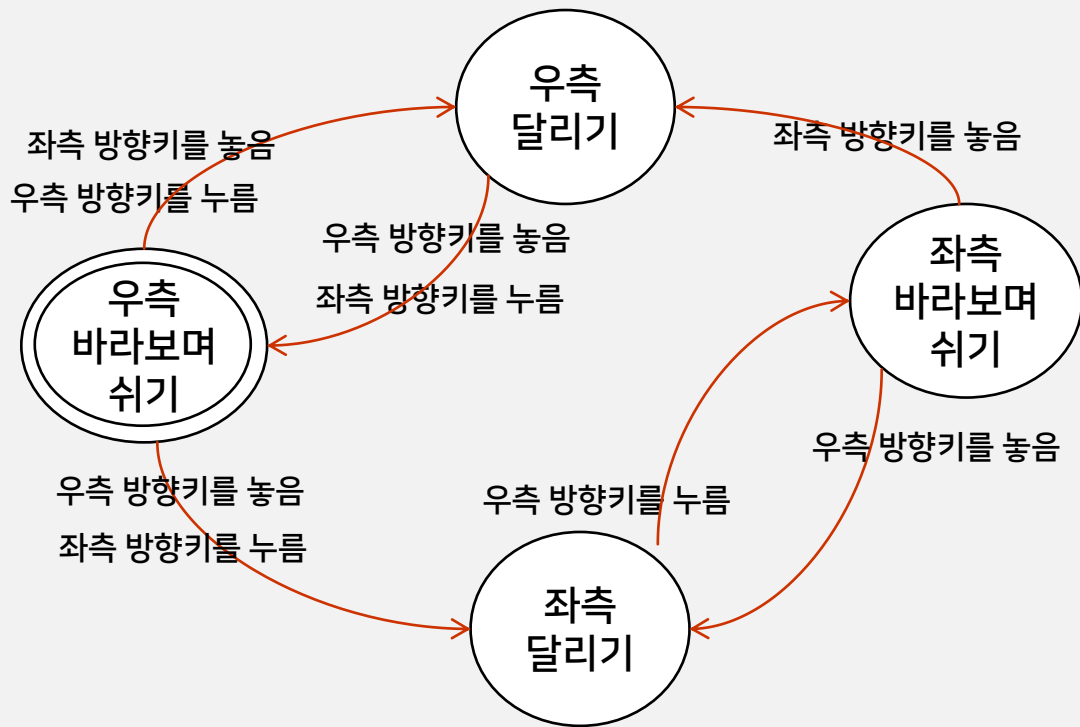


우리의 “주인공”은?

■ 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 #1



상태 구현

```
class Idle:
    @staticmethod
    def enter():
        print('Idle Enter')

    @staticmethod
    def exit():
        print('Idle Exit')

    @staticmethod
    def do():
        print('Idle Do')

    @staticmethod
    def draw():
        pass
```

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

이벤트 구현

- 튜플을 이용해서 상태 이벤트를 나타내도록 함.
 - (상태 이벤트 종류, 실제 이벤트값)
 - ('INPUT', 실제입력이벤트값)
 - ('TIME_OUT', 0)
 - ('NONE', 0)

```
def space_down(e):  
    return e[0] == 'INPUT' and e[1].type == SDL_KEYDOWN and e[1].key == SDLK_SPACE  
  
def time_out(e):  
    return e[0] == 'TIME_OUT'
```

상태 머신 구현

```
self.transitions = {  
    Idle: {right_down: Run, left_down: Run, left_up: Run, right_up: Run, time_out: Sleep},  
    Run: {right_down: Idle, left_down: Idle, right_up: Idle, left_up: Idle},  
    Sleep: {right_down: Run, left_down: Run, right_up: Run, left_up: Run, space_down: Idle}  
}
```

게임 월드 구현

```
# layer 0: Background Objects  
# layer 1: Foreground Objects  
objects = [[],[[]]]
```

```
def add_object(o, depth = 0):  
    objects[depth].append(o)
```

```
def add_objects(ol, depth = 0):  
    objects[depth] += ol
```

```
def remove_object(o):  
    for layer in objects:  
        if o in layer:  
            layer.remove(o)  
            return  
    raise ValueError('Cannot delete non existing object')
```

게임 월드에 담겨있는 모든 객체들을 담고 있는 리스트. Drawing Layer 에 따라서 분류. 필요에 따라 Layer를 추가하면 됨. 현재는 두개의 Layer만.

게임 월드에 객체 추가

게임 월드에 객체'들'을 추가

게임 월드에서 객체 제거

게임 모드

■ 게임 모드란?

- 게임 프로그램 실행 중에 지속적으로 머물러 있는 특정 상황, 씬,
- 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임 모드에 따라 달라짐.
- 작은 게임 루프로 볼 수 있음.



게임 프레임워크

- 게임 모드들을 효과적으로 연결하는 소프트웨어 구조.
- 일종의 Task Switching System
- 디자인 패턴 중, State Pattern 혹은 Strategy Pattern에 해당됨.

게임 모드 뼈대

```
def init(): pass

def finish(): pass

def update(): pass

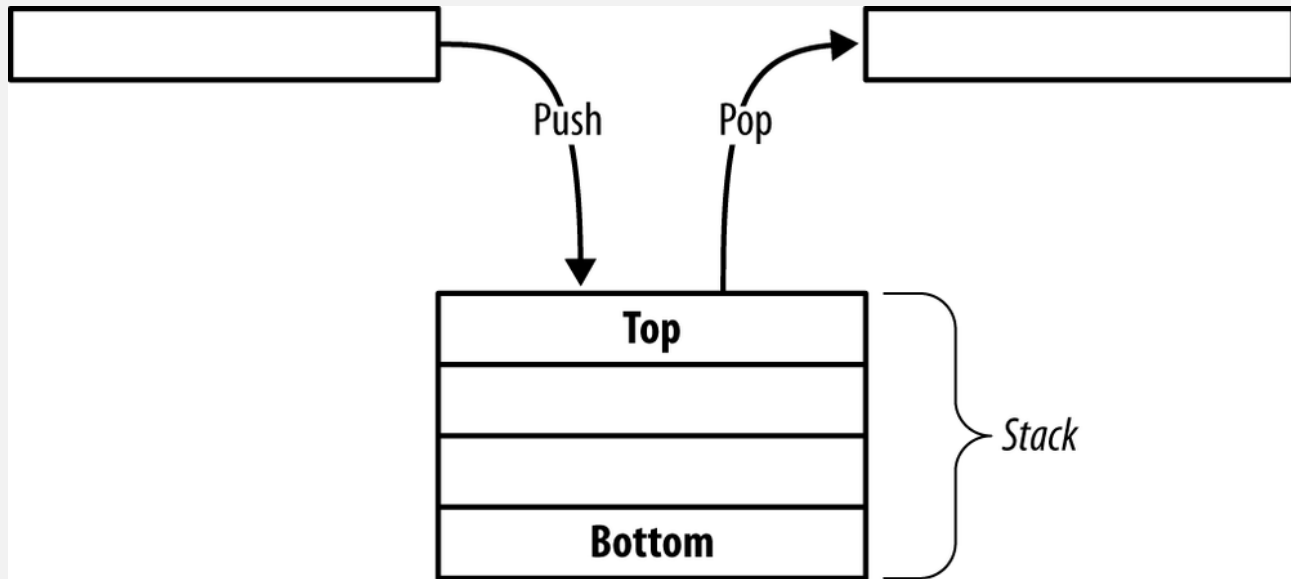
def draw(): pass

def handle_events(): pass

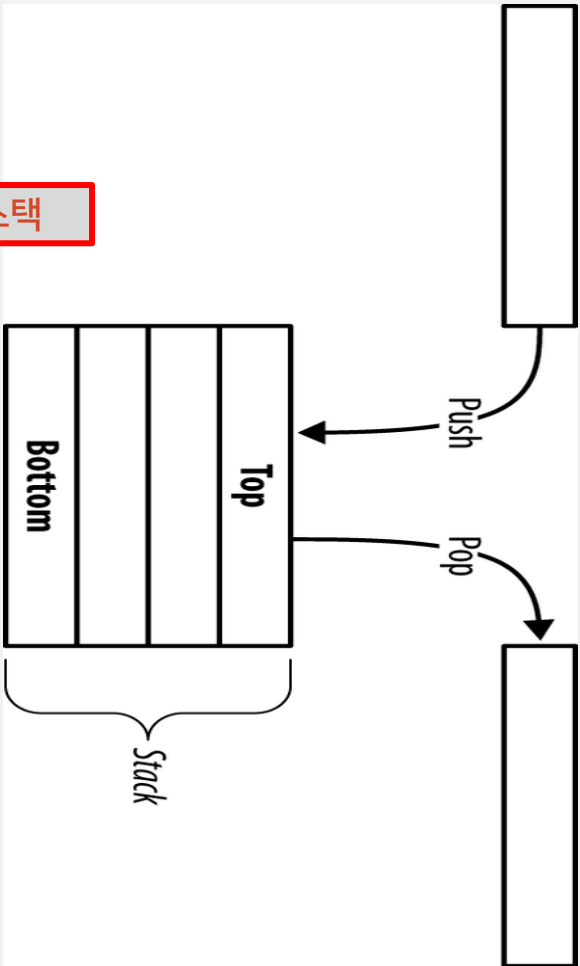
def pause(): pass

def resume(): pass
```

