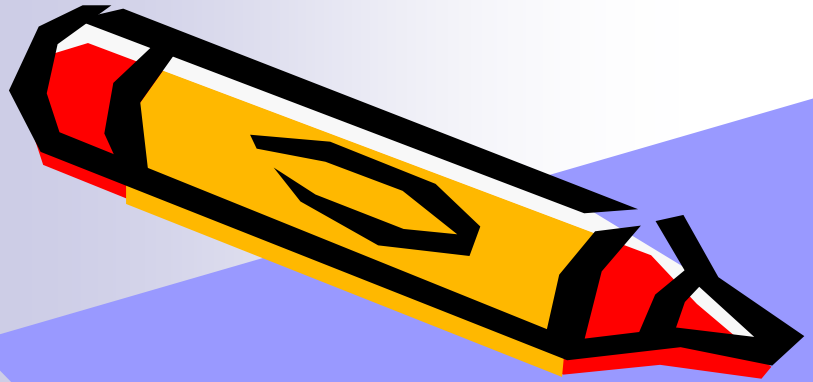


2D 게임 프로그래밍

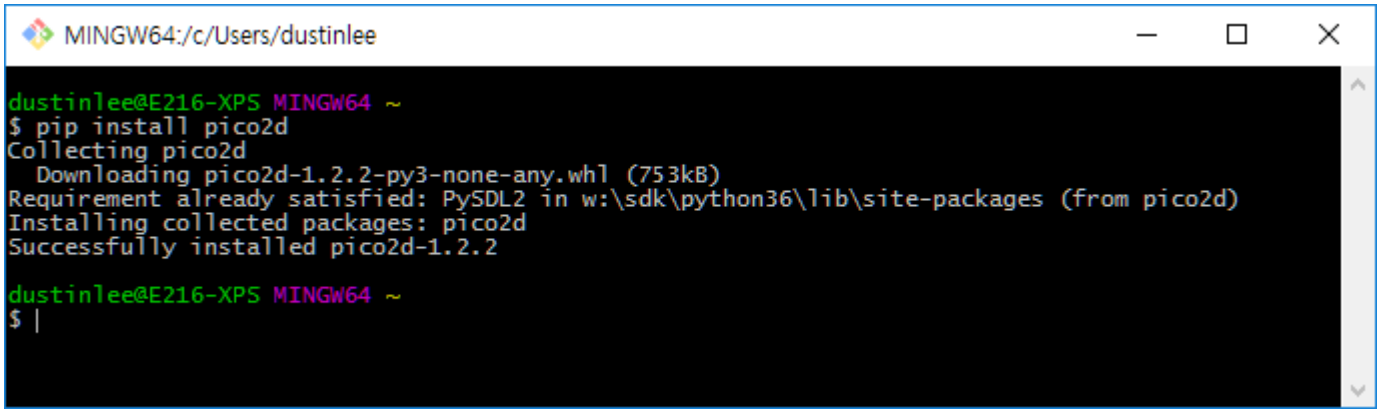
# 제8강 시간

이대현
한국산업기술대학교

# Pico2d 업데이트 (1.2.2)

- 기본 Pico2d 제거


- 콘솔 터미널 창에서
  - pip install pico2d

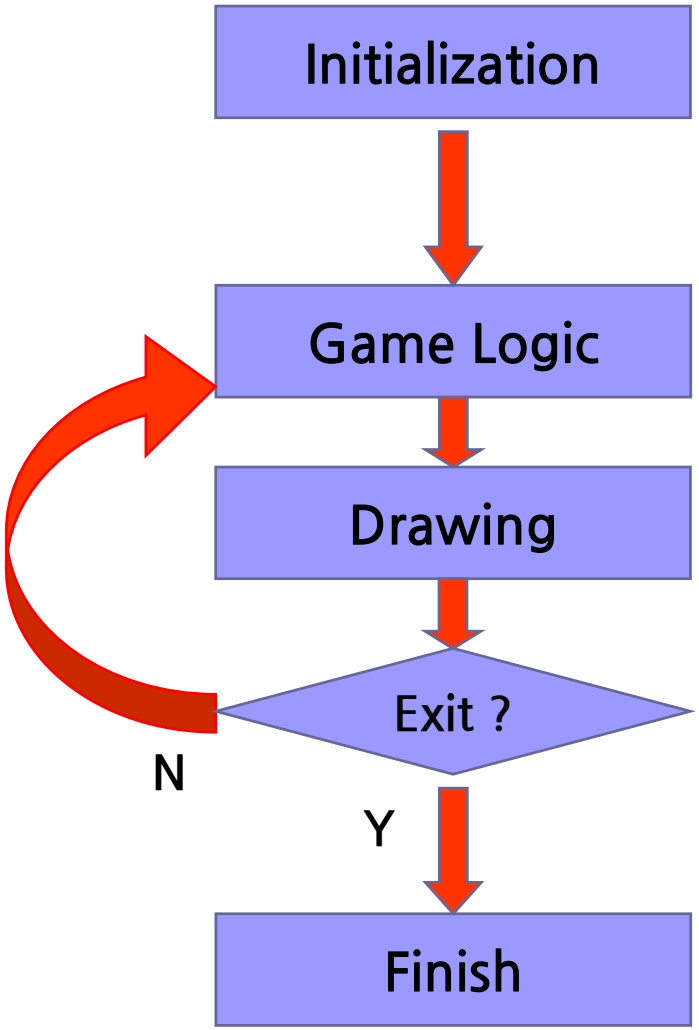


