

Lecture #12. 시간

2D 게임 프로그래밍

이대현 교수



한국공학대학교
TECH UNIVERSITY OF KOREA

학습 내용

- 프레임 시간
- 프레임 속도
- 프레임 시간을 활용한 객체 운동의 동기화

시간의 개념이 없는 코드의 문제점?

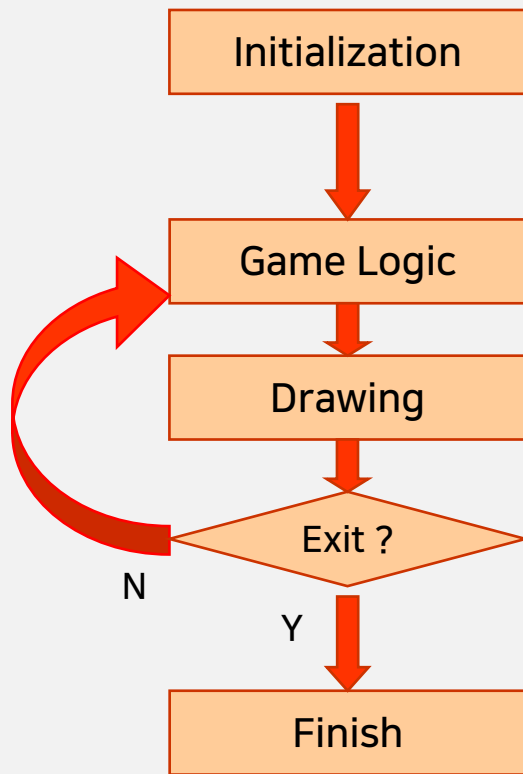
```
while running:  
  
    # Game Logic  
    player.x += 10  
  
    # Game Rendering  
    player.draw()
```

초창기의 CPU 종속적 게임

- 시간 개념이 없음.
- 그냥 물체의 움직임을 pixel 값의 변화로 표시
- 문제점은?
 - CPU 성능에 따라, 물체의 움직이는 속도가 달라짐.
 - single player 게임에서는 문제가 아닐수도...

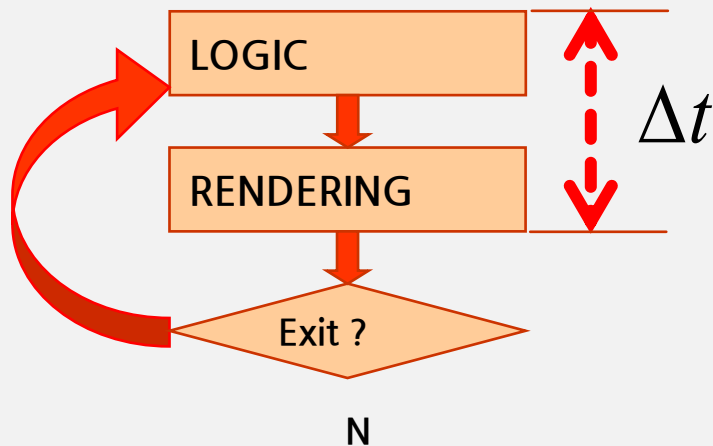
프레임(Frame)

- 특정 시점에서 씬(장면)을 화면에 그린 한장의 그림.
- 드로잉(렌더링)의 결과물
 - 드로잉(렌더링)이 끝나는 시점에 만들어짐.
- 스크린샷



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

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



프레임 시간과 프레임 속도 측정

game_framework.py



```
import time
frame_time = 0.0

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

    global frame_time
    frame_time = 0.0
    current_time = time.time()
    while running:
        stack[-1].handle_events()
        stack[-1].update()
        stack[-1].draw()
        frame_time = time.time() - current_time
        frame_rate = 1.0 / frame_time
        current_time += frame_time
        print(f'Frame Time: {frame_time}, Frame Rate: {frame_rate}')
```

