

2023 전국 청소년 오픈SW GAME 코딩대회 작품

THE CHROMATIC

[청소년] 팀_셰이픽셀

팀장:김택우 팀원:박세연,박민주



목차

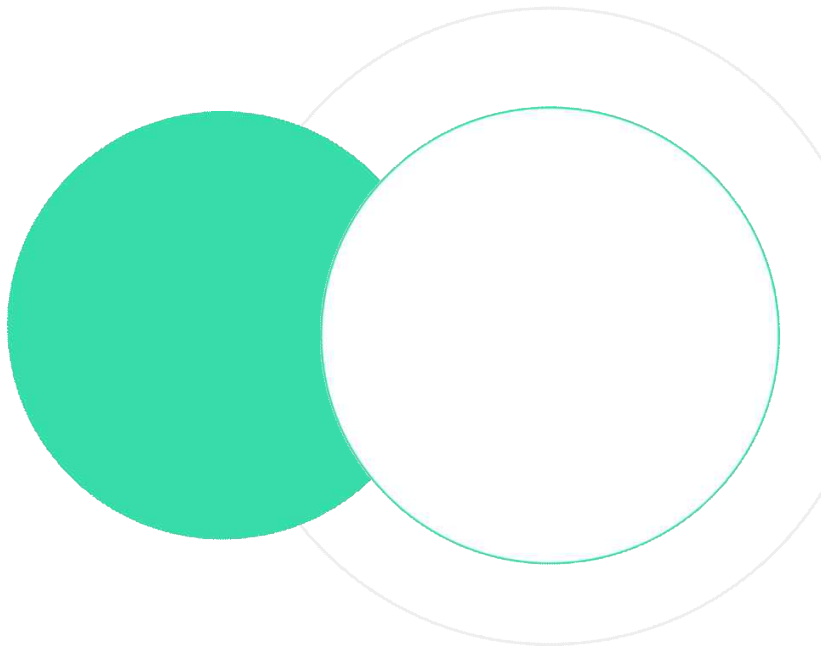
- 1 게임 소개
- 2 게임 방법
- 3 게임의 구조
- 4 마무리

팀원 역할

팀장_김택우 게임 기획,코딩,문서 작성

팀원_박세연 OST 제작, 효과음 제작, 개발, 문서 작성, 코딩

팀원_박민주 게임 스프라이트 제작, 게임 배경 제작



the Chromatic

게임 소개

주인공이 시간을 돌리는 능력을 사용하여,
맵의 장애물과 여러 적들을 피하며,
2D 액션 퍼즐 도트 게임입니다.



The
Chromatic

프로토타입 게임 소개

the Chromatic 인게임 화면



인트로



바탕화면



인 게임 화면



설정화면



일시정지 화면



게임오버

게임 방법

게임 조작 방법

이동



기본공격



점프&대화



아이템
사용



일시중지
&
게임종료



아이템
변경



스킬사용



기본 게임 조작 방법은

A, D / ←, → : 이동

W / ↑ / Space : 점프 / NPC와 대화

ESC : 일시중지 화면 표시 (인게임 내) / 게임 종료

J : 기본 공격 E : 아이템 사용 Shift : 아이템 변경

R : 스킬 사용(시간 되돌리기)

입니다.

게임 방법



주인공(플레이어)

직접적으로 게임 내에서 움직이고 행동 할 수 있는 플레이 캐릭터 입니다.



NPC 캐릭터

게임의 전반적인 설명과 조작법을 알려주는 캐릭터 입니다.



도움을 주는 NPC들



적대 캐릭터

플레이어 캐릭터를 추격, 공격하는 캐릭터 입니다.



적대 캐릭터들



피해 효과를 주는 장애물



아이템

잠긴 오브젝트를 열거나, 적대 캐릭터로 인해 입은 피해를 회복시키는 스프라이트입니다.



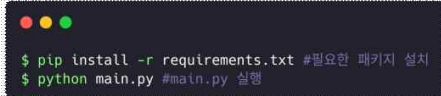
회복형 아이템



오브젝트 해제형 아이템

게임의 구조

패키지 설치 및
게임 실행

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of text: a command to install requirements and a command to run the main script.

```
$ pip install -r requirements.txt #필요한 패키지 설치  
$ python main.py #main.py 실행
```

`pip install -r requirements.txt` 명령어를 쉘에 입력하면
The Chromatic 게임의 필요한 패키지를 설치됩니다.

`python main.py` 명령어를 쉘에 입력하면 게임이 실행됩니다.

게임의 구조

전체 구조

The Chromatic 게임의 구조는 해당 사진과 동일합니다.

처음 `main.py` 파일에서 인수를 받아 `debug`, `fps`, `fullscreen`, `fullhd`, `quadhd`, `help` 명령어와 맞는 지 확인합니다.

이후 게임이 실행됩니다.



```

while CONFIG.is_running:
    CONFIG.clock.tick(CONFIG.FPS)
    CONFIG.surface.fill(CONST.COL_MAIN_BACKGROUND) # 정해놓은 백그라운드 색상으로 칠

    for event in pygame.event.get():
        match event.type:
            case pygame.KEYDOWN:
                match event.key:
                    case pygame.K_BACKQUOTE: # ` 키를 누를 시 인트로로 스왑
                        update_menu()
                        reload()

                    case pygame.K_ESCAPE: # ESC 키를 누를 시 게임 종료
                        CONFIG.is_running = False

    now = pygame.time.get_ticks() # 현재 시간 가져오기

    # region 인트로 : 움직 Saypixel ....
    if now - last >= cooldown: # 인트로 업데이트 간격보다 시간이 지난 경우
        if colon_count == 4: # 애니메이션이 끝난 경우 메인 메뉴로 이동할
            update_menu()
            reload()
            return

        last = now # 업데이트한 시간 갱신

        # 로고 애니메이션
        text = "Saypixel" + "." * colon_count
        colon_count += 1
    # endregion

    count += 1

    if count == 3: # 매 3프레임마다 호출
        count = 0
        player_icon.sprites.get_sprite_handler().sprite.update() # 스프라이트로 애니메이션 업데이트

    player_icon.move_x(1.2) # 1.2의 속도만큼 X 좌표로 이동
    player_icon.render() # 렌더링

    title = Font(FONTS.TITLE3, 48).render(text, CONST.COL_WHITE) # 로고 폰트 렌더링
    title_rect = title.get_rect(center=(480, 270)) # 로고의 좌표 & 크기 (Rect) 가져오기

    CONFIG.surface.blit(title, title_rect) # 로고를 화면에 렌더링
    CONFIG.update_screen() # 업데이트

process() # 공통 이벤트 처리

```

결과 ➡

매 프레임마다 정해놓은 백그라운드 색상으로 칠하고

인트로 로고를 업데이트 간격을 정해놓아 로고 애니메이션을 생성합니다.