Stack 자료 구조



list 를 이용한 스택



game_framework.py 분석(1)

```
def run(start_mode):  
    global running, stack  
    running = True  
    stack = [start_mode]  
    start_mode.init()
```

start_mode 를 담고 있는 스택을 생성

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

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

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

스택에 남아있는 모든 게임 모드들을 차례로 제거

game_framework.py 분석 (2)

```
def change_mode(mode):  
    global stack  
    if (len(stack) > 0):  
        # execute the current mode's finish function  
        stack[-1].finish()  
        # remove the current mode  
        stack.pop()  
    stack.append(mode)  
    mode.init()
```

현재 모드를 삭제한 후,
새로운 모드를 추가하고, init를 호출한다.

```
def push_mode(mode):  
    global stack  
    if (len(stack) > 0):  
        stack[-1].pause()  
    stack.append(mode)  
    mode.init()
```

현재 모드의 pause를 호출하고, 새로운 모드를 스택에
추가한 후, init 로 초기화함.

game_framework.py 분석 (3)

```
def pop_mode():
    global stack
    if (len(stack) > 0):
        # execute the current mode's finish function
        stack[-1].finish()
        # remove the current mode
        stack.pop()

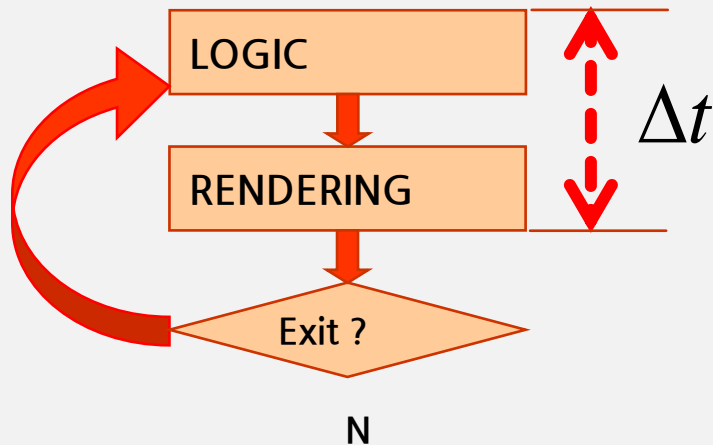
    # execute resume function of the previous mode
    if (len(stack) > 0):
        stack[-1].resume()

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

