

Lecture #19. 강의 정리

2D 게임 프로그래밍

이대현 교수

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, ...)

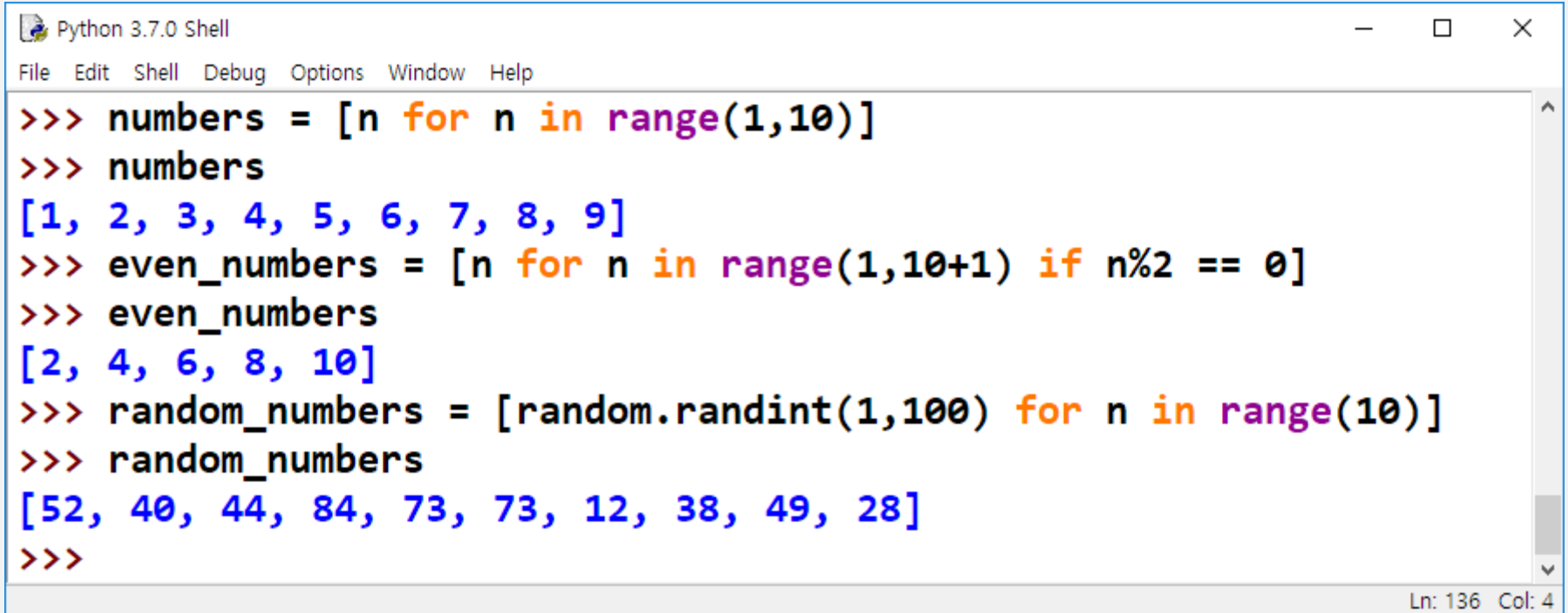
■ Set – set

- 중복을 허용하지 않는, 순서에 상관없는 데이터들의 집합

{ val1, val2, ... }

Python List Comprehension

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

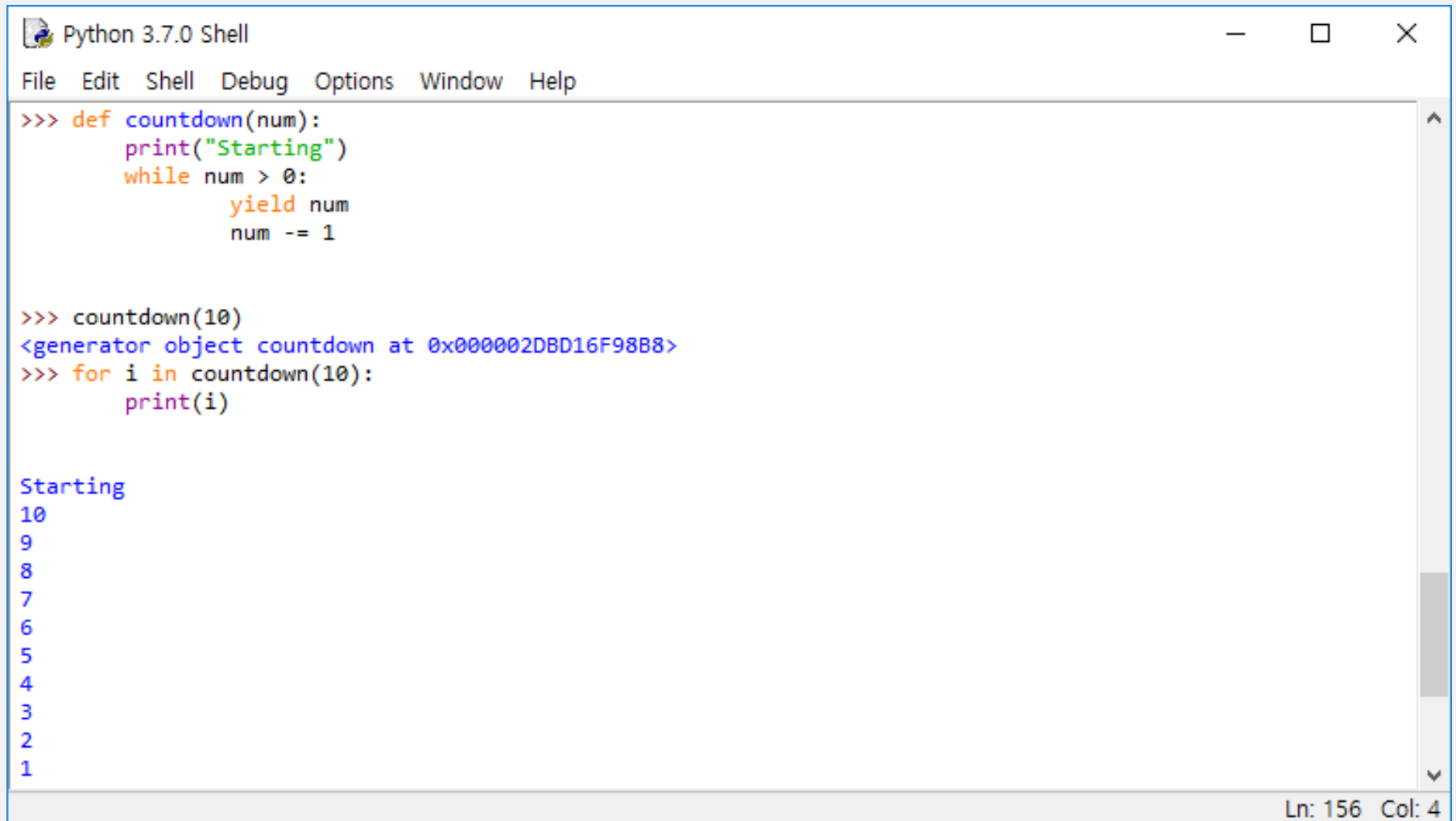
A screenshot of a Python 3.7.0 Shell window. The window has a title bar with the text 'Python 3.7.0 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains a series of Python commands and their outputs, demonstrating list comprehensions. The commands are: 1. 'numbers = [n for n in range(1,10)]' followed by 'numbers' on the next line, which outputs '[1, 2, 3, 4, 5, 6, 7, 8, 9]'. 2. 'even_numbers = [n for n in range(1,10+1) if n%2 == 0]' followed by 'even_numbers' on the next line, which outputs '[2, 4, 6, 8, 10]'. 3. 'random_numbers = [random.randint(1,100) for n in range(10)]' followed by 'random_numbers' on the next line, which outputs '[52, 40, 44, 84, 73, 73, 12, 38, 49, 28]'. The prompt '>>>' is used for all commands. 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>

Python Generator

- 객체들을 하나씩 만들어서(발전) 넘겨주는 기능
- for 문 등에서 효과적으로 사용.



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> def countdown(num):
    print("Starting")
    while num > 0:
        yield num
        num -= 1

>>> countdown(10)
<generator object countdown at 0x000002DBD16F98B8>
>>> for i in countdown(10):
    print(i)

Starting
10
9
8
7
6
5
4
3
2
1

Ln: 156 Col: 4
```

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()
```

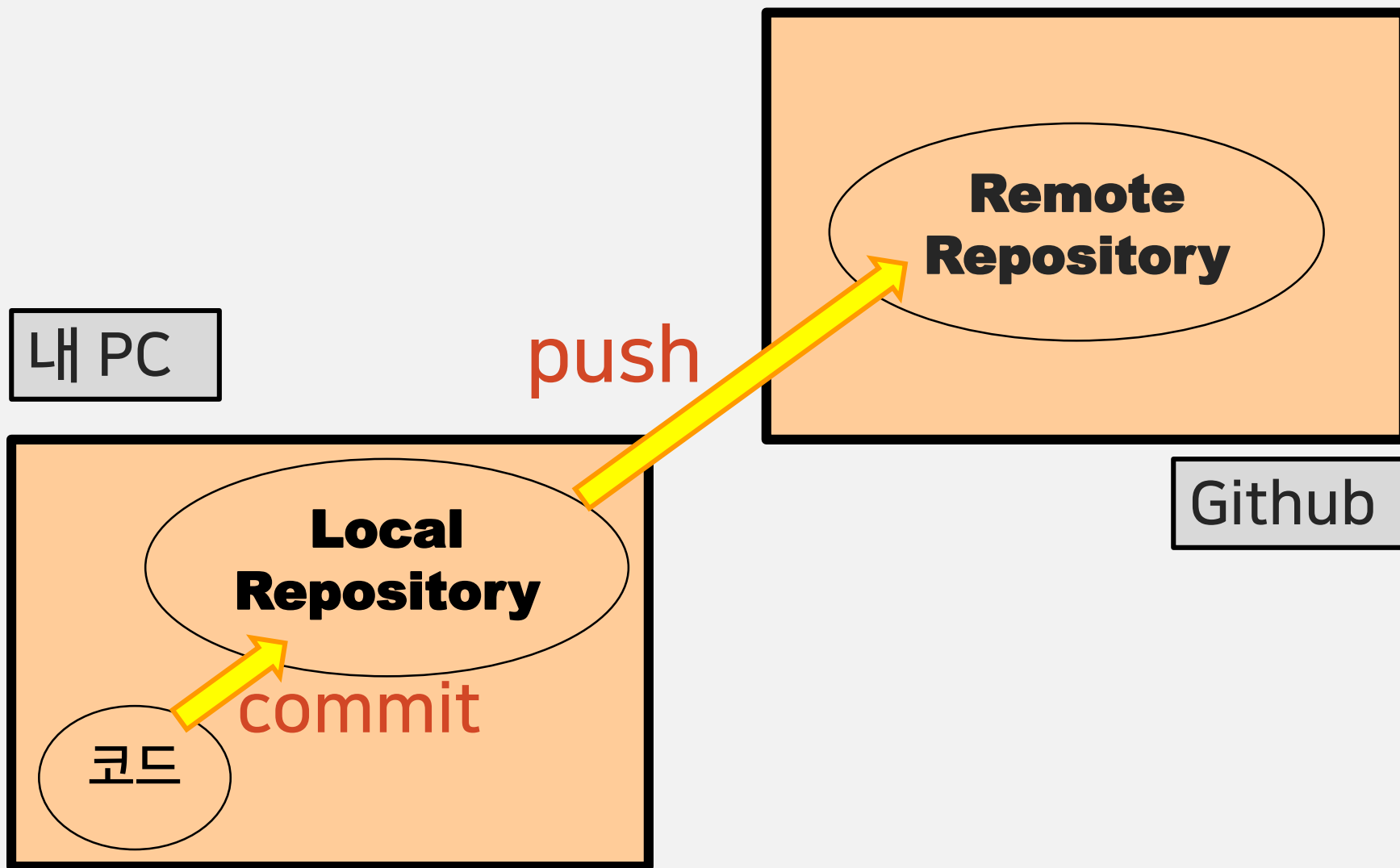
클래스 정의

함수 정의

문장

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

Commit 과 Push



게임이란?