로고 애니메이션이 완료된 경우 메인 메뉴 화면으로 이동됩니다.

또한 미니 플레이어의 스프라이트 애니메이션을 업데이트하여, 마치 미니 플레이어가 인트로 화면에서 뛰는 느낌을 줍니다.

로고 텍스트와 미니 플레이어를 렌더링하여 사용자에게 인트로 화면을 표시합니다.

게임의 구조

인트로

게임 시작전 인트로를 만드는 방식입니다.

Saypixel

게임의 구조

메인메뉴

메인 메뉴를 만드는 방식입니다.

```
def play_music():
    """음악을 재생합니다."""
    mixer.music.load("assets/audio/bg_daily.ogg")
    mixer.music.set_volume(SFX_VOLUME)
    mixer.music.play(-1)

global music_playing

need_to_exit = False # 메인 메뉴를 나가야 하는지 여부

background = pygame.image.load("assets/images/background.png") # 배경 불러오기
background = pygame.transform.scale(background, CONST.SCREEN_SIZE) # 화면 크기만큼 이미지 스케일링

# 제목
title = pygame.image.load("assets/images/menu_title.png")
title = pygame.transform.scale_by(title, 0.6) # 이미지 스케일링

# 버튼: 시작
button_play_image = pygame.image.load("assets/images/menu_play_rect.png")
button_play_image = pygame.transform.scale(button_play_image, (300, 80)) # 이미지 스케일링
button_play = Button(
    image=button_play_image,
    pos=(470, 380),
    text_input="시작",
    font=Font(Fonts.TITLE2, 60).to_pygame(),
    base_color="#ffffff",
    hovering_color="white",
)

# 버튼: 설정
button_settings_image = pygame.image.load(
    "assets/images/button_settings.png"
)
button_settings_image = pygame.transform.scale_by(button_settings_image, 0.4) # 이미지 스케일링
button_settings = Button(image=button_settings_image, pos=(580, 480))

# 버튼: 종료
button_exit_image = pygame.image.load("assets/images/menu_play_rect.png")
button_exit_image = pygame.transform.scale(button_exit_image, (200, 80)) # 이미지 스케일링
button_exit = Button(
    image=button_exit_image,
    pos=(420, 480),
    text_input="종료",
    font=Font(Fonts.TITLE2, 50).to_pygame(),
    base_color="#ffffff",
    hovering_color="white",
)

play_music() # 음악 재생
```

각 오브젝트의 이미지를 불러오고
스케일링하여 화면에 불러오고
렌더링할 수 있도록 합니다. ↗

```
while CONFIG.is_running and not need_to_exit:
    CONFIG.clock.tick(CONFIG.FPS)

    mouse_pos = CONFIG.get_mouse_pos() # 업스케일링 및 커매라 좌표가 보정된 마우스 좌표 가져오기

    if not music_playing: # 음악이 재생되고 있지 않은 경우 (메인 메뉴로 나가서 메인 메뉴로 돌아온 경우)
        music_playing = True
        play_music() # 음악 재생

    CONFIG.surface.blit(background, (0, 0)) # 배경 렌더링
    CONFIG.surface.blit(title, title.get_rect(center=(480, 140))) # 제목 렌더링

    for button in [button_play, button_settings, button_exit]:
        button.change_color(mouse_pos) # Hovering 시 (마우스 커서가 버튼에 올랐을 때) 색상 변경
        button.update(CONFIG.surface) # 버튼 렌더링

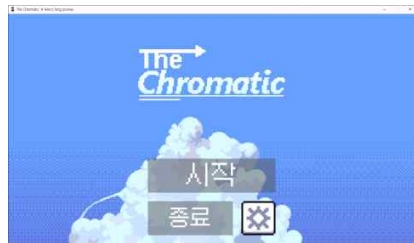
    CONFIG.update_screen() # 화면 업스케일링

process(process_menu) # 키보드 및 마우스 입력 이벤트 처리
```

↗ 메인 메뉴의 음악을 재생함으로써,

설정에서 정해진 음량으로 음량을
정하고

음악을 무한 반복합니다.



결과 →

```
def update_screen_resolution():
    """
    화면 해상도 업데이트
    """
    if CONFIG.is_fullscreen: # 전체화면으로 설정해야하는 경우
        CONFIG.screen = pygame.display.set_mode(CONFIG.window_size, pygame.FULLSCREEN)
    else:
        CONFIG.screen = pygame.display.set_mode(CONFIG.window_size) # 창모드로 설정
```

```
# 설정할 배경
background = pygame.image.load("assets/images/status3.png")
background = pygame.transform.scale_by(background, 0.45)
background = pygame.transform.scale(background, (background.get_width(), 580)) # 설정된 화면 크기 스케일링
background_rect = background.get_rect(center=(480 + CONFIG.camera_x, 270 + CONFIG.camera_y)) # 카메라 좌표 보정

# 텍스트: 화면
surface_resolution_2 = Font(Fonts.TITLE2, 36).render("화면", CONST.COL_WHITE)

# 화면 / 해상도: 이전
button_resolution_prev_image = pygame.image.load("assets/images/arrow_left.png")
button_resolution_prev_image = pygame.transform.scale_by(
    button_resolution_prev_image, 0.3
)
button_resolution_prev = Button(image=button_resolution_prev_image, pos=(340, 110))

# 화면 / 해상도: 다음
button_resolution_next_image = pygame.image.load("assets/images/arrow_right.png")
button_resolution_next_image = pygame.transform.scale_by(
    button_resolution_next_image, 0.3
)
button_resolution_next = Button(image=button_resolution_next_image, pos=(616, 110))

# 화면 / 전체화면
button_resolution_full_path = (
    "assets/images/button_checked.png"
    if is_fullscreen
    else "assets/images/button_unchecked.png"
)
button_resolution_full_image = pygame.image.load(button_resolution_full_path)
button_resolution_full_image = pygame.transform.scale_by(
    button_resolution_full_image, 0.4
)
button_resolution_full = Button(
    image=button_resolution_full_image,
    pos=(350, 170),
    text_offset=(100, 0),
    text_input="전체화면",
    font=Font(Fonts.TITLE2, 32).to_pygame(),
    base_color="#ffffff",
    hovering_color="white",
```

전체화면으로 설정해야하는 경우,

pygame.display.set_mode() 함수를
통해 전체화면으로 설정할지, 창모드로
설정할지 정합니다.