현재 모드를 finish 한 후, 현재 모드를 제거함.
이제 Stack Top에는 이전 모드가 있으므로, 이전
모드에 대해서 resume 을 호출함.

프레임 시간(Frame Time)

- 한장의 프레임을 만들어내는데 걸리는 시간.
- time delta 또는 delta time 이라고 함.



프레임 속도(Frame Rate)

■ 프레임 속도란?

- 얼마나 빨리 프레임(일반적으로 하나의 완성된 화면)을 만들어 낼 수 있는지를 나타내는 척도
- 일반적으로 초당 프레임 출력 횟수를 많이 사용한다.
- FPS(Frame Per Sec)
- 컴퓨터 게임에서는 일반적으로 최소 25~30 fps 이상이 기준이며, 최근엔 60/120fps

■ 프레임 시간과 프레임 속도의 관계

$$\text{Frame per sec} = 1 / \text{Frame time}$$

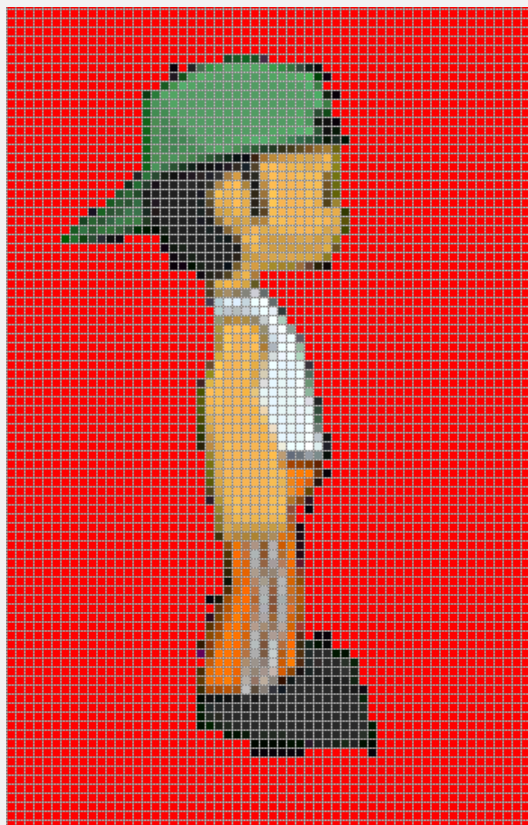
아주 아주 아주 근사한 방법

- 게임 객체들의 운동에 “시간”의 개념을 도입

$$\text{거리} = \text{경과시간} * \text{속도}$$

$$\text{위치} = \text{초기 위치} + \text{거리}$$

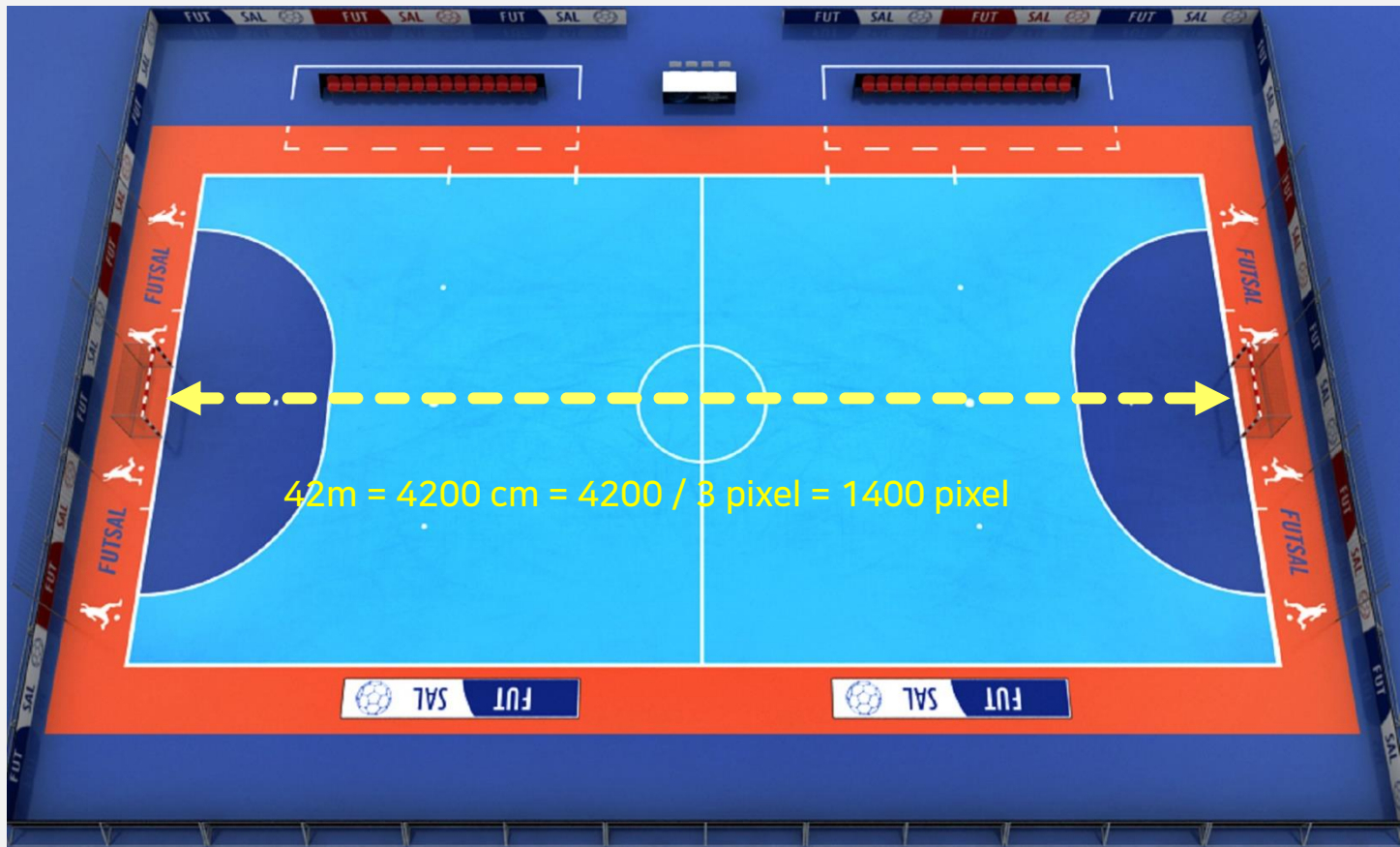
2D 공간의 물리값들을 먼저 결정할 필요



74 pixel = 약 2.2미터

10 pixel = 30 cm

게임 맵은 반드시 실제 물리값으로 크기가 표시되어야 함.



충돌 검사와 충돌 처리

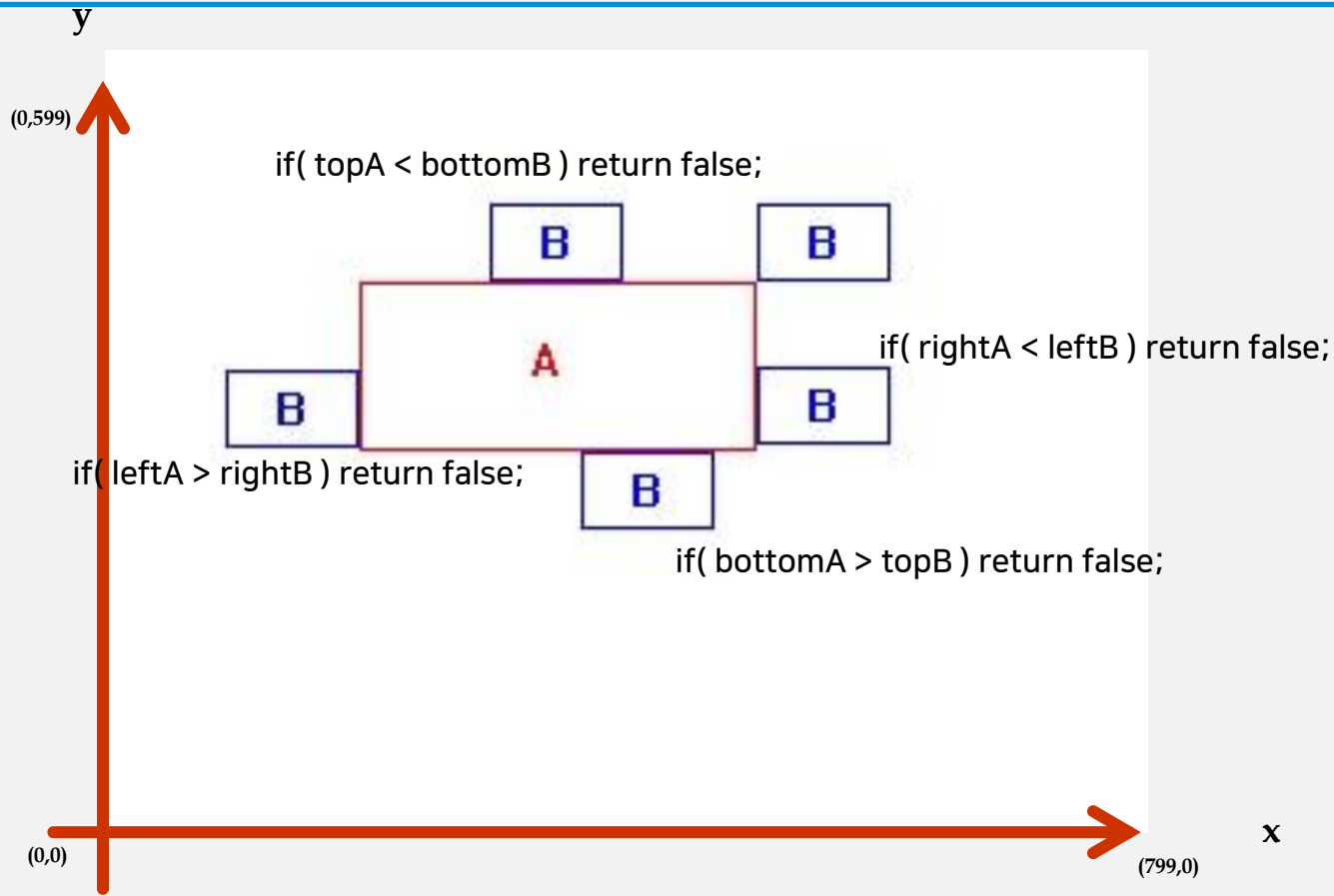
■ 충돌 검사

- 게임 상의 오브젝트 간에 충돌이 발생했는지를 검사하는 것.
- 모든 게임에서 가장 기본적인 물리 계산.
- 기본적으로 시간이 많이 소요되기 때문에, 게임의 오브젝트의 특성에 따라 각종 방법을 통해 최적화해줘야 함.

■ 충돌 처리

- 충돌이 확인 된 후, 이후 어떻게 할것인가?
 - 캐릭터와 아이템의 충돌에 대한 처리는??
 - 바닥에 떨어지는 적군 NPC가 바닥과 충돌하면??
 - 사선으로 움직이는 캐릭터가 맵의 벽과 충돌하면?

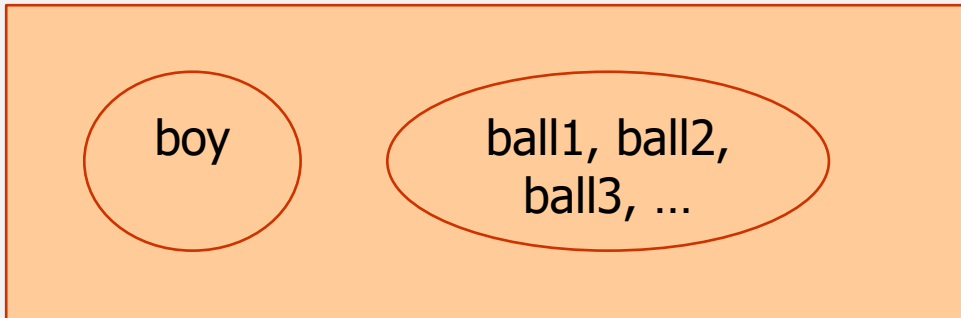
사각형과 사각형



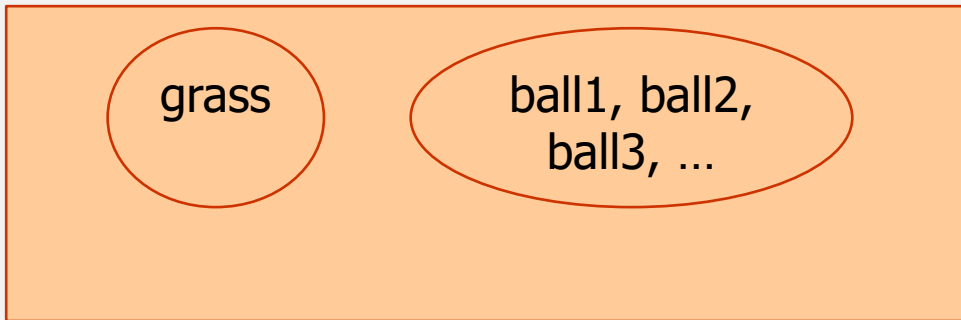
충돌 처리를 위한 그룹 등록

- 충돌 처리가 필요로 하는 두개의 객체 A, B 를 먼저 일일이 등록하고, 나중에 등록된 페어 전체에 대해서 충돌 검사를 수행한 후, 충돌이 있으면 A, B 에게 충돌 처리를 할 수 있도록 지시.
- 충돌 처리해야할 대상들을 그룹화해서 처리

boy:ball



grass:ball



충돌 페어 등록