- 가상 세계에 존재하는 게임 객체(오브젝트)들의 상호작용(인터랙션)을 시뮬레이션하고 그 결과를 보여주는 것
- 게임 객체의 상태를 변화시키는 3가지 요소는?
 - 물리
 - AI
 - 사용자입력

2D 게임?

■ 게임의 기본 구성 요소

- 배경
- 캐릭터, 오브젝트
- UI - GUI, 입력(키,마우스,터치, ...)
- AI
- 사운드

■ 2D 게임?

- 현재 진행 중인 게임 가상 월드의 내용을 화면에 2D 그림으로 보여주는 것
- 배경,캐릭터(오브젝트)의 표현(렌더링)을 2D 이미지들의 조합으로 구성함!

2D 게임의 기본 요소

GUI

배경(World)

캐릭터

오브젝트



2D 게임 개발 접근법

■ 플랫폼 종속적 방법

- Direct X
- OpenGL
- Simple Frame Buffer

■ 플랫폼 독립적 방법, Cross Platform

- Unity3D
- COCOS2D
- 그 외의 범용 2D 렌더링 라이브러리

- SDL !!!!

게임 개발 시작에 앞서 결정해야 할 것들

- 게임월드의 물리적 크기
- 게임오브젝트들의 물리적 값들
 - 크기
 - 속도
- 픽셀당 몇미터인가?
- 좌표계의 구성은?
- 피봇은 어디로 잡을 것인가?

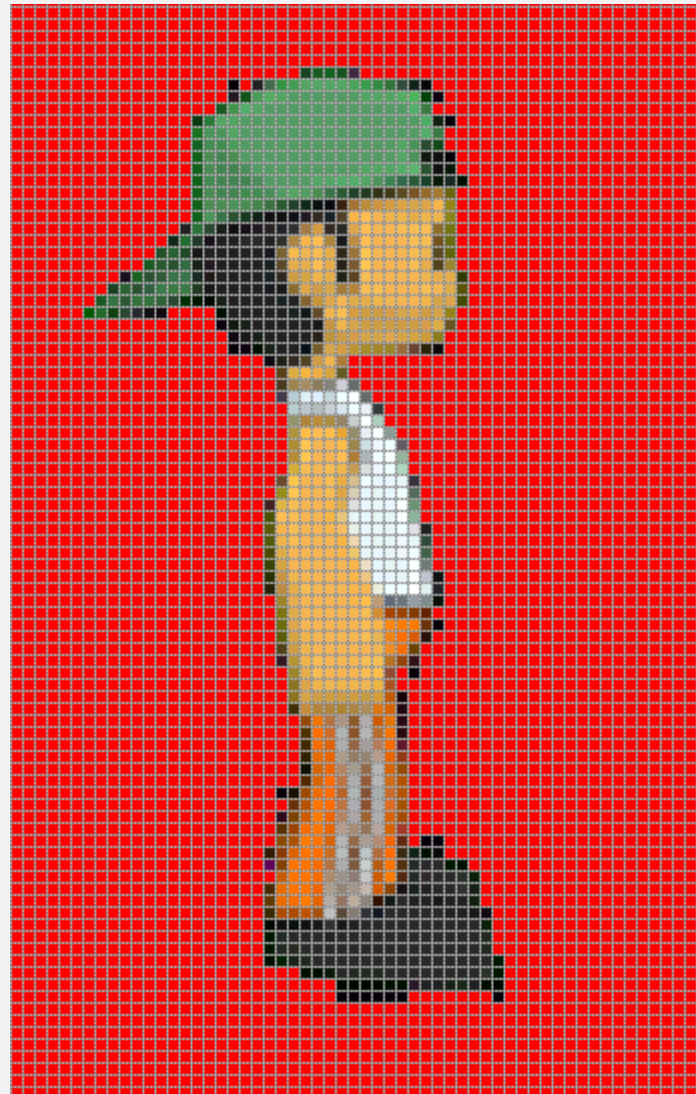
피봇(Pivot)



여기가 피봇입니다.

이 점을 피봇으로 삼기도 합니다

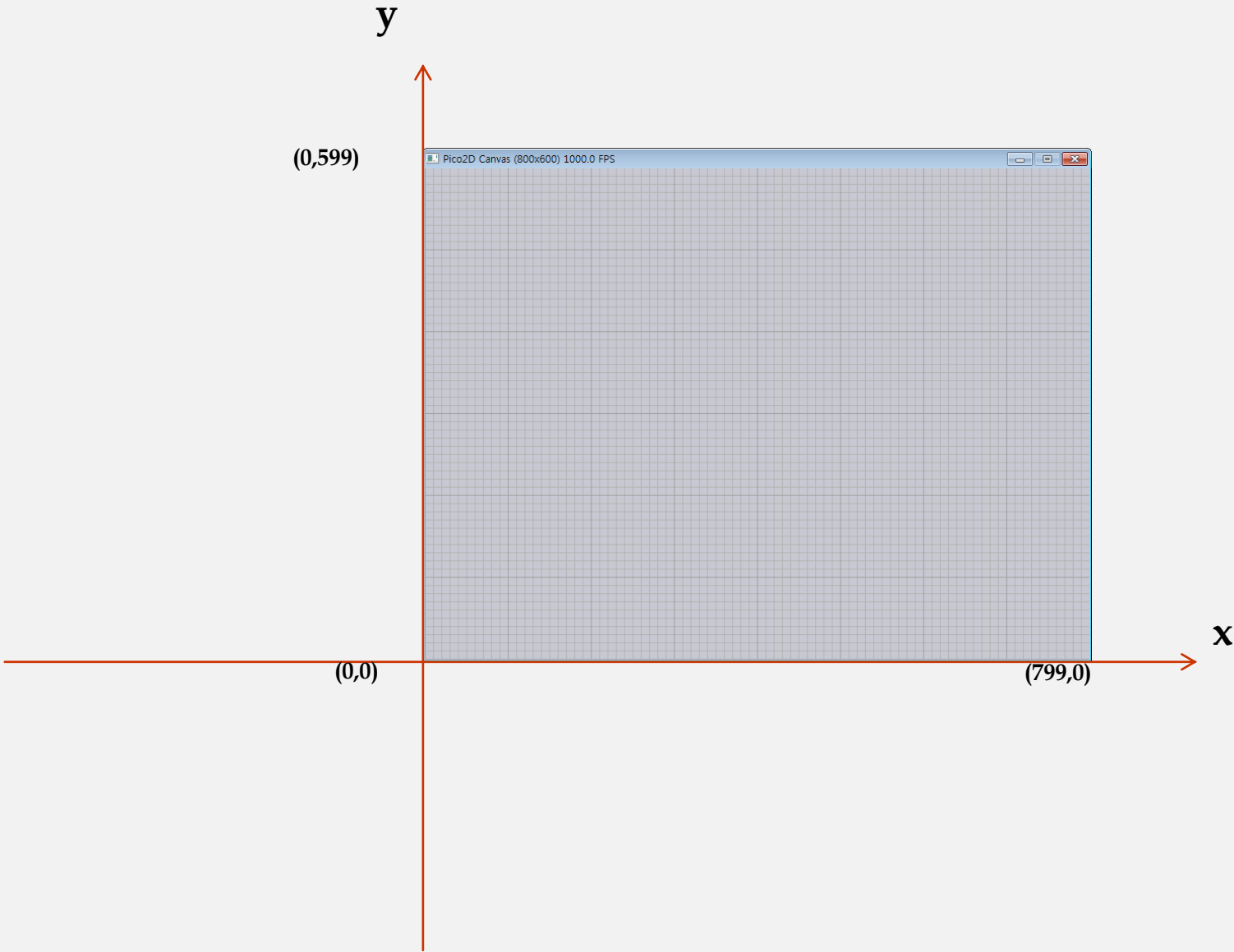
2D 공간의 물리값들을 먼저 결정할 필요



74 pixel = 약 2.2미터

10 pixel = 30 cm

캔버스의 좌표계



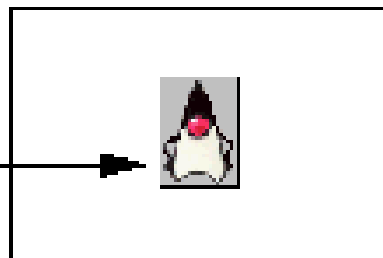
JPG vs PNG

더블 버퍼링(Double Buffering)

