REPORT

과목명 | 오픈소스SW활용

학과 | 정보통계학과

학번 | 32212924

이름 | 윤지수

제출일 | 2023년 05월 31일

1. 프로젝트 배경

- 학과 '데이터마이닝' 수업에서 Clustering(군집화)에 대해 배우는 과정 중 12개의 클래식 곡을 학생들에게 들려주고 선호도를 조사한 자료를 통해 군집분석을 실습해보았던 경험에서 '군집화의 결과를 프로그램 개발에 직접 이용해보자'라는 아이디어가 떠올랐다. 그래서 python을 이용한 간단한 게임제작이라는 주제로부터 출발하여 '군집분석'의 결과를 추가한, 전공 분야를 결합한 게임을 제작해보자는 생각까지 도달하였다. 옛날부터 클래식에 많은 관심이 있는 사람으로서 이번 게임의 첫 버전에 사용될 음악은 '클래식'으로 한정하고, 만약 상업적인 측면을 고려한다면 그 타겟층은 '클래식 음악에 관심이 있는 사람'으로 설정한다. 결과적으로, 클래식 외의 다양한 주제로 게임을 만들어서 단순한 게임 프로그램의 개발에서 나아가, 음악의 각 분야에 대한 사용자들의 선호도 데이터를 얻어, 다른 분야에도 활용할 수 있는 기회를 제공할 수 있다는 생각으로 이번 프로젝트를 추진하게 되었다.

- (실제 수업시간에 작성했던 선호도 조사지)

선호도 조사지 (2023.4.20)

이름	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
윤재우	5	4	3	10	7	6	4	5	8	8	10	10
KRR	7	2	3	5	6	8	3	2	10	7	8	9
Kong	7	5	2	6	8	9	3	0	5	4	7	2
GOEUN	7	0	2	8	5	4	1	3	8	3	5	7
KJK	3	5	3.5	6	7	8	8.5	8	9	9.5	8	9.5
KSW	3	7	2	9	8	2	5	6	9	4	3	2
YCY	7	10	5	9	4	10	0	4	8	7	9	3
BJH	4	6	3	7	7	8	3	4	6	4	10	5
CJW	8	3	1	10	7	2	2	7	9	10	4	1
YS	4	8	3	9	5	5	4	10	6	8	6	5
강원빈	5	0	1	4	8	10	6	3	10	3	8	4

※ 더욱 자세한 프로젝트의 배경, 목표, 공개 시의 기대 효과는 '프로젝트 제안서'를 참고해주세요!

2. 제공기능

- 게임의 메인 화면에서 사용자는 4가지(한국어, 영어, 일본어, 중국어(간체)) 언어 중 하나를 선택하여 플레이할 수 있다.

- 사용자는 12개의 곡에 대한 선호도를 1점~10점까지 입력하여 군집분석을 통해 자신과 비슷한 경향을 가진 사람의 군집이 있다면 그 군집에 대한 추천 곡을 플레이 가능한 곡으로 받을 수 있다.

- 사용자가 추천받은 3개의 곡 이외의 다른 곡을 플레이할 수 있도록 추천 곡을 메시지로 보여주고 플레이 가능한 곡은 모든 곡으로 보여준다.

- 게임이 시작되면 사용자는 가운데의 원 모양을 회전해가며 동서남북에서 날아오는 부채꼴들의 색을 원의 색과 형태에 일치시키면 점수를 획득할 수 있다.

- 게임이 끝나면 사용자는 자신이 획득한 점수와 해당 곡을 플레이 함으로서 얻을 수 있는 최고점수(퍼펙트 점수)를 화면에서 볼 수 있다.

- 만약 사용자가 퍼펙트 점수를 얻게 되면 곡의 선택창에서 해당 곡에 '퍼펙트 클리어!'라는 글자가 생긴다.

- 위와 같은 과정으로 사용자는 추천받은 3개의 클래식 곡을 포함한 총 12개의 클래식 곡을 들으며 재미있는 리듬게임을 플레이할 수 있게 된다.

3. 시스템 구조 및 설계

① 개략적인 게임의 구조

- 게임 초기 화면: 시작, 추천&도움말, 나가기, 언어
    - 추천&도움말: tkinter를 이용한 사용자의 음악 선호도 조사를 한 후, 군집분석을 통해 상위 3곡을 사용자에게 추천해주는 메시지메 출력한다. 정상적으로 진행된 경우, 선호도 조사 창이 종료되고, 메인 게임창에서 도움말을 볼 수 있다.
    - 언어: 사용자는 한국어, 영어, 중국어(간체), 일본어의 총 4가지 언어로 게임을 플레이할 수 있다.
- 음악 선택 화면: 음악 리스트, 최고점수, 플레이, 뒤로가기
    - 음악 리스트: M1, M2, M3 … M12까지 총 12개의 클래식 곡을 사용자가 선택하여 게임을 플레이 할 수 있다.
    - 최고점수: 사용자가 게임을 해서 얻은 최고점수를 표시한다.
    - 뒤로가기: 게임 초기 화면으로 돌아간다.
- 게임 결과 화면: 만점, 점수, 곡 선택, 다시하기
    - 만점: 이 곡을 플레이해서 얻을 수 있는 최고점수를 표시한다.
    - 점수: 사용자가 플레이한 게임의 점수를 표시한다.
    - 곡 선택: 곡 선택 화면으로 돌아간다.
    - 다시하기: 동일한 곡으로 게임을 한 번 더 플레이한다.

