

# Lecture #16. 스크롤링

2D 게임 프로그래밍

이대현 교수



한국공학대학교  
TECH UNIVERSITY OF KOREA

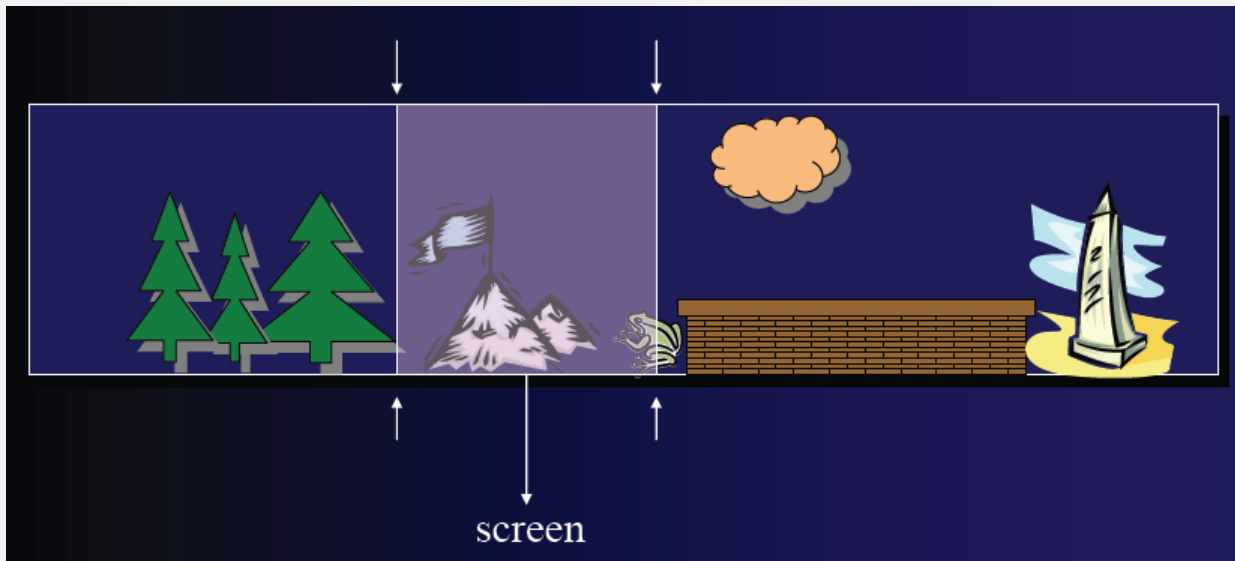
# 학습 내용

---

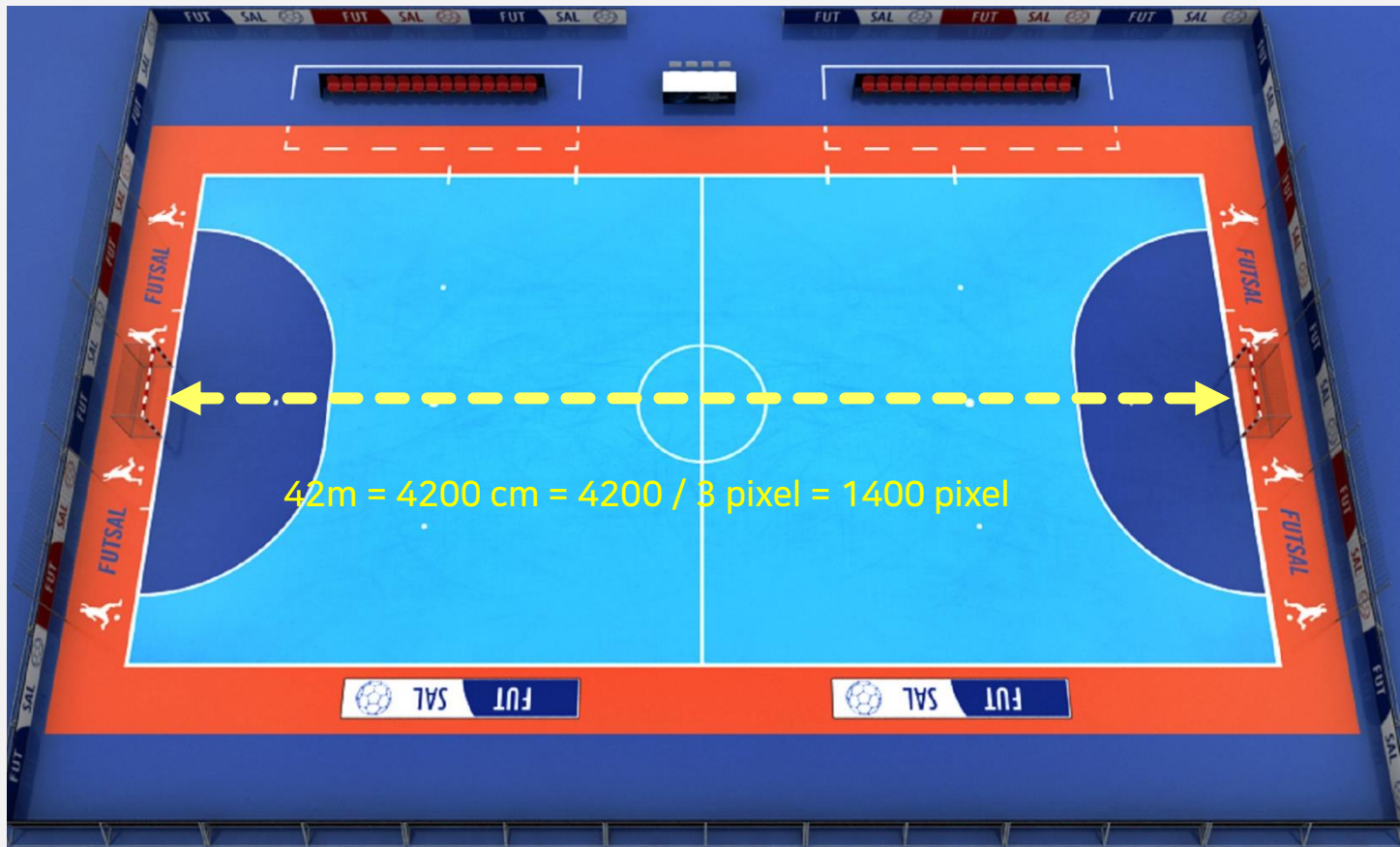
- 스크롤링
- 타일맵 기반 스크롤링
- 무한 스크롤링
- 시차 스크롤링

