Lecture #18. 스크롤링

2D 게임 프로그래밍

이대현 교수

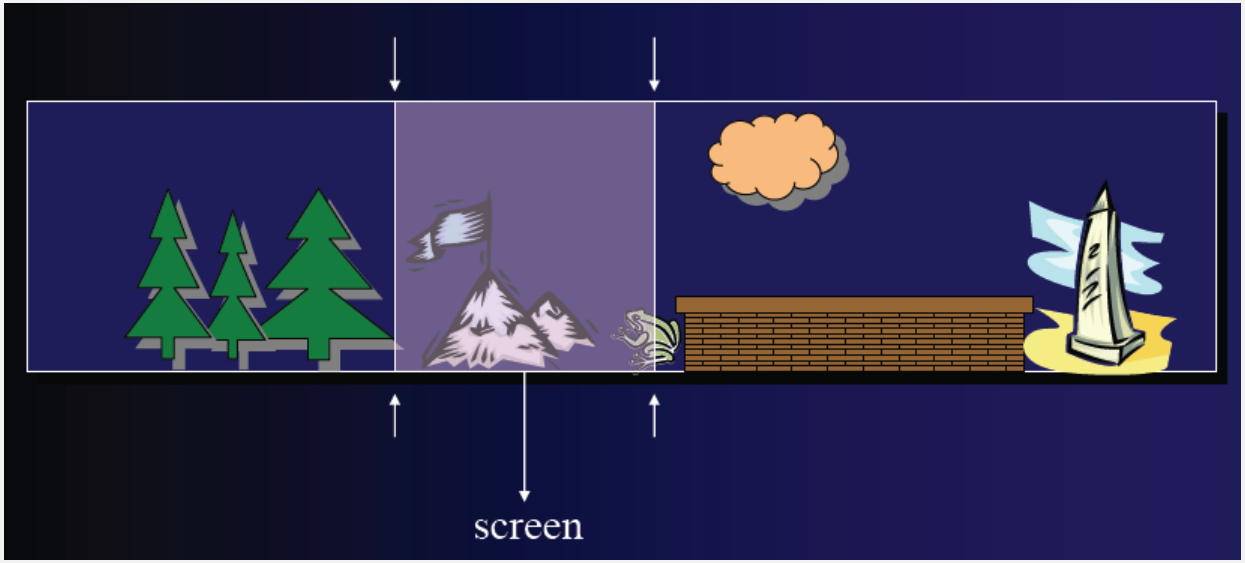
한국공학대학교
TECH UNIVERSITY OF KOREA

학습 내용

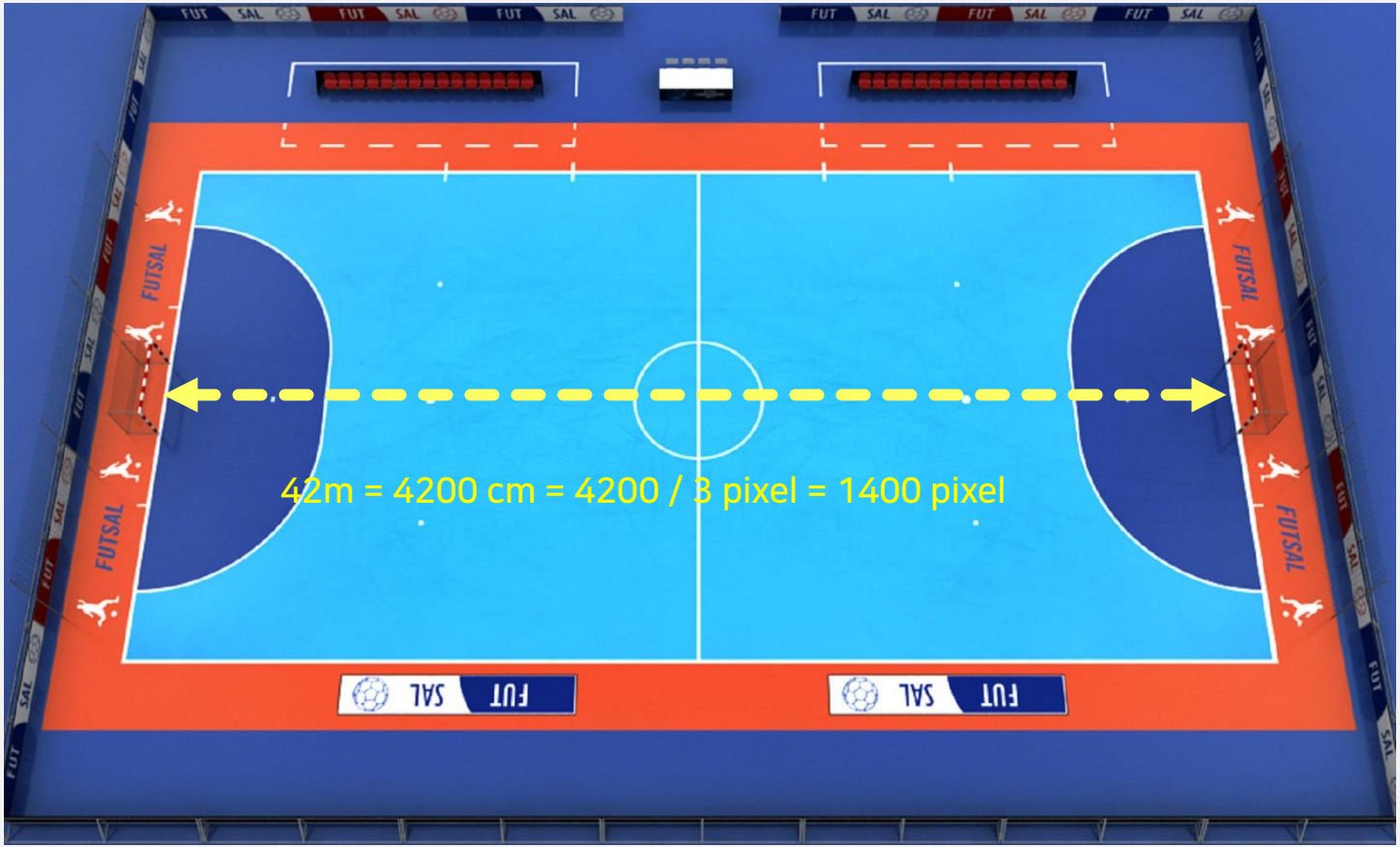
- 스크롤링
- 타일맵 기반 스크롤링
- 무한 스크롤링
- 시차 스크롤링

스크롤링(Scrolling)

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



게임 맵은 반드시 실제 물리값으로 크기가 표시되어야 함.