② 자세한 코드 설명을 포함한 시스템 설계 정보

- Game Start : def __init__(self)

  - 이 코드는 __init__ 함수가 시작되는 부분으로, Game 클래스의 초기화를 하게된다.
  - pg.init(): Pygame 라이브러리를 초기화한다.
  - pg.mixer.init(): 사운드 믹서를 초기화한다.
  - pg.display.set_caption(TITLE): 게임 창의 제목을 설정한다.
  - self.screen = pg.display.set_mode((WIDTH, HEIGHT)): 게임 창의 크기를 설정한다.
  - self.screen_mode = 0: 게임 창의 모드를 나타내는 변수로, 초기 모드를 0으로 설정하였다. (0: 로고 화면, 1: 로고2 화면, 2: 메인 화면)
  - self.screen_value = [-ALPHA_MAX, 0, 0, 0]: 화면 관리 값을 나타내는 변수로, ALPHA_MAX는 화면 전환에 사용되는 알파 값의 최대값이다.
  - self.clock = pg.time.Clock(): FPS(초당 프레임 수)를 제어하기 위한 Clock 객체를 생성하여 게임 루프 내에서 FPS를 조절하는 데 사용된다.
  - self.start_tick = 0: 게임 타이머의 시작 시간을 나타내는 변수로, 0으로 초기화하였다.
  - self.running = True: 게임의 초기화 여부를 나타내는 변수로, 이를 True로 설정함으로써 게임이 초기화된 상태에서 시작되게 한다.
  - self.language_mode = 1: 언어 모드를 나타내는 변수로, 0은 영어, 1은 한국어이다.
  - self.song_select = 1: 선택된 곡을 나타내는 변수로, 초기값은 1로 설정하였다.
  - self.load_date(): 데이터를 로드하는 함수로, 게임에서 사용되는 데이터를 로드하는 역할을 한다.
  - self.new(): new 함수를 호출하고, 게임 시작 시 초기 상태를 설정하는 역할이다.
  - pg.mixer.music.load(self.bg_main): bg_main 변수에 저장된 배경 음악 파일을 불러온다.

⇨ 위의 코드는 Game 클래스의 초기화 작업을 수행하는 부분으로, 게임을 시작하기 전에 필요한 설정을 초기화하고 데이터를 로드하는 등의 작업을 한다.

- Data Loading: def load_date(self)
    - self.dir = os.path.dirname(__file__): 현재 스크립트 파일의 디렉토리 경로를 self.dir 변수에 저장한다.
    - self.csv_dir = os.path.join(self.dir, 'song'): CSV 파일의 디렉토리 경로를 self.csv_dir 변수에 저장한다.
    - with open(os.path.join(self.fnt_dir, 'language.ini'), "r", encoding='UTF-8') as language_file: 언어 관련 설정이 포함된 'language.ini' 파일을 연다.
    - song_lists = [i for i in os.listdir(self.sng_dir) if i.split('.')[-1] in music_type]: os.listdir를 사용하여 self.sng_dir에서 파일 목록을 가져온 다음, 파일 확장자가 music_type에 포함되는지 확인하여 필터링한다.

⇨ 위의 코드는 게임에서 사용할 폰트, 이미지, 사운드, 곡 등의 데이터를 로드하는 작업을 수행한다.

- Game Initialize: def new(self)
    - self.circle_dir = 1: 원형(서클)의 방향을 나타내는 변수인 self.circle_dir을 1로 초기화한다. 흰색을 기준으로 아래, 오른쪽, 위, 왼쪽을 나타내는 숫자이다.
    - self.shots = pg.sprite.Group(): 샷을 관리하기 위한 스프라이트 그룹인 self.shots를 생성한다.

⇨ 위의 코드는 게임을 초기화할 때 필요한 변수 및 그룹을 생성하고 초기값을 설정한다.

- Game Loop : def run(self)
    - while self.playing: 게임이 실행 중인 동안 무한 루프를 실행한다.
    - self.events(): 키 입력, 마우스 동작 등의 이벤트를 처리하는 메서드를 호출한다.
    - self.update(): 게임 로직, 객체 상태 변화 등을 처리하며 게임 상태를 업데이트하는 메서드를 호출한다.
    - self.draw(): 게임 화면을 그리는 draw 메서드를 호출한다.
    - pg.mixer.music.fadeout(600): 현재 재생 중인 음악을 페이드아웃 시킨다. 600ms 동

안 점점 소리가 줄어들며 종료된다.

⇨ 위의 코드는 게임 루프를 구성하여 게임의 실행을 관리한다. 루프를 반복하면서 이벤트 처리, 게임 상태 업데이트, 그리기 작업 등을 수행하고 화면을 업데이트 한다. 이를 통해 게임의 지속적인 실행과 사용자와의 상호작용이 가능하다.