# 학습 내용

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

# 프레임(Frame)

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



# 시간의 개념이 없는 코드의 문제점?

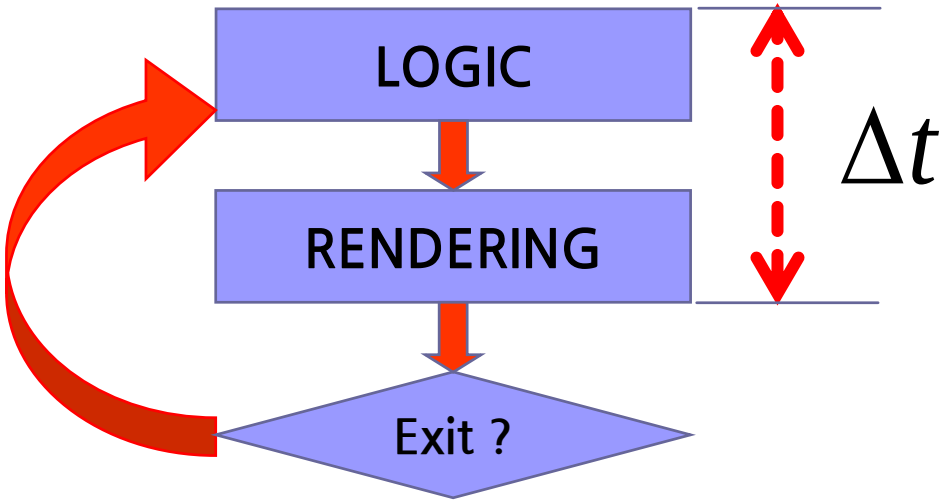
```