Double Buffering

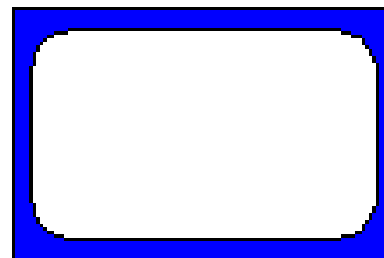
1. Draw

graphics



Image

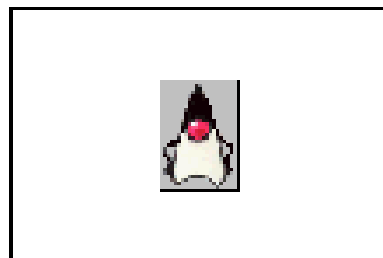
Back Buffer



Screen

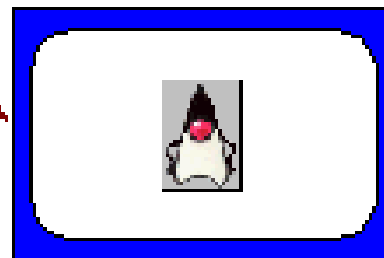
Primary Surface

2. Blt
(copy)



Image

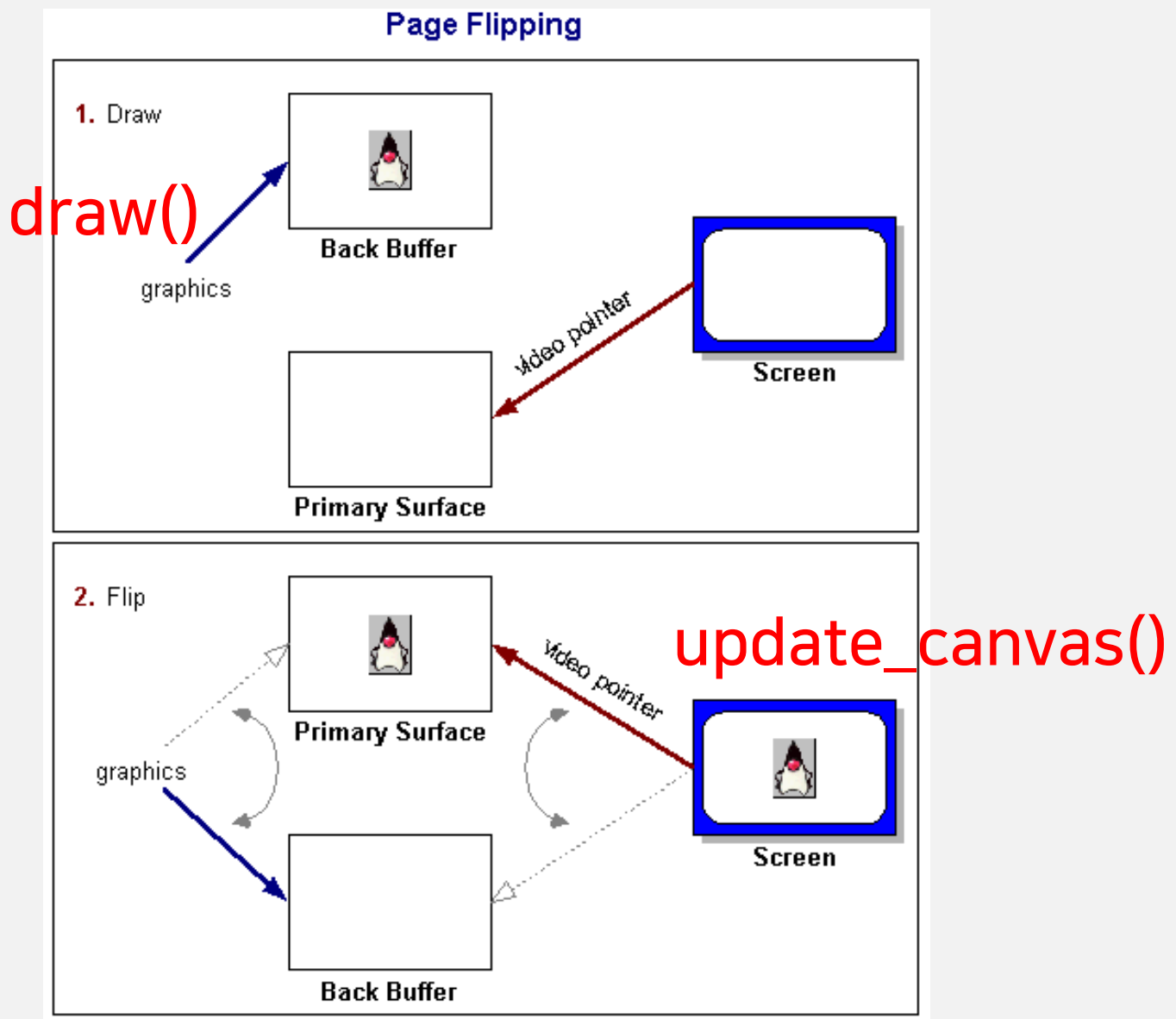
Back Buffer



Screen

Primary Surface

페이지 플리핑(Page Flipping)



스프라이트(Sprite)

■ 스프라이트란?

- 게임 장면안에서 보여지는 이미지 또는 애니메이션되는 오브젝트
- 2D 게임에서는 게임의 모든 캐릭터들과 이동하는 물체들을 표현하는 데 사용됨.
- 3D 게임에서는 2D로 표현될 수 있는 각종 오브젝트에 사용됨.
 - 불, 연기, 작은 물체들, UI 표시 등등.



Metal Slug 3

키보드 및 마우스 입력 처리 과정

Step1: 입력 이벤트들을 폴링한다.(get_events())



Step2: 이벤트의 종류를 구분한다.(event.type 을 이용)



Step3: 실제 입력값을 구한다.(event.key 또는 event.x, event.y 등 을 이용)

마우스 좌표의 획득과 변환

```
if event.type == SDL_QUIT:  
    running = False  
elif event.type == SDL_MOUSEMOTION:  
    x, y = event.x, KPU_HEIGHT - 1 - event.y  
elif event.type == SDL_KEYDOWN and event.key == SDLK_ESCAPE:  
    running = False
```

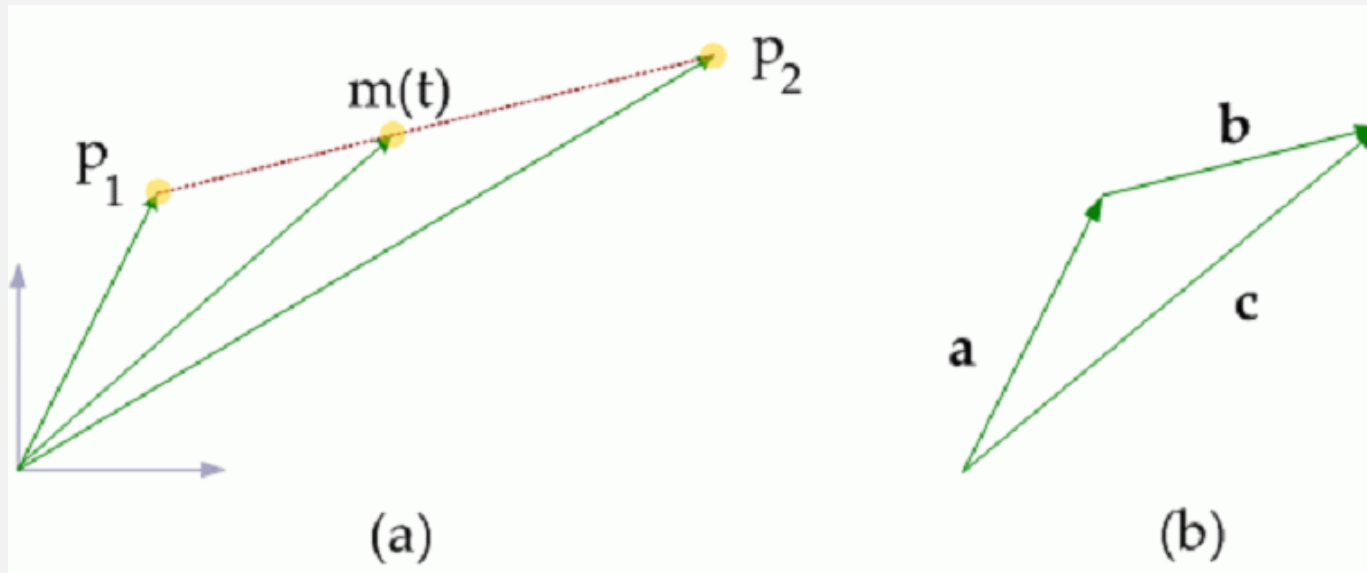
마우스가 이동하면, SDL_MOUSEMOTION 이벤트가 발생
event.x 및 y는, 윈도우 API 의 좌표계를 따름.
pico2d 좌표계 변환 필요.

Parametric Representation

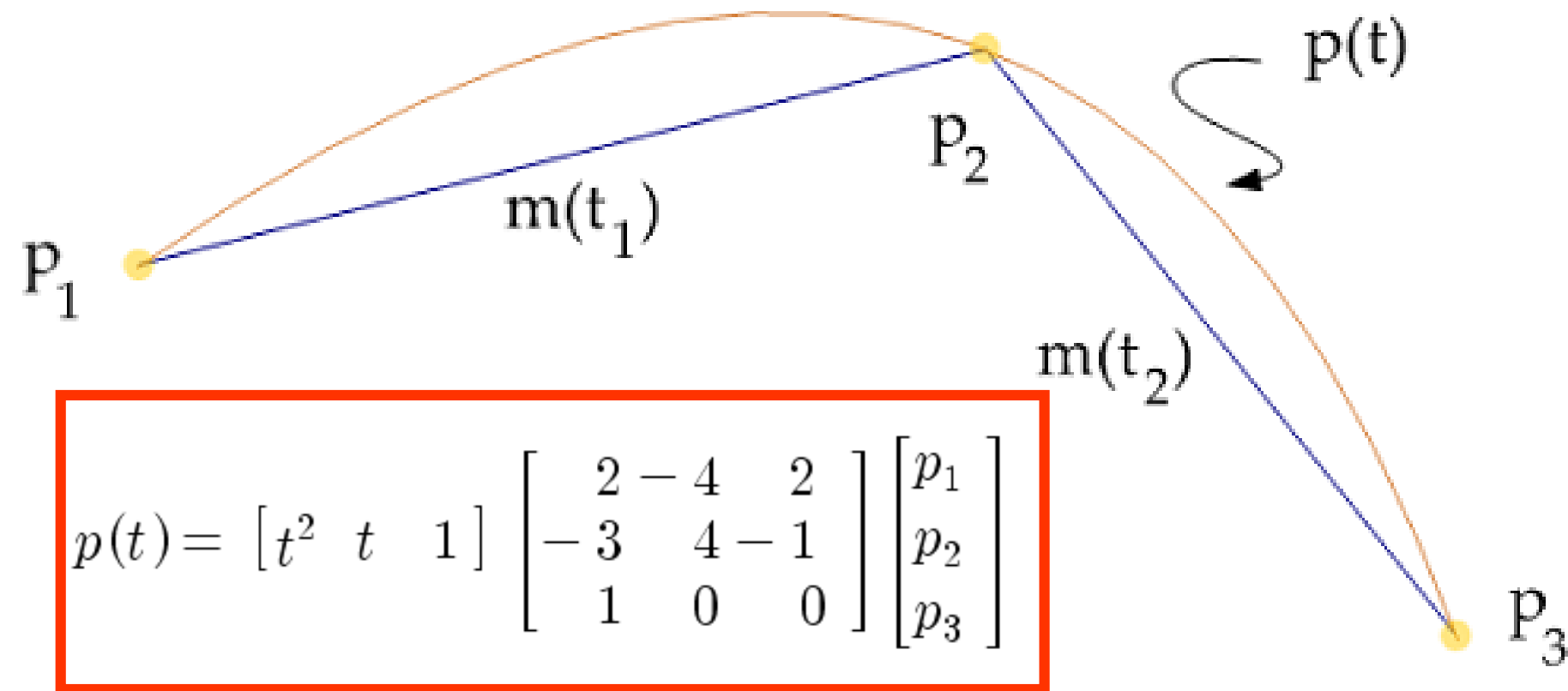
- 직선, 또는 곡선의 (x,y) 좌표를 공통적인 파라미터를 이용하여 표현하는 방법.
- 일반적인 수학적 표현에 비해, 컴퓨터를 이용하여 그리기가 편리함.
- 동일한 곡선에 대해, 파라미터 표현법은 여러 개 있음.

$$m(t) = (1 - t) * p_1 + t * p_2, \text{ t의 범위: } 0 \leq t \leq 1$$

파라미터 t로 표현

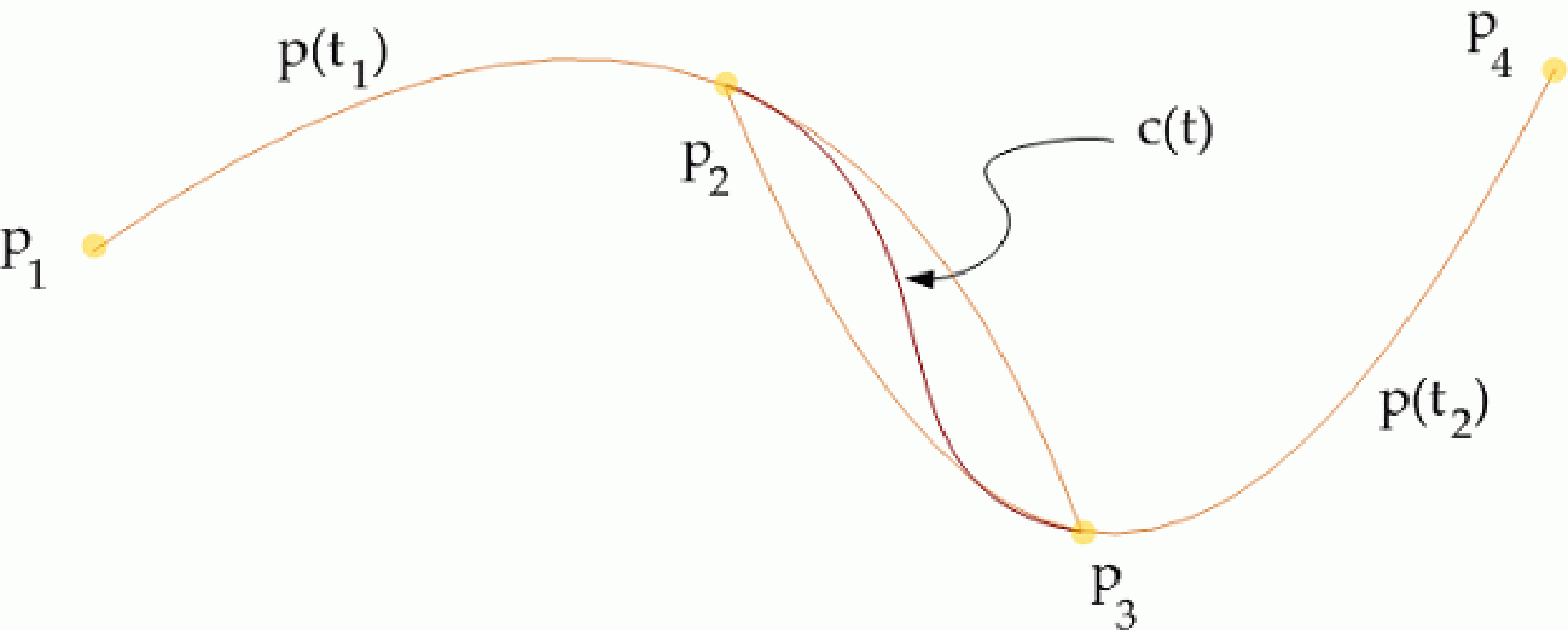


카디날 스플라인(Cardinal Spline)