각 오브젝트의 이미지를 불러오고
스케일링하여 화면에 불러오고
렌더링할 수 있도록 합니다.

렌더링할 때 업스케일링 기능과
동적 카메라 좌표 기능에 의해
좌표가 의도하지 않게 어긋날 수
있으므로,

2가지 기능을 고려하여 카메라
좌표를 보정합니다.

게임의 구조

메인메뉴

설정

게임의 구조

메인메뉴

화면 해상도 업데이트

```
while CONF10.is_running and not need_to_exit:
    CONF10.clock.tick(CONF10.FPS)

    mouse_pos = CONF10.get_mouse_pos() # 입스케일링 및 카메라 좌표가 보정된 마우스 커서 좌표 가져오기

    surface_resolution = Font(Fonts.ILLUST, 36).render( # 해상도 텍스트 폰트 렌더링
        CONF10.resolution_to_str(resolution), CONST_CD1_WHITE
    )
    surface_audio_text = Font(Fonts.ILLUST, 36).render( # 음향 텍스트 폰트 렌더링 (백분율로 표시)
        str(int(round(volume, 1)) * 100) + "%", CONST_CD1_WHITE
    )

    # 렌더링 (기타와 자료 보정)
    CONF10.surface.blit(background, background_rect)
    CONF10.surface.blit(
        surface_resolution, surface_resolution.get_rect(center=(480 + CONF10.camera_x, 110 + CONF10.camera_y))
    )
    CONF10.surface.blit(
        surface_resolution_2, surface_resolution_2.get_rect(center=(340 + CONF10.camera_x, 60 + CONF10.camera_y))
    )
    CONF10.surface.blit(surface_audio, surface_audio.get_rect(center=(340 + CONF10.camera_x, 290 + CONF10.camera_y)))
    CONF10.surface.blit(
        surface_audio_text, surface_audio_text.get_rect(center=(480 + CONF10.camera_x, 340 + CONF10.camera_y))
    )

    for button in [
        button_resolution_prev,
        button_resolution_next,
        button_resolution_full,
        button_fps,
        button_audio_prev,
        button_audio_next,
        button_cancel,
        button_ok,
    ]:
        button.change_color(mouse_pos) # Hovering 시 (마우스 커서가 버튼에 올라왔을 때) 색상 변경
        button.update(CONF10.surface) # 버튼 렌더링

    CONF10.update_screen() # 화면 입스케일링

    process(process_menu) # 키보드 키 및 마우스 입력 이벤트 처리

# 렌더링하는 화면 복구
CONF10.surface = surface_recovered
need_to_exit = False
```

각 오브젝트 및 버튼을 각각 렌더링하여 화면에 표시합니다.

렌더링 할 때
pygame.Surface.blit() 함수를
사용하였습니다.

```
# 버튼 / FPS 표시
button_fps_path = (
    "assets/images/button_checked.png"
    if fps
    else "assets/images/button_unchecked.png"
)
button_fps_image = pygame.image.load(button_fps_path)
button_fps_image = pygame.transform.scale_by(button_fps_image, 0.4)
button_fps = Button(
    image=button_fps_image,
    pos=(350, 225),
    text_offset=(100, 0),
    text_input="FPS 표시",
    font=Font(Fonts.ILLUST, 32).to_pygame(),
    base_color="ffffff",
    hovering_color="white",
)

# 텍스트: 소리
surface_audio = Font(Fonts.TITLE2, 36).render("소리", CONST_CD1_WHITE)

# 소리 / 음향: 이전
button_audio_prev_image = pygame.image.load("assets/images/arrow_left.png")
button_audio_prev_image = pygame.transform.scale_by(button_audio_prev_image, 0.3)
button_audio_prev = Button(image=button_audio_prev_image, pos=(340, 340))

# 소리 / 음향: 다음
button_audio_next_image = pygame.image.load("assets/images/arrow_right.png")
button_audio_next_image = pygame.transform.scale_by(button_audio_next_image, 0.3)
button_audio_next = Button(image=button_audio_next_image, pos=(410, 340))

# 버튼: 취소
button_cancel_image = pygame.image.load("assets/images/menu_play_rect.png")
button_cancel_image = pygame.transform.scale(button_cancel_image, (100, 50))
button_cancel = Button(
    image=button_cancel_image,
    pos=(400, 460),
    text_input="취소",
    font=Font(Fonts.OPTION, 50).to_pygame(),
    base_color="ffffff",
    hovering_color="white",
)

# 버튼: 확인
button_ok_image = pygame.image.load("assets/images/menu_play_rect.png")
button_ok_image = pygame.transform.scale(button_ok_image, (100, 50))
button_ok = Button(
    image=button_ok_image,
    pos=(400, 460),
    text_input="확인",
    font=Font(Fonts.OPTION, 50).to_pygame(),
    base_color="ffffff",
    hovering_color="white",
)
```

결과



게임의 구조

메인메뉴

화면 해상도
업데이트

```

def process_menu(event: pygame.event.Event):
    """인게임 이벤트 처리용 (process() child 함수)"""

    global need_to_exit

    match event.type:
        case pygame.MOUSEBUTTONDOWN: # 마우스 클릭 시
            if button_play.check_for_input(mouse_pos): # 돌아가기
                need_to_exit = True # ESC 화면 나가기
                return

            if button_settings.check_for_input(mouse_pos): # 설정
                update_settings() # 설정창 표시
                return

            if button_fullscreen.check_for_input(mouse_pos): # 전체화면
                CONF10.is_fullscreen = not CONF10.is_fullscreen

                path = (
                    "assets/images/button_window.png"
                    if CONF10.is_fullscreen
                    else "assets/images/button_fullscreen.png"
                )
                image = pygame.image.load(path)

                button_fullscreen.change_image(image) # 이미지 변경
                update_screen_resolution() # 전체화면으로 화면 업데이트
                return

            if button_unmute.check_for_input(mouse_pos): # 음소거
                path = (
                    "assets/images/button_unmute.png"
                    if SFX.muted
                    else "assets/images/button_mute.png"
                )
                image = pygame.image.load(path)
                image = pygame.transform.scale_by(image, 0.2) # 이미지 스케일링

                SFX.control_mute() # 음소거 조절
                button_unmute.change_image(image) # 이미지 변경

                if not SFX.muted:
                    SFX.UNMUTED.play() # 음소거 해제 효과음 재생

                return

            if button_exit.check_for_input(mouse_pos): # 나가기
                from .ingame import ingame

                need_to_exit = True
                ingame.default_need_to_exit = True # 인게임도 나가서 액션 메뉴로 이동
                return