while running:

    # 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 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**

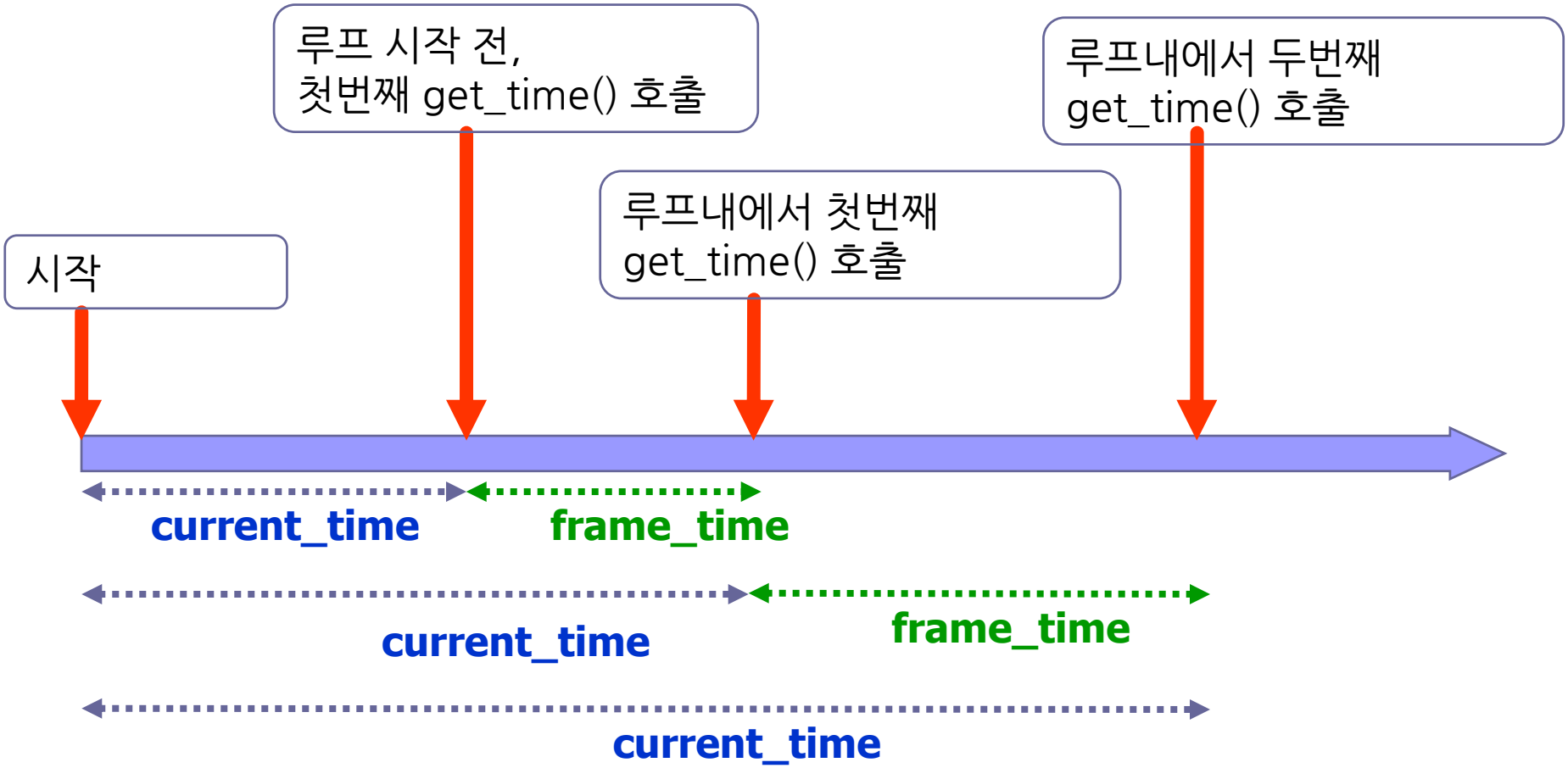
실습

# 프레임 시간과 프레임 속도 측정

# get_frame_time_rate.py