```
collision_pairs = {}
```

```
def add_collision_pair(group, a, b):  
    if group not in collision_pairs:  
        print(f'Added new group {group}')        collision_pairs[group] = [ [], [] ]  
    if a:  
        collision_pairs[group][0].append(a)  
    if b:  
        collision_pairs[group][1].append(b)
```

충돌 감지에 따른 충돌 처리

```
def handle_collisions():  
    for group, pairs in collision_pairs.items():  
        for a in pairs[0]:  
            for b in pairs[1]:  
                if collide(a, b):  
                    a.handle_collision(group, b)  
                    b.handle_collision(group, a)
```

충돌 검사 및 처리 기본 절차

- 충돌 처리가 필요한 객체에 대해서 충돌 영역 정의
- 디버그를 위해서 충돌 영역을 시각화할수 있도록 설정
- 공간에 이미 존재하는 여러 객체 들 중 충돌 처리가 필요한 두개의 객체 A, B 를 골라서 등록
- 만약 게임 실행 중에 생성된 객체에 대한 충돌 처리가 필요하다면, 그때마다 실시간으로 A, B 를 등록. 만약 한쪽이 이미 등록된 상태라면, None 으로 처리.
- 등록된 모든 충돌 페어에 대해서 매 프레임마다 충돌 검사를 실시.
- 충돌이 발생한 경우, A,B 각각에 대해서 충돌 처리하도록 지시

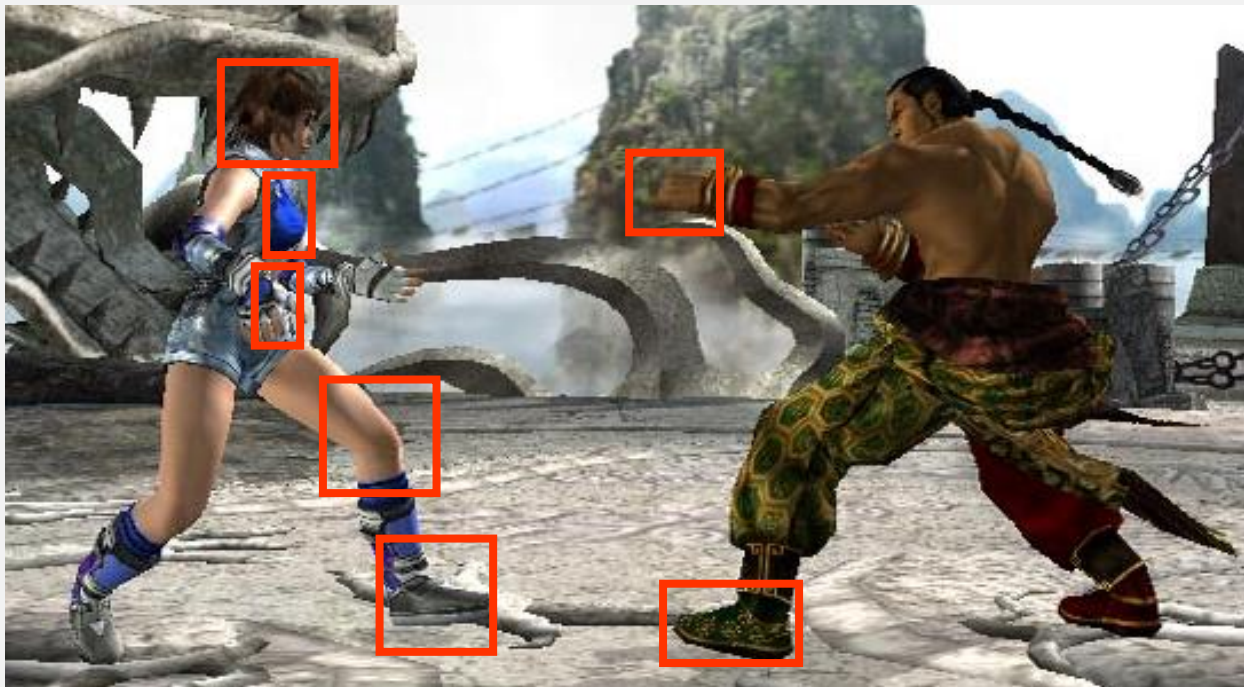
충돌 검사의 실제 적용 방법 (1)

- 정확도를 높이면 한편, 속도 측면에서도 효율적으로 하기 위해, 오브젝트를 적절한 개수의 바운딩 박스로 나눈다.