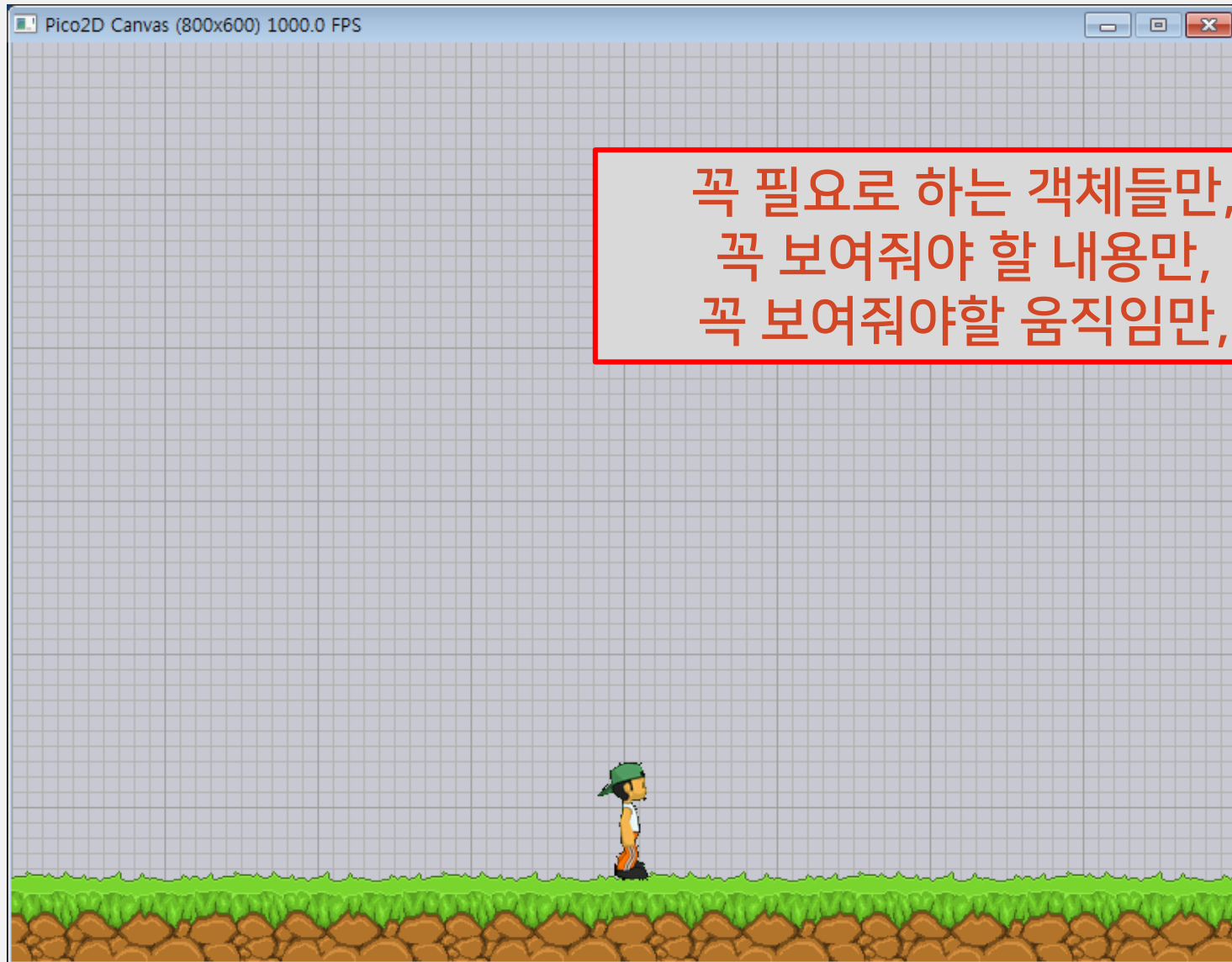


네 점을 연결하는 부드러운 곡선은?

아이디어: 두개의 곡선 $p(t_1)$ 과 $p(t_2)$ 를 $1-t:t$ 의 비율로 섞음.



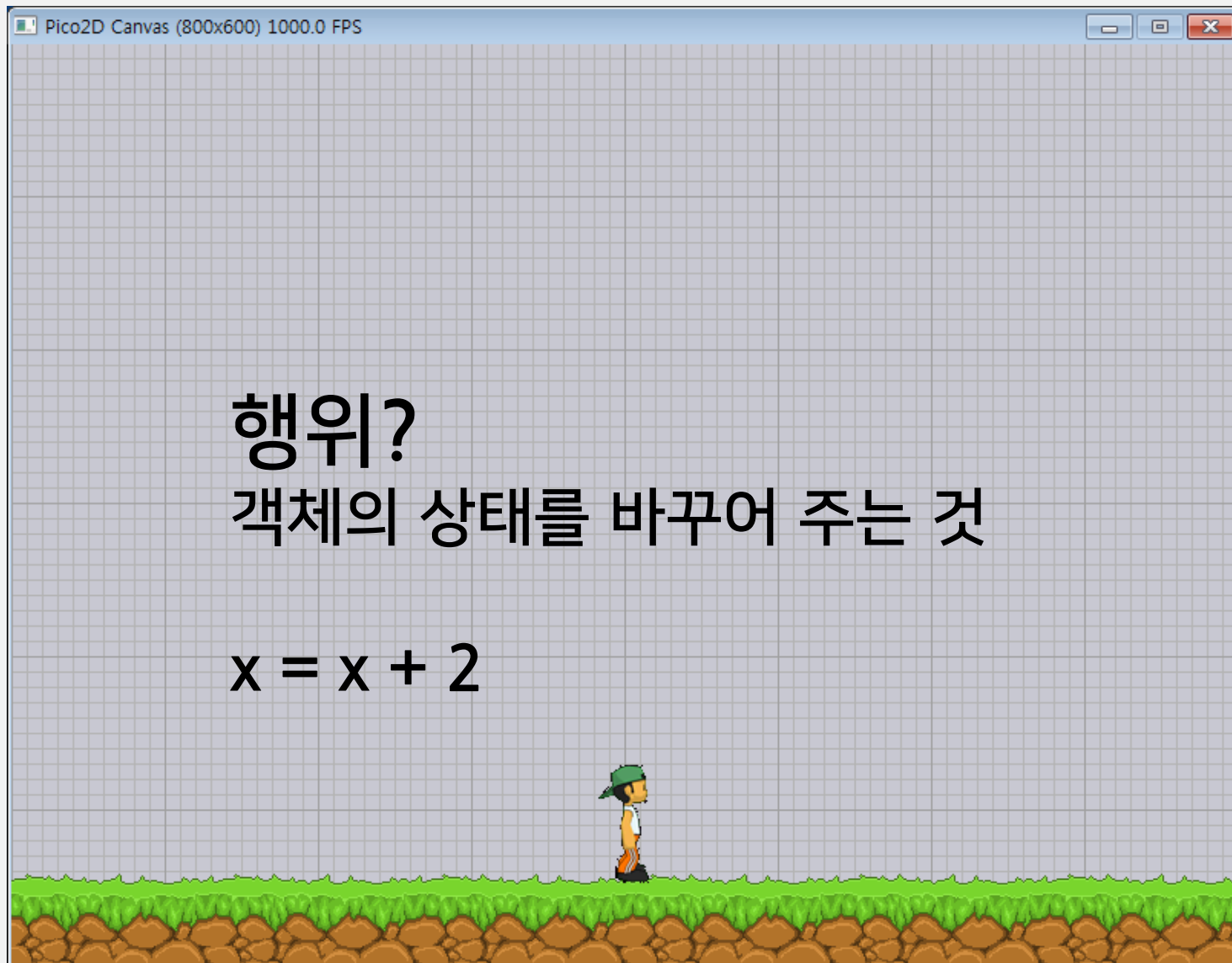
추상화(Abstraction)



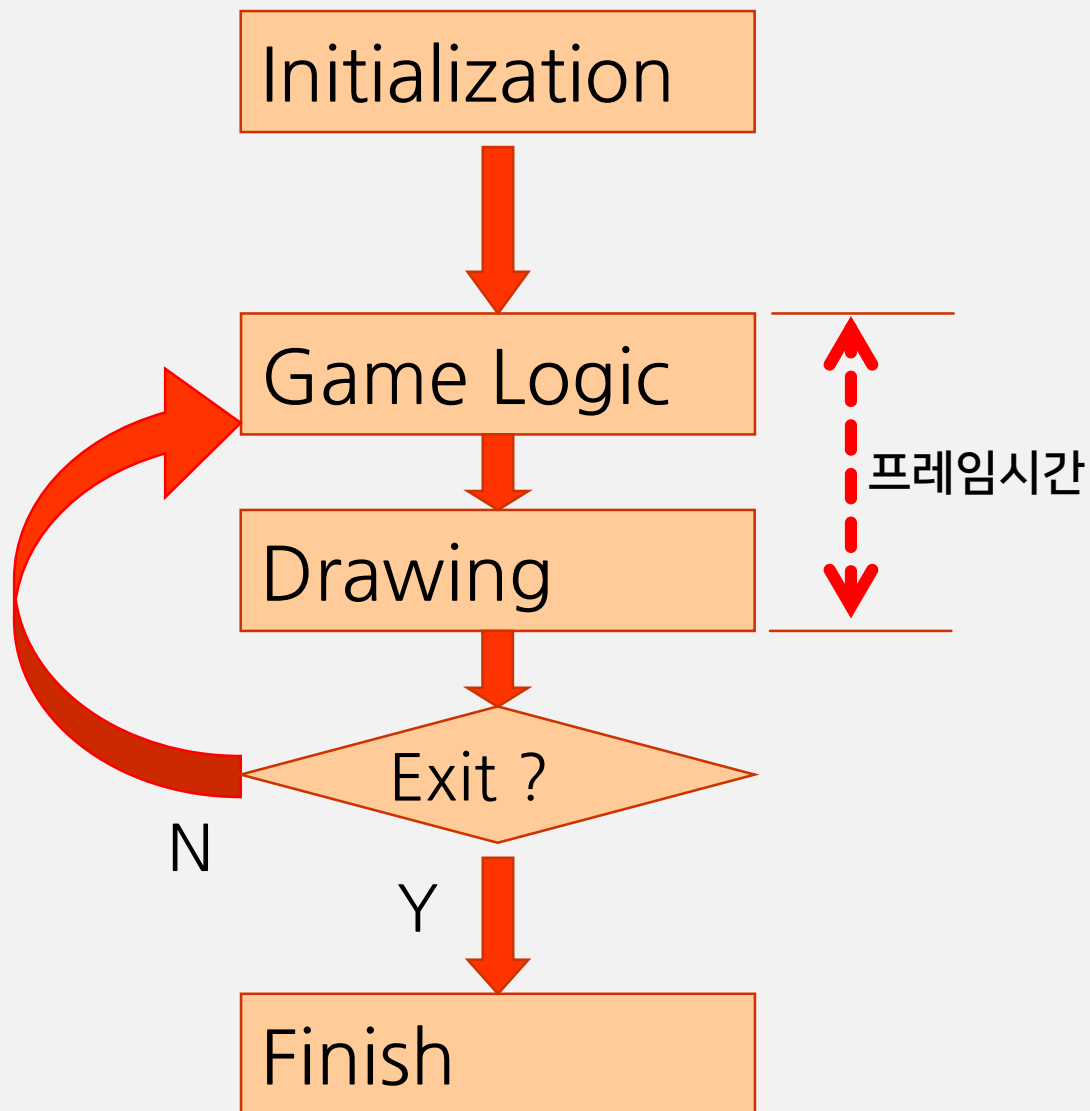


속성 + 행위 = 객체





게임 기본 구조 - 게임 루프



게임 상태(Game State)의 이해 (1)