- 선호도 조사를 위한 UI 구현: def draw_screen(self)
  - play_music: 주어진 곡 번호에 해당하는 음악을 재생한다.
  - stop_music: 현재 재생 중인 음악을 정지한다.
  - check_user_input: 사용자 입력을 주기적으로 확인하여 재생이 완료되면 저장 버튼을 활성화한다.
  - validate_value: 입력된 값이 1에서 10 사이의 부동소수점 값인지 확인한다.
  - save_values: 입력된 선호도 값을 저장하고, 상위 3곡을 가져와서 표시한다.
  - get_top_songs: 선호도 조사 데이터를 기반으로 상위 3곡을 추천합니다.
  - change_language: 선택된 언어에 따라 UI의 텍스트를 변경한다.
  - display_top_songs: 추천된 상위 3곡을 선택된 언어로 새 창에 표시한다.

⇨ 사용자가 음악 선호도를 입력하고 결과를 저장하며, 추천된 상위 3곡을 표시하는 기능을 가진 음악 선호도 조사 UI를 구현하는 부분이다.

- 선택한 음악 데이터 로드: def load_songData(self)
  - with open(...) 구문을 사용하여 선택한 파일을 연다.
  - self.song_dataPath[self.song_select - 1]: self.song_dataPath에서 song_select에 해당하는 인덱스의 파일 경로를 가리키는 코드로, 파일은 "r" 모드(읽기모드)로 열리며, UTF-8 인코딩으로 처리된다.

⇨ 선택한 파일을 열어서 노래 데이터를 로드하고, 각 데이터를 알맞게 처리하여 self.song_data 리스트에 저장한다.

- 현재 게임 틱에 해당하는 샷 데이터를 생성하고, 게임에 해당 샷을 추가: def create_shot(self)
    - 현재 게임 틱(self.game_tick)이 self.song_data[self.song_dataIndex][0]보다 크거나 같을 경우 실행되고, 이는 게임 틱이 해당 샷의 시간에 도달했다는 것을 의미한다.
    - self.song_data[self.song_dataIndex][1]이 -1이 아닐 경우는 샷 데이터가 존재하는 경우를 의미한다.
    - shot_num: 해당 샷 데이터의 개수를 저장한다.
    - shot_data: self.song_data[self.song_dataIndex][shot + 1]의 값을 저장한다. 여기서 shot은 0부터 시작하므로 인덱스를 맞추기 위해 1을 더한다.
    - obj_shot: Shot 클래스의 인스턴스를 생성하여 저장한다. 이 때, 생성자의 인자로는 샷 데이터의 색상(shot_data[0]), 모드(shot_data[1]), 방향(shot_data[2]), 속도(shot_data[3])를 전달한다.
    - 만약 self.song_data[self.song_dataIndex][1]이 -1인 경우는 샷 데이터가 없는 경우를 의미하고, 이는 해당 노래의 모든 샷 데이터를 처리했다는 것이다.
    - file_list: 'score:현재점수:최고점수' 형식의 문자열을 저장합니다.
    - self.song_dataPath[self.song_select - 1]의 파일을 'w+' 모드로 열어서 song_file에 저장하면 파일에 작성이 가능하다.
    - self.song_highScore[self.song_select - 1] 값을 현재 점수(self.score)로 업데이트한다.

⇨ 위의 코드는 현재 게임 틱에 해당하는 샷 데이터를 생성하고, 게임에 해당 샷을 추가한다. 또한, 모든 샷 데이터를 처리한 경우에는 최고 점수를 업데이트하고 파일에 저장한다.

- 스프라이트를 그리는 기능: def draw_sprite(self, coord, spr, alpha = ALPHA_MAX, rot = 0)
    - rot이 0인 경우(회전이 없는 경우)
        - spr의 투명도를 alpha로 설정한다.
        - self.screen에 spr를 주어진 좌표(coord)에 그린다.
        - round 함수를 사용하여 좌표를 반올림한다.

- rot이 0이 아닌 경우(회전이 있는 경우)
  - spr를 rot 각도만큼 회전한 rotated_spr를 생성한다.
  - rotated_spr의 투명도를 alpha로 설정한다.
  - self.screen에 rotated_spr를 주어진 좌표(coord)에 그린다.
  - 회전 중심점을 계산하기 위해, 회전 이전 스프라이트의 너비와 높이를 이용한다.
  - rotated_spr의 너비와 높이를 이용하여 중심점을 계산하고, 이를 이동시켜 회전 중심점을 조정한다.
  - round 함수를 사용하여 좌표를 반올림한다.

⇨ 위의 코드는 스프라이트를 그릴 수 있으며, 회전과 투명도를 적용할 수 있다. 회전이 없는 경우에는 원래 스프라이트를 주어진 좌표에 그리고, 회전이 있는 경우에는 회전한 스프라이트를 조정된 중심점으로 그린다.

- 텍스트를 그리는 기능: def draw_text(self, text, size, color, x, y, alpha = ALPHA_MAX, boldunderline = False)
  - 주어진 폰트 파일(self.gameFont)을 이용하여 pg.font.Font 객체를 생성한다.
  - 폰트의 스타일을 설정한다.
  - font.render(): 텍스트를 렌더링한 text_surface를 생성한다.
  - text_surface.get_rect(): 텍스트의 사각형 영역(text_rect)을 가져온다.
  - text_rect.midtop: x와 y 좌표를 이용하여 텍스트의 가로 중앙 상단 위치를 지정하고, round 함수를 사용하여 좌표를 반올림하여 텍스트의 위치를 설정한다.
  - self.screen.blit(): text_surface를 text_rect 위치에 그린다.