- 잘게 나누면 나눌수록, 정확도는 높아진다.



충돌 검사의 실제 적용 방법 (2)

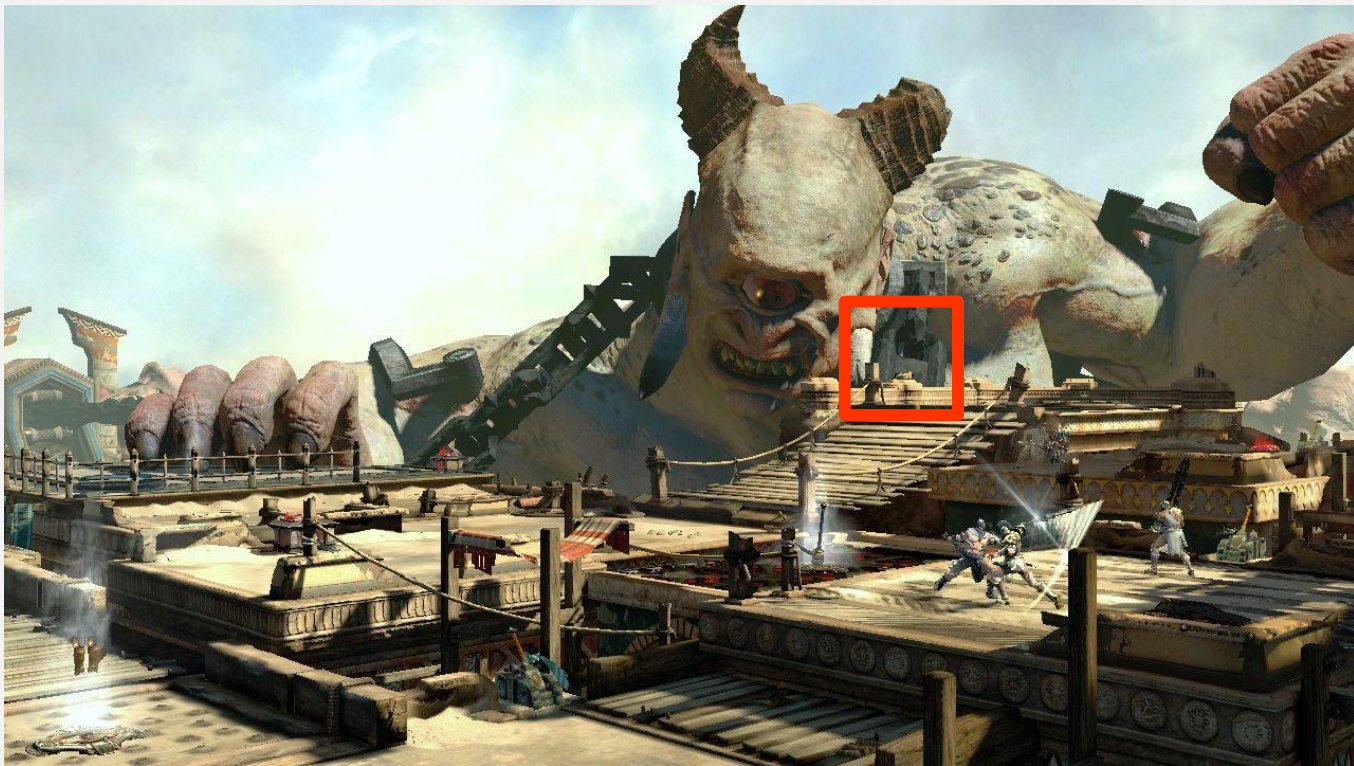
- 게임의 특성에 따라 필요한 부분만 바운딩 박스를 적용한다.
- 격투 대전 게임에서 가격에 사용되는 손 또는 발 부분, 가격이 가해지는 머리, 복부, 배 부분만을 바운딩 박스로 적용.



충돌 처리의 활용

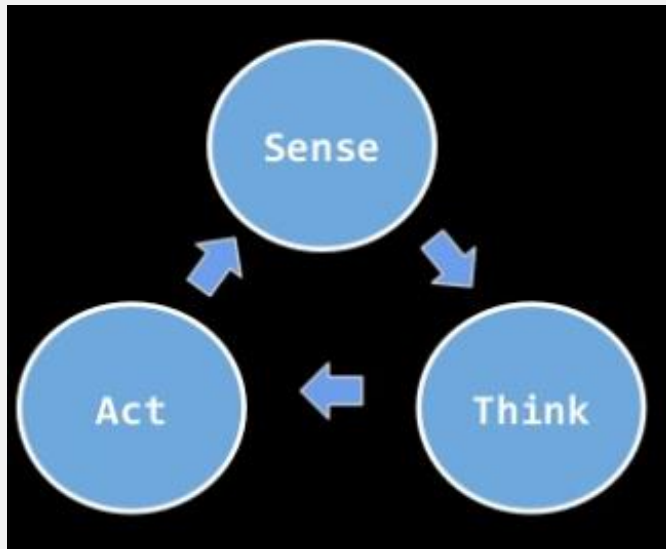
- 트리거(Trigger)

- 특정 위치에 캐릭터가 들어갈 경우, 이벤트를 발생



게임 인공지능

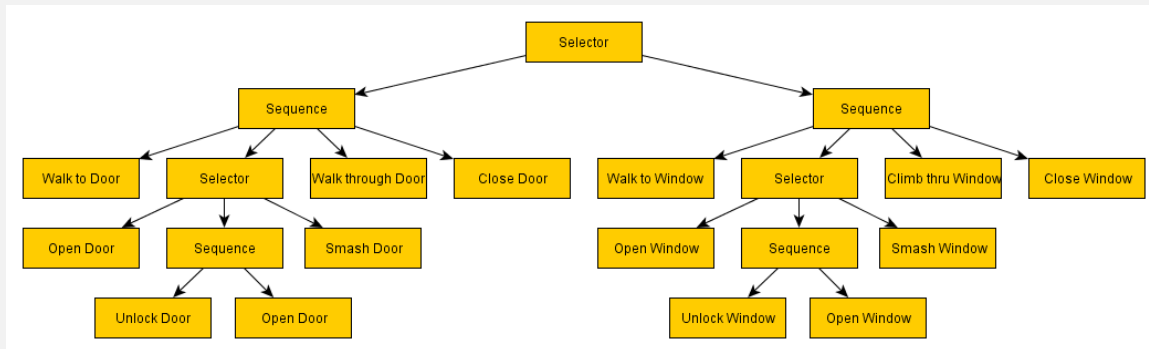
- 게임 객체는 주변의 상황을 인식(Sense)
- 인식된 결과를 바탕으로 행동을 결정(Think)
- 실제로 행동을 수행함(Act)



기본 구조

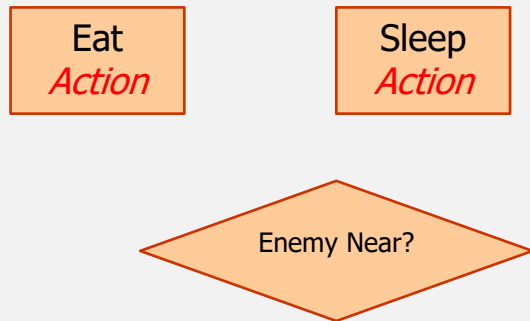
■ 트리 구조

- 말 그대로, 객체의 행위들을 tree 구조로 연결하여 나타냄.
- 매 프레임마다 tree 구조가 실행됨.
 - Root node 부터 시작해서, 아래로 실행되어 나감.
- node는 상태를 반환함.
 - SUCCESS, FAIL, RUNNING
- Node가 자식 노드가 있으면, 자식 노드들을 실행하고, 그 결과를 종합하여 노드의 최종 상태를 결정함.



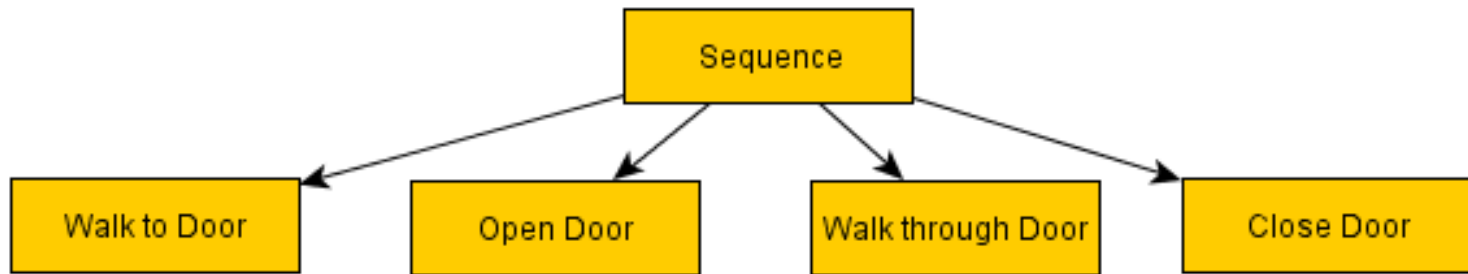
Leaf Node

- 단위 작업을 수행하는 노드로써, Action 또는 Condition 처리.
- Action
 - 어떤 일을 수행함.
 - 이동, 공격 등등
 - 목적을 달성하기 위해서 "매 프레임마다 해야 할 일"을 담음.
 - 수행 결과는 세 종류 : SUCCESS, FAIL, RUNNING(Task의 수행이 진행 중임. SUCCESS/FAIL 판단 유보)
- Condition
 - 여러가지 주변 상황, 상태등을 검사함.
 - 주인공과의 거리, 장애물 상태, 아이템 속성 등등
 - 조건 검사 결과, SUCCESS 또는 FAIL을 return함.



