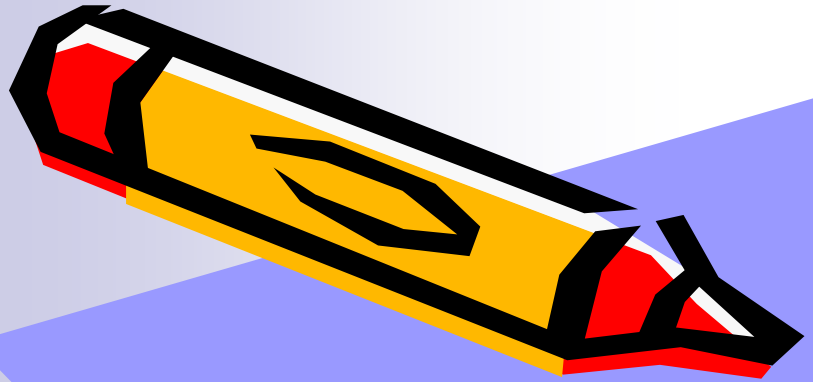


2D 게임 프로그래밍

제14강 강의 정리

이대현
한국산업기술대학교

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

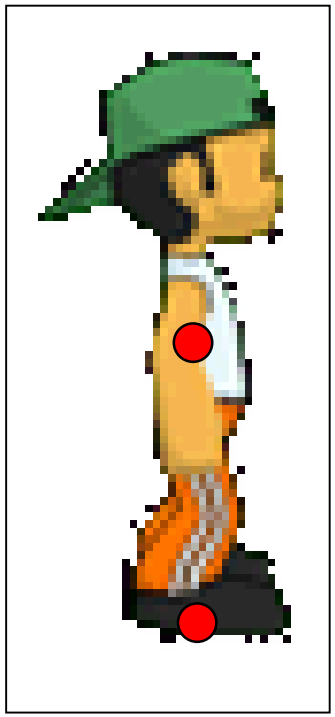
2D 게임 개발 접근법

- 플랫폼 종속적 방법
  - Direct X
  - OpenGL
  - Simple Frame Buffer

- 플랫폼 독립적 방법, Cross Platform
  - Unity3D
  - COCOS2D
  - 그 외의 범용 2D 렌더링 라이브러리

  - SDL !!!!

피봇(Pivot)



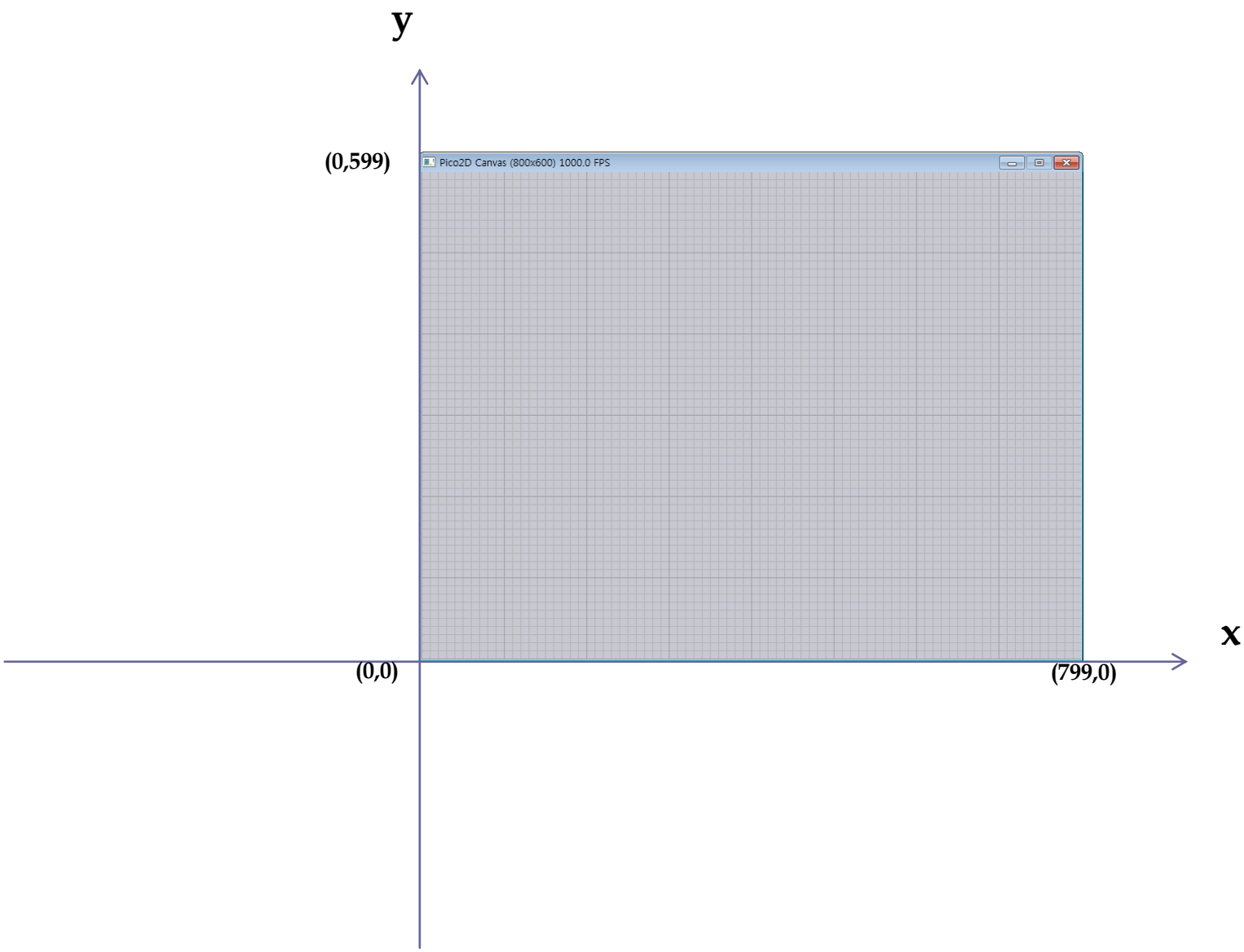
여기가 피봇입니다.