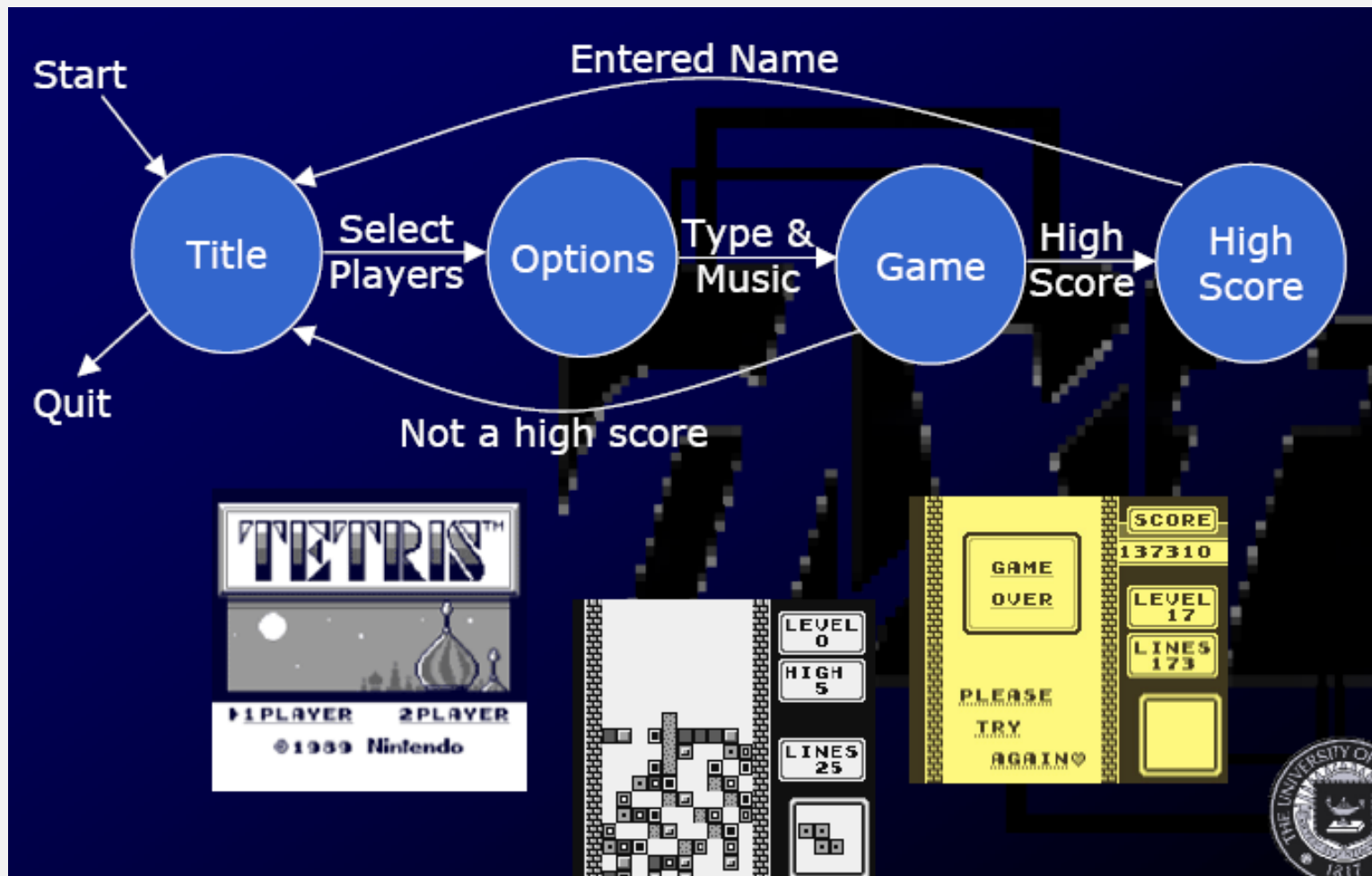
■ 게임 상태란?

- 게임 프로그램 실행 중의 어떤 특정 위치(또는 모드).
- 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임의 상태에 따라 달라짐.



- 맵 선택 상태.
- 방향키는 맵 선택을 처리.

- 게임 메인 플레이 상태..
- 방향키는 캐릭터의 이동을 처리.



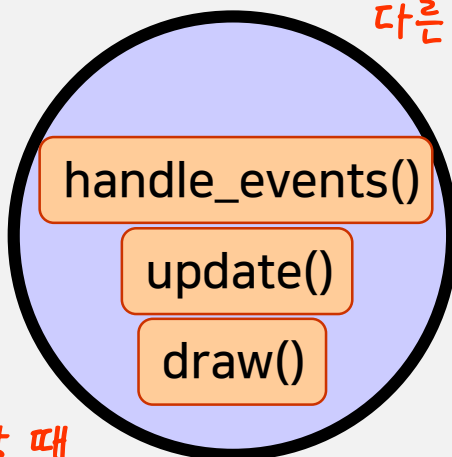
게임 상태의 구현

게임 상태에 들어올때 초기화

enter()

exit()

게임 상태에서 나갈 때
종료화

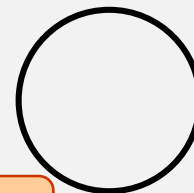


다른 상태로 잠깐 이동

pause()

resume()

현재 상태로 복귀



상태간의 전환: game_framework을 이용

run(state):

state를 시작 게임 상태로 하여, 게임 실행을 시작함.

change_state(state):

게임 상태를 state로 변화. 이전 게임 상태를 완전히 나눔.

push_state(state):

게임 상태를 state로 변화. 이전 게임 상태는 남아 있음.

pop_state(): 이전 게임 상태로 복귀

quit(): 게임을 중단

run(state): 게임을 state로 시작함.

캐릭터 컨트롤러(Character Controller)

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



클래스 변수

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

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

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.

게임 월드 game_world.py

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

게임 월드에 담겨있는 모든 객체들을 담고 있는 리스트.

Drawing Layer 에 따라서 분류.

```
def add_object(o, layer):  
    objects[layer].append(o)
```

게임 월드에 객체 추가

```
def remove_object(o):  
    for i in range(len(objects)):  
        if o in objects[i]:  
            objects[i].remove(o)  
            del o  
            break
```

게임 월드에서 객체 제거

```
def clear():  
    for l in objects:  
        l.clear()
```

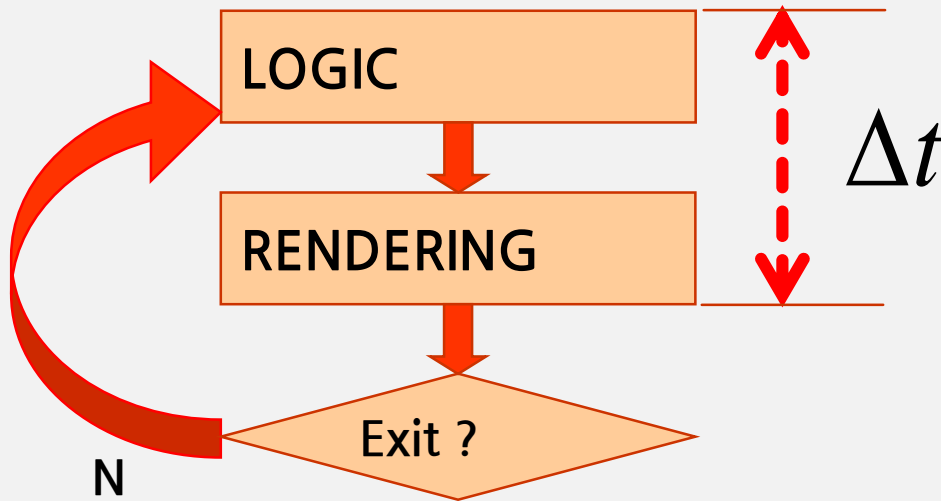
게임 월드의 모든 객체 제거

```
def all_objects():  
    for i in range(len(objects)):  
        for o in objects[i]:  
            yield o
```

게임 월드의 모든 객체들을 하나씩 꺼내오기

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 60fps

■ 프레임 시간과 프레임 속도의 관계

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

■ 프레임 시간은 균일하지 않다..

- 씬이 복잡하거나, 처리해야 할 계산이 많으면 시간이 많이 걸림
- 동일한 씬이라도, 컴퓨터의 성능에 따라서도 차이가 남.

■ 문제점은?

- 게임의 실행속도가 컴퓨터마다,,,, 또는 게임 내의 씬의 복잡도에 따라 달라지므로, 게임 밸런싱에 큰 문제를 야기함.
 - ex. 캐릭터의 이동속도가 달라짐..

해결 방법은?

■ 아예 고정하기...

□ 그래픽 라이브러리 자체에서 싱크를 조정하도록...