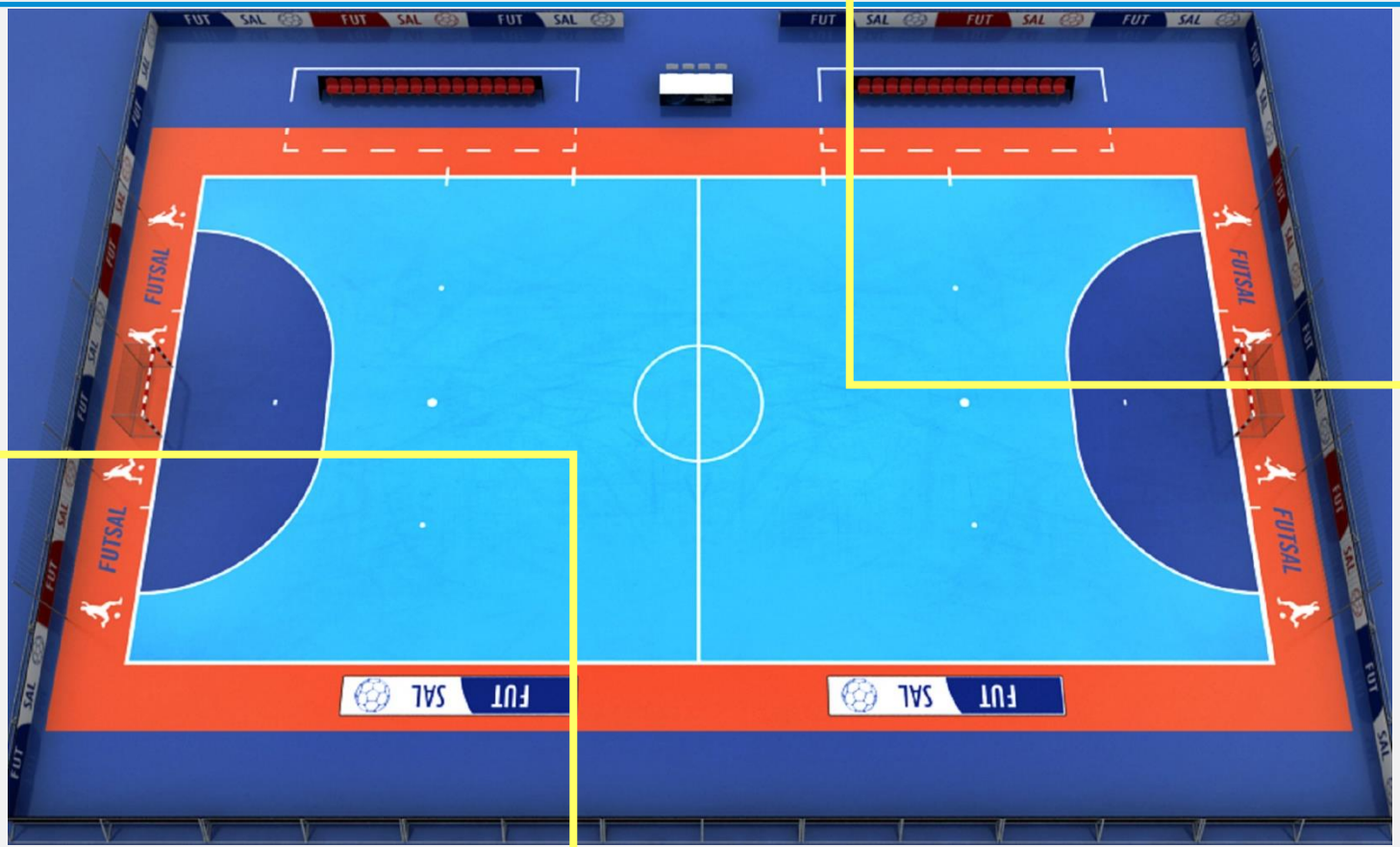
실제 좌표와 화면 좌표를 분리 처리

실제 공간 좌표 – 객체의 실제 좌표 계산할 때,

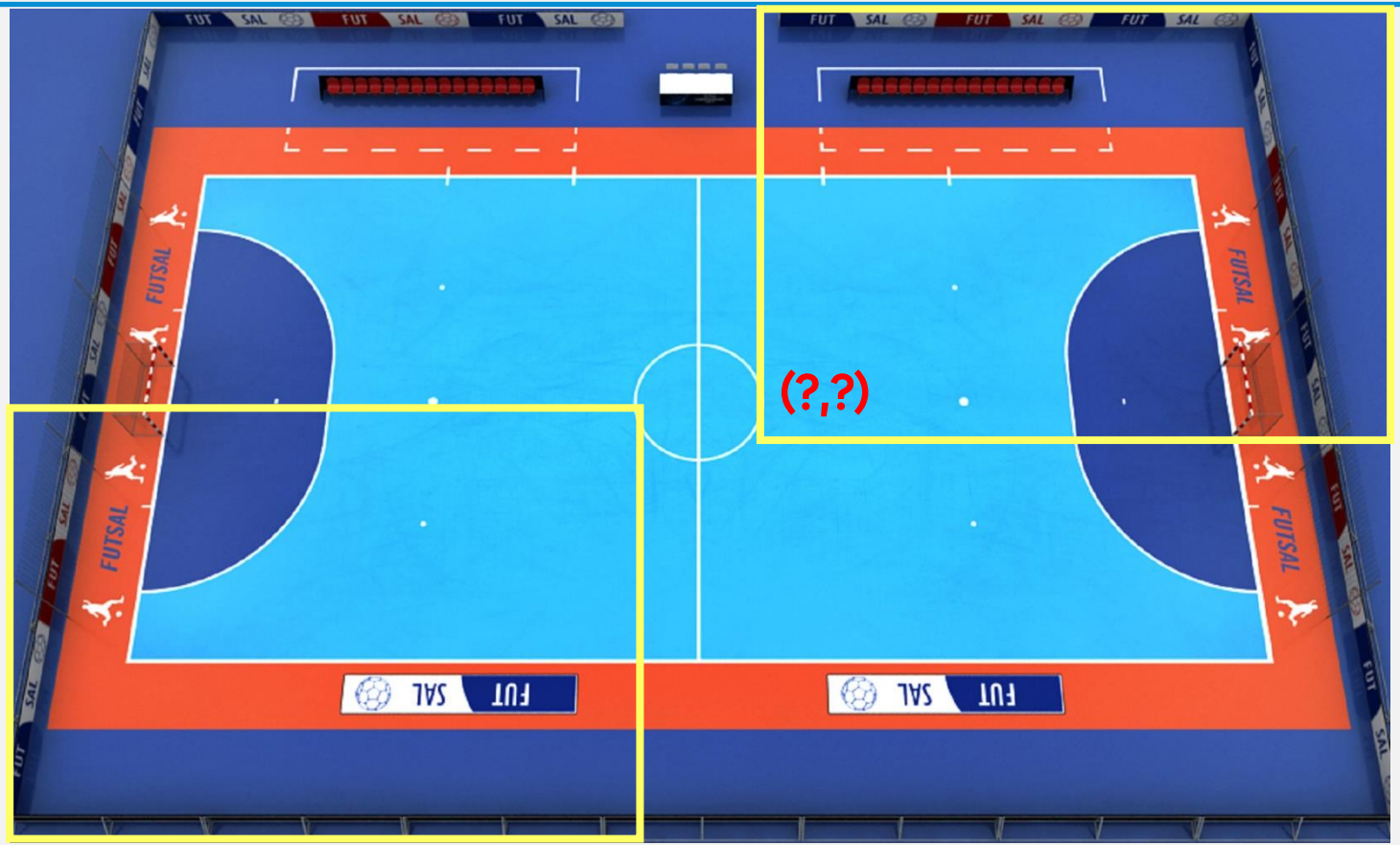
화면 좌표 – 화면 상에 그릴 때

퀴즈 – 클리핑 영역 계산

클리핑 영역이 물리 공간을 넘어서면?



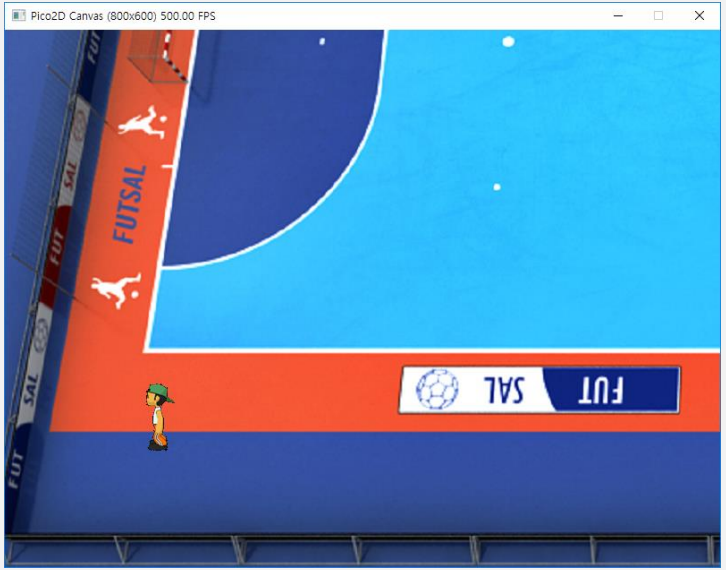
실제 가능한 클리핑 영역은?



스크린 윈도우를 이용한 스크롤링

window_left, window_bottom

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



상하좌우 스크롤링 #1

clamp 함수

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

화면의 정중앙에 캐릭터를 그림


```
def draw(self):
    sx, sy = get_canvas_width() // 2, get_canvas_height() // 2
    self.image.clip_draw(int(self.frame) * 100, self.action * 100, 100, 100, sx, sy)
    self.font.draw(sx - 100, sy + 60, f'({self.x:5.5}, {self.y:5.5})', (255, 255, 0))
```

boy.py - 물리 좌표계와 화면 좌표의 분리


```
def __init__(self):
    self.x, self.y = get_canvas_width() / 2, get_canvas_height() / 2

    # 물리 좌표계로 바꿔야 함.
    self.x, self.y = server.background.w / 2, server.background.h / 2
```

```
def update(self):
    self.state_machine.update()
    self.frame = (self.frame + FRAMES_PER_ACTION * ACTION_PER_TIME * game_framework.frame_time) % 8
    self.x += math.cos(self.dir) * self.speed * game_framework.frame_time
    self.y += math.sin(self.dir) * self.speed * game_framework.frame_time
    self.x = clamp(50.0, self.x, get_canvas_width()-50.0)
    self.y = clamp(50.0, self.y, get_canvas_height()-50.0)

    # 물리 좌표계로 바꿔야 함.
    self.x = clamp(get_canvas_width()/2, self.x, server.background.w - get_canvas_width()/2)
    self.y = clamp(get_canvas_height()/2, self.y, server.background.h - get_canvas_height()/2)
```

background.py


```
class FixedBackground:

    def __init__(self):
        self.image = load_image('futsal_court.png')
        self.cw = get_canvas_width()
        self.ch = get_canvas_height()
        self.w = self.image.w
        self.h = self.image.h

    def draw(self):
        self.image.clip_draw_to_origin(self.window_left, self.window_bottom, self.cw, self.ch, 0, 0)

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

background.py

```
def draw(self):
    self.image.clip_draw_to_origin(self.window_left, self.window_bottom, self.cw, self.ch, 0, 0)



def update(self):

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

피봇(중심)을 무시하고, 왼쪽 아래 원점을 피봇으로 간주.