이 점을 피봇으로 삼기도 합니다

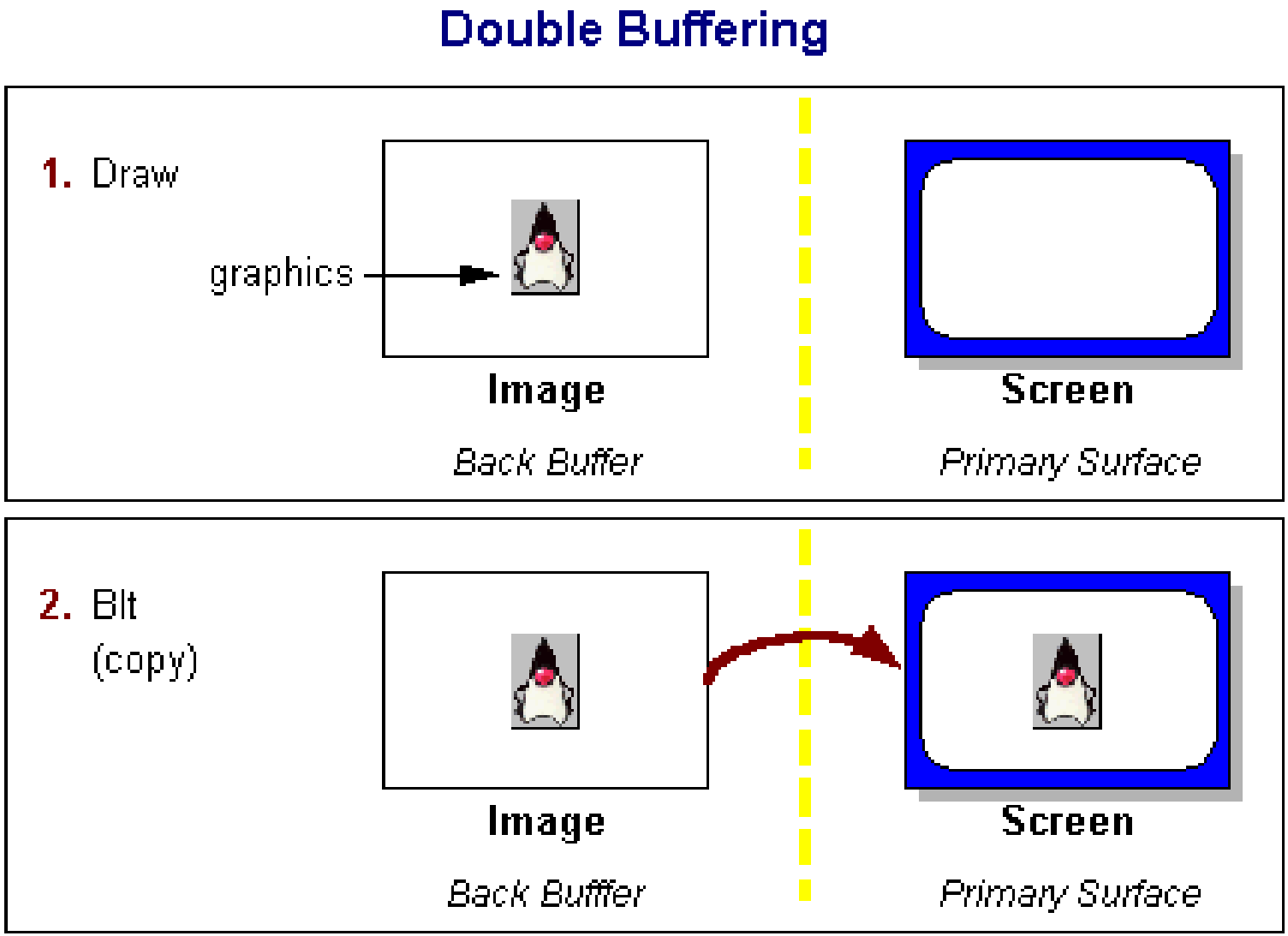
게임 개발 시작에 앞서 결정해야 할 것들

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

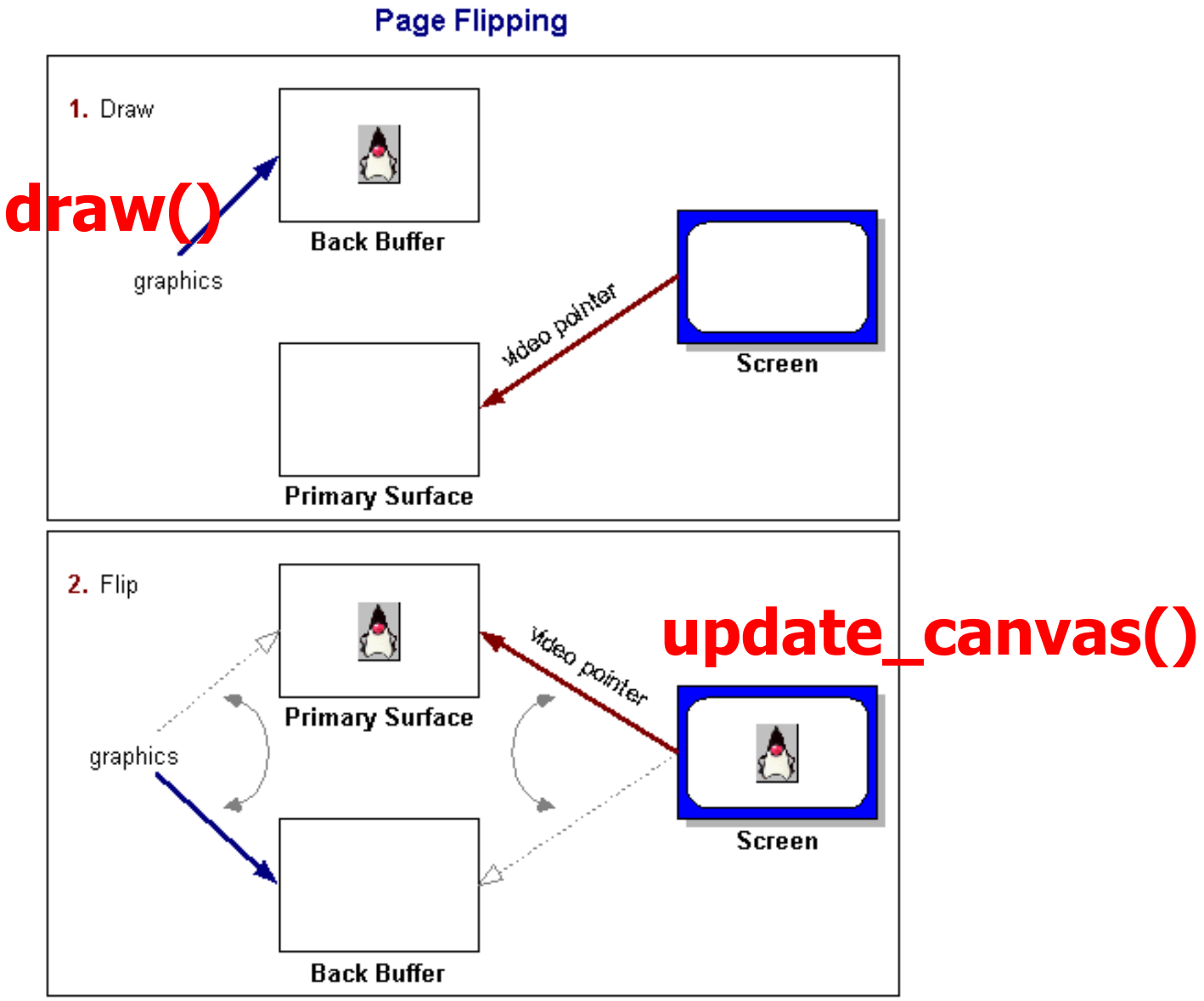
캔버스의 좌표계



더블 버퍼링(Double Buffering)



페이지 플리핑(Page Flipping)



스프라이트(Sprite)

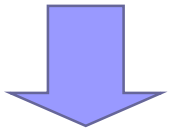
- 스프라이트란?
  - 게임 장면안에서 보여지는 이미지 또는 애니매이션되는 오브젝트
  - 2D 게임에서는 게임의 모든 캐릭터들과 이동하는 물체들을 표현하는 데 사용됨.
  - 3D 게임에서는 2D로 표현될 수 있는 각종 오브젝트에 사용됨.
    - 불, 연기, 작은 물체들, UI 표시 등등.