# 스크롤링(Scrolling)

- 그림이나 이미지의 일부분을 디스플레이 화면 위에서 상하좌우로 움직이면서 나타내는 기법.



# 게임 맵은 반드시 실제 물리값으로 크기가 표시되어야 함.



# 실제 좌표와 스크린 좌표를 분리 처리



# 스크린 윈도우를 이용한 스크롤링



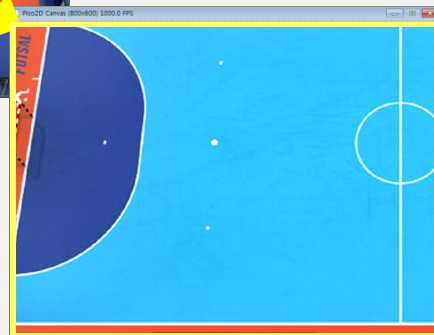
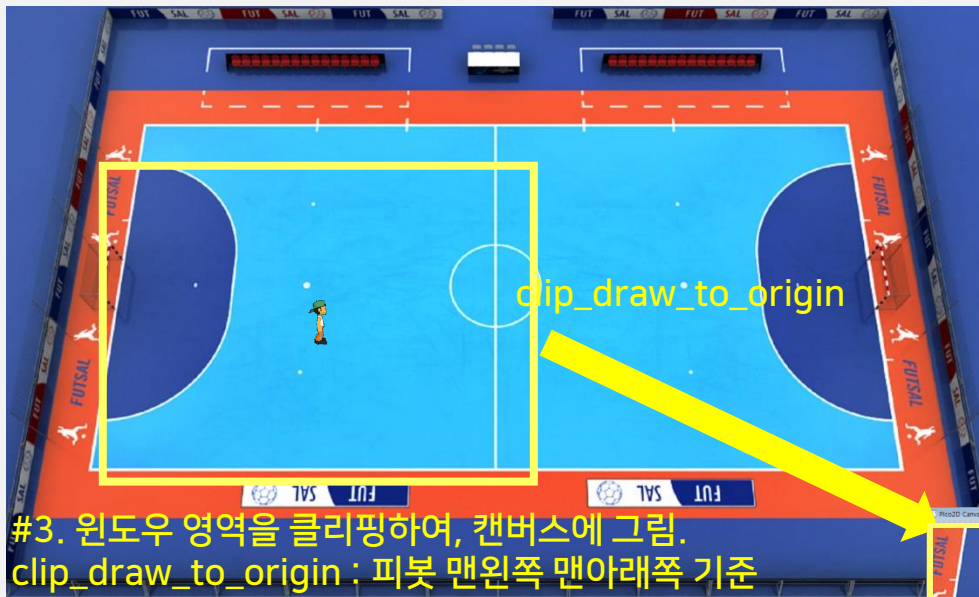


#2. 플레이어를 가운데에 놓고, 맵 상의 윈도우 좌표를 계산

window\_left,

window\_bottom

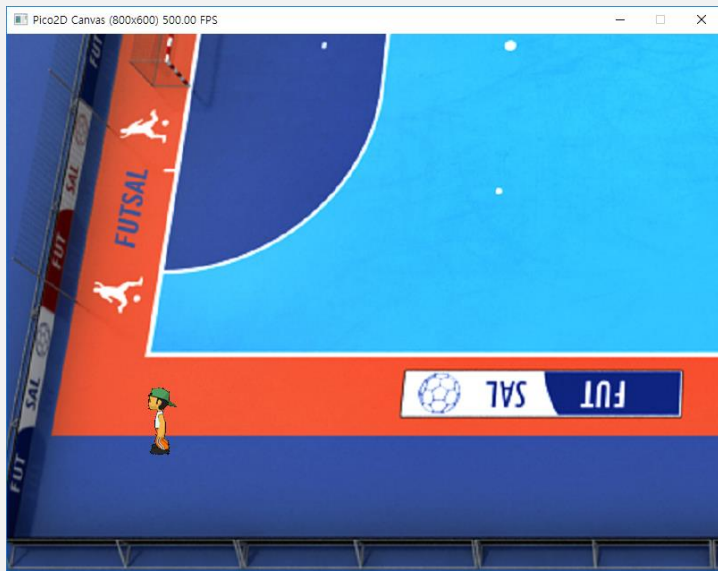
x-canvas\_width//2, y-canvas\_height//2)







y - window\_bottom



x - window\_left



상하좌우 스크롤링!