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

상하좌우 스크롤링 #2

```
sx = self.x - server.background.window_left
sy = self.y - server.background.window_bottom
```

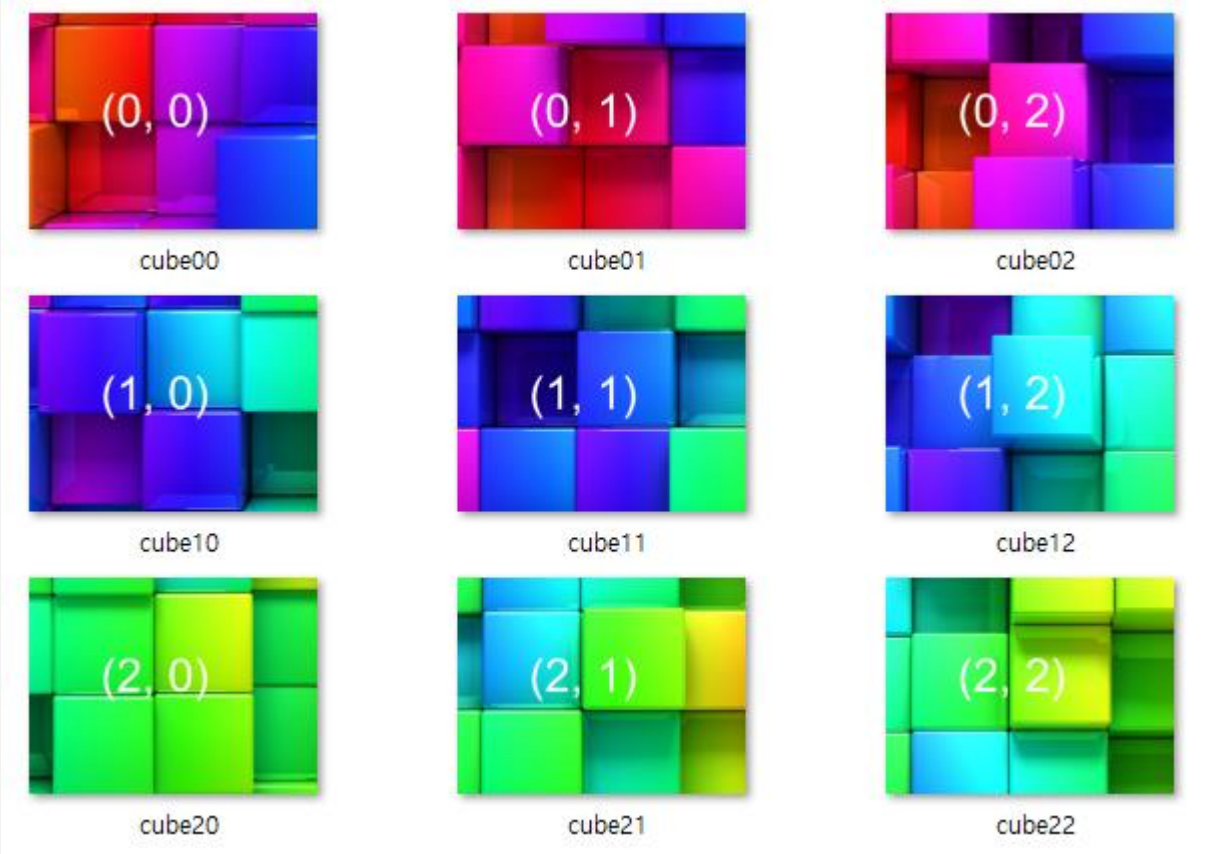
boy.py (1)


```
def update(self):
    self.state_machine.update()
    self.frame = (self.frame + FRAMES_PER_ACTION * ACTION_PER_TIME * game_framework.frame_time) % 8
    self.x += math.cos(self.dir) * self.speed * game_framework.frame_time
    self.y += math.sin(self.dir) * self.speed * game_framework.frame_time
    self.x = clamp(50.0, self.x, server.background.w - 50.0)
    self.y = clamp(50.0, self.y, server.background.h - 50.0)
```