```
current_time = get_time()

while running:

    # Game Logic
    handle_events()
    for player in team:
        player.update()

    # Game Rendering
    clear_canvas()
    grass.draw()
    for player in team:
        player.draw()
    update_canvas()

    frame_time = get_time() - current_time
    frame_rate = 1.0 / frame_time
    print("Frame Rate: %f fps, Frame Time : %f sec, " %(frame_rate,frame_time))

    current_time += frame_time
```

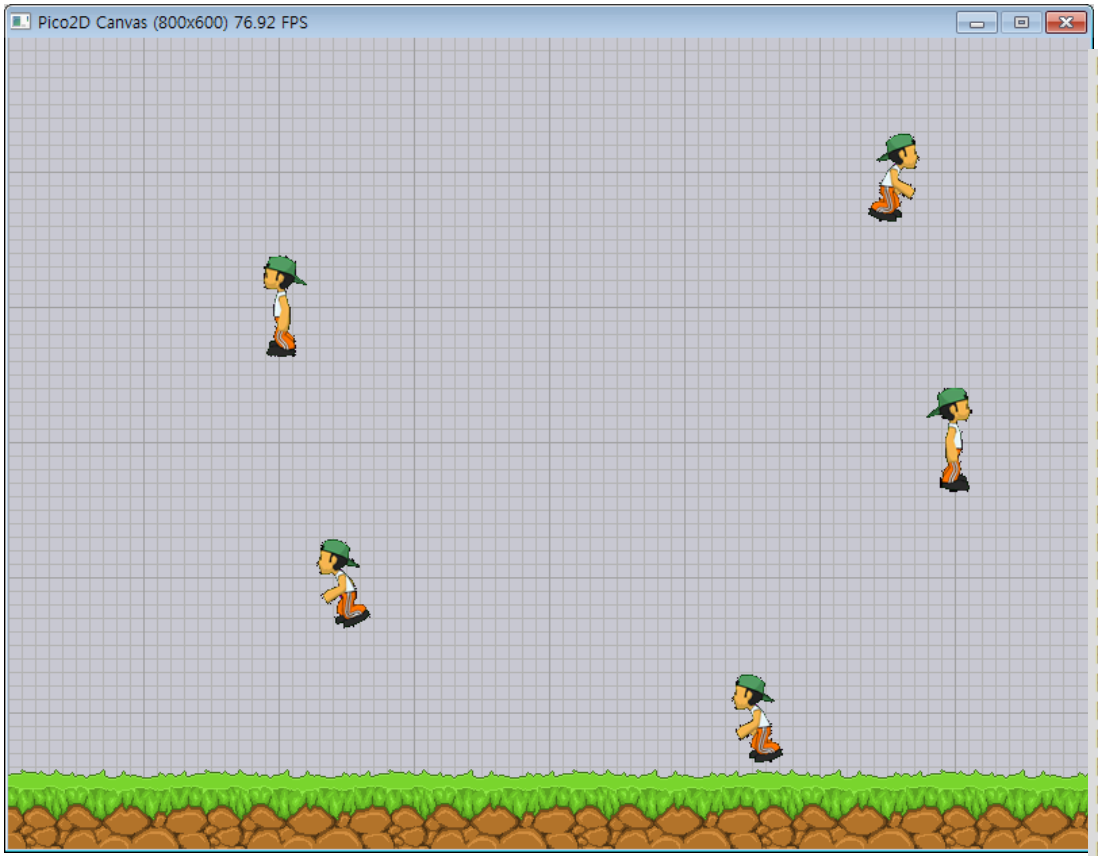
# 프레임 시간의 계산




프레임 시간: frame_time = get_time() - current_time

프레임 속도: frame_rate = 1.0 / frame_time

# 프레임 속도와 프레임 시간




```
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 125.000000 fps, Frame Time : 0.008000 sec,
Frame Rate: 90.909091 fps, Frame Time : 0.011000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 111.111111 fps, Frame Time : 0.009000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 125.000000 fps, Frame Time : 0.008000 sec,
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 125.000000 fps, Frame Time : 0.008000 sec,
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 125.000000 fps, Frame Time : 0.008000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 100.000000 fps, Frame Time : 0.010000 sec,
Frame Rate: 125.000000 fps, Frame Time : 0.008000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 100.000000 fps, Frame Time : 0.010000 sec,
Frame Rate: 166.666667 fps, Frame Time : 0.006000 sec,
Frame Rate: 83.333333 fps, Frame Time : 0.012000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 111.111111 fps, Frame Time : 0.009000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 100.000000 fps, Frame Time : 0.010000 sec,
Frame Rate: 111.111111 fps, Frame Time : 0.009000 sec,
Frame Rate: 142.857143 fps, Frame Time : 0.007000 sec,
Frame Rate: 100.000000 fps, Frame Time : 0.010000 sec,
```

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(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

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)

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

이동거리 = 경과시간 * 속도


```
distance = frame_time * SPEED
x = x + distance
```


# 프레임 시간에 따른 객체 이동 구현

# adjust_run_and_action.py (1)


```
class Boy:

    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)

    font = None
    image = None

    LEFT_RUN, RIGHT_RUN = 0, 1
```

# adjust_run_and_action.py (2)


```
def __init__(self):
    self.x, self.y = 0, 90
    self.frame = random.randint(0, 7)
    self.total_frames = 0.0
    self.dir = 1
    self.state = self.RIGHT_RUN
    if Boy.image == None:
        Boy.image = load_image('animation_sheet.png')
    if Boy.font == None:
        Boy.font = load_font('ENCR10B.TTF', 16)
```

# adjust_run_and_action.py (3)


```
def update(self, frame_time):
    distance = Boy.RUN_SPEED_PPS * frame_time
    self.total_frames += 1.0
    self.frame = (self.frame + 1) % 8
    self.x += (self.dir * distance)