```

```

case pygame.KEYUP:
    match event.key:
        case pygame.K_ESCAPE: # ESC 키를 누른 경우 ESC 화면 나가기
            need_to_exit = True

```

```

surface_recovered: pygame.Surface
"""백업된 화면 (인게임 화면에서 덮어쓰는걸 방지)"""

```

```

need_to_exit = False
"""설정창을 닫아야 하는 경우"""

```

```

def update_pause_menu():
    """ESC 화면을 표시합니다. """

```

함수를 이용하여 각 버튼마다
클릭했는지 확인 후,
각 버튼에 맞는 설정 적용합니다.

게임의 구조

ESC 화면

키보드 및 마우스
입력 처리 이벤트

게임의 구조

ESC 화면

ESC화면 표시

1. 각 오브젝트를 스케일링 후 화면에 렌더링할 수 있도록 오브젝트 미리 초기화합니다.

2. 보정이 필요하다면 카메라 좌표 보정 및 이미지 스케일링을 합니다.

```
surface_recovered = CONF10.surface.copy() # 렌더링한 화면 복사 후 백업

# ESC 화면 배경
background = pygame.image.load("assets/images/status.png")
background = pygame.transform.scale_by(background, 0.35) # 이미지 스케일링
background = pygame.transform.rotate(background, 90) # 이미지를 시계방향으로 90도만큼 회전
background_rect = background.get_rect(center=(488 + CONF10.camera_x, 270 + CONF10.camera_y)) # 카메라 좌표 보정

# 버튼: 플레이기
button_play_image = pygame.image.load("assets/images/button_play.png")
button_play_image = pygame.transform.scale_by(button_play_image, 0.4) # 이미지 스케일링
button_play = Button(image=button_play_image, pos=(425, 216))

# 버튼: 설정
button_settings_image = pygame.image.load("assets/images/button_settings.png")
button_settings_image = pygame.transform.scale_by(button_settings_image, 0.4) # 이미지 스케일링
button_settings = Button(image=button_settings_image, pos=(531, 216))

# 버튼: 전체화면 / 종료
button_fullscreen_image = pygame.image.load("assets/images/button_fullscreen.png")
button_fullscreen_image = pygame.transform.scale_by(button_fullscreen_image, 2)
button_fullscreen = Button(image=button_fullscreen_image, pos=(371, 320))

# 버튼: 음소거 / 음소거 해제
button_unmute_image = pygame.image.load("assets/images/button_unmute.png")
button_unmute_image = pygame.transform.scale_by(button_unmute_image, 0.4) # 이미지 스케일링
button_unmute = Button(image=button_unmute_image, pos=(476, 320))

# 버튼: 계속하려면 나가기
button_exit_image = pygame.image.load("assets/images/button_exit.png")
button_exit_image = pygame.transform.scale_by(button_exit_image, 2)
button_exit = Button(image=button_exit_image, pos=(585, 320))
```

```
while CONF10.is_running and not need_to_exit:
    CONF10.clock.tick(CONF10.FPS)

    mouse_pos = CONF10.get_mouse_pos() # 컨트롤러와 및 카메라 보정이 포함된 마우스 좌표 가져오기

    # 전체화면 인 토글로
    path = (
        "assets/images/button_window.png"
        if CONF10.is_fullscreen
        else "assets/images/button_fullscreen.png"
    )
    image = pygame.image.load(path)
    button_fullscreen.change_image(image) # 모든 것에 맞춰 이미지 변경

    # 음소거 인 토글로
    path = (
        "assets/images/button_mute.png"
        if SPX.muted
        else "assets/images/button_unmute.png"
    )
    image = pygame.image.load(path)
    image = pygame.transform.scale_by(image, 0.2) # 이미지 스케일링
    button_unmute.change_image(image) # 모든 것에 맞춰 이미지 변경

    CONF10.surface.blit(background, background_rect) # ESC 화면 배경 렌더링

    for button in [
        button_play,
        button_settings,
        button_fullscreen,
        button_unmute,
        button_exit,
    ]:
        button.change_color(mouse_pos) # hovering 시 (마우스 커서가 버튼에 올라왔을 때) 색상 변경
        button.update(CONF10.surface) # 버튼 업데이트

    CONF10.update_screen() # 모든 컨트롤러링

    process(process_name) # 키보드 및 마우스 입력 이벤트 처리

# 렌더링하는 화면 복사
CONF10.surface = surface_recovered
need_to_exit = False
```

카메라 좌표 보정이 필요하다면 보정 후, 화면에 렌더링합니다.

게임의 구조

ESC 화면

ESC화면 표시

결과



```
class CONST:
    SCREEN_SIZE = [960, 540]
    """화면 (카메라) 크기 (960x540)"""

    SURFACE_SIZE = [1920, 1080]
    """세계 크기 (1920x1080)"""
```

먼저 화면 및 카메라 크기를
정해줍니다.

이후 주인공이 움직이는 맵의 크기도
정해줍니다.

카메라 좌표 선언

카메라의 위치가 어디까지
움직였는지에 대해 저장할 수 있는
좌표 변수를 선언합니다.

```
camera_x = 0
camera_y = 0
"""카메라 좌표 (동기화됨)"""
```

카메라 좌표 선언

```
# 카메라 좌표 업데이트
x_to_start = max(0, CONFIG.player_x - (CONST.SCREEN_SIZE[0] // 2)) # 카메라가 움직일 시작 범위 (x좌표)
x_to_end = CONST.SURFACE_SIZE[0] - CONST.SCREEN_SIZE[0] # 카메라가 최대 좌표로 이동하여 가만히 있게 될 시작 범위 (x좌표)

CONFIG.camera_x = min(x_to_start, x_to_end) # 카메라가 움직일 범위를 벗어나면 x_to_end에서 멈출
CONFIG.camera_y = 0 # y 좌표는 고정
```

매 프레임 업데이트마다 플레이어의
이동 거리에 대한 X좌표를 계산합니다.

```
cropped_screen = pygame.Surface(CONST.SCREEN_SIZE)
cropped_screen.blit(CONFIG.surface, (0, 0), CONFIG.get_camera_bound())
```

카메라 좌표가 플레이어 중심으로
움직이도록 설정 합니다.

```
def get_camera_bound() -> tuple[int, int, int, int]:
    """
    현재 카메라가 위치해있는 좌표와 크기 (Rect)를 가져옵니다.
    :return: 좌표와 크기 (Rect)
    """
    return (CONFIG.camera_x, CONFIG.camera_y, CONST.SCREEN_SIZE[0], CONST.SCREEN_SIZE[1])
```

카메라의 화면크기 만큼 잘라
세계 좌표를 카메라로 변환시킵니다.