# clamp 함수

---

```
def clamp(minimum, x, maximum):  
    return max(minimum, min(x, maximum))
```



```
from boy import Boy
# fill here
from background import FixedBackground as Background
```



```
class FixedBackground:
```

```
def draw(self):
```

```
    self.image.clip_draw_to_origin(  
        self.window_left, self.window_bottom,  
        self.canvas_width, self.canvas_height,  
        0, 0)
```

```
def update(self, frame_time):
```

```
    self.window_left = clamp(0,  
        int(server.boy.x) - self.canvas_width//2,  
        self.w - self.canvas_width - 1)  
    self.window_bottom = clamp(0,  
        int(server.boy.y) - self.canvas_height//2,  
        self.h - self.canvas_height - 1)
```

# boy.py (1)



```
def do(boy):  
    boy.frame = (boy.frame + FRAMES_PER_ACTION * ACTION_PER_TIME *  
game_framework.frame_time) % FRAMES_PER_ACTION  
    boy.x += boy.x_velocity * game_framework.frame_time  
    boy.y += boy.y_velocity * game_framework.frame_time  
  
    # fill here  
    boy.x = clamp(0, boy.x, server.background.w-1)  
    boy.y = clamp(0, boy.y, server.background.h-1)
```

# boy.py (2)



```
def draw(boy):  
    sx, sy = boy.x - server.background.window_left, boy.y - server.background.window_bottom  
  
    boy.font.draw(sx - 40, sy + 40, '(%d, %d)' % (boy.x, boy.y), (255, 255, 0))
```

# background.py

```
class FixedBackground:

    def draw(self):
        self.image.clip_draw_to_origin(
            self.window_left, self.window_bottom,
            self.canvas_width, self.canvas_height,
            0, 0)

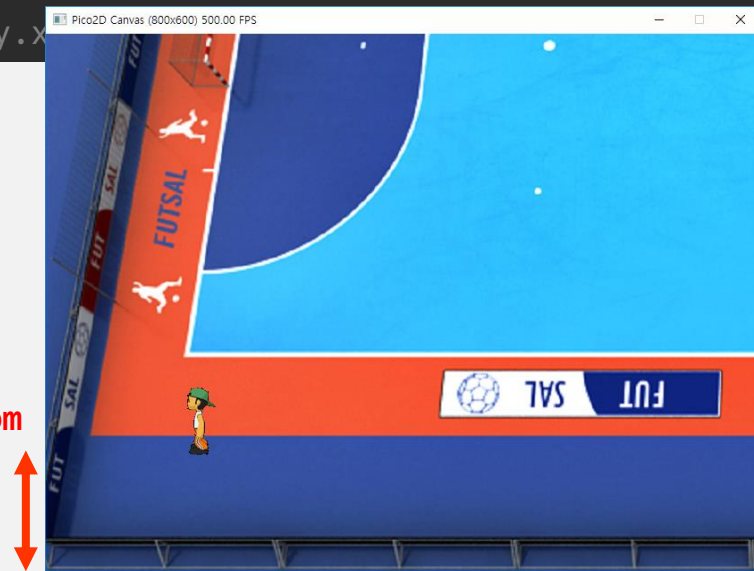
    def update(self, frame_time):
        self.window_left = clamp(0,
            int(server.boy.x) - self.canvas_width//2,
            self.w - self.canvas_width - 1)
        self.window_bottom = clamp(0,
            int(server.boy.y) - self.canvas_height//2,
            self.h - self.canvas_height - 1)
```

window의 left x 좌표의 최대값은, 전체 배경 너비에서 화면의 너비를 뺀 값.



```
def draw(boy):  
    sx, sy = boy.x - server.background.window_left, boy.y - server.background.window_bottom  
  
    boy.font.draw(sx - 40, sy + 40, '(%d, %d)' % (boy.x,
```

$y - \text{window\_bottom}$



$x - \text{window\_left}$



상하좌우 스크롤링!  
(타일링 배경)