    if self.x > 960:
        self.dir = -1
        self.x = 960
        self.state = self.LEFT_RUN
        print("Change Time: %f, Total Frames: %d" %
            (get_time(), self.total_frames))
    elif self.x < 0:
        self.dir = 1
        self.x = 0
        self.state = self.RIGHT_RUN
        print("Change Time: %f, Total Frames: %d" %
            (get_time(), self.total_frames))
```

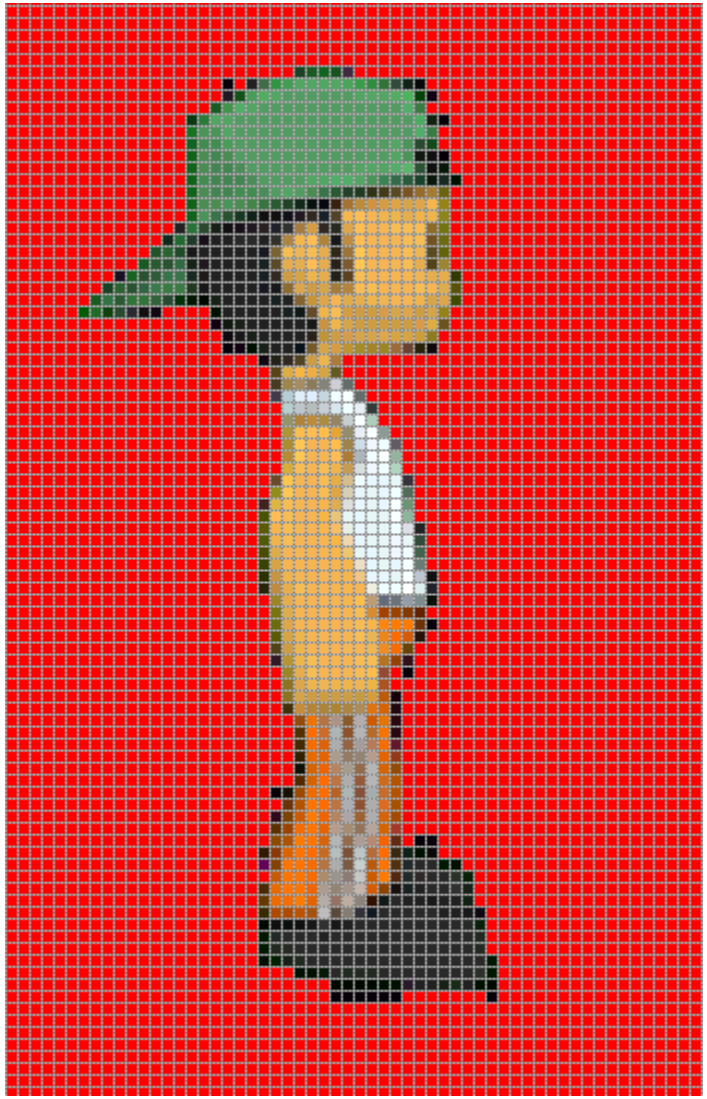
# adjust_run_and_action.py (4)


```
def draw(self):
    Boy.font.draw(self.x - 40, self.y + 50, 'Time: %3.2f' % get_time(), (255, 255, 0))
    self.image.opacify(random.random())
    self.image.clip_draw(self.frame * 100, self.state * 100, 100, 100, self.x, self.y)
```

# adjust_run_and_action.py (5)


```
while running:

    # Game Logic
    frame_time = get_frame_time()
    handle_events(frame_time)
    hero.update(frame_time)

    # Game Rendering
    clear_canvas()
    field.draw()
    hero.draw()
    update_canvas()
```

# 2D 공간의 물리값들을 먼저 결정할 필요



10 pixel = 30 cm

# 폰트 출력

```
font = load_font(폰트파일, 싸이즈)

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.

# 이미지 반투명화

```
image.opacify(o)

o: 0~1까지의 값. 0이면 완전 투명, 1이면 불투명
```

# 강제적인 프레임 시간 증가


```
while running:

    # Game Logic
    frame_time = get_frame_time()
    handle_events(frame_time)
    hero.update(frame_time)

    # Game Rendering
    clear_canvas()
    field.draw()
    hero.draw()
    update_canvas()

    delay(0.07)
```