⇨ 위의 코드는 주어진 텍스트를 원하는 크기와 색상으로 그릴 수 있게 한다. 또한, 폰트 파일이 존재하지 않는 경우에는 기본 폰트를 사용하며, 텍스트에는 경우에 따라 밑줄과 굵기를 변경할 수 있다.

- 스프라이트 시트 이미지를 처리하여 개별 이미지를 반환: class Spritesheet

  - __init__(self, filename): filename은 스프라이트 시트 이미지 파일의 경로이다. 해당 파일을 pg.image.load() 함수를 사용하여 로드하고, convert() 메서드를 사용하여 이미지를 변환하여 self.spritesheet 속성에 저장한다.
  - get_image(self, x, y, width, height): 스프라이트 시트에서 특정 영역을 잘라내어 개별 이미지를 반환한다.

    (x, y: 자르기 시작할 위치의 x 및 y 좌표)

    (width, height: 자르고자 하는 영역의 너비와 높이)
  - image.blit(self.spritesheet, (0,0), (x, y, width, height)): self.spritesheet에서 지정된 영역을 image에 복사하고, image를 반환한다.

⇨ 위 코드는 프라이트 시트에서 필요한 이미지 영역을 잘라내어 반환하는 기능을 한다. get_image() 메서드를 호출하여 스프라이트 시트에서 원하는 이미지를 얻을 수 있다.

- Shot 클래스: class Shot(pg.sprite.Sprite)

  - __init__(self, game, color, mode, direction, speed): game은 게임 객체, color는 샷의 색상 값(WBDR), mode는 샷의 모드(DRUL), direction은 샷의 방향(DRUL), speed는 샷의 이동 속도이다.
  - update(self): 프레임마다 호출되며, correct 변수에 따라 점수를 계산하고, 효과음을 재생하는 등 스프라이트의 상태를 업데이트한다. 또한, 스프라이트가 화면 밖으로 나가면 삭제된다.

- 메인 루프 관련

  - game = Game(): Game 클래스의 인스턴스를 생성하여 game 변수에 할당한다.
  - while game.running: 게임이 실행 중인 동안 game.run() 메서드를 반복적으로 호출하여 게임을 실행한다.
  - pg.quit(): 게임 루프가 종료된 후, Pygame 라이브러리를 종료한다.

4. 구현 (발생한 문제 및 해결내용)

- IndexError: index 20 is out of bounds for axis 0 with size 20

```
# 새로운 데이터에 Id 열 번호 설정
new_data['Id'] = range(prefer_data['Id'].max() + 1, prefer_data['Id'].max() + 2)
```

➔ 새로운 선호도 조사 자료를 기존 데이터프레임에 추가할 때, 사용자의 입력 값은 list형태로 들어가지만, id열은 따로 추가를 해줘야 한다. 따라서 기존 데이터프레임에 있는 Id 열의 최대값을 기준으로 Id 열 번호를 설정하여 중복되지 않는 연속적인 번호가 설정되는 방법으로 오류를 해결하였다.

- 선호도 조사 창의 스크롤바 추가 문제

```
# 선호도 입력 프레임 생성
preference_frame = tk.LabelFrame(root, text=translations["Preference"][current_language])
preference_frame.pack(pady=20)

# 선호도 입력 레이블 및 엔트리 생성
entries = []
for i in range(1, 13):
    label = tk.Label(preference_frame, text=f"M{i}")
    label.grid(row=i, column=0, padx=10, pady=5)
    entry = tk.Entry(preference_frame)
    entry.grid(row=i, column=1, padx=10, pady=5)
    entries.append(entry)

scroll_frame = ttk.Frame(root)
scroll_frame.pack(fill=tk.BOTH, expand=True)

error_label = tk.Label(root, text="", fg="red")
error_label.pack()

# 저장 버튼 생성
save_button = tk.Button(root, text=translations["Save"][current_language], command=save_values)
save_button.pack(pady=20)
```

➔ 선호도 조사 창에 스크롤바를 추가하는 과정에서 오류가 많이 발생하여 선호도 조사 창의 구조를 다르게 하면 스크롤 하지 않아도 한 화면에 다 들어올 수 있으므로 코드를 수정하여 문제를 해결하였다.

- 사용자가 값을 입력하고 종료버튼을 눌렀을 때 창이 무한 생성되는 현상

```
# 새로운 유저의 데이터를 기존 데이터에 추가
update_data = pd.concat([prefer_data, new_data], ignore_index=True)

# 업데이트된 데이터를 csv 파일로 저장
csv_file_path = os.path.join(self.csv_dir, 'moonbeat_music_preference.csv')
update_data.to_csv(csv_file_path, index=False)

num = num+1
```

```
if num == 1:
    root.withdraw()
    root.deiconify()
else:
    root.mainloop()
```

➔ 변수 num과 if문을 이용하여 사용자의 입력 값이 제대로 입력된 후, 군집분석을 통한 3곡이 추천되고 저장도 잘 되었으면 num에 +1을 한다. 그래서 num 값이 1이면 선호도 조사 창은 종료하고 메인 게임창만 남아있도록 하고, 값이 제대로 입력되지 않을 경우에만 선호도 조사 창이 계속 뜨도록 코드를 수정하여 문제를 해결하였다.