# Tile image



cube00



cube01



cube02



cube10



cube11



cube12



cube20

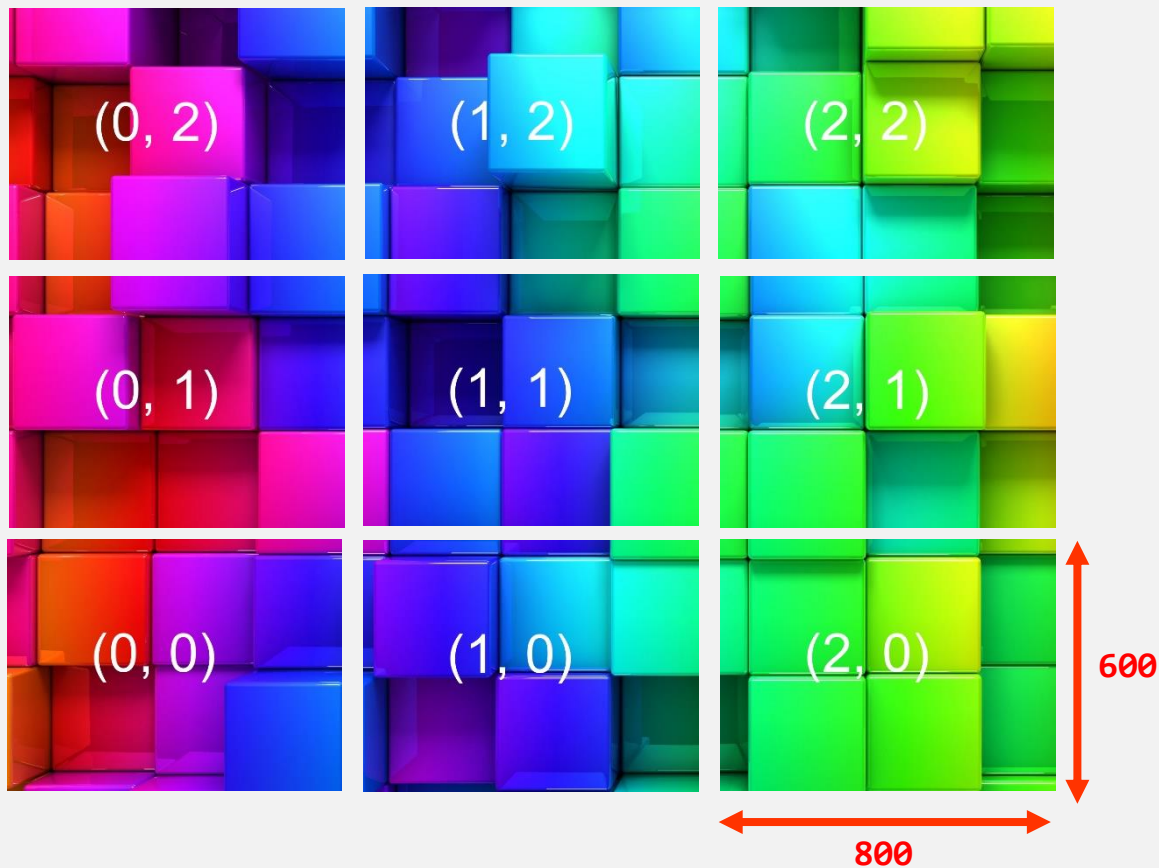


cube21



cube22

# 타일맵 구조





```
from boy import Boy
# fill here
from background import TileBackground as Background
```

# background.py (1)



```
class TileBackground:

    def __init__(self):
        self.canvas_width = get_canvas_width()
        self.canvas_height = get_canvas_height()
        self.w = 800 * 3
        self.h = 600 * 3

        self.tiles = [ [ load_image('cube%d%d.png' % (x, y)) for x in range(3) ] for y in range(3) ]
```

# background.py (2)



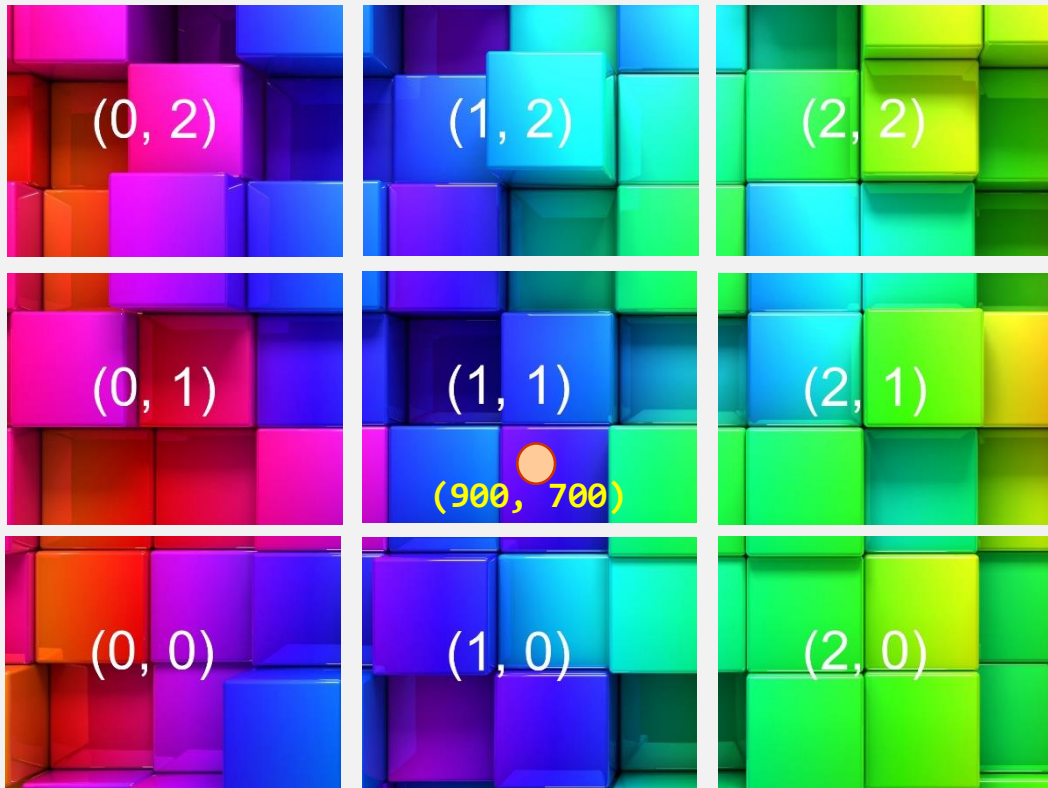
```
def draw(self):
    self.window_left = clamp(0,
                             int(server.boy.x) - self.canvas_width // 2,
                             self.w - self.canvas_width - 1)
    self.window_bottom = clamp(0,
                               int(server.boy.y) - self.canvas_height // 2,
                               self.h - self.canvas_height - 1)

    tile_left = self.window_left // 800
    tile_right = (self.window_left + self.canvas_width) // 800
    left_offset = self.window_left % 800

    tile_bottom = self.window_bottom // 600
    tile_top = (self.window_bottom + self.canvas_height) // 600
    bottom_offset = self.window_bottom % 600

    for ty in range(tile_bottom, tile_top+1):
        for tx in range(tile_left, tile_right+1):
            self.tiles[ty][tx].draw_to_origin(-left_offset + (tx-tile_left)*800, -bottom_offset+(ty-tile_bottom)*600)
```