게임의 구조

동적 카메라

세계 좌표는 움직이지 않고
고정으로 작동하며,

카메라 좌표만 움직이도록 만드는
동적 카메라를 구현하는 방식입니다.

게임의 구조

동적 카메라

해당 방법을 적용한다면,
해당 사진같이 모든 UI의 좌표를
변경하지 않아도 된다는 장점이 있어,

유지 보수에 용의하다는 장점이 있습니다.



↑ 카메라 좌표 이동을 이용한 동적 카메라 구현 방식.

```
class CONST:
    SCREEN_SIZE = [960, 540]
    """화면 (카메라) 크기 (960x540)"""
```

업스케일링

```
window_size = [1440, 810] # CONST.SCREEN_SIZE * 1.5
window_scale = 1.5

resolutions = [
    [480, 270],
    [960, 540],
    [1440, 810],
    [1920, 1080],
    [2560, 1440],
    [3840, 2160]
]
"""적용할 수 있는 해상도 배열"""

surface = pygame.Surface(CONST.SCREEN_SIZE)
"""크기가 [960, 540]으로 고정된 화면
월드 좌표는 [1920, 1080]에 한정됨
surface에 렌더링하고 업스케일링 후 screen으로 화면 표시"""

screen = pygame.display.set_mode(window_size)
"""업스케일링된 실제로 플레이어에게 보여주는 화면"""
```

화면 출력

```
transformed_screen = pygame.transform.scale(
    cropped_screen, CONFIG.window_size
) # 업스케일링
CONFIG.screen.blit(transformed_screen, (0, 0)) # 화면 표시
```

렌더링 되는 화면의 크기를 정합니다.



업스케일링 할 크기를 정해줍니다.
이후 배수만큼 늘릴 비율을 정해줍니다.



1. **pygame.transform.scale()**
함수로 **surface** 변수에 렌더링 후 업스케일링합니다

2. 업스케일링한 화면을 **screen** 변수에 저장하여 최종적으로 **screen** 변수로 화면 출력합니다.

게임의 구조

업스케일링

업스케일링은 게임 화면을 해상도에 맞춰
게임 화면 비율을 조정하는
방식입니다.

게임의 구조

마우스 좌표

마우스의 좌표를 설정합니다.

```
def get_mouse_pos() → tuple[int, int]:  
    """  
    업스케일링과 월드 좌표가 적용된 마우스 좌표 가져오기  
    :return: 업스케일링과 월드 좌표가 적용된 마우스 좌표  
    """  
  
    mouse_pos = pygame.mouse.get_pos()  
  
    # 업스케일링  
    upscaled = (  
        mouse_pos[0] // CONFIG.window_scale,  
        mouse_pos[1] // CONFIG.window_scale  
    )  
  
    # 월드 좌표 구현에 따른 오프셋 적용  
    cameraed = (  
        upscaled[0] + CONFIG.camera_x,  
        upscaled[1] + CONFIG.camera_y  
    )  
    return cameraed
```

1. **pygame.mouse.get_pos()** 함수로
창 기준 마우스 좌표를 가져옵니다

2. 사용자 지정 해상도에서 고정된 960x540 해상도 기준
마우스 좌표로 변환 해줍니다.

3. 마우스의 좌표는 카메라가 움직일 때마다
오프셋이 추가되어야하므로 카메라가 움직인만큼 오프셋 추가시켜줍니다.

게임의 구조

스킬 사용

시간을 되돌리는
스킬

```
# region 시간 관리
def process_time_event(self):
    """시간 관리 이벤트를 처리합니다."""
    if TimeEvent.is_rewind: # 시간을 되감아야 할 경우
        values = TimeEvent.rewind() # 이전의 플레이어와 적의 위치 데이터 가져오기

        if values is not None: # 되감을 값들이 아직 있는 경우
            for chr in values:
                x_prev = chr.character.x # 전 프레임의 캐릭터 X 좌표

                # 현재 프레임과 전 프레임의 캐릭터 X 좌표 비교하여 현재 속도값 지정
                velocity = -1 # 현재 속도값
                compared = chr.x - x_prev

                if compared == 0:
                    velocity = 0
                elif compared < 0:
                    velocity = 1

                if chr.character.name == "player": # 캐릭터가 플레이어인 경우
                    if velocity != 0: # 속도에 맞게 플레이어의 스프라이트 애니메이션 지정
                        chr.character.sprites.status = "walk"
                    else:
                        chr.character.sprites.status = "stay"

                chr.character.velocity_x = velocity # 캐릭터의 현재 속도값 지정
                chr.character.set_pos(chr.x, chr.y) # 캐릭터 좌표 지정

            else: # 더 이상 되감을 값들이 없는 경우
                TimeEvent.is_rewind = False # 시간 되감는 변수 갱신
                mixer.music.unpause() # 메인 음악 다시 재생

        else:
            characters = MapManager.current.enemies + [self.player]
            TimeEvent.update(characters) # 매 프레임마다 캐릭터들의 위치 지정

# endregion
```

```
case pygame.K_r: # 시간 관리
    if TextEvent.dialog_closed: # 대화창이 닫혀 있는 경우 (버그 방지)
        TimeEvent.is_rewind = True # 시간 관리 변수 갱신
        mixer.music.pause() # 메인 음악 일시정지
        SFX.REWIND.play() # 시간 관리 효과음 재생
```

1. 시간 관리 키를 누른 경우 대화창이 닫혀 있는지 확인합니다.
(버그 방지를 하기 위해서 입니다.)

2. 시간 관리 변수 갱신 후 메인 음악을 일시정지하고
시간 관리 효과음 재생 합니다.