Metal Slug 3

키보드 및 마우스 입력 처리 과정

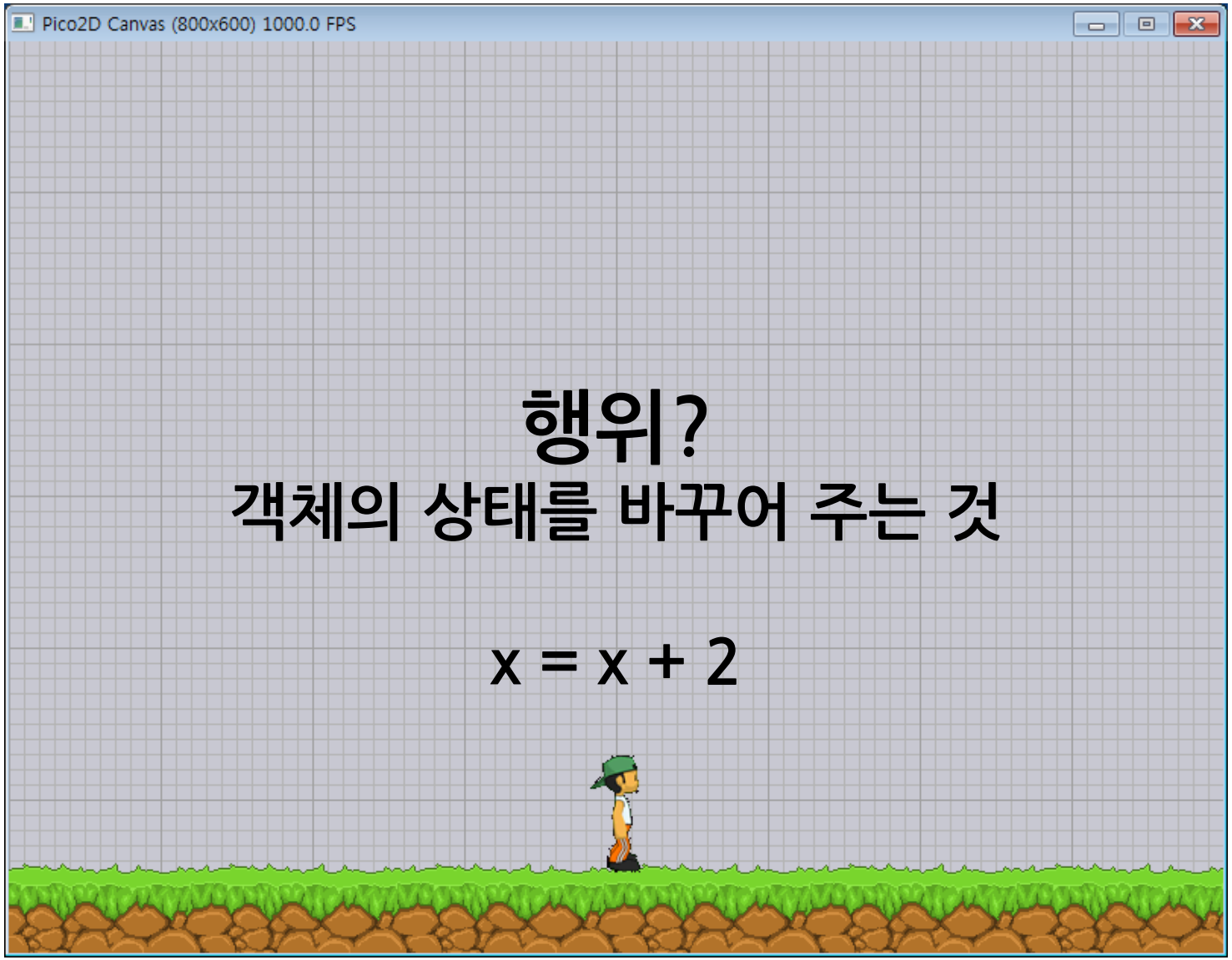
Step1: 입력 이벤트들을 폴링한다.(get_events())

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

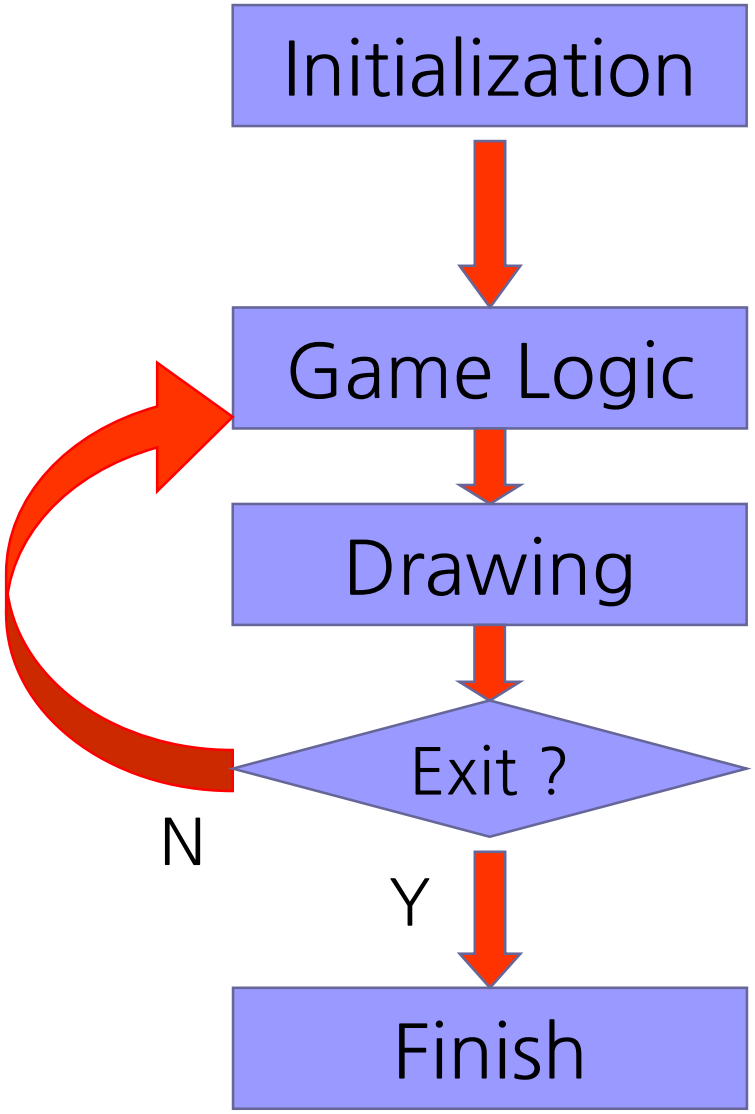


Step3: 실제 입력값을 구한다.(event.key 또는 event.x, event.y 등 을 이용)

속성 + 행위 = 객체



게임 루프



게임 상태(Game State)의 이해 (1)

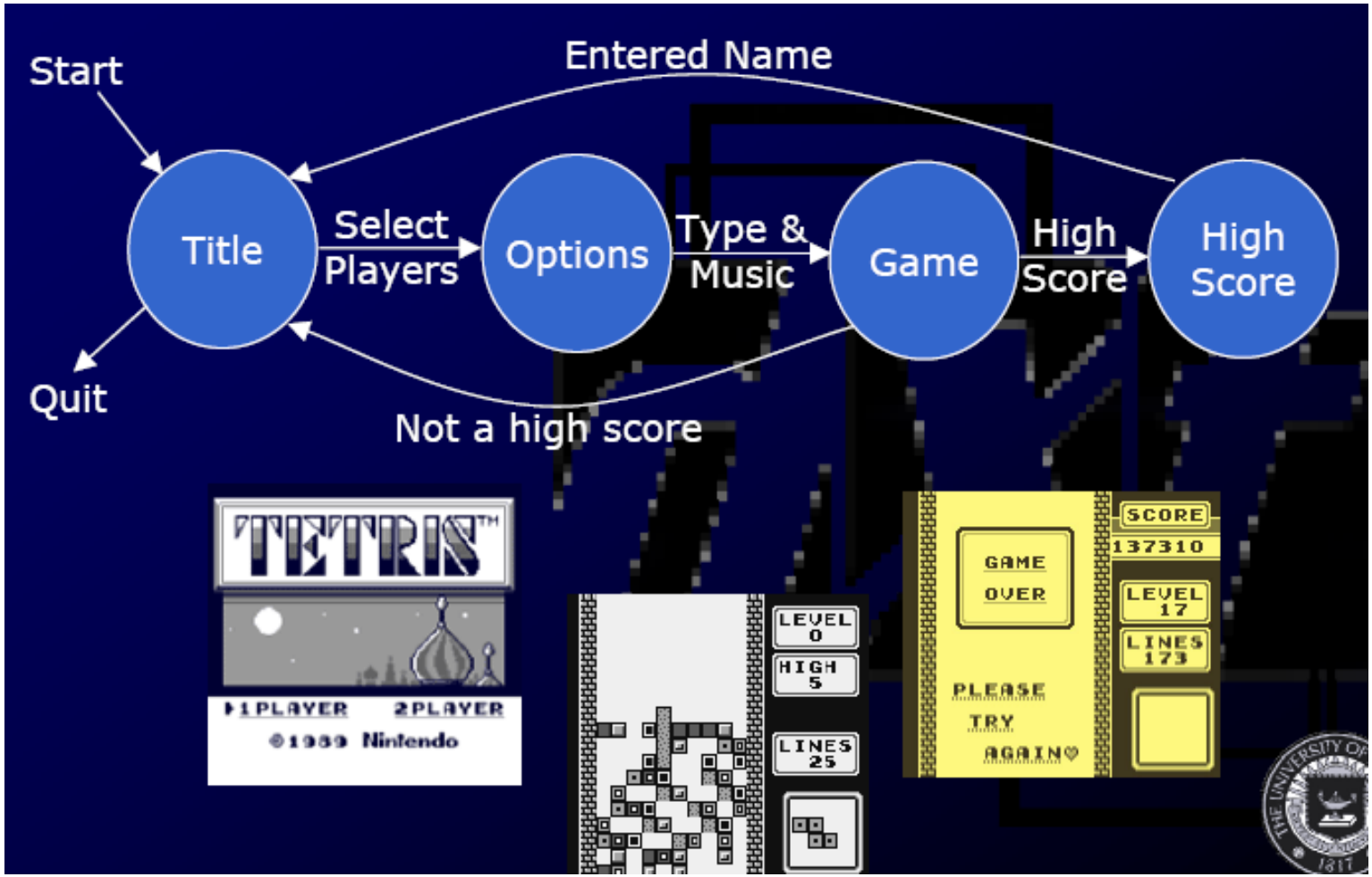
- 게임 상태란?
  - 게임 프로그램 실행 중의 어떤 특정 위치(또는 모드).
  - 사용자 입력(키보드 또는 마우스 입력)에 대한 대응 방식은 게임의 상태에 따라 달라짐.