# 전체 맵 좌표로부터, 타일맵 좌표의 계산



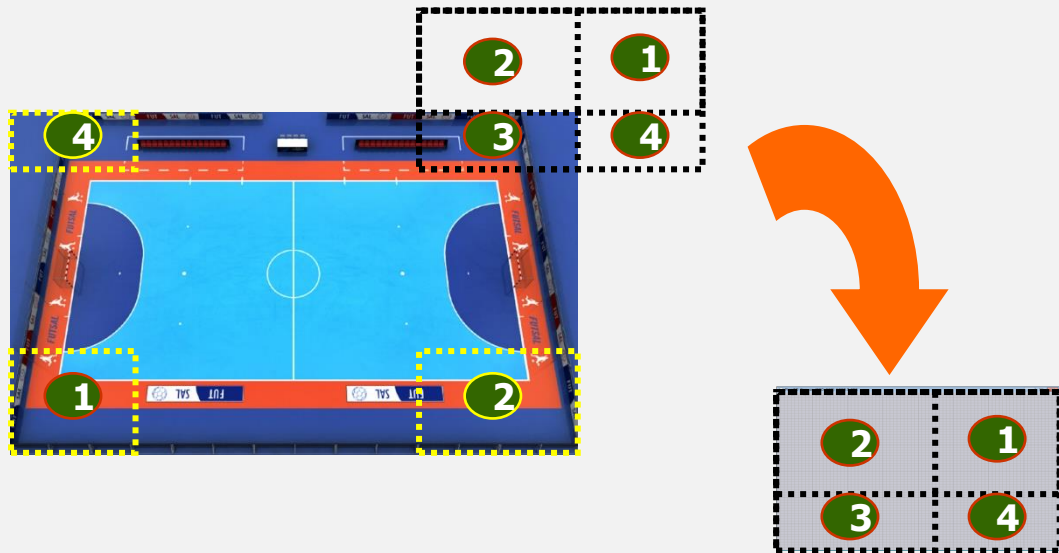
```
tx = 900 // 800  
ty = 700 // 600
```