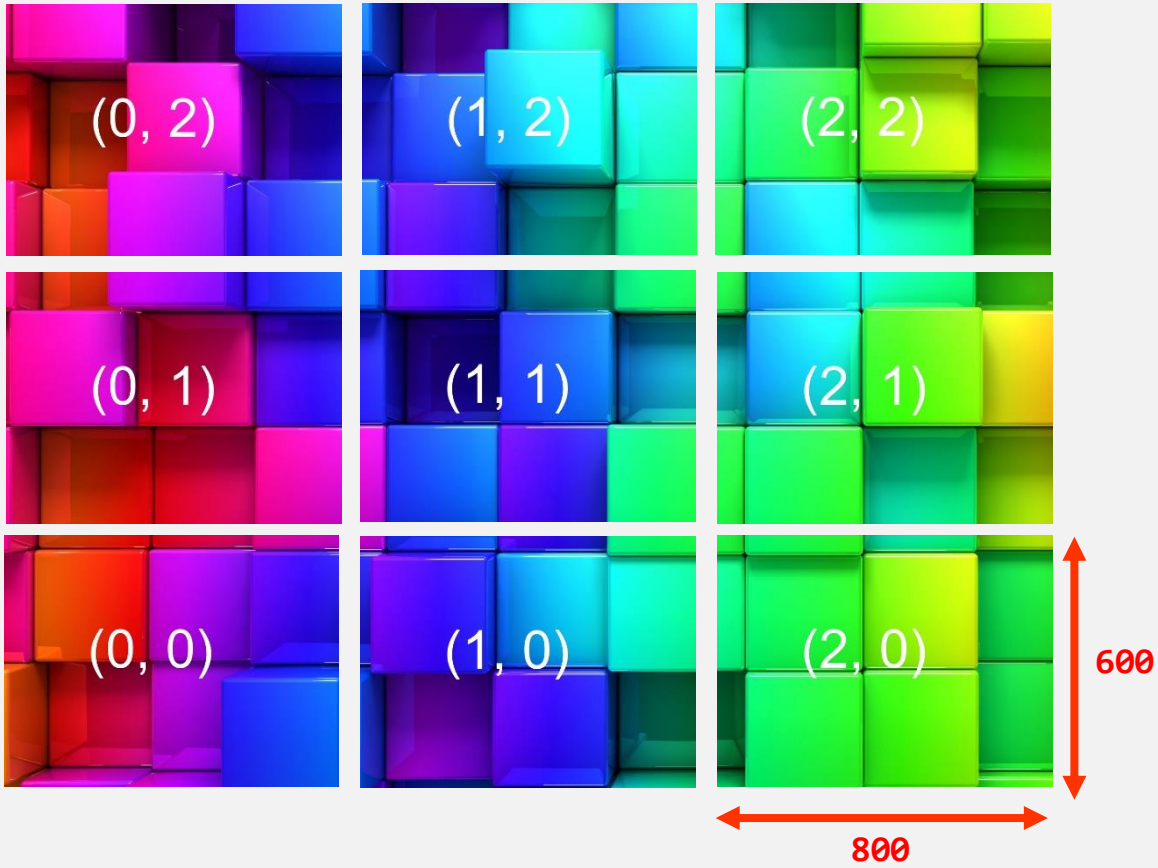
실습

상하좌우 스크롤링 (타일링 배경)

Tile image



타일맵 구조



play_mode.py


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

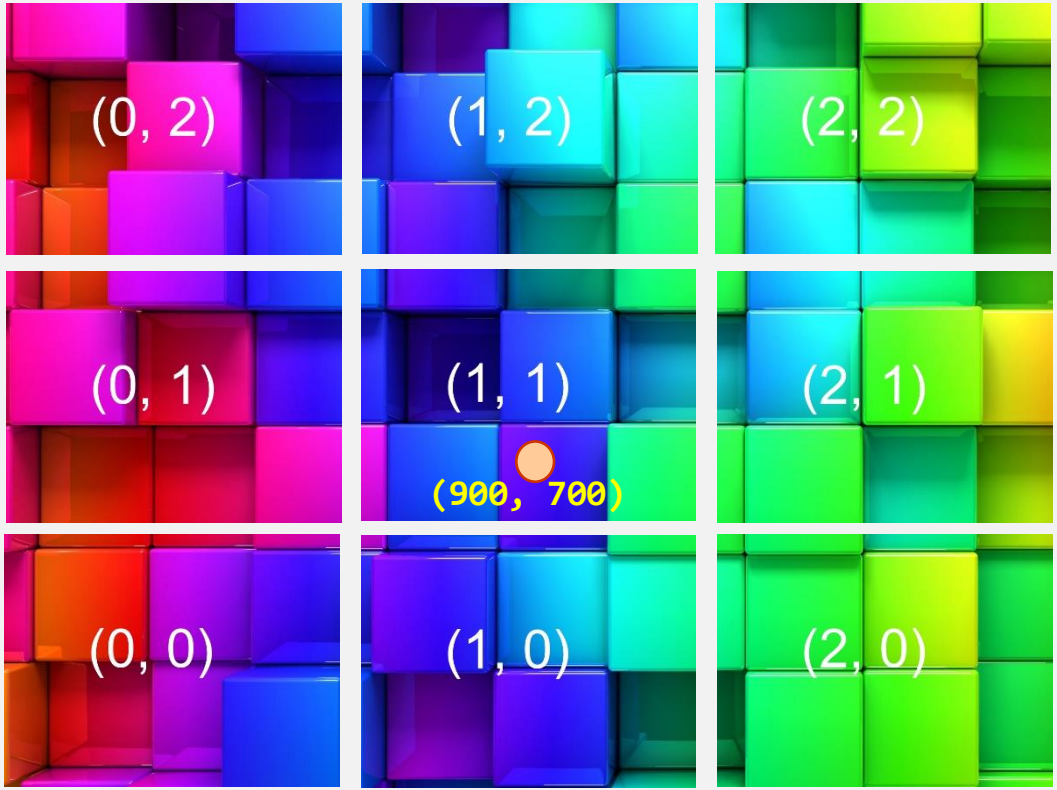
background.py (1)


```
class TileBackground:

    def __init__(self):
        self.cw= get_canvas_width()
        self.ch = 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.cw // 2, self.w - self.cw - 1)