□ `open_canvas(sync=True)`

□ 60fps 로 고정할 수 있음.

```
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 55.555556 fps, Frame Time : 0.018000 sec
Frame Rate: 55.555556 fps, Frame Time : 0.018000 sec
Frame Rate: 71.428571 fps, Frame Time : 0.014000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 58.823529 fps, Frame Time : 0.017000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
Frame Rate: 62.500000 fps, Frame Time : 0.016000 sec
```


아주 아주 아주 근사한 방법

- 게임 객체들의 운동에 “시간”의 개념을 도입

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

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

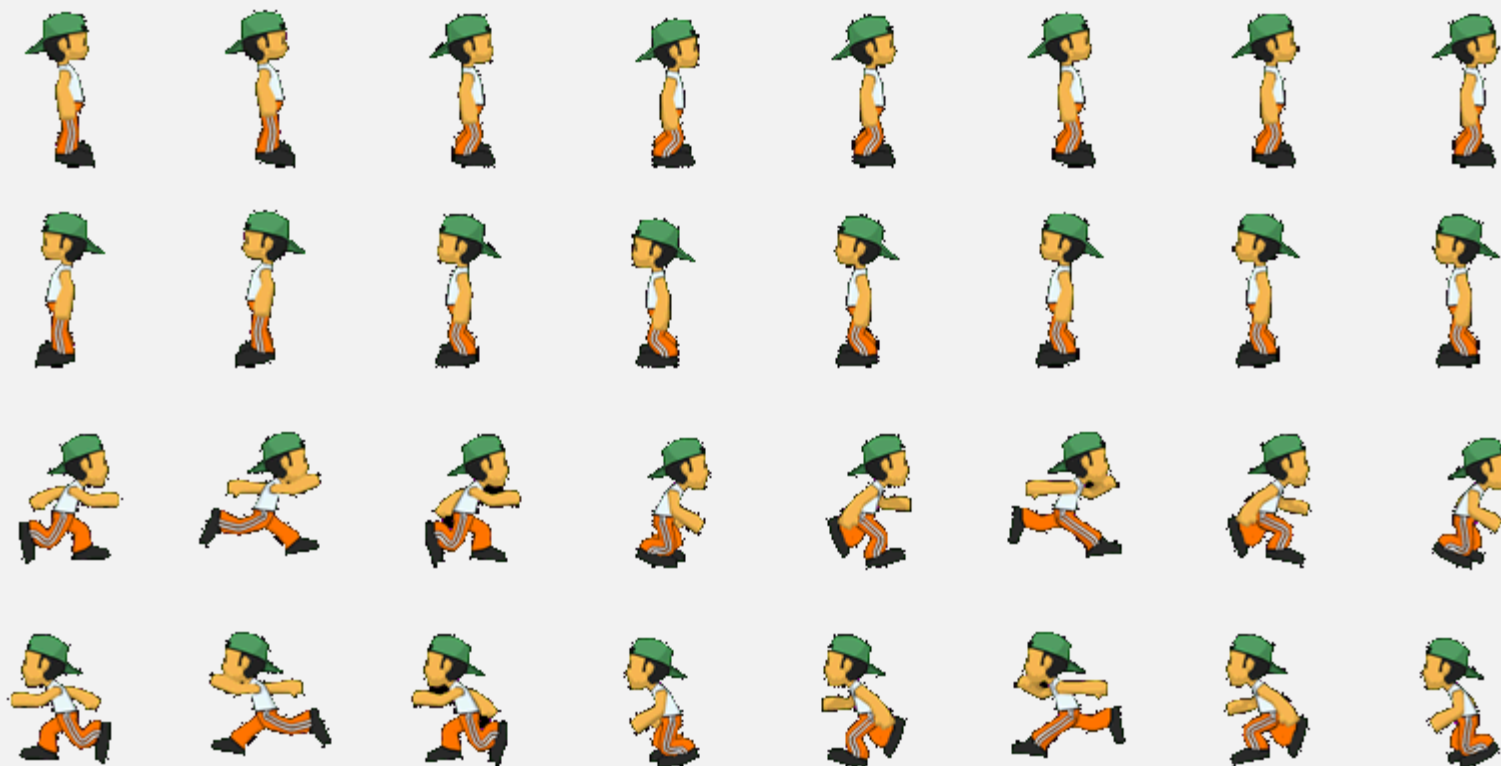
frame time을 이용한 객체 위치 계산

- 그 시간 동안 이동한 거리를 구한다.

- x : 객체의 위치
- v : 객체의 속도(등속 운동 가정)

$$X_{\text{다음프레임}} = X_{\text{현재프레임}} + v\Delta t$$

애니메이션 프레임의 경우도, 속도를 반영해야 함.



TIME_PER_ACTION = 0.5

ACTION_PER_TIME = 1.0 / TIME_PER_ACTION →

FRAMES_PER_ACTION = 8

충돌 검사(Collision Detection)

■ 충돌 검사

□ 게임 상의 오브젝트 간에 충돌이 발생했는지를 검사하는 것.

□ 모든 게임에서 가장 기본적인 물리 계산.

- 슈팅, 발차기, 펀치, 때리기, 자동차 충돌
- 맵 상의 길 이동

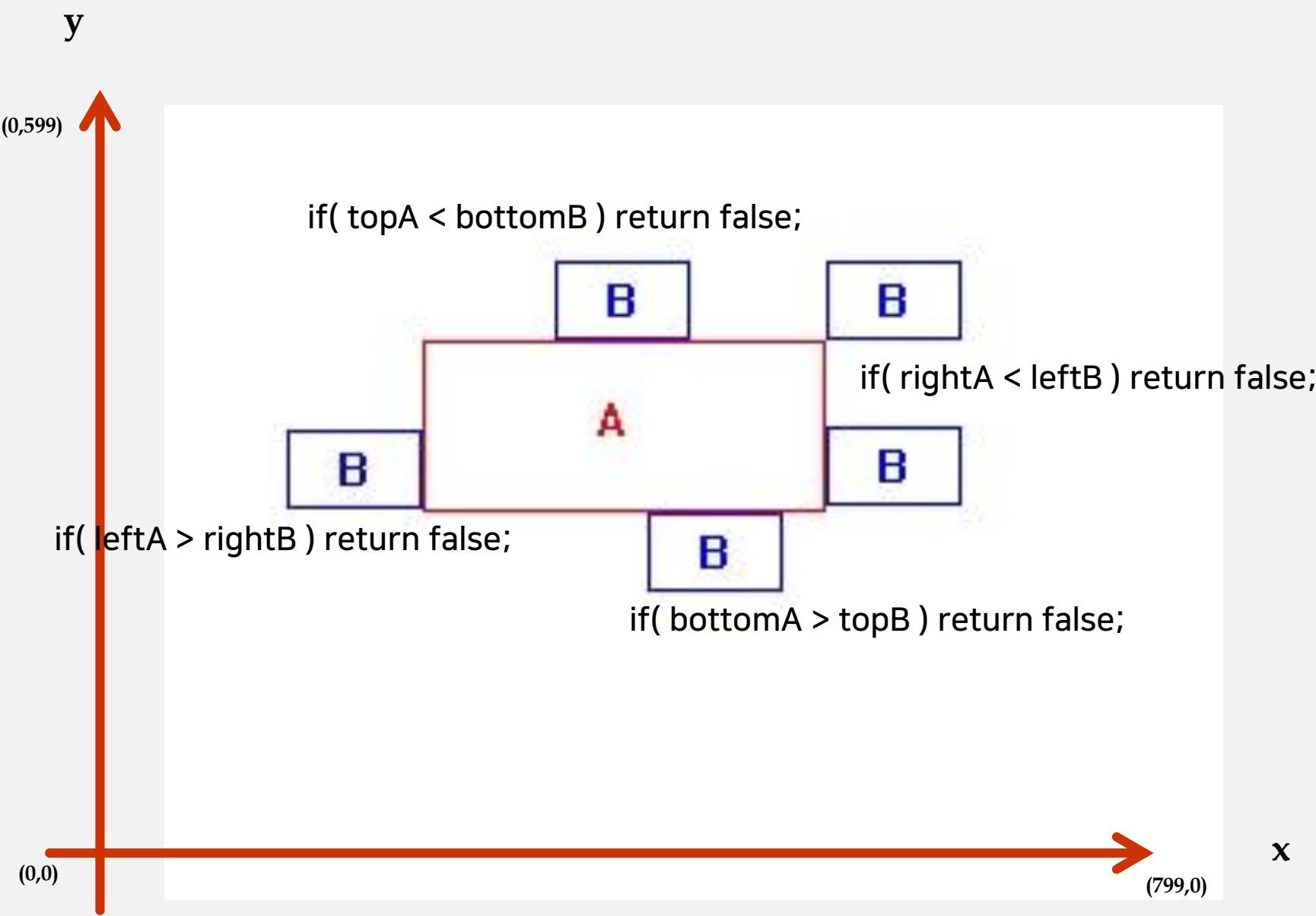
□ 기본적으로 시간이 많이 소요되기 때문에, 게임의 오브젝트의 특성에 따라 각종 방법을 통해 최적화해주어야 함.

- $O(N^2)$ 알고리즘
- $nC2 = \frac{n(n-1)}{2}$

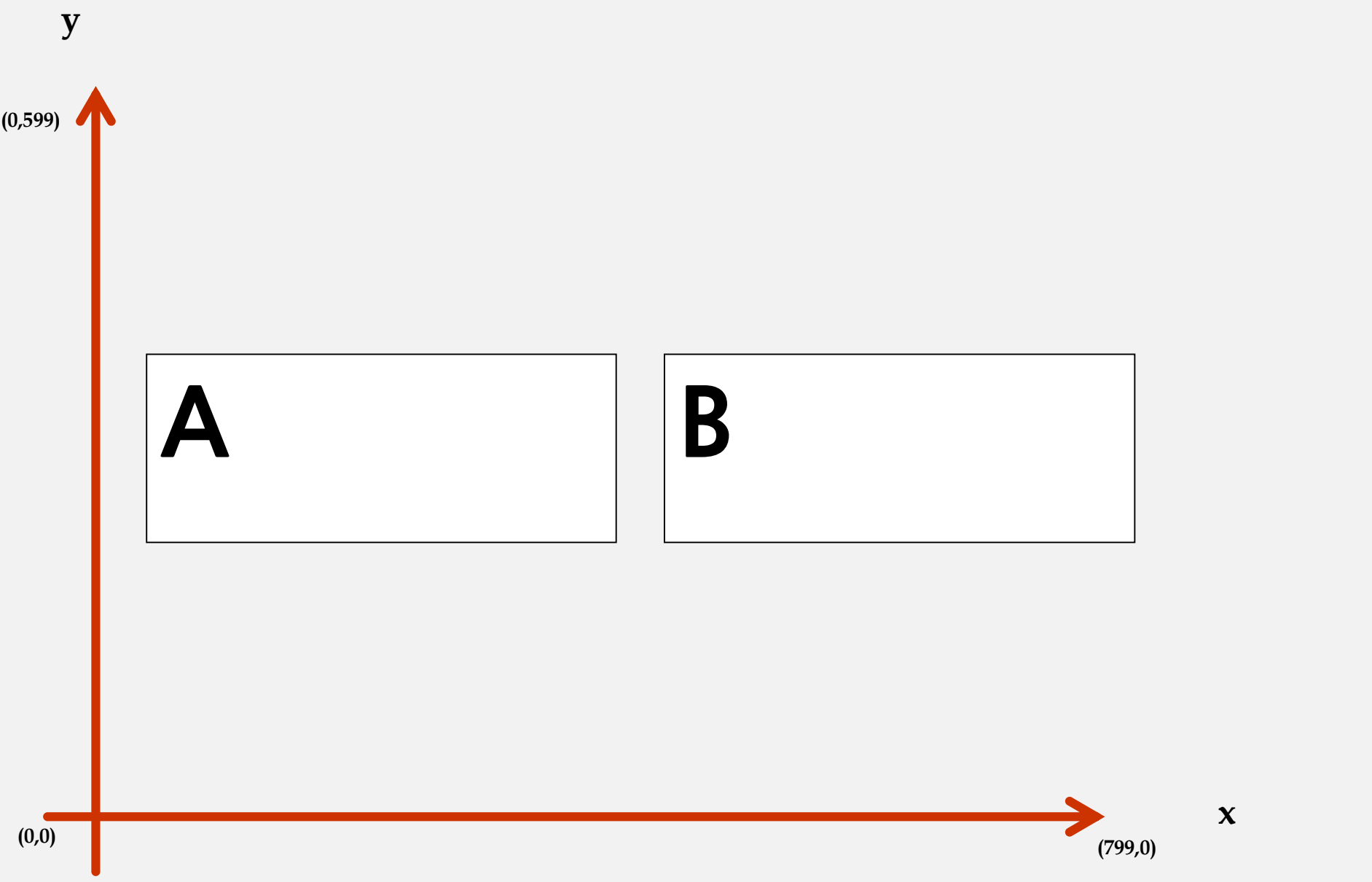
충돌 처리(Collision Handling)