- 불러올 데이터의 경로가 개발자의 컴퓨터에서만 작동될 수 있도록 설정되어 있는 문제

```
def load_date(self): ########################## Data Loading
    self.dir = os.path.dirname(__file__)

    ### font
    self.fnt_dir = os.path.join(self.dir, 'font')
    self.gameFont = os.path.join(self.fnt_dir, DEFAULT_FONT)
    self.csv_dir = os.path.join(self.dir, 'song')

    with open(os.path.join(self.fnt_dir, 'language.ini'), "r", encoding = 'UTF-8') as language_file:
        language_lists = language_file.read().split('\n')

    self.language_list = [n.split("_") for n in language_lists]
```

```
def get_top_songs(new_pref):
    os.chdir(f"{self.csv_dir}")
    prefer_data = pd.read_csv("moonbeat_music_preference.csv")
```

➔ load_date(self) 함수를 정의하여 이 코드를 다운받는 사용자의 컴퓨터 환경에 맞추어 데이터와 파일들을 불러올 수 있도록 코드를 수정하였다.

- 구문오류는 없지만 논리오류가 발생한 경우

```
def save_values():
    global new_pref  # 전역 변수를 사용하도록 선언
    global num
    global new_data
    global prefer_data
    global value
```

```
if not error_message_shown:
    global top_songs
    new_pref = values  # 입력받은 값 저장
```

➔ 선호도 조사 창과 군집분석을 구현하는 화면은 다른 창에서 코드를 작성하고 메인 게임 코드에 추가하는 방식으로 게임을 개발하였는데, 이 과정에서 구문오류는 발생하지 않았지만 논리오류가 발생하였다. 여기서 발생한 오류는 대부분 필요한 변수들을 전역변수로 생성하지 않아 오류가 발생함을 알게 되어 global (변수명)을 선언하는 방법으로 오류를 해결하였다.

- 음악을 재생하면서 값을 입력하려고 하면 '응답없음'이라고 뜨는 문제

```
def check_user_input():
    root.update()  # UI 업데이트
    if pygame.mixer.music.get_busy():
        root.after(100, check_user_input)  # 100ms마다 사용자 입력 체크
    else:
        save_button.config(state=tk.NORMAL)  # 저장 버튼 활성화
```

➔ Tkinter의 'after()' 메서드를 사용하여 음악이 재생되는 동안에도 사용자가 값을 입력할 수 있도록 하여 문제를 해결하였다.

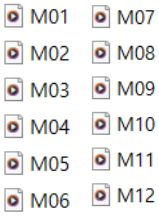
- 선호도 조사 창이 종료될 때, 메인 게임 창이 함께 종료되는 현상

```
num = num+1

# 선호도 조사 창 숨기기
root.withdraw()
# 메인 게임 창 보이기
root.deiconify()
```

➔ 코드 끝에 있는 root.destroy()는 메인 게임 창을 닫게 하므로 제거하고, 대신 선호도 조사 창을 숨기는 root.withdraw()와 메인 게임 창을 보여주는 root.deiconify()를 사용하여 문제를 해결하였다.

- 사용자가 곡을 선택하는 화면에서 곡이 로딩될 때 이름 순서대로 나타나지 않고 M1, M10, M11, M12, M2, M3, M4, M5, M6, M7, M8, M9로 나타나는 문제



➔ 이는 python에서 숫자를 문자로 인식하여 발생한 문제임을 알고, 01, 02, 03과 같은 방식으로 음악 파일 이름을 수정하여 문제를 해결하였다.

- 사용자가 잘못된 값을 입력했을 때의 오류 메시지와 군집분석을 통한 3개의 추천 곡을 보여주는 메세지가 제대로 번역되지 않는 문제

```
def validate_value(value):
    global current_language
    try:
        value = float(value)
        if value < 1 or value > 10:
            raise ValueError
        return value
    except ValueError:
        error_message = translations["Invalid value. Please enter a float value between 1 and 10."][current_language]
        error_label.config(text=error_message)
        return None
```

```
def display_top_songs(top_songs):
    global top_songs_label, current_language
```

➔ 'current_language'변수가 전역변수로 선언되지 않아서 문제가 발생함을 알고, global current_language를 추가하여 문제를 해결하였다.

- 선호도 조사 창이 처음 켜졌을 때, default가 한국어로 설정되어 있어서 만약 한국인 사용자가 한국어를 언어로 직접 선택하지 않으면 사용자가 언어를 선택하지 않았기 때문에 오류가 발생

```
# 언어 선택 콤보 박스 생성
language_combo = ttk.Combobox(language_frame, values=list(translations["Moonbeats - Music Preference Survey"].keys()))
language_combo.set('------')  # 기본값 설정
language_combo.pack(side=tk.LEFT, padx=10)
```

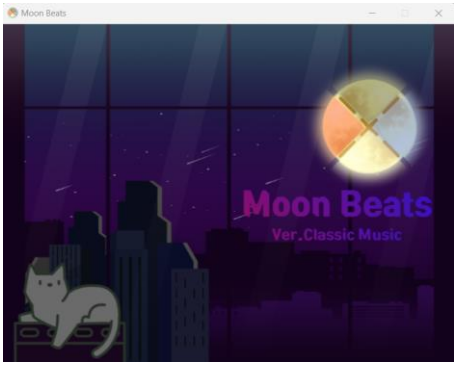
➔ 선호도 조사 창이 처음 켜졌을 때, 언어 선택 창의 default를 '------' 로 나타내어 한국어를 사용하는 사용자도 한국어를 직접 선택하게 한다. 그런 다음, 다른 언어와 마찬가지로 선호도 값을 입력하게 되어 정상적으로 선호도 조사가 진행되도록 하여 오류를 해결하였다.