Time 모듈

- **Unix epoc time**

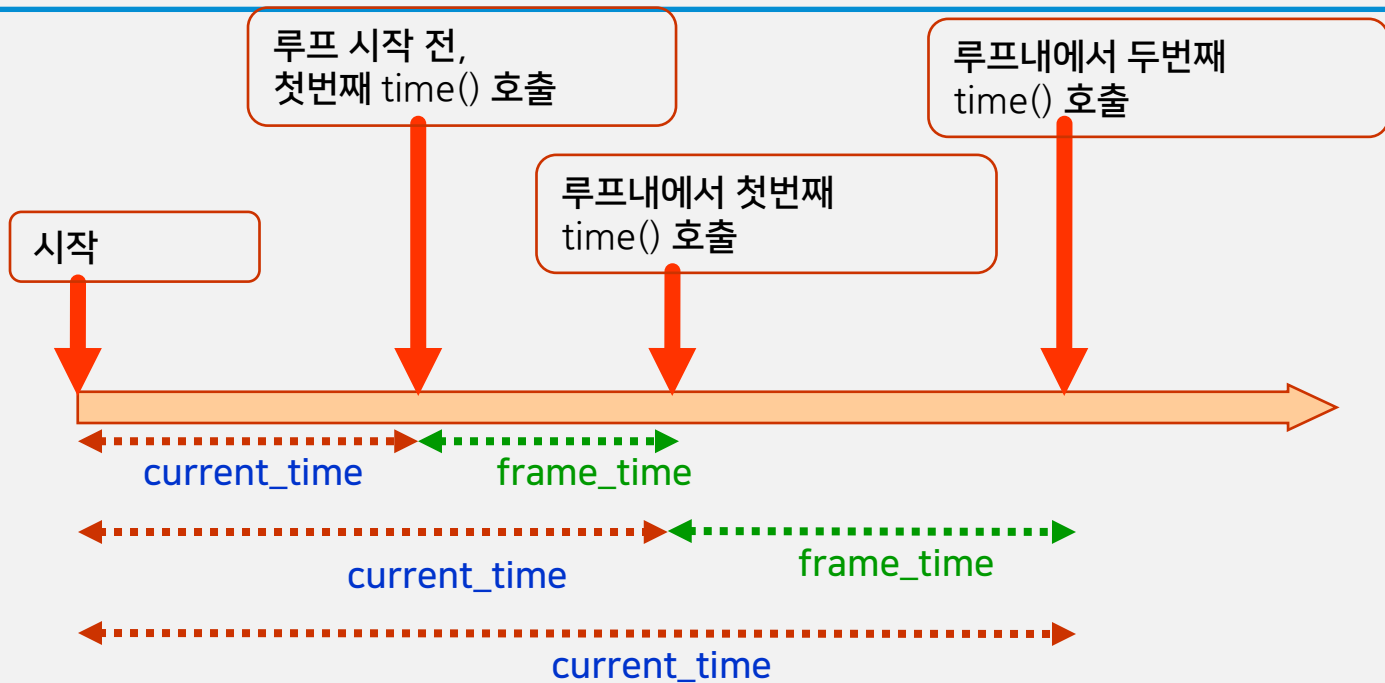
- 1970년 1월 1일 AM 0시 기준.
- 많은 프로그램들의 기준 시작 시간으로 활용됨.

- **time.time()**

- Unix epoc time에서 시작되어 경과된 시간(초)

```
>>> import time
>>> time.time()
1621468685.019336
```