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

사각형과 사각형

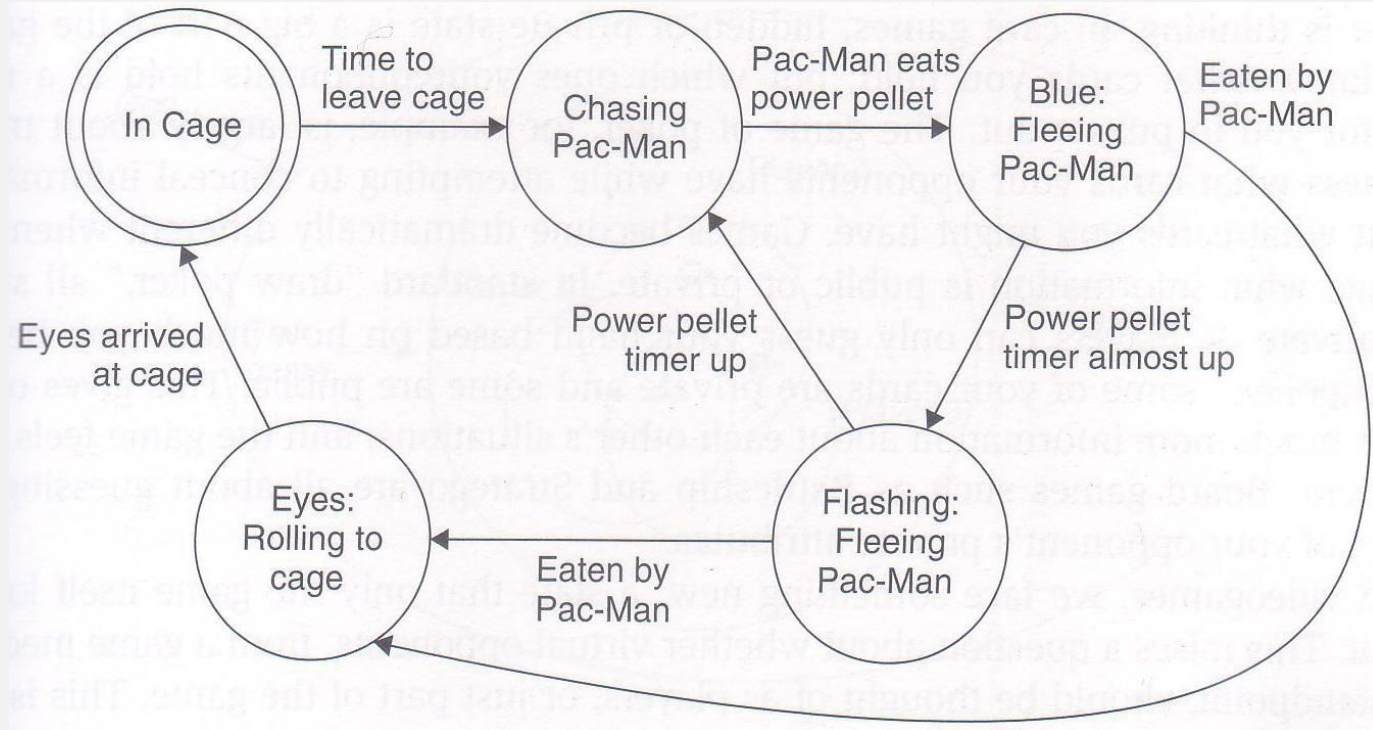


사각형과 사각형



FSM – 가장 전통적인 게임 AI 구현 방식

- 상태의 개수가 늘어남에 따라, 와이어링(이벤트의 변화 추적)이 복잡해짐.
- 정확히 상태를 분리해서, 추출하는 것이 어려움.
- HFSM(Hierarchical FSM)이 실전에서는 사용됨.



Behavior Tree

- 객체의 인공지능행동을 트리 구조로 구현한 것.
- FSM 방식 - 상태와 이벤트에 따라서, 다음 상태를 결정
- BT 방식 - Goal 을 달성하기 위한 Task들을 구성. 재사용이 쉽게 직관적임.
- HALO 에서 사용된 후, 기본 구조가 공개됨.
- GTA 등에서도 사용

https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php



기본 구조

■ 트리 구조

□ 말 그대로, 객체의 행위들을 tree 구조로 연결하여 나타냄.

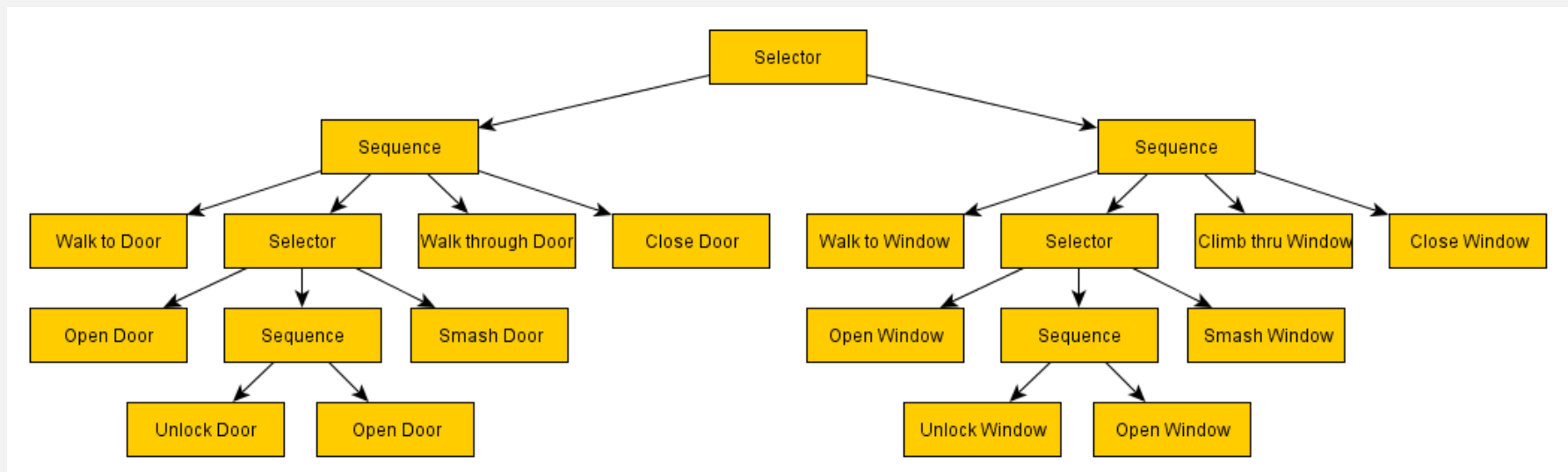
■ 매 프레임마다 tree 구조가 실행됨.

□ Root node 부터 시작해서, 아래로 실행되어 나감.

■ node는 상태값을 반환함.

□ SUCCESS, FAIL, RUNNING

■ Node가 자식 노드가 있으면, 자식 노드들을 실행하고, 그 결과를 종합하여 노드의 최종 상태 값을 결정함.



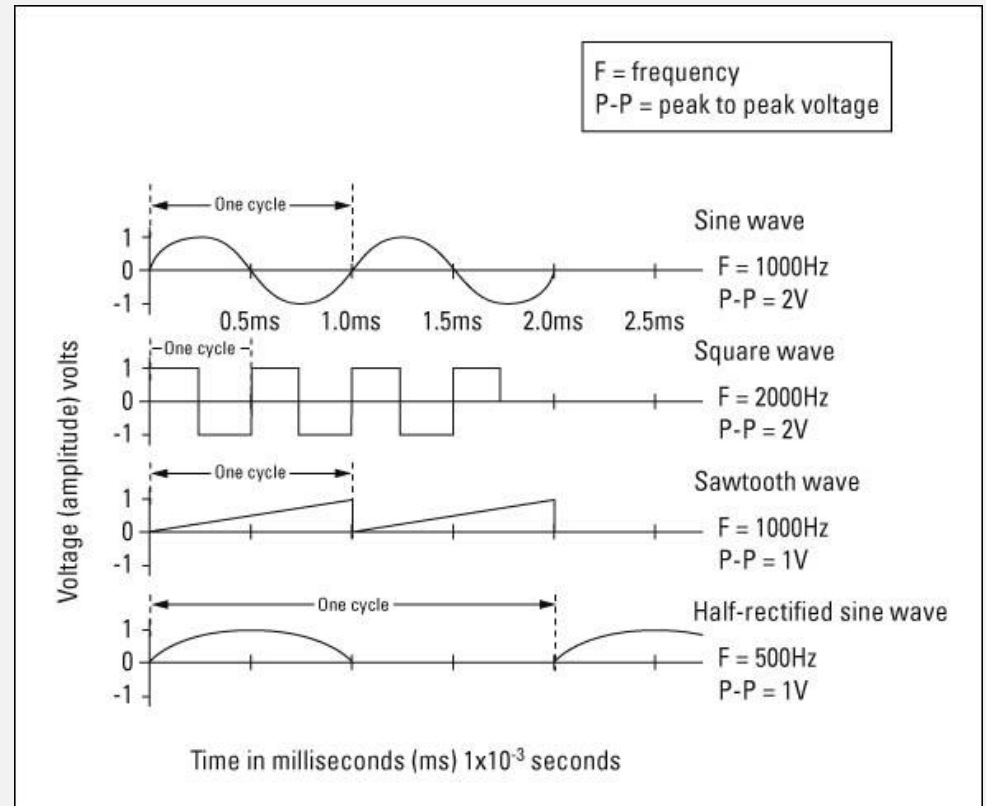
진폭과 주파수

■ 진폭(Amplitude)

- 파형의 크기

■ 주파수(Frequency)

- 초당 특정 파형이 반복되는 횟수. 단위는 Hz
- 가청 주파수: 20 - 20,000 Hz
- 남자: 20 - 20,000 Hz
- 여자: 70 - 30,000 Hz



디지털 사운드와 합성 사운드

■ 디지털 사운드(Digital Sound)

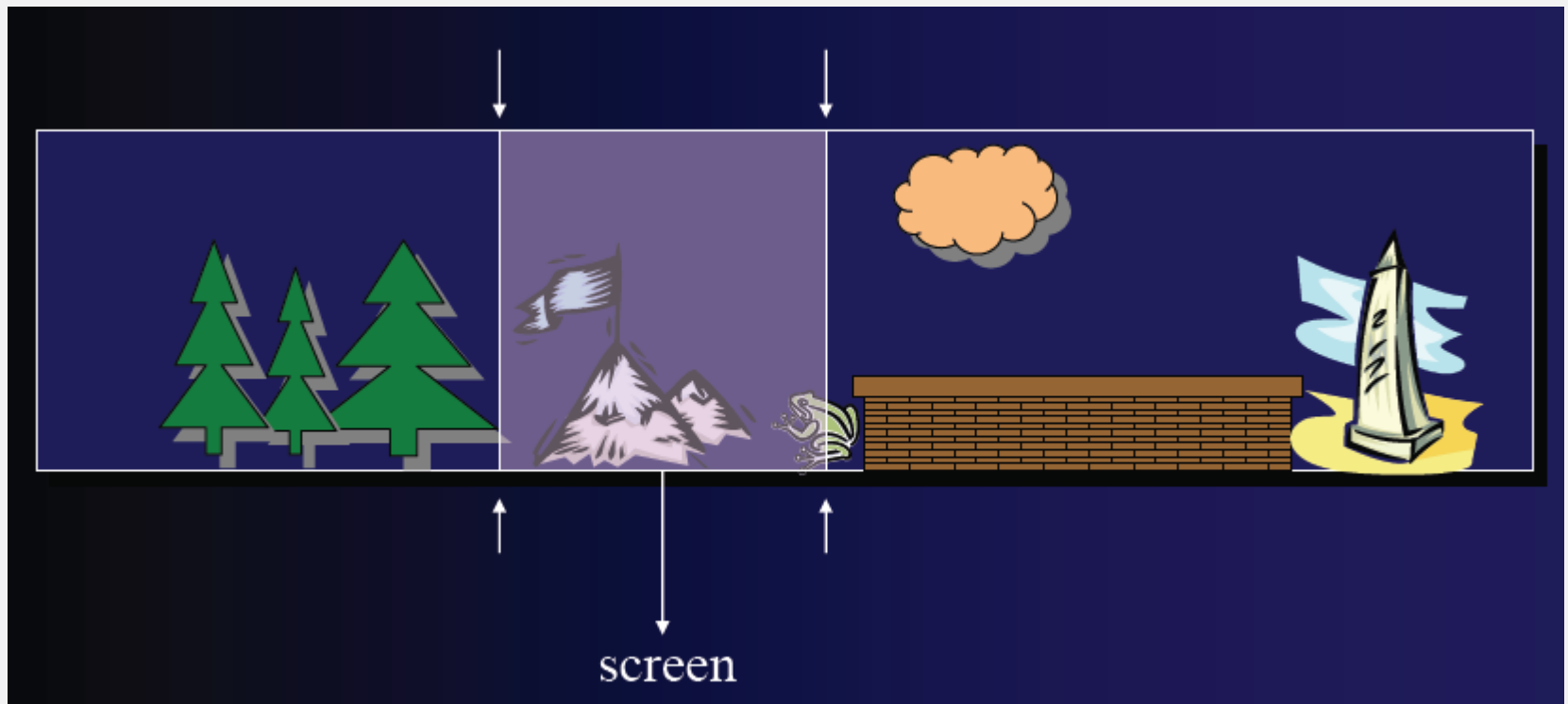
- 소리의 직접적인 녹음
- 효과음등에 사용(폭발, ...)