게임의 구조

스킬 사용

시간을 되돌리는
스킬

결과



R키를 눌러 시간을 되돌릴수 있습니다.

```
class MutualText:
    text = ""
    prefix = ""
    delay = 1

    index = 0

    def __init__(self, text: str, prefix: str, delay: int):
        self.text = text
        self.prefix = prefix
        self.delay = delay
```

Mutual Text



Mutual Text 클래스는
상호작용할 텍스트를 구분하기
위해 만들어진 클래스로,
접두어를 기준으로 나눕니다.

게임의 구조

동적 애니메이션
텍스트

Mutual Text & Text

Text



Text는 *Mutual Text* 배열로,
렌더링할 텍스트 (문자열)
한 단위를 말합니다.

```
class Text:
    raw: str = ""
    """날 것의 텍스트, 접두어 포함"""

    pure: str = ""
    """순수 텍스트, 접두어 미포함"""

    has_prefix: bool = False
    """접두어가 포함되어있는가?"""

    texts: List[MutualText]
    """텍스트 리스트 (상호작용 배열)"""

    delay: int = 30
    """기본 지연 시간 (ms)"""

    FONT = fonts.TITLE2
    """기본 폰트"""

    PT = 18
    """기본 폰트 크기"""

    SYNTAX: dict[str, tuple[Font, int]] = {
        "A": (fonts.TITLE2, PT),
        "B": (fonts.TITLE2, PT + 4),
        "/": (fonts.DIALOG, PT - 4)
    }

    """구문"""

    """
    https://gist.github.com/ihoneymon/652be052a0727ad59601 : 일부 문법은 깃허브 마크다운 문법을 참조하였습니다.

    * : 굵게
    # : 크기
    / : 각개
    < : 줄바꿈 (New Line)
    """
```


게임의 구조

동적 애니메이션
텍스트

초기화

초기화

```
def __init__(self, text: str, delay: Optional[int] = 30):
    """
    Text 클래스를 생성합니다.
    :param text: 텍스트
    """
    self.texts = []

    self.raw = text
    self.pure = text # 일단 저장. 여차피 replace() 함수를 통해 접두어를 제거하기 때문.
    self.delay = delay

    for prefix in ["^", "#", "/"]:
        if prefix in self.pure: # 텍스트에 접두어가 있는 경우
            self.has_prefix = True
            self.pure = self.pure.replace(prefix, "") # 순수 텍스트는 접두어가 없으므로 접두어만 삭제
```

텍스트 안에 접두어가 있는 경우,
접두어만 삭제한 순수 텍스트
저장합니다.

```
is_started = False # 접두어가 있는가?
mutual_text_ = "" # 각 접두어별 텍스트
prefix = "" # 접두어

for ch in self.raw:
    match ch:
        case "^" | "#" | "/": # 접두어
            is_started = not is_started # 접두어가 없었으면 시작, 있었으면 끝

            if mutual_text_: # mutual text가 비어있지 않은 경우
                mutual = MutualText(mutual_text_, prefix, delay) # 상호작용 텍스트 생성
                mutual_text_ = ""
                self.texts.append(mutual) # texts 배열에 MutualText 추가

            prefix = ch if is_started else "" # 접두어 저장

        case _: # 접두어가 아닌 순수 텍스트인 경우
            mutual_text_ += ch

# 접두어가 없는 일반 텍스트가 있는 경우
if mutual_text_: # mutual text가 비어있지 않은 경우
    mutual = MutualText(mutual_text_, "", delay)
    self.texts.append(mutual)
```

게임의 구조

동적 애니메이션
텍스트

문자 렌더링

```
def write(  
    self,  
    mutual: MutualText,  
    index: int,  
    text_position: tuple[int, int],  
    char_position: tuple[int, int],  
    surface: pygame.Surface,  
) → tuple[int, int]:  
    """  
    특정 문자를 렌더링 (출력)합니다.  
    :param mutual: 상호작용 테스트  
    :param index: 상호작용 텍스트의 index  
    :param text_position: 화면 위치  
    :param char_position: 문자 위치 (렌더링할 위치)  
    :param surface: 화면  
    :return: 갱신해야할 문자 위치  
    """
```

점두어별 맞는 폰트 지정하고 pt를 픽셀로 변환합니다.

```
if index + 1 > len(mutual.text): # 상호작용 텍스트 끝의 범위를 벗어난 경우 종료  
    return # 이 상황은 대화를 너무 빨리 넘길 때 발생함  
  
ch_x = char_position[0]  
ch_y = char_position[1]  
  
font = Text.FONT  
pt = Text.PT  
px = 0  
  
if mutual.prefix in Text.SYNTAX: # 점두어별 맞는 폰트 지정  
    syntax = Text.SYNTAX[mutual.prefix]  
    font = syntax[0]  
    pt = syntax[1]  
  
px = pt / 3.0 * 4.0 # pt를 픽셀로 변환
```

```
chs = mutual.text[index] # 문자  
  
if chs == "<": # 줄바꿈 점두어  
    ch_x = text_position[0] # x 좌표를 처음 좌표로 이동  
    ch_y += px # y 좌표를 일정 이동  
  
    # x 좌표와 y 좌표를 각각 이동시켜 마치 줄바꿈이 된 것처럼 행동  
else:  
    ch = Font(font, pt).render(chs, CONST.COL_BLACK) # 감정적인 텍스트 생성  
    surface.blit(ch, (ch_x, ch_y)) # 화면에 렌더링  
  
    ch_x += px # x 좌표를 일정 이동  
  
return (ch_x, ch_y) # 갱신해야할 문자 위치를 반환
```

줄바꿈 점두어면 좌표를 지정하고, 감정인 문자 생성 후 화면에 렌더링합니다.

X 좌표를 변환한 픽셀 수만큼 일정 이동하고, 새롭게 갱신된 문자 좌표 반환합니다.

게임의 구조

동적 애니메이션
텍스트

TextCollection

TextCollection: **Text** 클래스 배열,
여러 말을 해야할 때 쓰이는 클래스

Text 속 **MutualText**를 열거합니다.

접두어를 확인하여 각 접두어별 맞는 폰트 크기를 지정하고, 폰트 크기를 픽셀로 변환합니다.

문자의 크기를 텍스트 너비에 더한 후 말풍선 너비와 비교하여 범위를 벗어난 경우, 텍스트에 줄바꿈 접두어와 문자를 추가하여 자동 줄바꿈을 기능합니다.