- 맵 선택 상태.
- 방향키는 맵 선택을 처리.

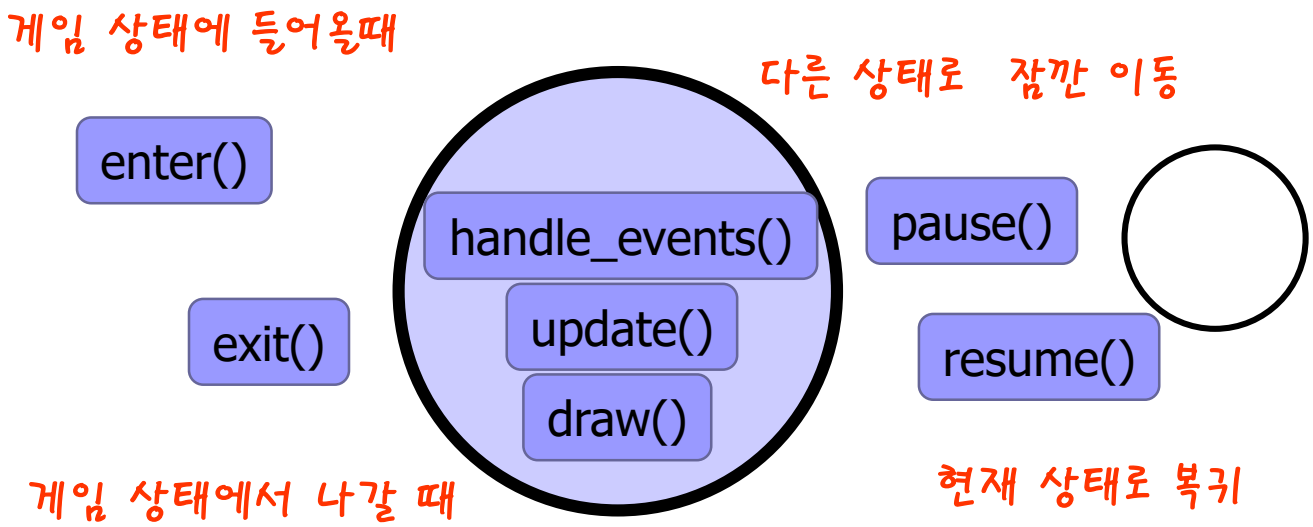
- 게임 메인 플레이 상태.
- 방향키는 캐릭터의 이동을 처리.

게임 상태(Game State)의 이해 (2)

- 게임 프로그램은 게임 상태의 집합으로 구현됨.
  - 예) 테트리스 게임



게임 상태의 구현



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 나옴.

push_state(state):
게임 상태를 state로 변화. 이전 게임 상태는 남아 있음.

pop_state(): 이전 게임 상태로 복귀

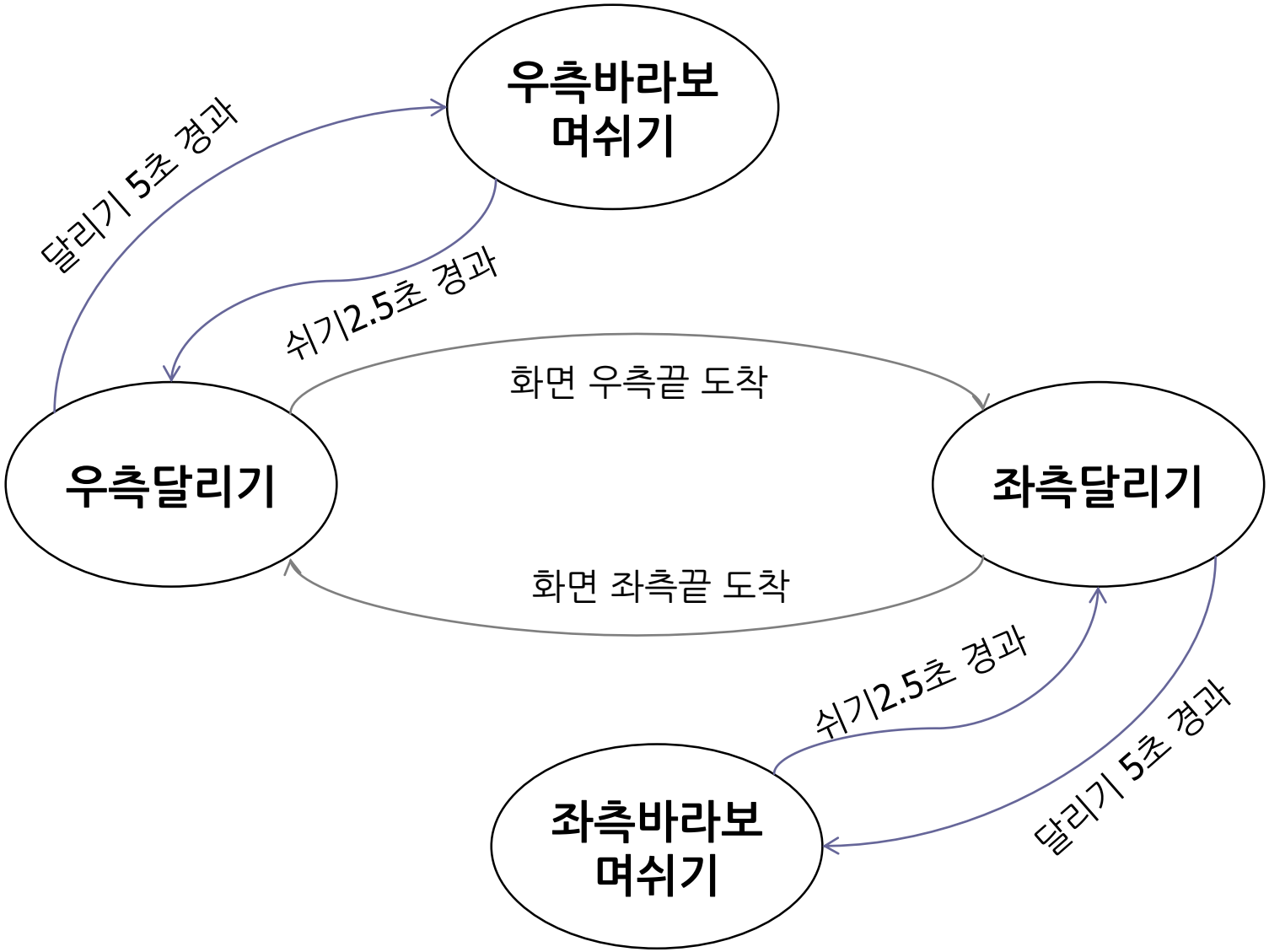
quit(): 게임을 중단

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.

인공 지능 - FSM으로 구현한다



흔히 하는 “무식한 방법”: 하드코딩(Hard Coding)

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

문제점은?

왜 무식한가?