■ 합성 사운드

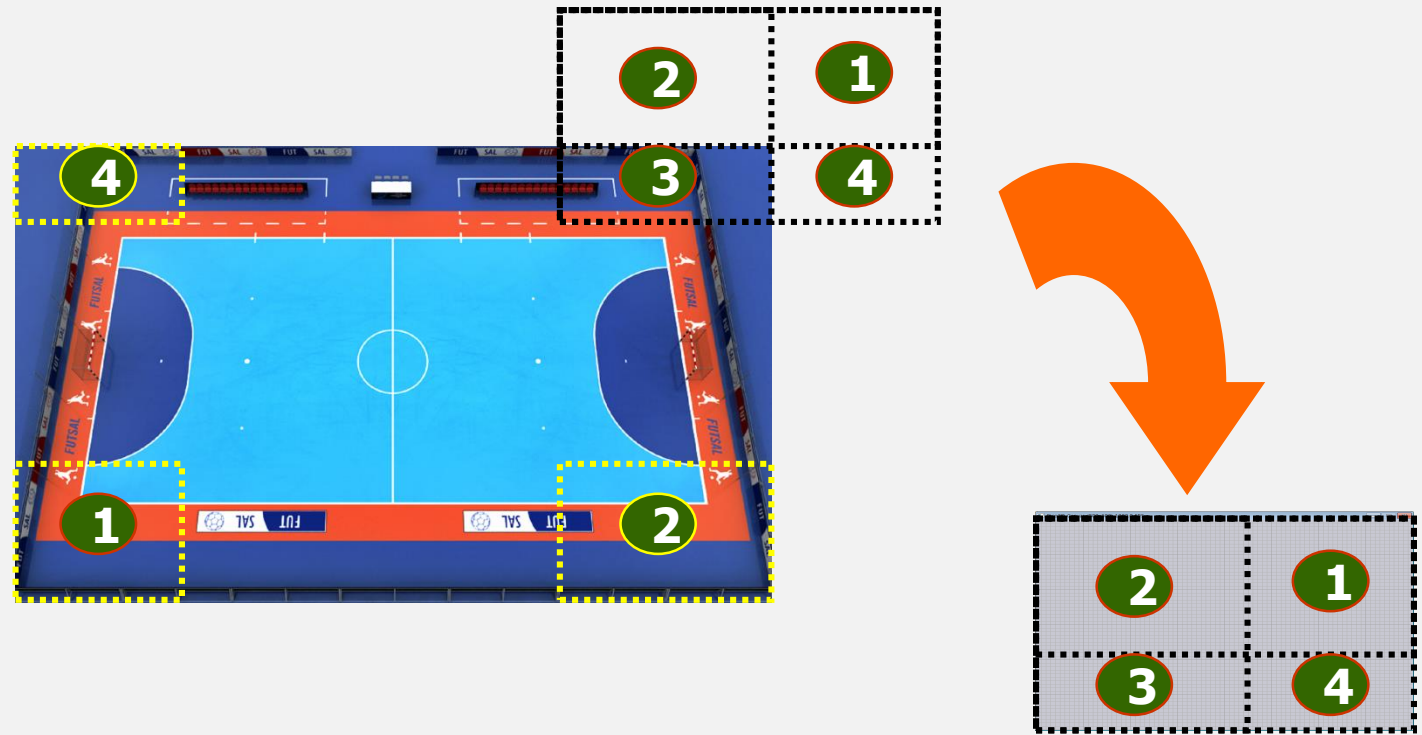
- 알고리즘과 톤 발생기에 의하여 합성된 소리.
- 주로 음악의 재생에 사용.

스크롤링(Scrolling)

- 그림이나 이미지의 일부분을 디스플레이 화면 위에서 상하좌우로 움직이면서 나타내는 기법.
- 슈팅 게임, 고전 RPG 게임에서 주로 사용됨.
- 캐릭터를 직접 이동하는 대신, 배경을 이동함으로써, 캐릭터의 이동을 표현.



상하좌우 무한스크롤링



시차(視差) 스크롤링(Parallax Scrolling)

- 물체와 눈의 거리에 따라, 물체의 이동속도가 달라보이는 효과를 이용하여, 3차원 배경을 흉내내는 기법.
- 1982년 “Moon Patrol”이라는 게임에서 세계 최초로 사용됨.



- 밤하늘, 뒷산, 앞산의 스크롤링 속도를 다르게 함으로써, 3차원적인 깊이 효과를 구현.

직렬화(Serialization)

- 프로그램 내의 객체 데이터를 외부에 저장 또는 보내는 행위.
- 나중에 다시 복구(de-serialization)할 수 있어야 함.
- 게임 플레이 상황을 저장(Save)하고 로드(Load)하는 것도 직렬화로 볼 수 있음.

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)

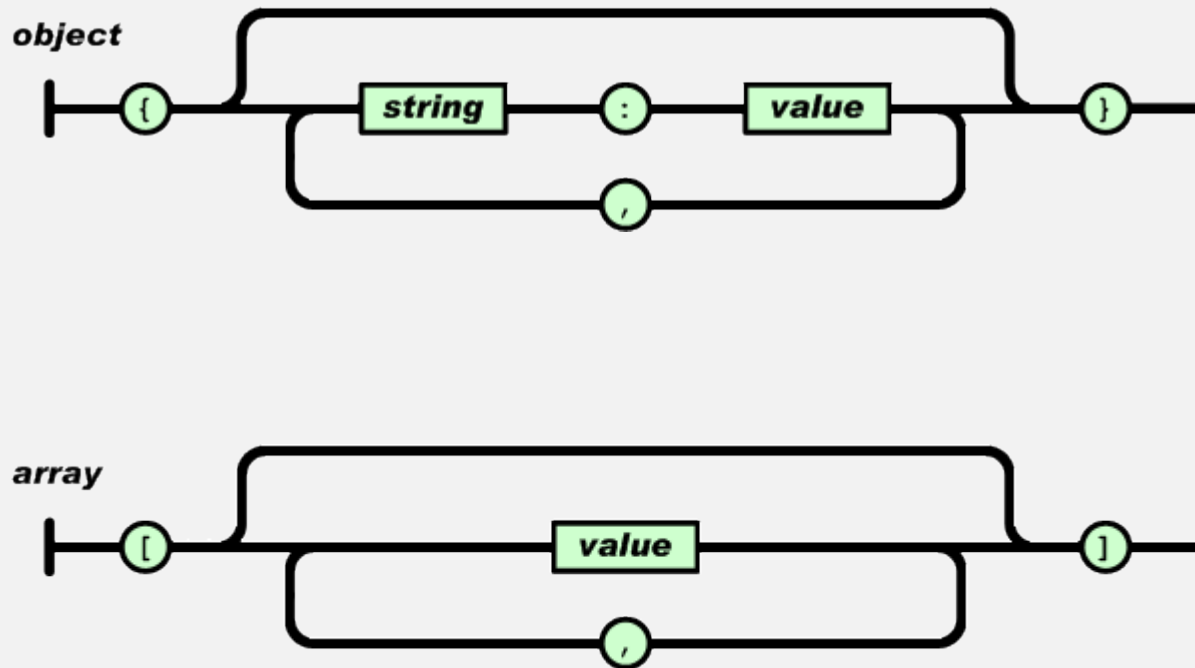
    data_struct["total_spam"]+=messagetype
    data_struct["total_ham"]+=(1-messagetype)
    #K is laplacian smoother
def predict(message,K):
    bag_of_words = data_struct['bag_of_words']
    k=K
    #total messages
    total_messages=data_struct["total_spam"]+data_struct["total_ham"]
    #prior probability 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
- 단, 문자열은 " "을 사용해야 함.



Pickle(<https://docs.python.org/ko/3/library/pickle.html>)

- 파이썬이 제공하는 객체 직렬화 모듈
- 대부분의 파이썬 내부 데이터들을 직렬화할 수 있음.



어떤 것이 피클 되고 역 피클 될 수 있을까요?

다음 형을 피클 할 수 있습니다:

- `None`, `True` 와 `False`
- 정수, 실수, 복소수
- 문자열, 바이트열, 바이트 배열(`bytearray`)
- 피클 가능한 객체만 포함하는 튜플, 리스트, 집합과 딕셔너리
- 모듈의 최상위 수준에서 정의된 함수 (`lambda` 가 아니라 `def` 를 사용하는)
- 모듈의 최상위 수준에서 정의된 내장 함수
- 모듈의 최상위 수준에서 정의된 클래스
- 그런 클래스의 인스턴스 중에서 `__dict__` 나 `__getstate__()` 를 호출한 결과가 피클 가능한 것들 (자세한 내용은 클래스 인스턴스 피클링 절을 참조하세요).

시게루 미야모토의 게임 제작 원칙

- Start with a simple concept

- "running, climbing, jumping"

- Design around the computer's limitations

- Character wears dungarees(작업복) so easier to see arms move
- Wears a hat because don't have to have hair
- Has mustache because couldn't draw nose and mouth

- Minimize the player's confusion

- What do to should be clear without consulting a manual

- The importance of play testing

- Incorporate a smooth learning curve

- Accommodate all skill levels



Sid Meier의 게임 설계

- Sid Meier의 훌륭한 게임 (Great Game)의 정의:

- "A great game is a series of interesting and meaningful choices made by the player in pursuit of a clear and compelling goal."

- Player should have fun, not designer, programmer, or computer

- Begin your game with a great first few minutes

- The inverted pyramid of decision making(have few decisions to deal with first, and then let them multiply until the player is totally engrossed)

- Put the player in his dreams, where he or she is the hero

프로그래밍 비법

- **프로그램은 항상 실행가능한 상태여야 한다.**

- → 실행 가능한 상태에서, 그날의 프로그래밍을 중단해야 한다.

- **최소의 기능단위만을 작성하고 컴파일한다.**

- 한줄, 두줄, 세줄... 정도만...

- **새로운 기능을 넣을 때는 직접 넣지 않고, 미리 조그마한 테스트 프로젝트를 만들어서 테스트 한다.**

- 테스트 결과, 패스하면,,, 메인 프로젝트에 담는다.

- **프로그램의 실행 상황을 표시할 수 있는 창을 만든다.**