프레임 시간의 계산



프레임 시간: $\text{frame_time} = \text{time}() - \text{current_time}$

프레임 속도: $\text{frame_rate} = 1.0 / \text{frame_time}$

프레임 속도와 프레임 시간

```
Frame Time : 0.002991 sec, Frame Rate: 334.287399 fps
Frame Time : 0.002006 sec, Frame Rate: 498.431848 fps
Frame Time : 0.003978 sec, Frame Rate: 251.381720 fps
Frame Time : 0.001994 sec, Frame Rate: 501.531030 fps
Frame Time : 0.001995 sec, Frame Rate: 501.351183 fps
Frame Time : 0.002993 sec, Frame Rate: 334.074393 fps
Frame Time : 0.001995 sec, Frame Rate: 501.171466 fps
Frame Time : 0.001994 sec, Frame Rate: 501.411118 fps
Frame Time : 0.003989 sec, Frame Rate: 250.720545 fps
Frame Time : 0.001994 sec, Frame Rate: 501.531030 fps
Frame Time : 0.002992 sec, Frame Rate: 334.234122 fps
Frame Time : 0.002993 sec, Frame Rate: 334.127619 fps
Frame Time : 0.001995 sec, Frame Rate: 501.231358 fps
Frame Time : 0.002993 sec, Frame Rate: 334.101004 fps
Frame Time : 0.002989 sec, Frame Rate: 334.607419 fps
Frame Time : 0.002995 sec, Frame Rate: 333.941401 fps
Frame Time : 0.002991 sec, Frame Rate: 334.367347 fps
Frame Time : 0.002992 sec, Frame Rate: 334.180862 fps
Frame Time : 0.002992 sec, Frame Rate: 334.260759 fps
Frame Time : 0.002993 sec, Frame Rate: 334.154238 fps
Frame Time : 0.002991 sec, Frame Rate: 334.367347 fps
Frame Time : 0.003989 sec, Frame Rate: 250.660111 fps
```

균일하지 않다? 왜?

문제점은?

■ 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(1600, 600, sync=True)`
- 60fps 로 고정할 수 있음.
- 문제점은???

```
Frame Time : 0.016958 sec, Frame Rate: 58.970053 fps
Frame Time : 0.016488 sec, Frame Rate: 60.649016 fps
Frame Time : 0.016920 sec, Frame Rate: 59.101341 fps
Frame Time : 0.016955 sec, Frame Rate: 58.978345 fps
Frame Time : 0.016955 sec, Frame Rate: 58.978345 fps
Frame Time : 0.016024 sec, Frame Rate: 62.406880 fps
Frame Time : 0.016958 sec, Frame Rate: 58.970053 fps
Frame Time : 0.016257 sec, Frame Rate: 61.512686 fps
Frame Time : 0.016387 sec, Frame Rate: 61.022260 fps
Frame Time : 0.017181 sec, Frame Rate: 58.203294 fps
Frame Time : 0.015956 sec, Frame Rate: 62.673580 fps
Frame Time : 0.017683 sec, Frame Rate: 56.552159 fps
Frame Time : 0.015957 sec, Frame Rate: 62.669834 fps
Frame Time : 0.016971 sec, Frame Rate: 58.922833 fps
Frame Time : 0.017946 sec, Frame Rate: 55.721968 fps
Frame Time : 0.014961 sec, Frame Rate: 66.839368 fps
Frame Time : 0.016957 sec, Frame Rate: 58.972541 fps
Frame Time : 0.016952 sec, Frame Rate: 58.991617 fps
Frame Time : 0.016956 sec, Frame Rate: 58.976687 fps
Frame Time : 0.015956 sec, Frame Rate: 62.673580 fps
```

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)

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

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



$$\begin{aligned} \text{distance} &= \text{frame_time} * \text{velocity} \\ x &= x + \text{distance} \end{aligned}$$



프레임 시간에 따른
객체 이동 구현

boy.py (1) - 속력 계산



```
# Boy Run Speed
PIXEL_PER_METER = (10.0 / 0.3) # 10 pixel 30 cm
RUN_SPEED_KMPH = 20.0 # Km / Hour
RUN_SPEED_MPM = (RUN_SPEED_KMPH * 1000.0 / 60.0)
RUN_SPEED_MPS = (RUN_SPEED_MPM / 60.0)
RUN_SPEED_PPS = (RUN_SPEED_MPS * PIXEL_PER_METER)
```

boy.py (2) – 위치 계산



```
class Run:

    def do(self):
        self.boy.frame = (self.boy.frame + 1) % 8
        self.boy.x += self.boy.dir * RUN_SPEED_PPS * game_framework.frame_time
```

boy.py (3) - 폰트 로딩 및 렌더링



```
class Boy:
```

```
    def __init__(self):  
        self.font = load_font('ENCR10B.TTF', 16)
```