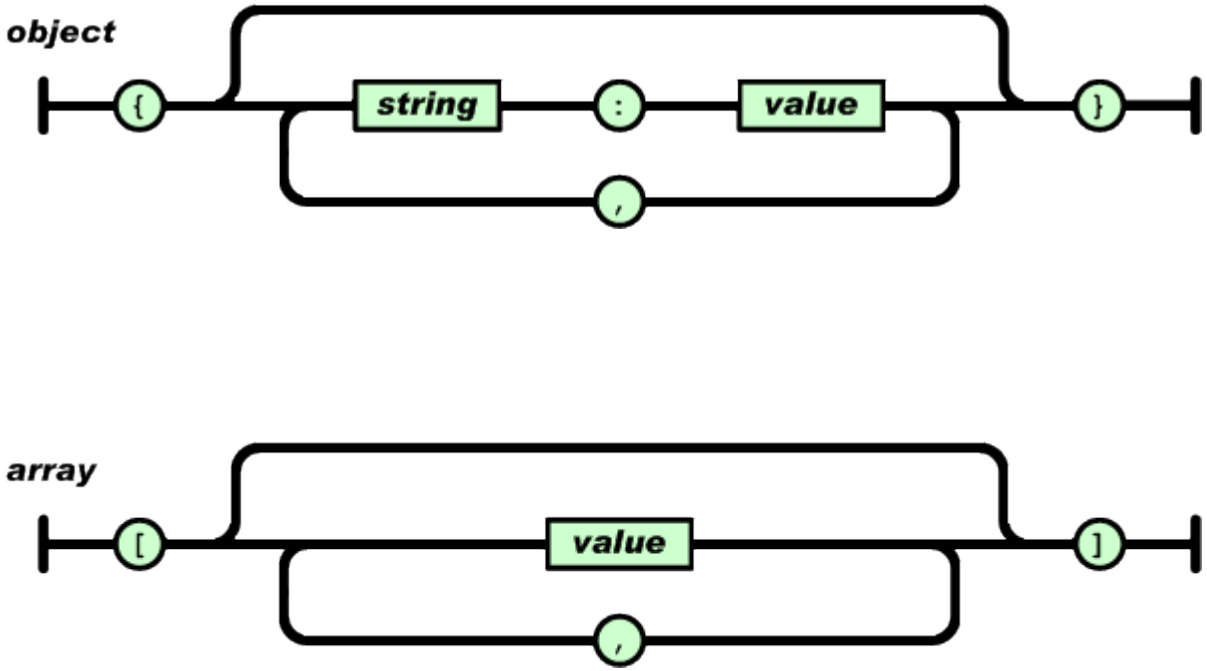
소프트 코딩(Soft Coding)

```
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)
```

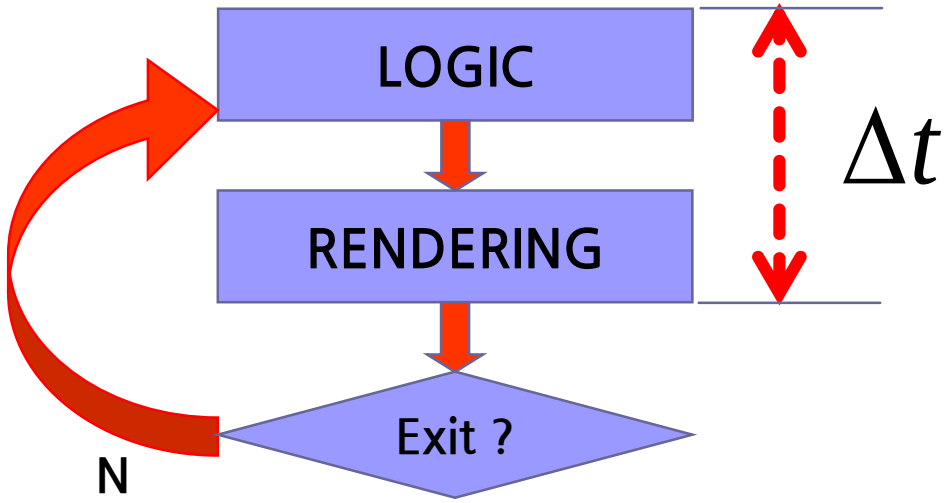
JSON(Java Script Object Notation)

- 객체를 교환(저장 및 전송 등)하기 위한 텍스트 형식 표준
- 파이썬의 리스트와 딕셔너리와 거의 동일
- 단, 문자열은 “ ”을 사용해야 함.



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

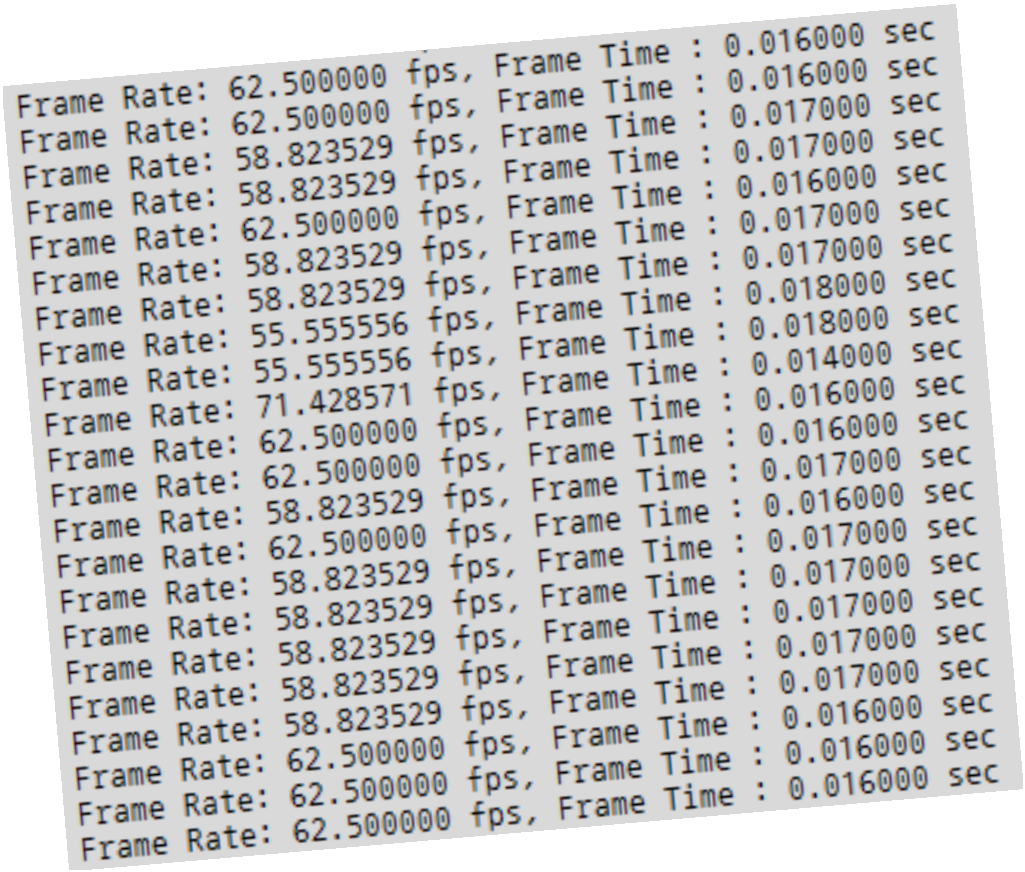
Frame per sec = 1 / Frame time

- 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.