실습

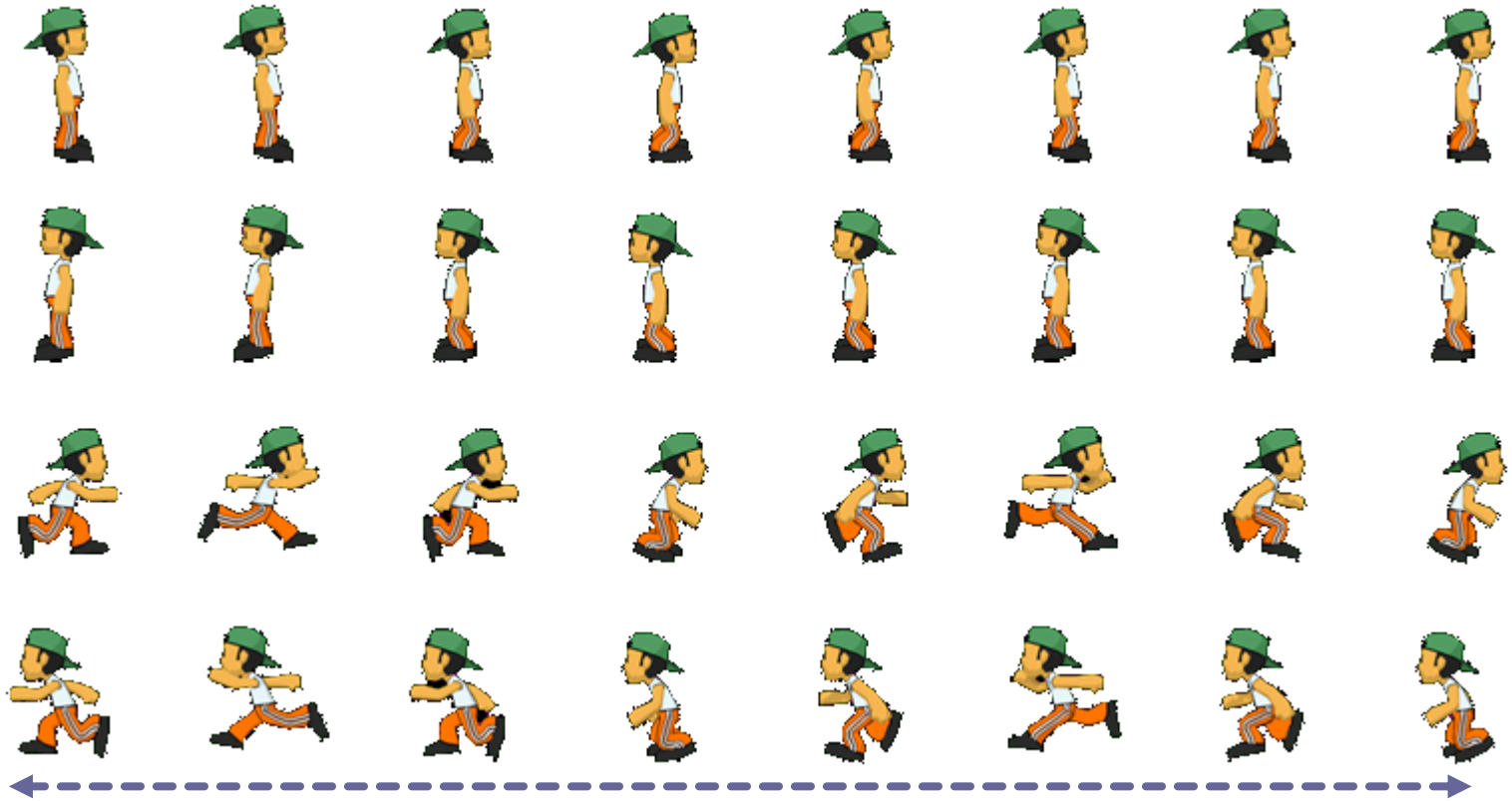
# 프레임 시간에 따른 액션 프레임의 조절

# adjust_run_and_action.py (1)


```
class Boy:

    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)


    TIME_PER_ACTION = 0.5
    ACTION_PER_TIME = 1.0 / TIME_PER_ACTION
    FRAMES_PER_ACTION = 8

    image = None

    LEFT_RUN, RIGHT_RUN = 0, 1
```

# adjust_run_and_action.py (2)


```
def update(self, frame_time):
    distance = Boy.RUN_SPEED_PPS * frame_time
    self.total_frames +=
        Boy.FRAMES_PER_ACTION * Boy.ACTION_PER_TIME * frame_time
    self.frame = int(self.total_frames) % 8
    self.x += (self.dir * distance)

    if self.x > 960:
        self.dir = -1
        self.x = 960
        self.state = self.LEFT_RUN
        print("Change Time: %f, Total Frames: %d" %
            (get_time(), self.total_frames))
    elif self.x < 0:
        self.dir = 1
        self.x = 0
        self.state = self.RIGHT_RUN
        print("Change Time: %f, Total Frames: %d" %
            (get_time(), self.total_frames))
```



TIME_PER_ACTION = 0.5

ACTION_PER_TIME = 1.0 / TIME_PER_ACTION ➔

FRAMES_PER_ACTION = 8