그 다음에는 범위를 벗어나지 않은 경우 텍스트에 문자만 추가하고 기존 텍스트를 줄바꿈이 들어간 텍스트로 변경합니다.

반복문을 돌면서 텍스트 너비 초기화 후 **TextCollection** 클래스에 **Text** 클래스 추가하고 현재 출력할 **Text** 클래스와 다음 출력할 **Text** 클래스를 지정합니다

```
def __init__(
    self,
    texts: list[Text],
    sign_width: int,
    pos: Optional[tuple[int, int]] = (320, 180),
):
    """
    Text 배열을 TextCollection로 초기화합니다.
    :param texts: Text 배열
    :param sign_width: 말풍선 너비 (줄바꿈할 때 쓰임)
    :param pos: 텍스트를 표시할 절대좌표
    """

    self.textList = []
    self.position = pos

    pt = Text.PT # 폰트 단위
    text_width = 0.0 # 현재 텍스트 너비
    modified_text = "" # 줄바꿈이 들어간 텍스트
```

```
for text in texts: # Text
    for mutual in text.texts: # Mutual Text
        if mutual.prefix in Text.SYNAX: # 접두어별 맞는 폰트 지정
            syntax = Text.SYNAX[mutual.prefix]
            pt = syntax[1]

        px = pt / 3.0 * 4.0 # pt를 픽셀로 변환

        for ch in mutual.text: # Mutual Text의 문자
            text_width += px

            if text_width > sign_width: # 말풍선 너비보다 텍스트 너비가 더 큰 경우 (범위를 벗어난 경우)
                modified_text += "<" # 텍스트에 줄바꿈 추가
                text_width = 0 # 줄바꿈 했으니 텍스트 너비 초기화

            if ch == "\n": # 공백이 아닌 경우
                modified_text += ch # 텍스트에 문자 추가
                text_width += px
            else:
                modified_text += ch # 텍스트에 문자 추가

        mutual.text = modified_text # 기존 텍스트를 줄바꿈이 들어간 텍스트로 변경
        modified_text = "" # 줄바꿈이 들어간 텍스트 초기화

    text_width = 0 # 텍스트 너비 초기화 (새로운 Text 등장)
    self.textList.append(text) # TextCollection에 Text 추가

if len(self.textList) >= 1: # 텍스트 배열 크기가 1 이상인 경우
    self.current = self.textList[0] # 현재 텍스트 지정
if len(self.textList) >= 2: # 텍스트 배열 크기가 2 이상인 경우
    self.next = self.textList[1] # 다음 텍스트 지정
```

대화 (텍스트) 이동

```
def jump_to_next(self) → bool:
    """
    다음 대화로 이동 (텍스트를 넘김)
    """

    if len(self.textList) ≤ self.index + 1: # 모든 텍스트를 다 본 경우
        self.index = 0 # index 초기화

        # 현재 출력할 텍스트와 다음 출력할 텍스트 지정
        if len(self.textList) ≥ 1:
            self.current = self.textList[0]
        if len(self.textList) ≥ 2:
            self.next = self.textList[1]

        return False # 다음 대화로 이동할 수 없으므로 False 반환

    self.index += 1

    # 다음 출력할 텍스트를 현재 출력할 텍스트로 지정
    # 다음 출력할 텍스트 지정
    self.current = self.next
    self.next = (
        self.textList[self.index + 1]
        if self.index + 2 ≤ len(self.textList)
        else None
    )

    return True # 다음 대화로 이동할 수 있으므로 True 반환
```

모든 텍스트를 다 본 경우
텍스트 순서 관련 변수인
index 변수를 초기화합니다.

모든 텍스트를 다 보지 않은
경우 **index** 변수 값을 추가
하여 텍스트 순서를
갱신합니다.

게임의 구조

동적 애니메이션
텍스트

대화 (텍스트) 이동

게임의 구조

동적 애니메이션
텍스트

TextEvent

```
class TextEvent(object):  
    """인게임 내에서 처리될 공통 텍스트 이벤트"""  
  
    dialog_delayed = True  
    """대화창의 텍스트 출력이 지연되어야 하는가? (REFRESH 이슈 & 텍스트 미리 출력 대응)"""  
  
    dialog_paused = True  
    """대화창의 텍스트 출력이 완성되었는가?"""  
  
    dialog_closed = True  
    """대화창 텍스트가 닫혀있는가?"""  
  
    dialog: TextCollection  
    """대화창 (Text 배열)"""
```

**TextEvent 클래스는
대화 이벤트 처리입니다.**

**각 변수에 따라 대화창 출력
을 지연시키지 않고
완성시켜야 할지,**

**다음 대화창으로 넘겨야할지
이벤트를 처리합니다.**

```
@classmethod  
def process_next_event(cls):  
    # 다음 대화창 이벤트 구현  
  
    if cls.dialog_paused: # 대화창의 텍스트 출력이 완성되었을 때  
        if cls.dialog_index == 0 and cls.dialog_closed: # 대화창이 처음으로 출력될 때  
            cls.dialog_closed = False # 대화창 열림  
            cls.dialog_paused = False # 텍스트 출력 미완성  
  
        else:  
            if cls.dialog.jump_to_next(): # 대화창을 넘길 때  
                cls.dialog_paused = False # 텍스트 출력 미완성  
  
            else: # 대화창의 텍스트가 더이상 없을 때  
                cls.dialog_closed = True # 대화창 닫힘  
                cls.dialog_paused = True # 텍스트 출력 완성  
  
                cls.dialog_delayed = True # 텍스트 출력 지연  
  
    else:  
        cls.dialog_delayed = False # 텍스트 미리 모두 출력  
        cls.dialog_paused = True # 텍스트 출력 완성
```

게임의 구조

동적 애니메이션
텍스트

대화 애니메이션 이벤트

write_each()

각 문자를 출력하는 함수입니다.
이 때, 출력하기 위해 기다리는 지연을 구현하기 위해 `threading.Timer()` 클래스를 이용하여 비동기적으로 구현하였습니다

process_animation_event()