아주 아주 아주 근사한 방법

- 게임 객체들의 운동에 “시간”의 개념을 도입

거리 = 경과시간 * 속도

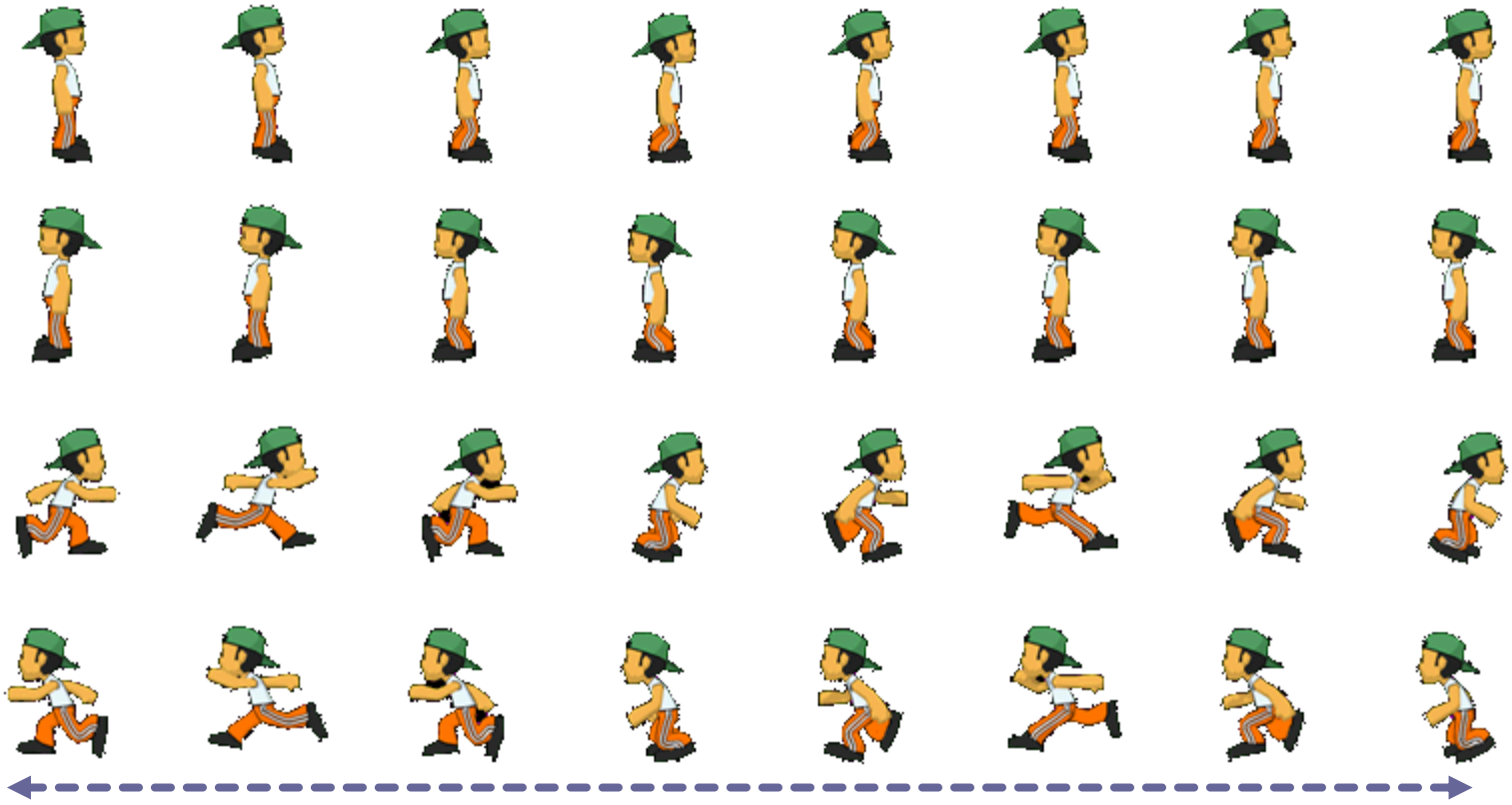
위치 = 초기 위치 + 거리

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(N2) 알고리즘

충돌 처리(Collision Handling)

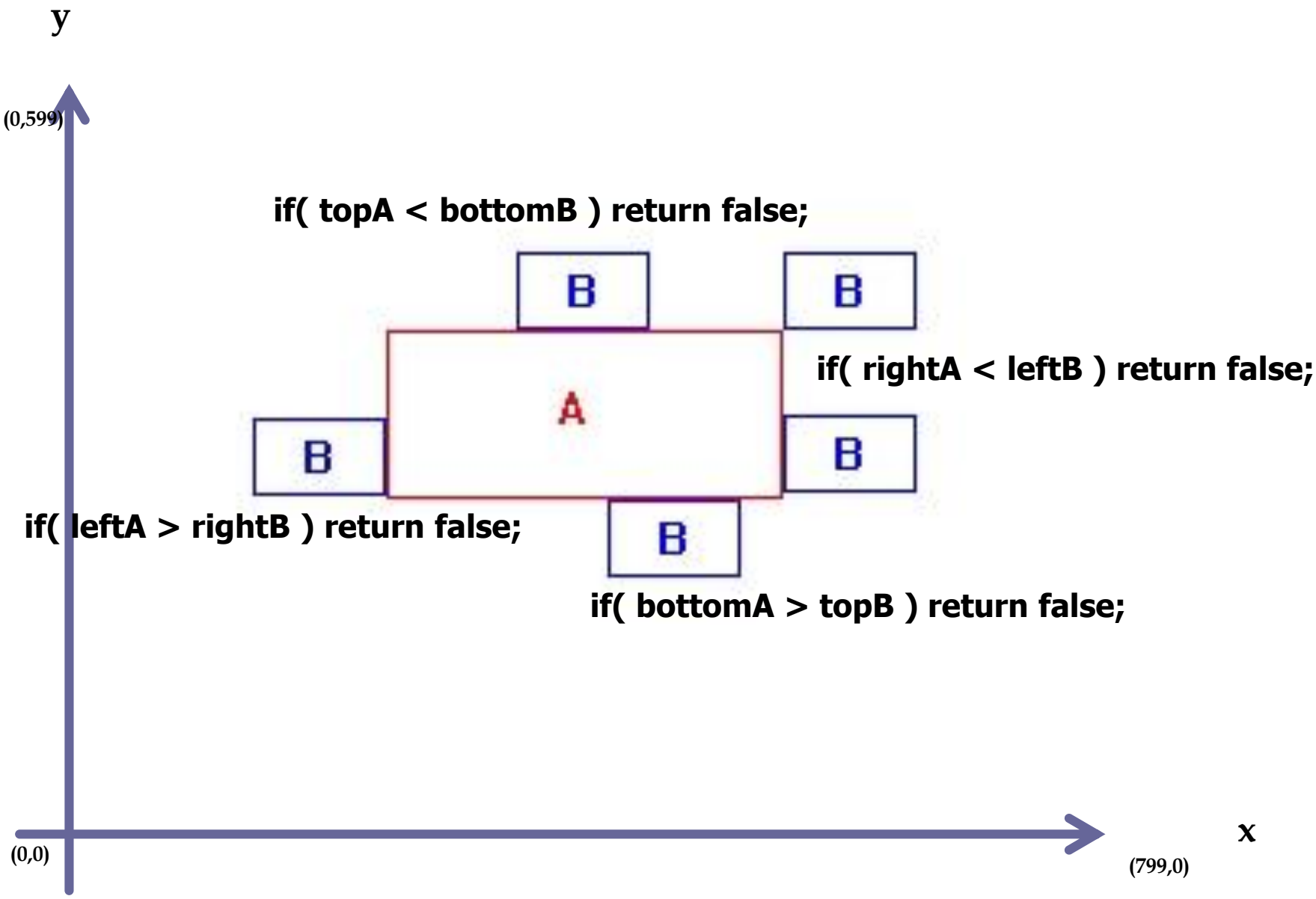
- 충돌이 확인 된 후, 이후 어떻게 할것인가?
  - 충돌 응답(Collision Response)

- 캐릭터와 아이템의 충돌에 대한 처리는??