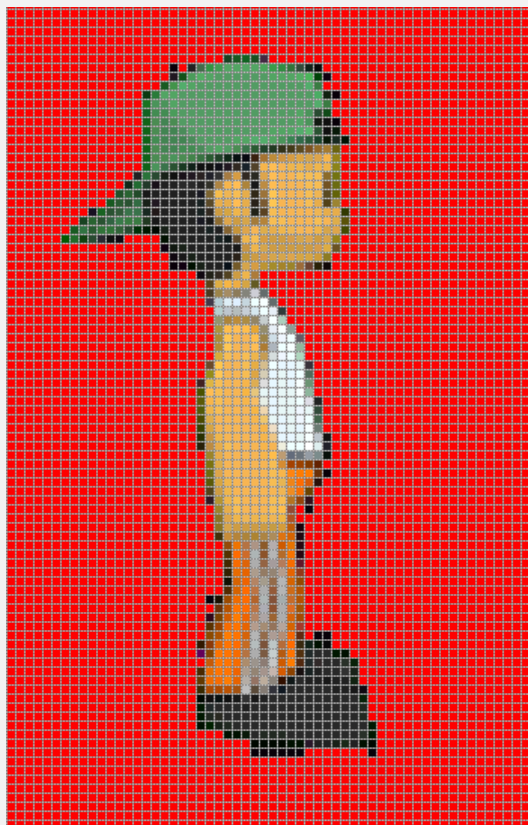
```
    def draw(self):  
        self.cur_state.draw(self)  
        self.font.draw(self.x - 60, self.y + 50, f'(Time: {get_time():.2f})', (255, 255, 0))
```

3M 6M 9M 12M 15M 18M 21M 24M 27M 30M 33M 36M 39M 42M 45M

(x1m): 59.24)



2D 공간의 물리값들을 먼저 결정할 필요



74 pixel = 약 2.2미터

10 pixel = 30 cm

폰트 출력

```
font = load_font(폰트파일, 사이즈)
```

```
font.draw(x, y, 'Your Text', (R,G,B) )
```

r	the red component in the range 0-255
g	the green component in the range 0-255
b	the blue component in the range 0-255

후면 버퍼(back buffer)에 그리기 때문에,
나중에 update_canvas() 할 때, 표시됨.

play_mode.py - 강제적인 프레임 시간 증가



```
def update():  
    game_world.update()  
    delay(0.1)
```



프레임 시간에 따른
액션 프레임의 조절

boy.py (1)



```
# Boy Action Speed
TIME_PER_ACTION = 0.5
ACTION_PER_TIME = 1.0 / TIME_PER_ACTION
FRAMES_PER_ACTION = 8
```

boy.py (2) - frame 증가량 계산



- Idle, Run, Sleep 모두 관련 부분 수정 업데이트 필요.