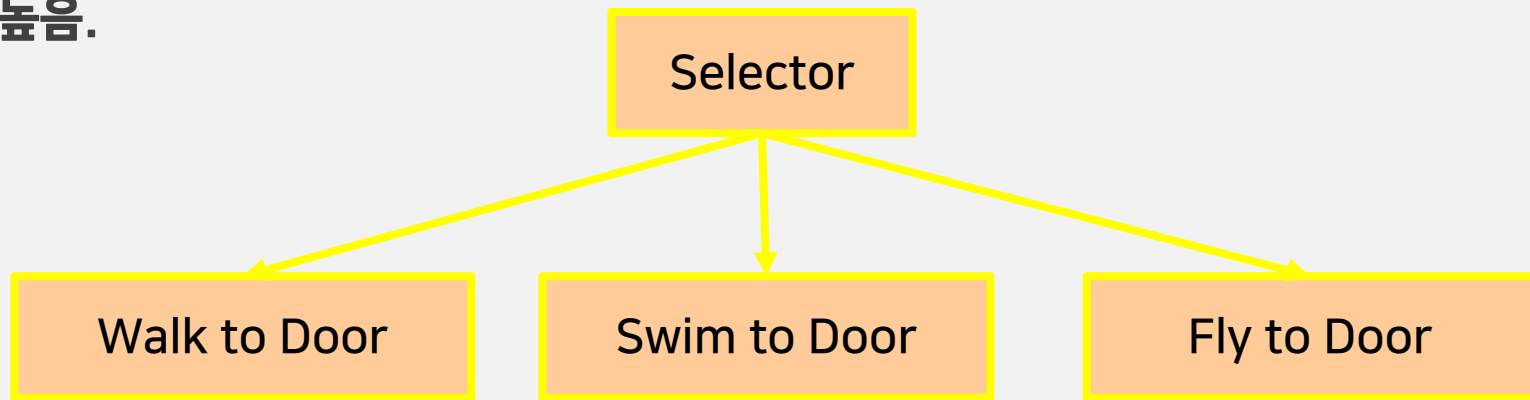
Sequence Node

- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하면서 실행됨.
- 모든 자식 노드가 다 SUCCESS 되면, 노드도 성공
- 여러 개의 작업이 모두 다 차근 차근 진행되어야 하는 경우 - AND 조건
- 하나라도 FAIL 되면, 실행 중단. Sequence Node 도 FAIL
- 실행 결과, 처음으로 RUNNING이 나오면, 자식 노드의 위치를 기록함. 결과는 RUNNING임.
- 어떤 목표를 달성하기 위해 수행해야 하는 Task 들을 차례로 모두 완수해야 하는 경우에 사용 됨.

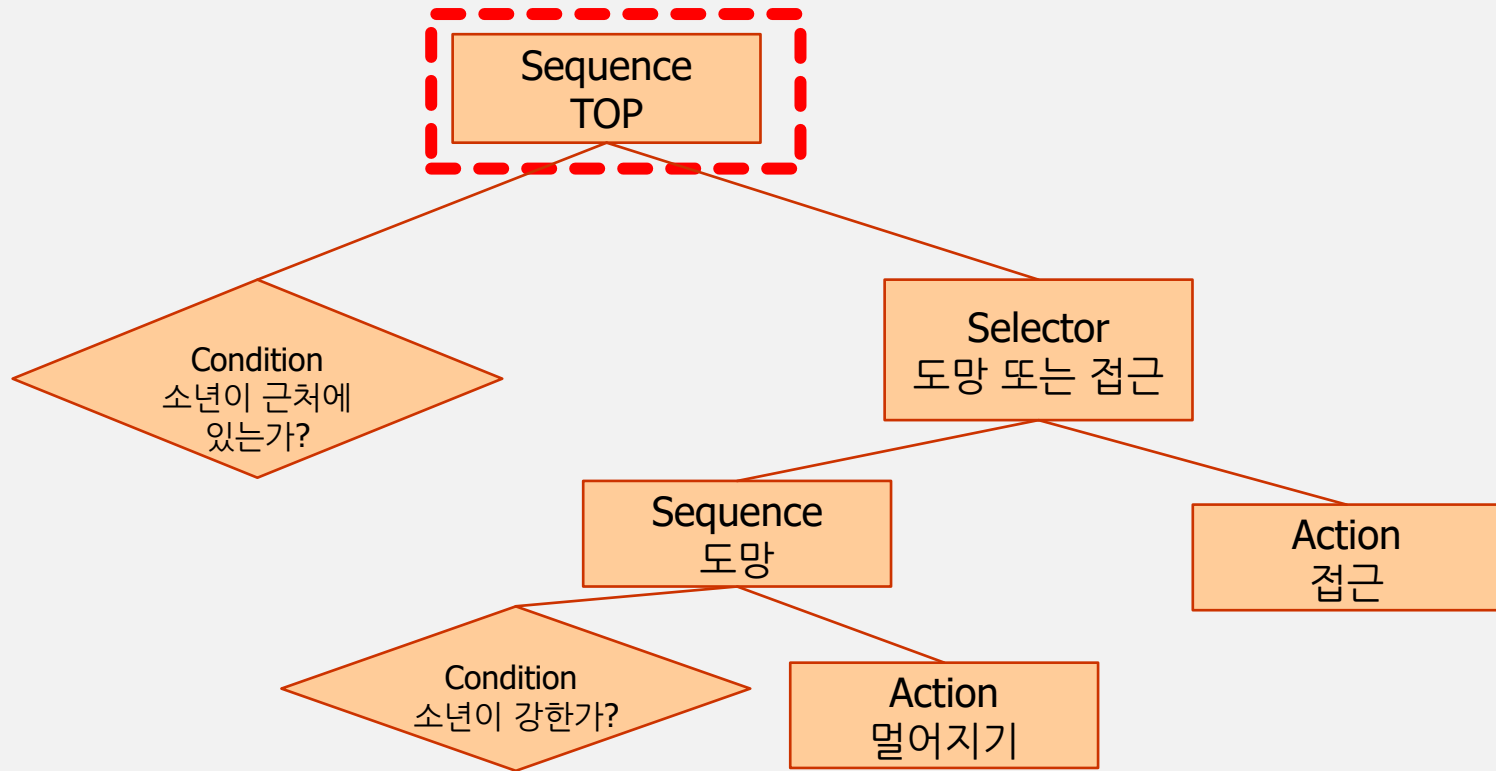


Selector Node

- 자식 노드 중, 하나만 성공하면 성공
- 여러 개의 작업 중, 하나를 선택하는 개념 - OR
- 실행은, 맨 왼쪽 자식 노드부터 오른쪽으로 진행하면서 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.
- 작업에 우선 순위를 부여할 때 사용됨. 즉, 왼쪽에 있는 노드가, 오른쪽에 있는 노드보다 우선 순위가 높음.



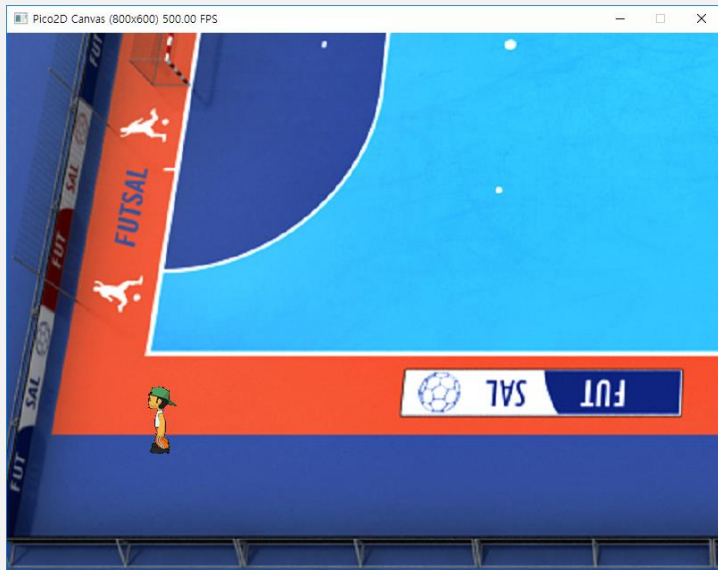
소년과 좀비의 힘 대결



기본 스크롤링 구현



y - window_bottom



x - window_left

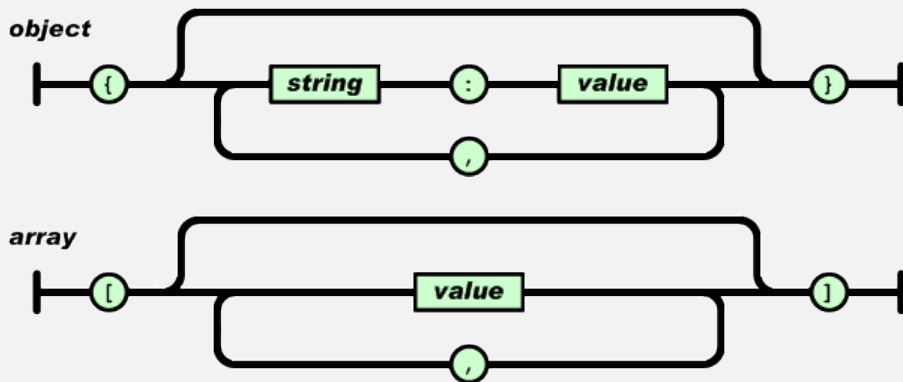


직렬화(Serialization)

- 프로그램 내의 객체 데이터를 외부에 저장 또는 보내는 행위.
- 나중에 다시 복구(de-serialization)할 수 있어야 함.
- 직렬화의 활용
 - 게임 플레이 상황을 저장(Save)하고 로드(Load).
 - 맵 데이터의 출력(Export) 및 입력(Import)
 - 게임내 객체들의 초기화 데이터 로딩
 - 게임 결과 및 기록 저장

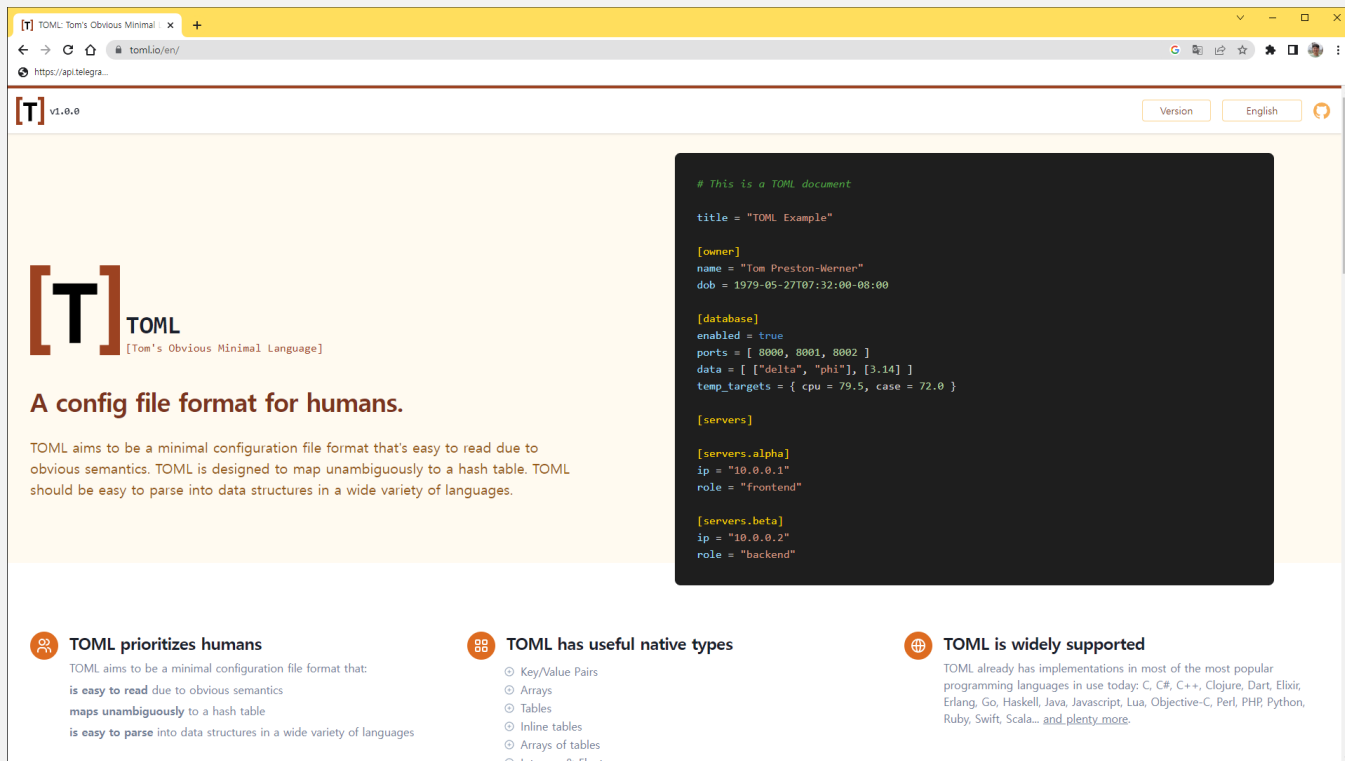
JSON(Java Script Object Notation)

- File write/read 의 경우, 문자열을 실제 데이터로 변환하는 작업이 필수적임. -> 까다롭고 복잡하고, 에러 발생의 소지가 많음.
- 객체를 교환(저장 및 전송 등)하기 위한 텍스트 형식 표준
- 파이썬의 리스트와 딕셔너리와 거의 동일



TOML

■ 설정 데이터를 좀 더 읽기 쉽게 저장하기 위한 텍스트 파일 형식



The screenshot shows the TOML website. At the top, there's a navigation bar with a logo, version (v1.0.0), and language (English) options. The main content area features the TOML logo and the tagline "A config file format for humans." Below this, a paragraph explains that TOML is a minimal configuration file format designed for readability and easy parsing. To the right, a dark-themed code block displays a sample TOML document. At the bottom, three key features are highlighted with icons: TOML prioritizes humans, TOML has useful native types, and TOML is widely supported.

[T] TOML
[Tom's Obvious Minimal Language]

A config file format for humans.

TOML aims to be a minimal configuration file format that's easy to read due to obvious semantics. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages.