    self.window_bottom = clamp(0, int(server.boy.y) - self.ch // 2, self.h - self.ch - 1)

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

    tile_bottom = self.window_bottom // 600
    tile_top = (self.window_bottom + self.ch) // 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

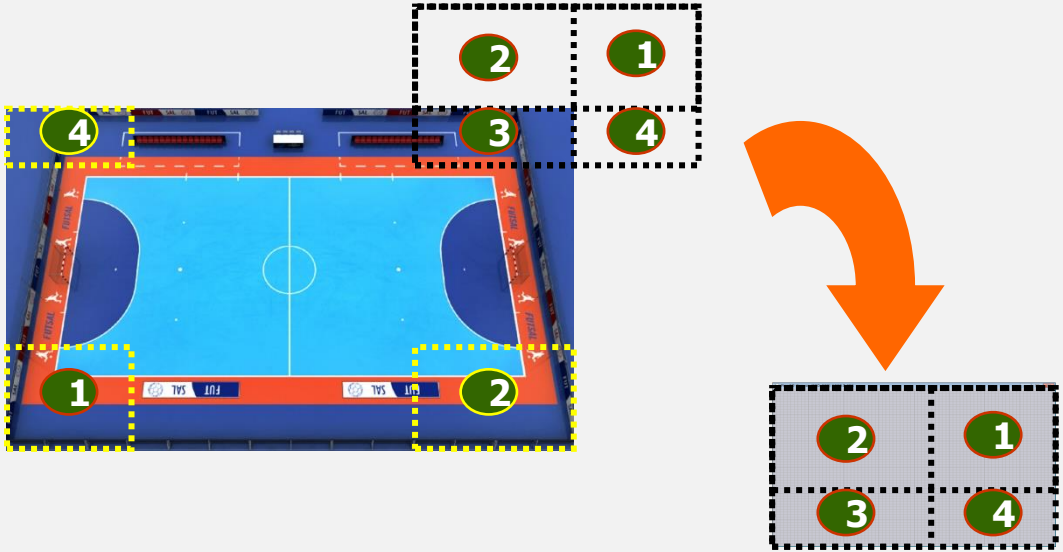


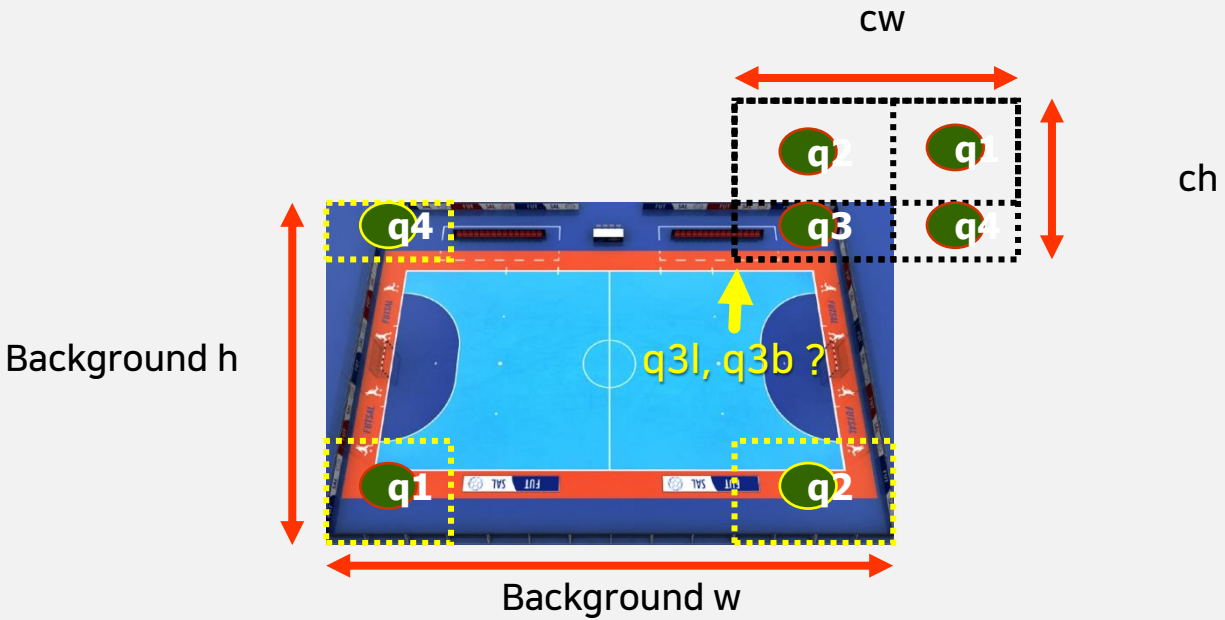