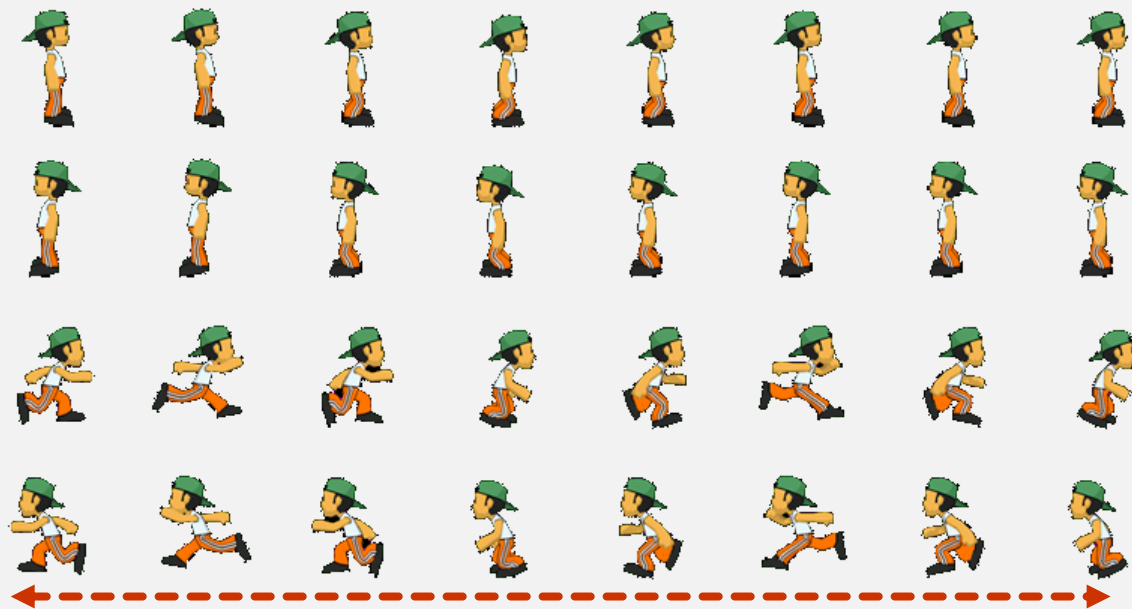
```
class Idle:

    def do(self):
        self.boy.frame = (self.boy.frame + FRAMES_PER_ACTION * ACTION_PER_TIME * game_framework.frame_time) % 8
        if get_time() - self.boy.wait_time > 3:
            self.boy.state_machine.handle_state_event(('TIMEOUT', None))

    def draw(self):
        if self.boy.face_dir == 1: # right
            self.boy.image.clip_draw(int(self.boy.frame) * 100, 300, 100, 100, self.boy.x, self.boy.y)
        else: # face_dir == -1: # left
            self.boy.image.clip_draw(int(self.boy.frame) * 100, 200, 100, 100, self.boy.x, self.boy.y)
```

frame 값은 float, 즉 실수로 계산됨.

clip_draw 함수는 int 로 받기 때문에, 형변환 수행 필요.



$\text{TIME_PER_ACTION} = 0.5$

$\text{ACTION_PER_TIME} = 1.0 / \text{TIME_PER_ACTION} \rightarrow$
 $\text{FRAMES_PER_ACTION} = 8$



공 드로잉 구현

ball.py - 수정



```
PIXEL_PER_METER = (1.0 / 0.03) # 1pixel = 3cm, 1m = 33.33 pixel  
GRAVITY = 9.8 # 중력 가속도 (m/s2)
```

```
class Ball:  
    image = None  
  
    def __init__(self, x = 400, y = 300, throwin_speed = 15, throwin_angle = 45):  
        if Ball.image == None:  
            Ball.image = load_image('ball21x21.png')  
        self.x, self.y = x, y  
        self.xv = throwin_speed * math.cos(math.radians(throwin_angle))  
        self.yv = abs(throwin_speed * math.sin(math.radians(throwin_angle)))  
  
    def draw(self):  
        self.image.draw(self.x, self.y)  
  
    def update(self):  
        # y 축 속도에 중력 가속도 적용  
        self.yv -= GRAVITY * game_framework.frame_time # m/s  
  
        # 위치 업데이트  
        self.x += self.xv * game_framework.frame_time * PIXEL_PER_METER  
        self.y += self.yv * game_framework.frame_time * PIXEL_PER_METER  
  
        # 지면 충돌 처리  
        if self.y < 60:  
            game_world.remove_object(self)
```