```
# This is a TOML document

title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3,14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"

[servers.beta]
ip = "10.0.0.2"
role = "backend"
```

👤 TOML prioritizes humans
TOML aims to be a minimal configuration file format that:
is easy to read due to obvious semantics
maps unambiguously to a hash table
is easy to parse into data structures in a wide variety of languages


🔧 TOML has useful native types

- ⌚ Key/Value Pairs
- 📦 Arrays
- 📄 Tables
- 📄 Inline tables
- 📄 Arrays of tables
- 📄 Integers, 8, Floats

🌐 TOML is widely supported
TOML already has implementations in most of the most popular programming languages in use today: C, C#, C++, Clojure, Dart, Elixir, Erlang, Go, Haskell, Java, Javascript, Lua, Objective-C, Perl, PHP, Python, Ruby, Swift, Scala... and plenty more.

TOML

■ 설정 데이터를 좀 더 읽기 쉽게 저장하기 위한 텍스트 파일 형식



A config file format for humans.

TOML aims to be a minimal configuration file format that's easy to read due to obvious semantics. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages.

```
# This is a TOML document

title = "TOML Example"


[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3,14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"


[servers.beta]
ip = "10.0.0.2"
role = "backend"
```



TOML prioritizes humans


TOML aims to be a minimal configuration file format that:

- is **easy to read** due to obvious semantics
- maps **unambiguously** to a hash table
- is **easy to parse** into data structures in a wide variety of languages



TOML has useful native types

- Key/Value Pairs
- Arrays
- Tables
- Inline tables
- Arrays of tables
- Integers, B, Floats



TOML is widely supported

TOML already has implementations in most of the most popular programming languages in use today: C, C#, C++, Clojure, Dart, Elixir, Erlang, Go, Haskell, Java, Javascript, Lua, Objective-C, Perl, PHP, Python, Ruby, Swift, Scala... [and plenty more.](#)

__dict__

- 클래스를 이용해서 생성한 객체는 모든 속성(멤버 변수)을 dictionary 형태로 내부적으로 저장하여 사용함.
- obj.__dict__ 라는 내부변수가 바로 그것임.
- 이것을 통해, 객체의 속성을 쉽게 바꿀 수 있음. - 다른 dictionary 데이터를 이용해서.

```
class NPC:
    def __init__(self, name, x, y):
        self.name = name
        self.x, self.y = x, y

yuri = NPC('Yuri', 100, 200)

print(yuri.__dict__)

new_data = {"name": "Jusu", "x": 400, "y": 900}

yuri.__dict__.update(new_data)

print(yuri.__dict__)
print(yuri.name, yuri.x, yuri.y)
```

객체의 저장과 복구

- 객체의 멤버 변수들을 모두 저장할 필요가 없는 경우가 대부분.
- 저장이 필요한 내용만 선택적으로 골라서 저장할 수 있음.
- 피클링이 필요한 멤버 변수만 저장하고 복구함.

`__getstate__` : 객체를 저장할 때, 즉 피클링할 때 필요한 내용을 결정.
피클 모듈은 객체의 `__getstate__` 함수를 이용해서, 저장할 내용을 가져옴.

```
def __getstate__(self):  
    state = {'name': self.name, 'x': self.x, 'y': self.y, 'size': self.size}  
    return state
```

```
def __setstate__(self, state):  
    self.__init__()  
    self.__dict__.update(state)
```

`__setstate__` : 객체를 복구할 때, 즉, 역피클링할 때, 복구할 내용을 결정.

`pickle.load` 를 이용해서 객체를 만들 경우, `__init__()` 가 자동 호출되지 않으므로, 수동으로 호출함. 저장된 내용 이외의 값들을 채우기 위함.