```
tx = 900 // 800
ty = 700 // 600
```

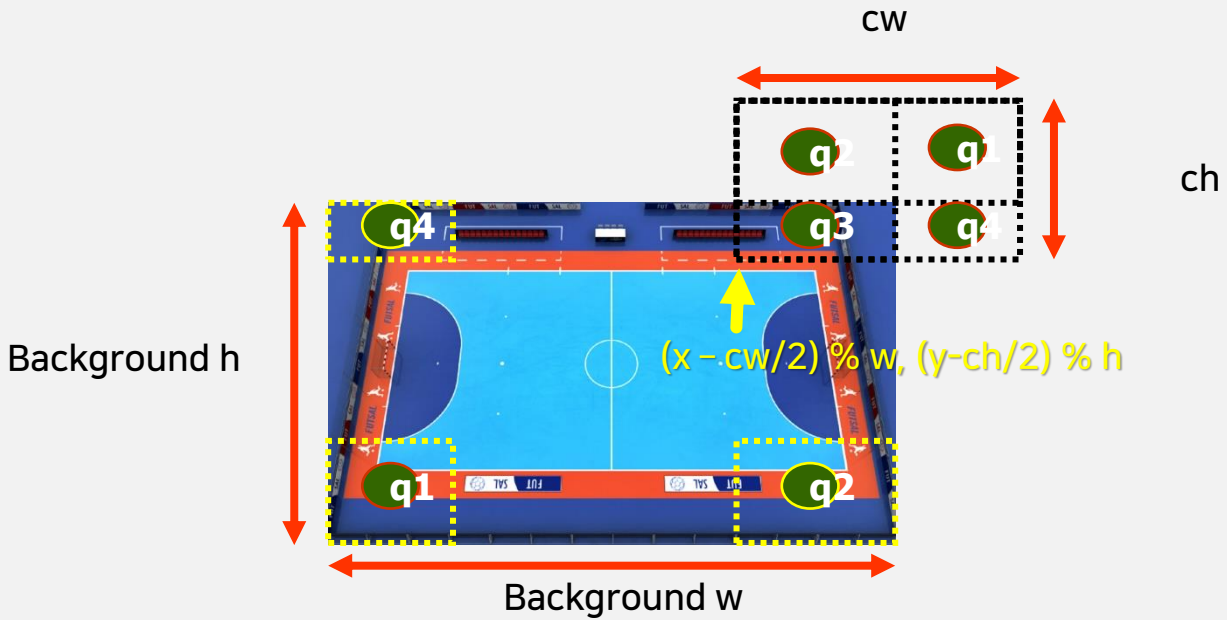
실습

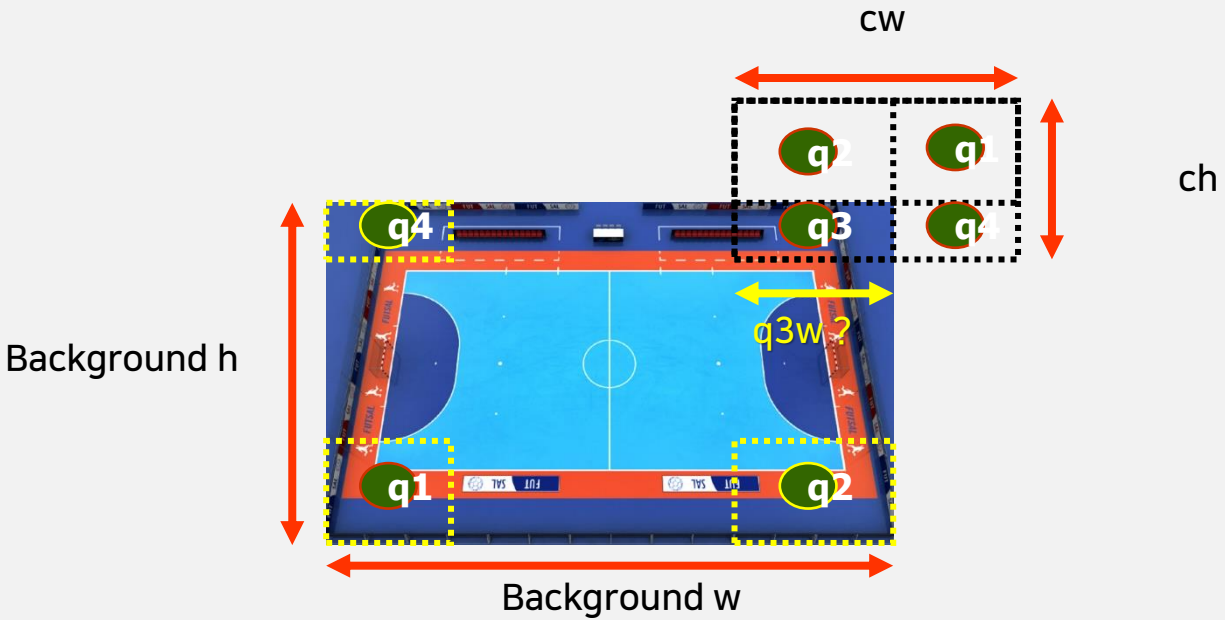
상하좌우 무한 스크롤링

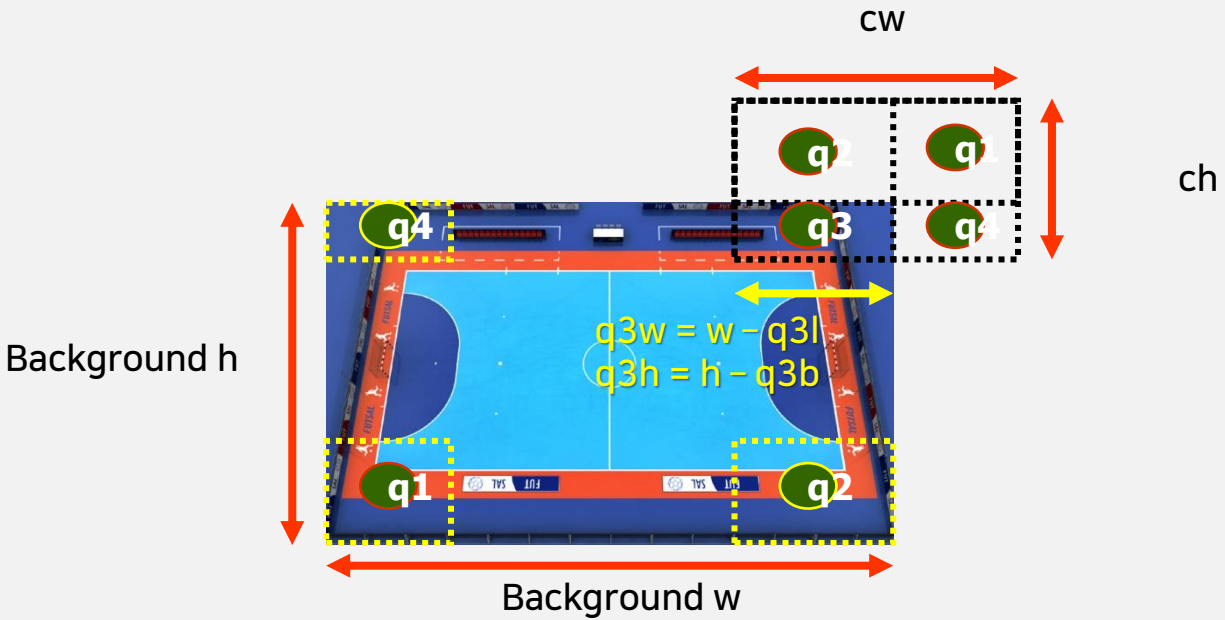
상하좌우 무한스크롤링 공식











play_mode.py


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

boy.py