5. 테스트 결과 및 분석

- 시작 로딩 화면

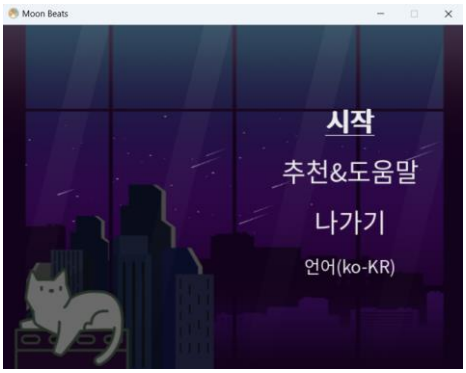
(저장해 놓은 이미지 파일대로 로고가 잘 출력된다.)

- 메인화면



(저장해 놓은 배경화면이 잘 출력되고, 로고 이미지가 잘 움직인다.)

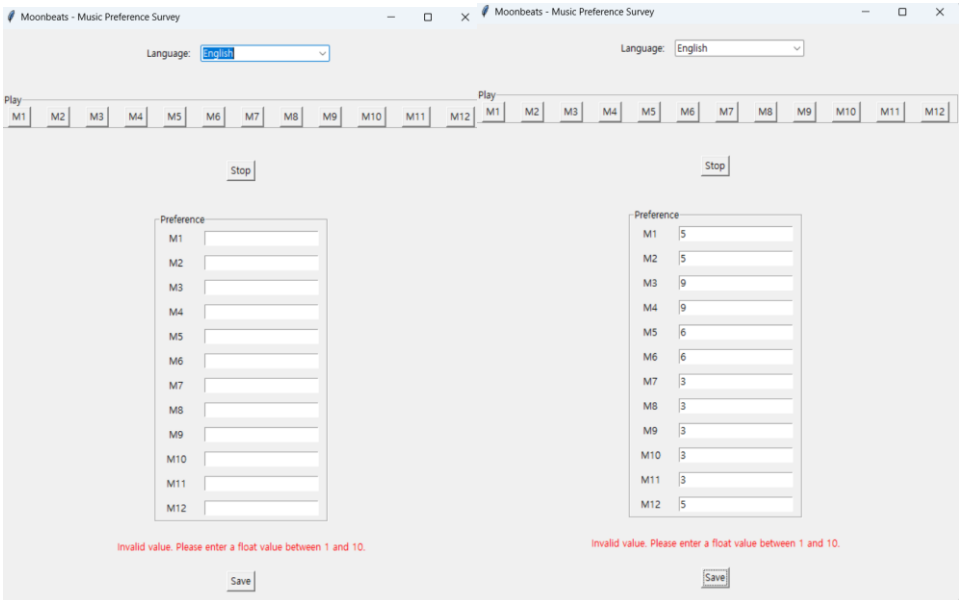
- 메인



(모든 메뉴들이 정상적으로 잘 작동한다.)

- 선호도 조사 화면

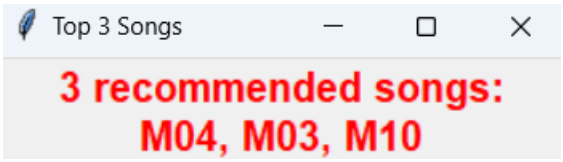
('추천&도움말'을 클릭하면 선호도 조사 화면이 정상적으로 등장하고, 언어 선택 칸이 default값인 '-----'로 설정되어 있다.)



(언어를 영어로 변경했을 때, 잘 번역되는 것을 알 수 있고, 값을 입력하지 않고 저장을 눌렀을 때 출력되는 오류 메시지도 정상적으로 번역되는 것을 알 수 있다.)

(값을 정상적으로 입력하고 저장을 누르면 선호도 조사 창이 종료되고 메인 게임창은 정상적으로 작동하는 것을 확인하였다.)

- 군집분석을 통한 추천 화면



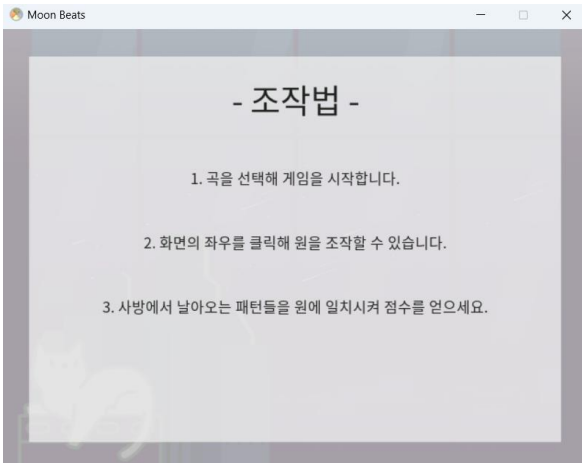
(언어를 영어로 선택했을 때, 군집분석을 통한 추천 곡 3곡이 정상적으로 화면에 번역되어 나타난다.)