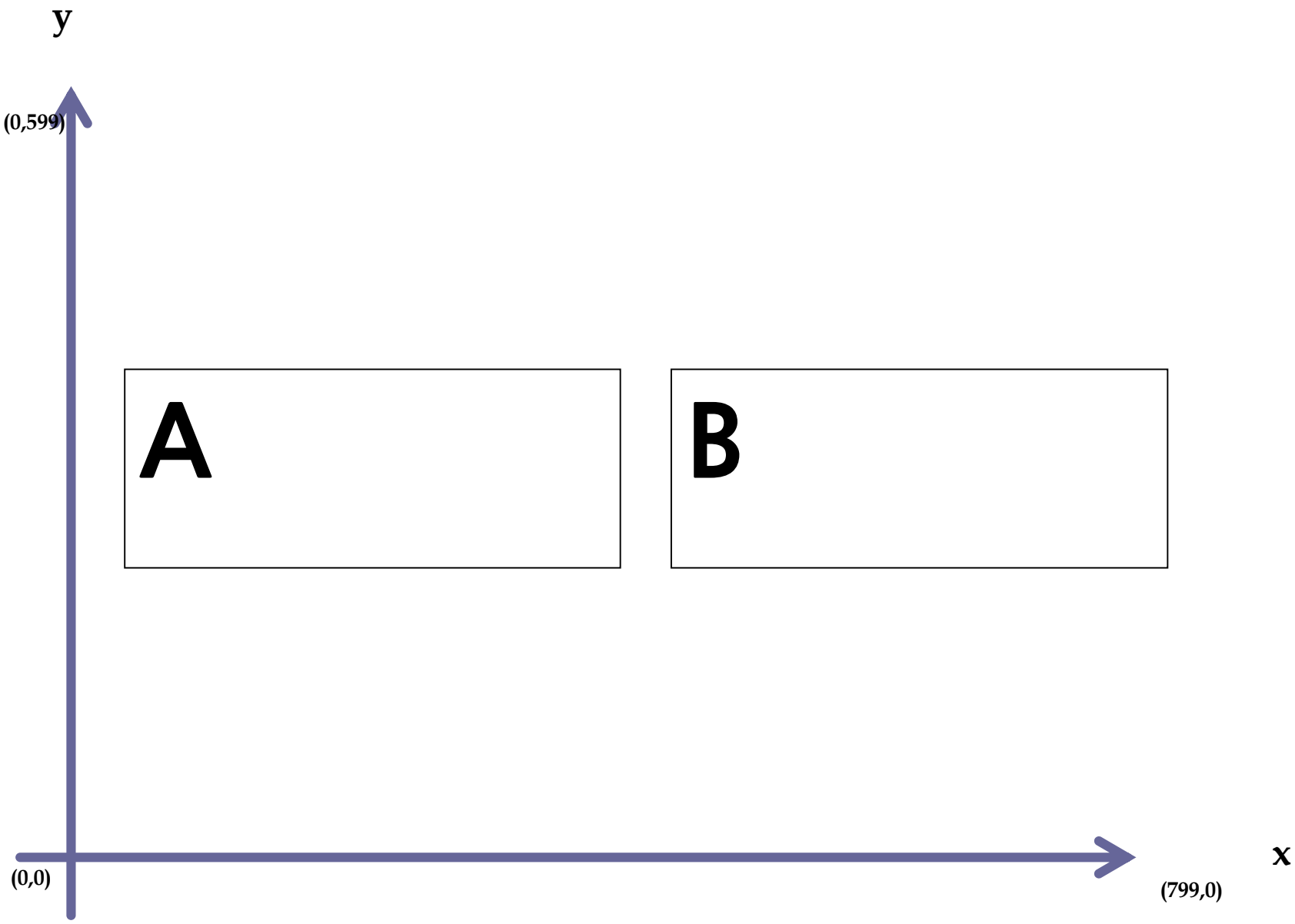
- 바닥에 떨어지는 적군 NPC가 바닥과 충돌하면??

- 사선으로 움직이는 캐릭터가 맵의 벽과 충돌하면?

사각형과 사각형

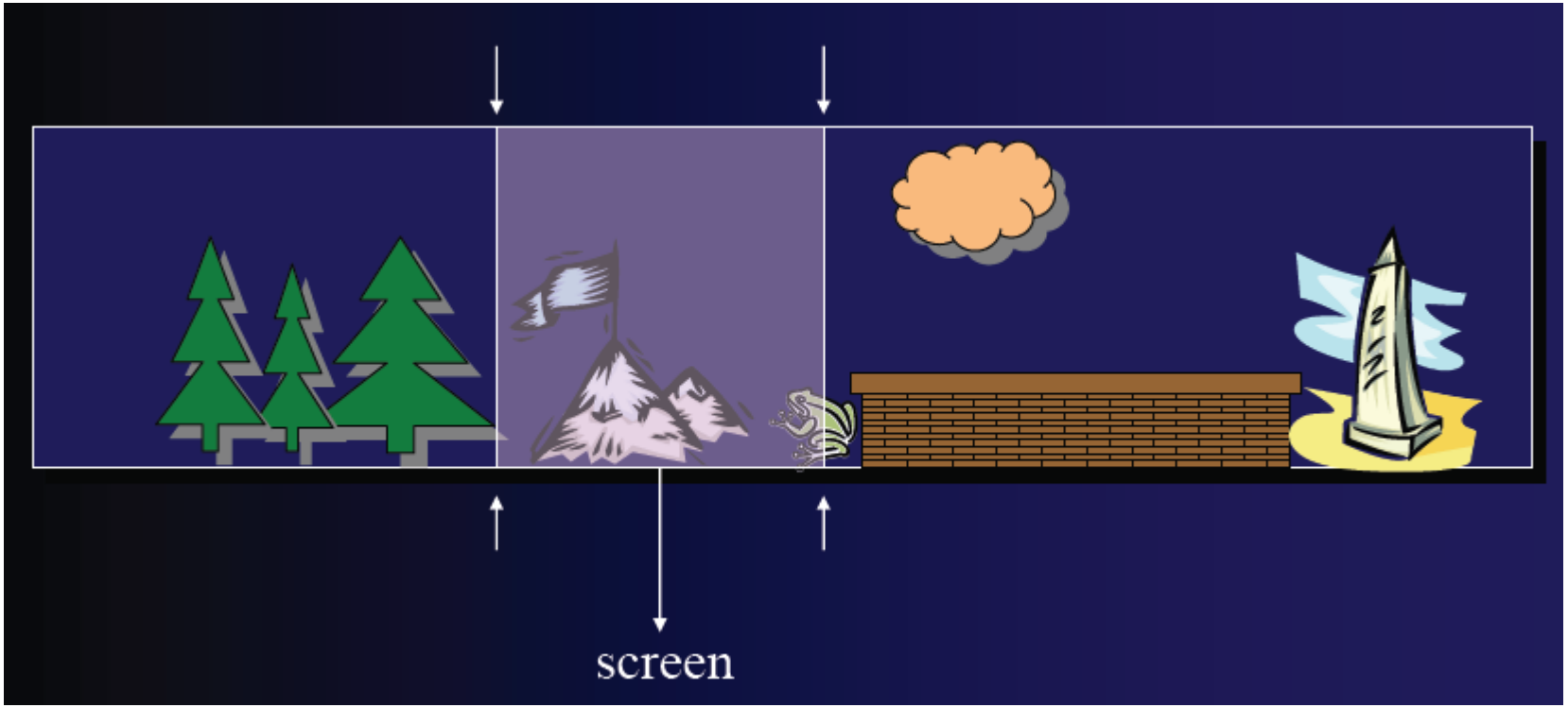


사각형과 사각형



스크롤링(Scrolling)

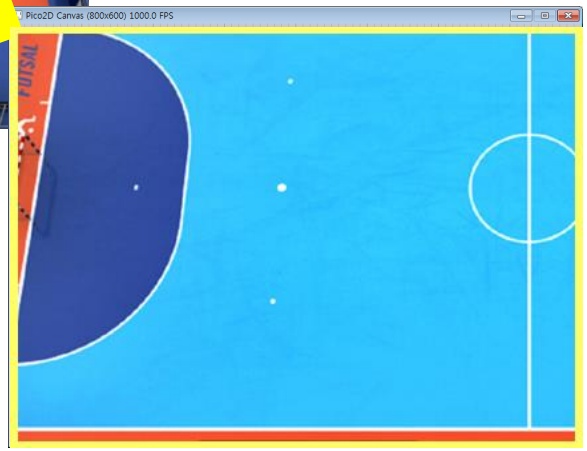
- 그림이나 이미지의 일부분을 디스플레이 화면 위에서 상하좌우로 움직이면서 나타내는 기법.
- 슈팅 게임, 고전 RPG 게임에서 주로 사용됨.
- 캐릭터를 직접 이동하는 대신, 배경을 이동함으로써, 캐릭터의 이동을 표현.