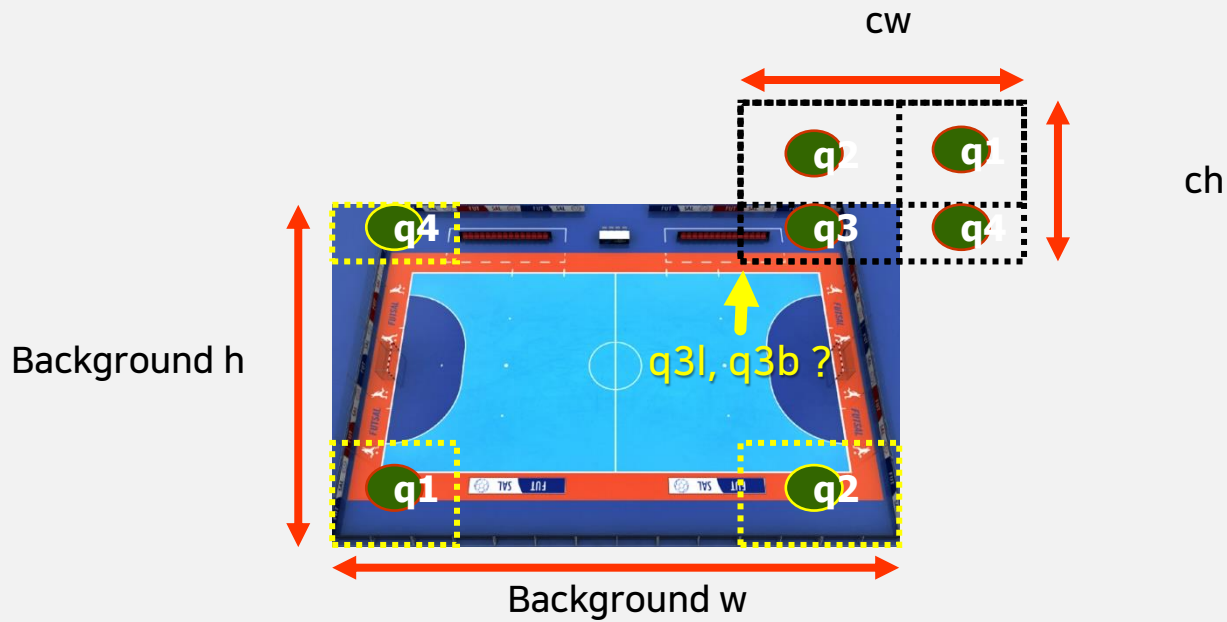


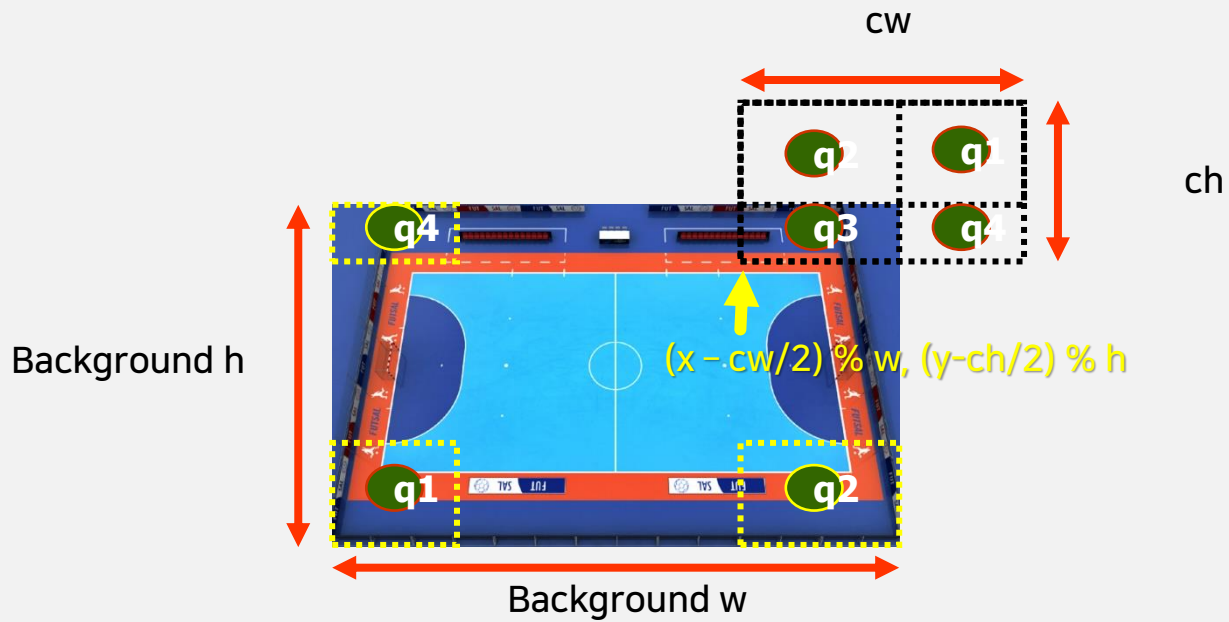


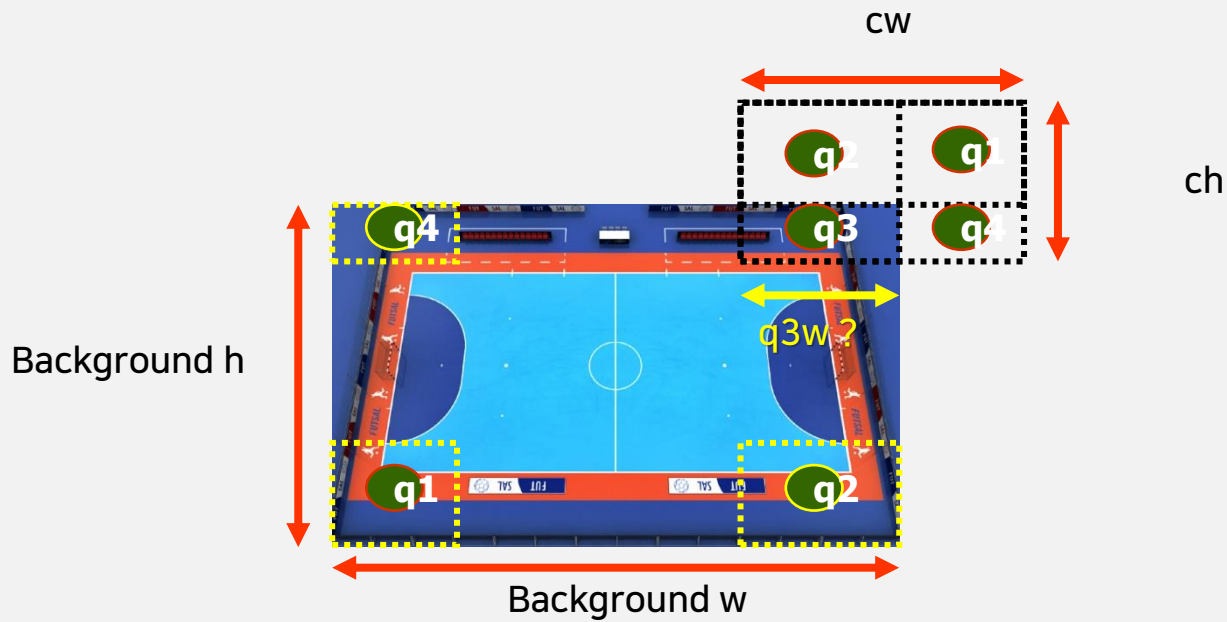
상하좌우 무한 스크롤링

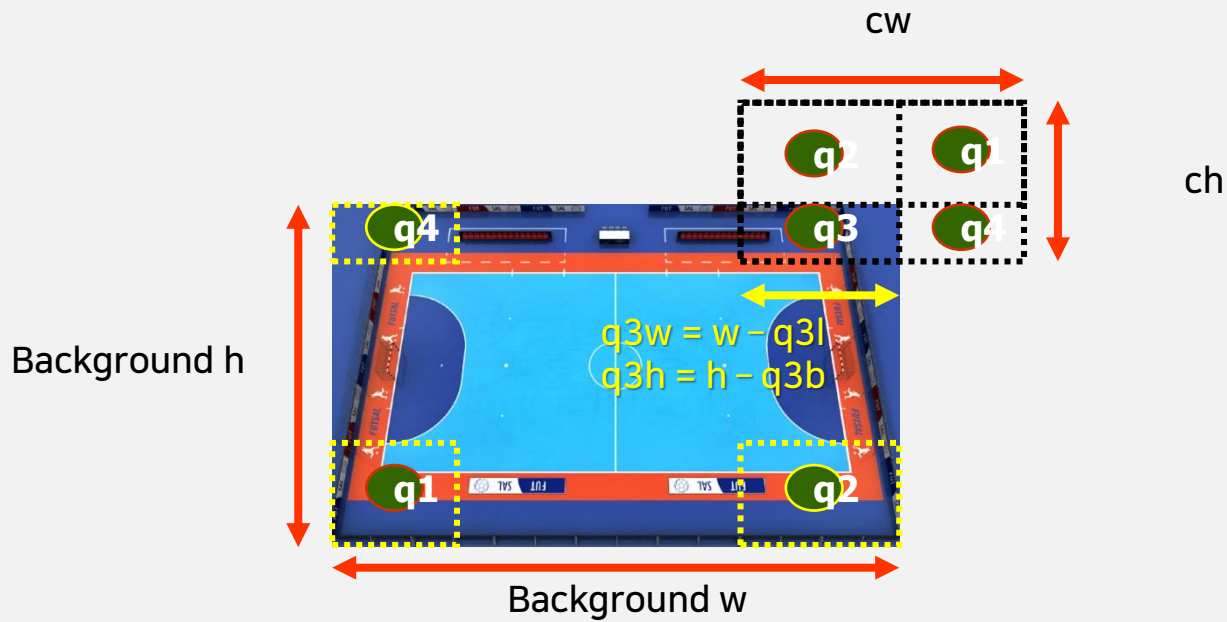
# 상하좌우 무한스크롤링 공식













```
# from background import TileBackground as Background
# from background import FixedBackground as Background
from background import InfiniteBackground as Background
```



```
def update(self):
    self.cur_state.do(self)
    if len(self.event_que) > 0:
        event = self.event_que.pop()
        self.cur_state.exit(self, event)
        self.cur_state = next_state_table[self.cur_state][event]
        self.cur_state.enter(self, event)
```

```
# fill here
```

```
self.x = clamp(50, self.x, server.background.w-50)
self.y = clamp(50, self.y, server.background.h-50)
```

```
def draw(boy):
    sx, sy = server.background.canvas_width // 2, server.background.canvas_height // 2
    # sx, sy = boy.x - server.background.window_left, boy.y - server.background.window_bottom
```





```
class InfiniteBackground:
```

```
    def update(self, frame_time):
```

```
        # quadrant 3
```

```
        self.q3l = (int(server.boy.x) - self.canvas_width // 2) % self.w
```

```
        self.q3b = (int(server.boy.y) - self.canvas_height // 2) % self.h
```

```
        self.q3w = clamp(0, self.w - self.q3l, self.w)
```

```
        self.q3h = clamp(0, self.h - self.q3b, self.h)
```

```
        # quadrant 2
```

```
        self.q2l = ?
```

```
        self.q2b = ?
```

```
        self.q2w = ?
```

```
        self.q2h = ?
```

```
        # quadrant 4
```

```
        self.q4l = ?
```

```
        self.q4b = ?
```

```
        self.q4w = ?
```

```
        self.q4h = ?
```

```
        # quadrant 1
```