- 사용자 입력 데이터 저장 확인

17	16	4	8	3	2	9	8.5	4	3	9	3	4	2.5
18	17	6	2	7	7.5	7	6	5	3	5.5	7	4	1
19	18	8.5	1	2	3	3	3	3	8	2.5	4	9	7.5
20	19	7.5	2	3	5	3	3	4.5	9	2.5	5.5	8.5	6
21	20	3.5	9.5	8	7	8.5	8.5	6	3	9	6.5	3	2
22	21	5	5	9	9	6	6	3	3	3	3	3	5

(사용자가 입력한 선호도 값이 song폴더에 있는 'moonbeat_music_preference.csv'파일에 정상적으로 추가된다.)

- 도움말 화면



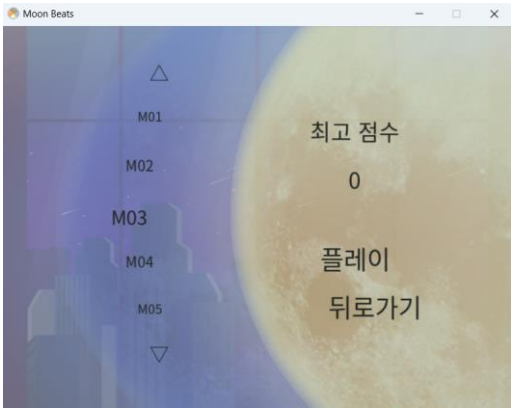
(선호도 조사를 마치고 추천 곡이 정상적으로 출력된 후, 선호도 조사 창을 종료하면 게임의 조작법이 나오는 것을 확인하였다.)

(다른 언어를 선택했을 때도 조작법이 잘 번역되어 나오는 것을 확인하였다.)

- 언어 변환 화면

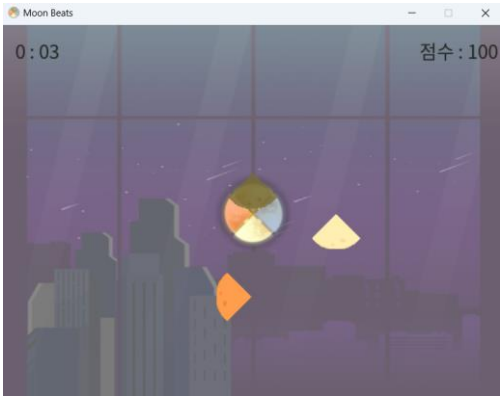
(한국어, 영어, 중국어(간체), 일본어 총 4개의 언어로 메뉴가 잘 번역되는 것을 확인하였다.)

- 곡 선택 화면



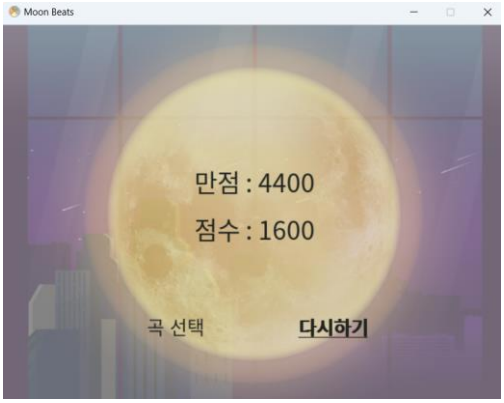
(클래식 곡들이 이름 순서대로 나열됨을 알 수 있다.)

- 게임 플레이 화면



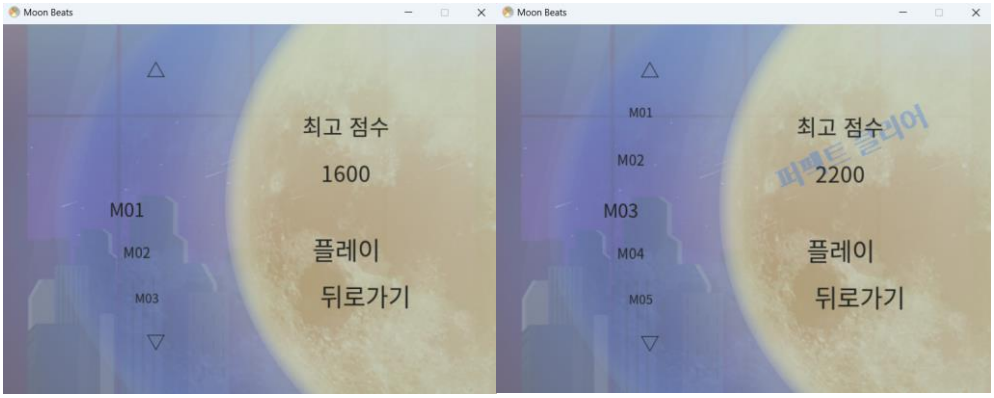
(모양을 맞추면 점수가 정상적으로 올라감을 확인하였다.)

- 게임 결과 화면



(만점 점수와 얻은 점수가 잘 출력되고, '곡 선택', '다시하기' 기능이 잘 작동함을 확인하였다.)

- 게임 진행 후 점수 표시 & 퍼펙트 스코어 표시



(게임을 플레이 한 후, 기존 점수보다 높은 점수를 얻었다면 그 점수로 표시되고, 만점과 동일한 점수를 얻은 경우 '퍼펙트 클리어'라는 글씨가 파란색으로 점수 화면에 출력됨을 확인하였다.)