카메라 윈도우를 이용한 스크롤링

(x-canvas_width//2, y-canvas_height//2)

clip_draw_to_origin



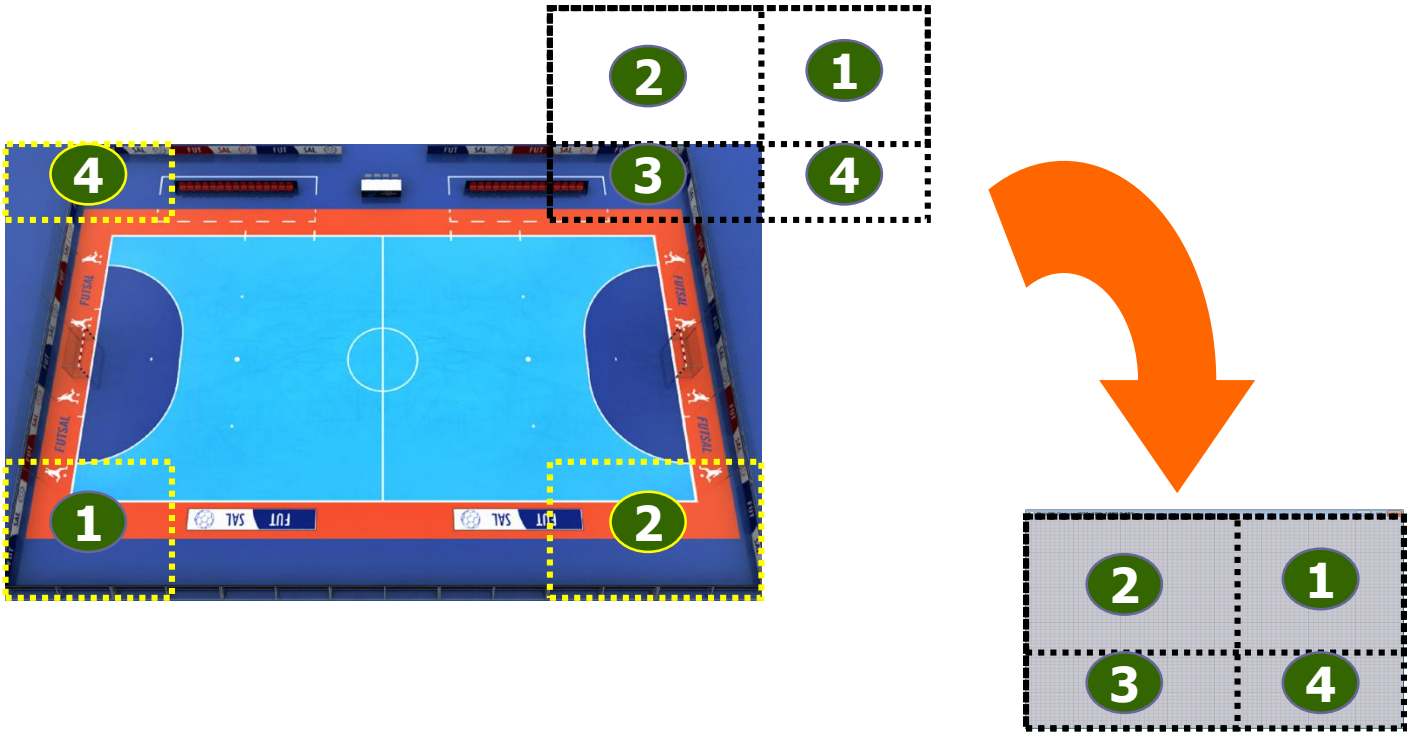
#3. 윈도우 영역을 클리핑하여, 캔버스에 그림.

Clip_draw_to_origin : 피봇 맨왼쪽 맨아래쪽 기준

#4. 플레이어를 캔버스에 그림.

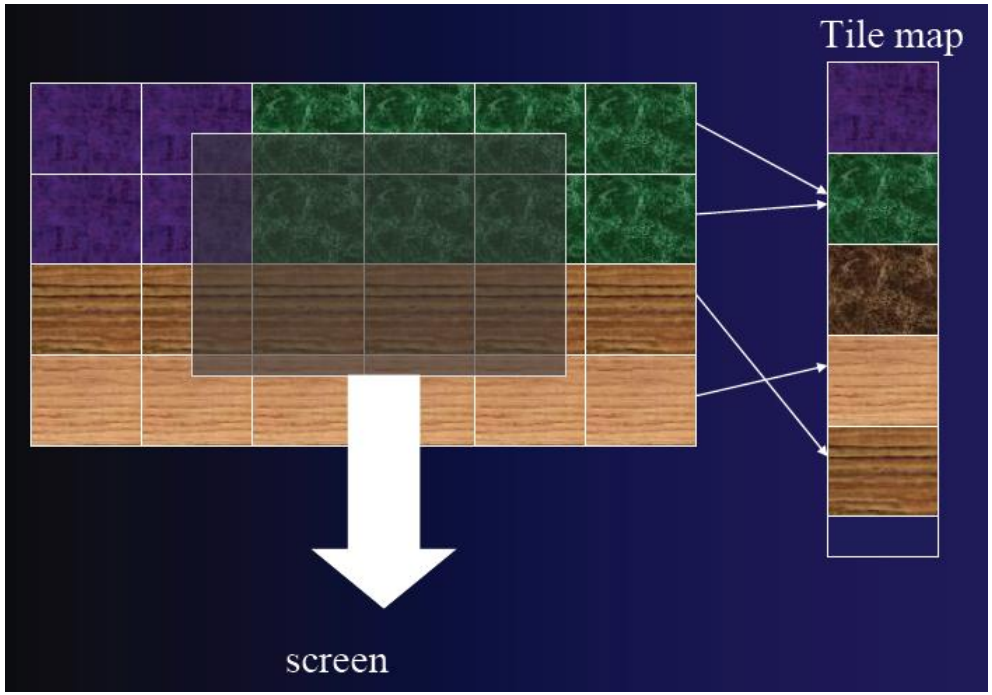
(canvas_width//2, canvas_height//2)

상하좌우 무한스크롤링



타일링(Tiling)

- 게임의 배경을 단일 이미지로 구성하지 않고, 여러 개의 타일을 이용하여 구성하는 방법. 거의 모든 RPG 게임에 사용됨.



- 장점
  - 맵의 크기를 대폭 줄일 수 있다.
  - 사용자의 구미에 맞게 맵을 편집하여 사용할 수 있다.
- 단점
  - 정교한 배경을 만드는데 많은 시간이 소요되고 어렵다.

타일링 구현

- 타일 이미지를 이용한 tile set 의 준비

desert_tileset.json

```
{
 "type": "tileset",
 "name": "Desert",
 "image": "desert_tiles.png",
 "imageheight": 192,
 "imagewidth": 256,
 "tileheight": 32,
 "tilewidth": 32,
 "columns": 8,
 "tilecount": 48
}
```

- 타일맵 렌더링
  - 1차원의 layer의 data를 미리 2차원 배열로 준비할 필요.(위 아래 뒤집기 필요)
  - 경계 부분을 클리핑하는 대신, 전체 타일을 그대로 그리고, 대신 오프셋만큼 쉬프트 해 줌.

desert_map.json

```
{
 "orientation":"orthogonal",
 "width":40,
 "height":40,
 "tilewidth":32,
 "tileheight":32,
 "tilesets":[ { "firstgid":1,"source":"desert_tileset.json"} ],
 "layers":[
        {
         "name":"Ground",
         "width":40,
         "height":40,
         "data":[30, 30, 30, 30, 30, ……],
         "opacity":1,
         "type":"tilelayer",
         "visible":true,
         "x":0,
         "y":0
        }]
}
```

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.

디지털 사운드의 샘플링 주파수와 데이터 비트수

- 샘플링 속도
  - 디지털 사운드를 기록할 때, 초당 몇번의 샘플링을 하는가?
  - 샘플링되는 사운드 주파수보다 2배 이상으로 샘플링을 해야함.

- 샘플 당 비트수(bits ber samples)
  - 8 비트 샘플: 256개의 진폭 크기. 게임 효과음 등에는 충분.
  - 16 비트 샘플: 65536개의 진폭 크기. 음악 등에 사용.

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 테스트한다.
  - 테스트 결과, 패스하면,,, 메인 프로젝트에 담는다.

- 프로그램의 실행 상황을 표시할 수 있는 창을 만든다.