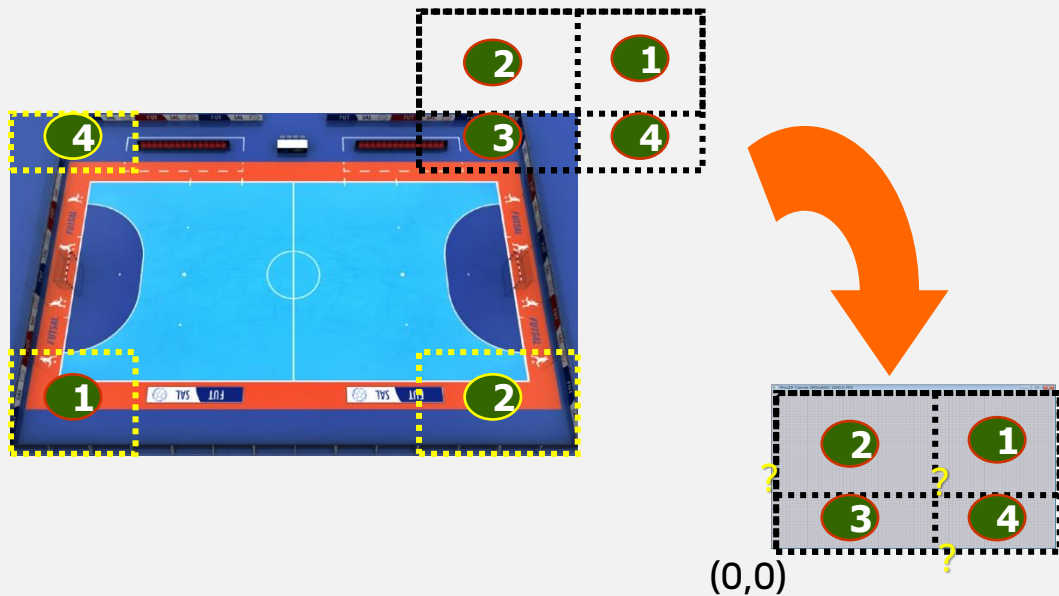
```
        self.q1l = ?
```

```
        self.q1b = ?
```

```
        self.q1w = ?
```

```
        self.q1h = ?
```

# 상하좌우 무한스크롤링 공식





**class InfiniteBackground:**

```
def draw(self):  
    self.image.clip_draw_to_origin(self.q3l, self.q3b, self.q3w, self.q3h, 0, 0)  
    self.image.clip_draw_to_origin(self.q2l, self.q2b, self.q2w, self.q2h, ?, ?)  
    self.image.clip_draw_to_origin(self.q4l, self.q4b, self.q4w, self.q4h, ?, ?)  
    self.image.clip_draw_to_origin(self.q1l, self.q1b, self.q1w, self.q1h, ?, ?)
```

# 시차(視差) 스크롤링(Parallax Scrolling)

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



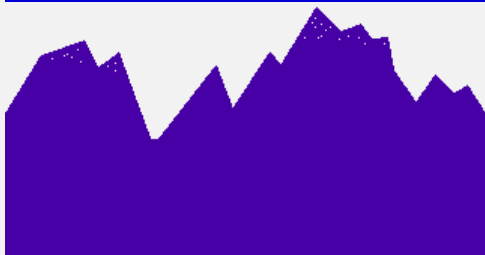
- 밤하늘, 뒷산, 앞산의 스크롤링 속도를 다르게 함으로써, 3차원적인 깊이 효과를 구현.



# 시차 스크롤링 방법



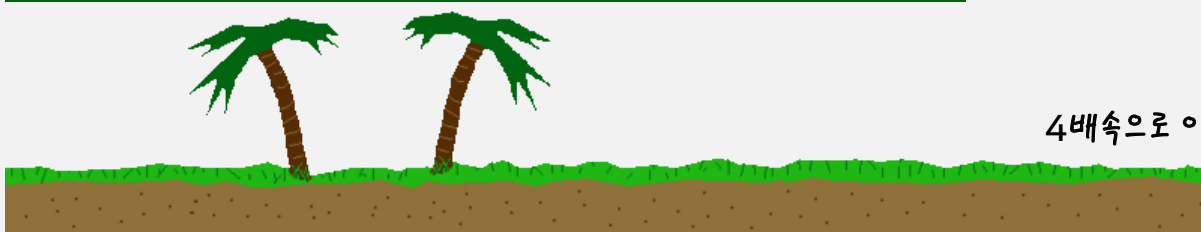
1배속으로 이동



2배속으로 이동



3배속으로 이동



4배속으로 이동

```
def draw(self):
    self.image.clip_draw_to_origin(self.q3l, self.q3b, self.q3w, self.q3h, 0, 0)           # quadrant 3
    self.image.clip_draw_to_origin(self.q2l, self.q2b, self.q2w, self.q2h, 0, self.q3h)      # quadrant 2
    self.image.clip_draw_to_origin(self.q4l, self.q4b, self.q4w, self.q4h, self.q3w, 0)      # quadrant 4
    self.image.clip_draw_to_origin(self.q1l, self.q1b, self.q1w, self.q1h, self.q3w, self.q3h) # quadrant 1

def update(self):
    # quadrant 3
    self.q3l = (int(server.boy.x) - self.canvas_width // 2) % self.w
    self.q3b = (int(server.boy.y) - self.canvas_height // 2) % self.h
    self.q3w = clamp(0, self.w - self.q3l, self.w)
    self.q3h = clamp(0, self.h - self.q3b, self.h)
    # quadrant 2
    self.q2l = self.q3l
    self.q2b = 0
    self.q2w = self.q3w
    self.q2h = self.canvas_height - self.q3h
    # quadrant 4
    self.q4l = 0
    self.q4b = self.q3b
    self.q4w = self.canvas_width - self.q3w
    self.q4h = self.q3h
    # quadrant 1
    self.q1l = 0
    self.q1b = 0
    self.q1w = self.q4w
    self.q1h = self.q2h
```