⇨ 그동안 게임을 플레이하면서 얻은 점수를 포함한 게임 로그를 확인할 수 있는 기능을 추가해도 좋을 것 같다.

- 전반적인 게임의 목표 달성 여부를 체크해 보면 아래의 표와 같다.

번호	프로젝트 목표 달성 여부 판단 기준	달성여부 (O / X)
1	프로그램 구성에 필요한 클래식 곡들을 직접 연주한 녹음파일이 pygame에서 지원하는 포맷으로 준비되어 있는가?	O
2	초기 실행화면에 4개의 언어(한국어, 영어, 일본어, 중국어(간체))가 정상적으로 나타나고 선택 가능한가?	O
3	사용자가 게임 방법을 알 수 있는 도움말이 보기 좋게 정돈되어 나타나는가?	O
4	게임 실행 시 초기화면에 사용자의 음악 선호도를 입력 받을 수 있는 인터페이스가 잘 구성되어 있는가?	O
5	클래식 음악 선호도에 관련된 군집분석이 잘 수행되었는가?	O
6*	군집분석 결과에 따라 사용자에게 3개의 음악이 추천되는 메시지가 나오고, 그 3개의 곡을 포함한 12개의 곡을 플레이할 수 있는가?	O
7	리듬게임 과정 중 사용자가 리듬을 맞추면 점수가 정상적으로 오르는가?	O
8	게임을 시작하여 종료하는 모든 과정에서 오류가 발생하거나 논리오류가 발생하는 부분을 모두 해결하였는가?	O
9	게임에 사용되는 모든 요소에서의 저작권, 라이선스 등에서 문제가 없도록 명시하였는가?	O
10	테스트 이후 수정된 게임에 테스트시 이루어진 피드백 내용이 반영되었는가?	O

* 6번은 프로그램을 제작하며 기존 '프로젝트 제안서'의 기준에서 변경된 평가 기준

6. 후기

python을 이용하여 해보았던 것은 기본적인 패키지들을 사용한 간단한 프로젝트, 통계적인 패키지 사용 등 밖에 없던 상태에서 다양한 패키지를 사용한 게임 개발을 혼자 하는 것은 정말 많은 힘이 들었다. 하지만 이전에 파이썬 프로젝트를 팀플로 진행해본 경험이 있기 때문에 이번 프로젝트는 혼자의 힘으로 많이 생각해보고 찾아보며 성장하고 싶었다. 혼자 프로그램을 개발할 수 있었던 가장 큰 도움은 인터넷에 무수히 많이 나와 있는 정보들(이번 프로젝트에서 사용한 기본게임 코드도 이에 해당한다.)과 chat gpt등이 있다. 물론, 이런 정보를 무조건적으로 믿는 건 옳지 않음을 알기에 그만큼 더 많이 찾아보고, 코드를 하나하나 뜯어보며 오류를 해결하려 많이 노력했다. 이를 통해 구현하고자 하는 기능을 코드로 옮기는 것이 생각보다 어렵다는 것, 코드를 다 완성하고 나서도 끝이 아니라 예상하지 못한 오류가 존재할 수도 있다는 점을 새로 알 수 있었다. 이와 관련하여 평소에 진행했던 프로젝트들은 보통 개발자 자신이 구현하면 끝이기때문에 파일 경로 같은 경우도 개발자의 컴퓨터 환경에만 맞추면 됐지만, 이제는 '사용자'가 게임을 플레이 할 입장에서 프로젝트를 진행하다 보니 어떤 사용자가 이 코드를 다운받아도 그 사용자의 컴퓨터 환경에서 파일 경로 등이 제대로 작성되어야 한다는 점이 새롭게 다가왔다. 혼자 프로젝트를 진행하면서 많은 우여곡절이 있었지만 그것을 해결하는 과정에서 값진 경험을 했기 때문에 앞으로 비슷한 프로젝트를 진행하더라도 절차에 맞추어 무리 없이 진행할 수 있을 거라는 생각이 들었다. 비록 첫 게임 개발이므로 A부터Z까지 내 힘으로 만든 프로그램은 아니지만, '모방은 창조의 어머니이다.'라는 말이 있듯이(물론 프로젝트에 있어서의 모방은 어디까지나 개발자가 수정을 허용하고 사용자가 그 출처를 밝혔다는 전제이다.), 이번 프로젝트의 경험을 토대로 나중에는 정말 처음부터 끝까지 온전히 나의 힘으로 프로그램을 개발해볼 수 있을 거라 생각한다. 또한, 이러한 점이 바로 오픈소스를 이용한 프로젝트의 가장 큰 장점임을 알 수 있었다.

7. 참고 문헌

- https://printed.tistory.com/ (게임 기본 틀 출처)
- https://smecsm.tistory.com/147 (파이썬 숫자 정렬 문제 해결)
- https://python-forum.io/thread-17965.html (파이게임 내부에서 tkinter 사용과 관련한 자료)
- https://stackoverflow.com/questions/60817473/warning-uncondensed-distance-matrix-in-python (군집화 과정에서 발생한 warning 해결)

8. 소스코드 GitHub링크 & 프로젝트 소개 동영상 링크

- 소스코드 GitHub링크: https://github.com/YjisuY/MoonBeats-ClassicVer.git
- 프로젝트 소개 동영상 링크: https://youtu.be/lvui6jX8O7Y