```
def update(self):

    self.x = clamp(50.0, self.x, server.background.w - 50.0)
    self.y = clamp(50.0, self.y, server.background.h - 50.0)
```

```
def draw(self):
    sx, sy = get_canvas_width() // 2, get_canvas_height() // 2
    self.image.clip_draw(int(self.frame) * 100, self.action * 100, 100, 100, sx, sy)
```

background.py


```
class InfiniteBackground:

    def update(self, frame_time):

        # quadrant 3
        self.q3l = (int(server.boy.x) - self.cw // 2) % self.w
        self.q3b = (int(server.boy.y) - self.ch // 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 = ?
```

background.py

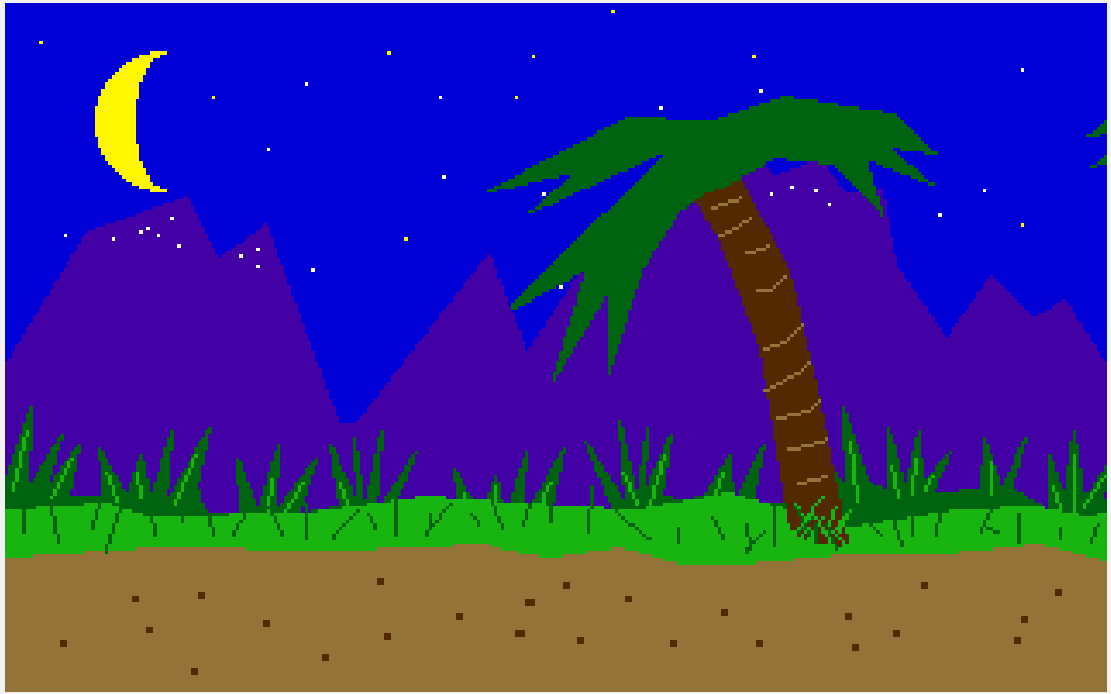

```
class InfiniteBackground:

    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)

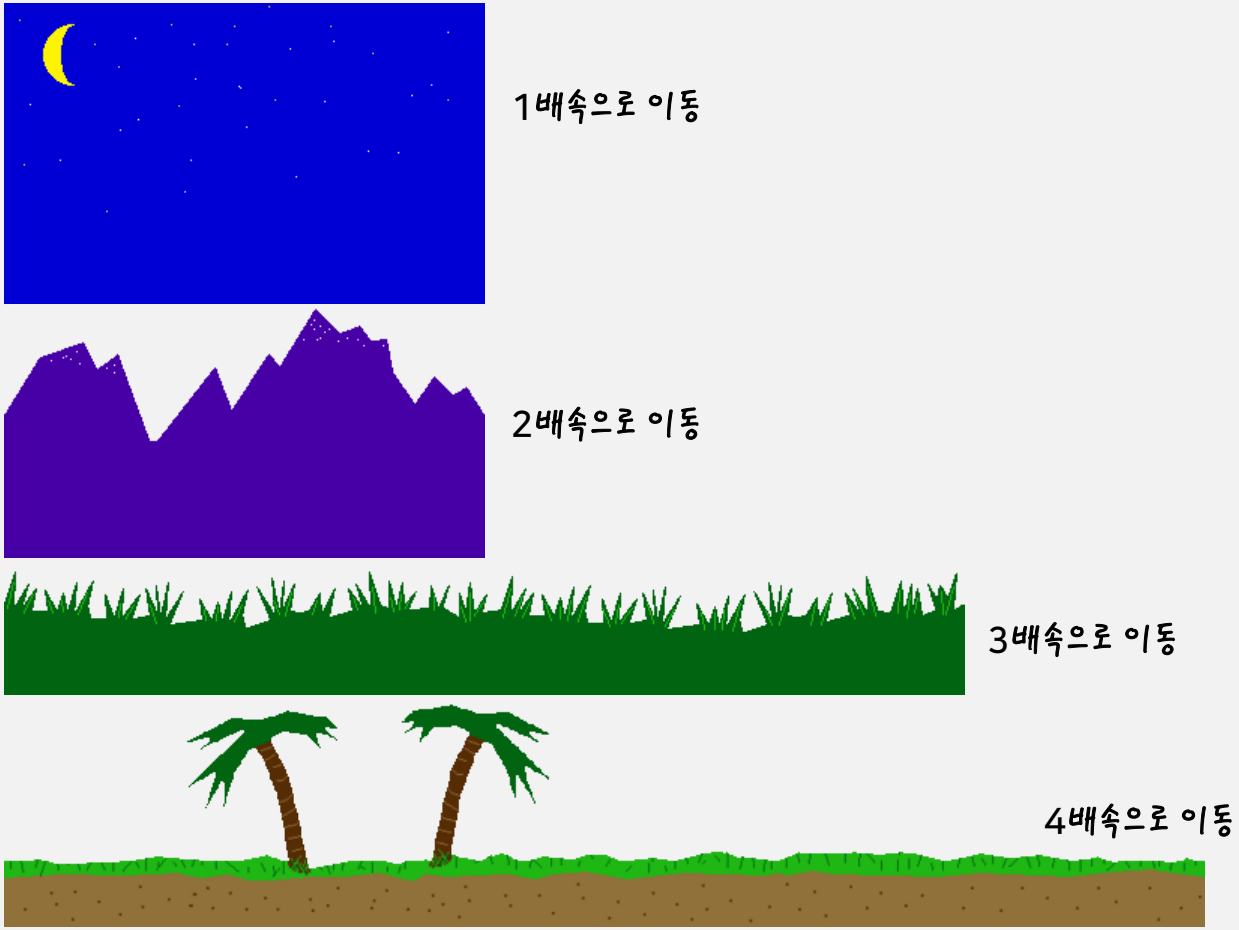
- 물체와 눈의 거리에 따라, 물체의 이동속도가 달라보이는 효과를 이용하여, 3차원 배경을 흉내 내는 기법.

- 1982년 “Moon Patrol”이라는 게임에서 세계 최초로 사용됨.

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



시차 스크롤링 방법



정답

```
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.cw // 2) % self.w
    self.q3b = (int(server.boy.y) - self.ch // 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.ch - self.q3h
    # quadrand 4
    self.q4l = 0
    self.q4b = self.q3b
    self.q4w = self.cw - self.q3w
    self.q4h = self.q3h
    # quadrand 1
    self.q1l = 0
    self.q1b = 0
    self.q1w = self.q4w
    self.q1h = self.q2h
```