사용자가 지정한 화면이 없으면 화면을 기본
지정된 화면으로 나오고,
텍스트 출력이 미완성인 경우에는
`write_each()` 함수 실행하여 텍스트를
처음부터 출력합니다.

```
@classmethod
def process_animation_event(cls, surface: pygame.Surface = None):
    """
    화면에 애니메이션 이벤트 구현 (V2)
    pygame surface, 렌더링할 화면 (기본: COMFIS.surface)
    """
    mutual_index = 0 # MutualText 클래스의 index
    mutual_text_index = 0 # MutualText 클래스의 텍스트 index
    ch_x = cls.dialog.position[0] # 대화창 시작 X좌표
    ch_y = cls.dialog.position[1] # 대화창 시작 Y좌표

    if cls.dialog.is_none: # 대화를 난무 할지 아닌지 여부
        return # 출력할 내용이 없으므로 종료

    def write_each():
        """각 문자 렌더링 (출력)"""
        nonlocal mutual_index, mutual_text_index, ch_x, ch_y

        if mutual_index + 1 > len(cls.dialog.current.texts): # MutualText 배열의 범위를 벗어난 경우
            cls.dialog.paused = True # 텍스트 출력 완료
            cls.dialog.delayed = True # 텍스트 지연

            return # 함수를 재귀로 세 번만 호출을 통한 렌더링으로 반환

        mutual = cls.dialog.current.texts[mutual_index] # MutualText 클래스의 가져오기
        new_ch_pos = cls.dialog.current.write(
            mutual, mutual_text_index, cls.dialog.position, (ch_x, ch_y), surface
        ) # 텍스트 렌더링 후 위치 갱신

        if new_ch_pos.is_none: # 대화를 난무 할지 아닌지 여부
            return # 출력할 내용이 없으므로 종료

        ch_x = new_ch_pos[0]
        ch_y = new_ch_pos[1]

        mutual_text_index += 1

        if mutual_text_index + 1 > len(mutual.text): # MutualText 클래스의 문자를 범위를 벗어난 경우
            mutual_index += 1 # 새로운 MutualText 클래스를 가져오기 위해 delay를 1씩 증가
            mutual_text_index = 0 # 새로운 MutualText 클래스를 가져오므로 index 초기화

        if mutual_delay > 0 and cls.dialog.delayed: # 지연 시간이 되는 경우
            delay_per_second = mutual_delay / 1000.0 # 1s 내 threading.Timer 클래스의 시간 변수는 단위가 초이므로 단위를 ms에서 초로 변환
            Timer(delay_per_second, write_each).start() # threading.Timer 클래스를 특정 시간 후 비동기적으로 함수 재귀
        else:
            write_each() # 특정 시간 없이 함수 재귀

    if surface.is_none: # 사용자 지정한 화면이 없으므로
        surface = COMFIS.surface # 렌더링할 화면을 기본 지정한 화면으로 지정

    if not cls.dialog.paused: # 텍스트 출력이 미완성인 경우
        write_each() # 텍스트를 처음부터 출력
```

게임의 구조

대화 애니메이션
이벤트

인게임 대화 애니메이션
이벤트

```
case CONST.PYGAME_EVENT_DIALOG: # 텍스트 애니메이션 이벤트
    if CONFIG.is_interactive(): # 플레이어와 상호작용이 가능한 경우
        TextEvent.process_animation_event(self.sign.image) # 텍스트 애니메이션 이벤트 처리
```

플레이어와 상호작용이 가능한 경우에만

TextEvent.process_animation_event() 함수를 이용하여
텍스트 애니메이션 이벤트 처리합니다.

```
def speech_npc(self) -> bool:
    """
    대화 키를 누른 경우 대화 관련 이벤트를 처리합니다.
    :return: NPC와 대화를 하고 있는지 여부
    """
    for npc in MapManager.current.NPCs:
        if npc.is_bound(88, 88): # 플레이어가 NPC 일정 범위 안에 들어와 있는 경우
            if TextEvent.dialog is None and npc.dialog is not None: # 현재 대화창이 NPC의 대화로 설정 되어있지 않은 경우
                TextEvent.dialog = npc.dialog
            if TextEvent.NPC is None or TextEvent.NPC is not npc: # 주변에 NPC가 있는 경우 대화하는 NPC 변수 갱신
                TextEvent.NPC = npc

            if TextEvent.dialog is not None: # 현재 대화창이 설정되어 있는 경우
                TextEvent.process_next_event() # 다음 대화창 이벤트 처리

                if TextEvent.dialog.delayed: # 다음 대화로 진행되어야 하는 경우
                    self.sign.refresh() # 말풍선 재렌더링

            if not TextEvent.dialog_closed: # 대화창이 닫혀 있지 않은 경우
                pygame.time.set_timer(CONST.PYGAME_EVENT_DIALOG, 1, 1) # 대화 하는동안 관련 이벤트 처리

            # 주연공 애니메이션을 기본으로 설정
            MapManager.current.player.sprites.status = "stay"

            return True

    return False
```

현재 맵에 있는 NPC가
일정 범위에 해당하는 경우,

대화 애니메이션 이벤트를
처리합니다.

```
if CONFIG.is_interactive():
    match event.key:
        case pygame.K_SPACE | pygame.K_w | pygame.K_UP:
            speeched = self.speech_npc() # 키가 눌려진 대화 관련 이벤트 처리 후 대화를 하고 있는지 여부를 변수로 저장

            if not speeched: # 주변에 NPC가 있는 경우 대화하는 NPC 변수 갱신
                TextEvent.NPC = None
```

대화 이벤트 처리 후 대화를 하고 있는지의
여부를 **speeched** 변수에 저장합니다.
주변에 대화할 NPC가 없는 경우 대화하는
NPC 변수를 초기화합니다.

**NPC에게 대화 텍스트를
위에 설명한 클래스 형식에
맞게 적용됩니다.**

```
self.emilia.dialog = TextCollection(  
    [  
        Text("*안녕!*"),  
        Text("나는 에밀리아야."),  
        Text("*J키*는 #기본공격#이야!"),  
        Text("그럼 즐거운 여행되길 바래!")  
    ],  
    self.sign.width  
)
```

결과



게임의 구조

대화 애니메이션
이벤트

인게임 대화 애니메이션
이벤트



플레이 영상

<https://youtu.be/hkwRBZQYmuw>



감사합니다!

게임속 요소들인 BGM,스프라이트 소스, 배경 이미지,코딩,효과음은 순수창작으로 제작됨을 알려드립니다.

게임 라이브러리인 pygame과 디버그 라이브러리인 icecream의 도움을 받았으며, 글꼴(폰트)로 도스고딕체와 도스샘물체(leedheo 제작) 등을 사용하였습니다.

마지막으로 Python 3.11 버전을 기반으로 코딩을 작성함을 알려드립니다.

