**BMS 패턴의 구성과 패턴 자동 제작 알고리즘**

‘구조로 알아보는 BMS포맷의 역사와 발전방향’ 글로 BMS의 개략적인 소개를 하였다. BMS 포맷은 텍스트 파일과 구조가 동일하므로 접근과 편집이 쉽다. 구조에 대한 지식이 있다면 메모장으로도 데이터를 작성, 편집할 수 있다. 물론 이는 다소 불편하기에 BMS 편집 프로그램을 주로 이용하지만, 그 BMS 편집 프로그램을 작성하기 위해서는 BMS 파일을 텍스트로 접근해야 한다.

BMS 포맷은 건반형 리듬게임을 위한 오픈 소스 포맷이다. 텍스트 파일이 리듬게임 데이터가 되는 것에 의문이 들 수 있다. 사실 BMS 파일 자체는 음악 파일 혹은 그래픽 파일을 특정 시각에 특정 시간동안 재생하도록 정의만 내린다. 따라서 BMS를 완전히 즐기려면 BMS 파일에서 요구하는 음악 파일인 키음과, 이미지나 영상 등의 그래픽 파일도 필요하다.

게임 데이터 작성이 모두 그렇듯이 BMS 제작 또한 인력을 소모한다. MIDI 음악 파일에서 키음을 추출하는 과정은 최소 10년 전부터 자동화 프로그램이 배포되었지만, 패턴은 그러한 프로그램이 배포되지 않았으며, 모 고난이도 패턴 제작자의 개인적인 사용에 그친 것으로 보인다.

따라서 고난이도뿐만 아닌 다양한 난이도의 패턴을 자동으로 만드는 프로그램 작성을 목표한다. 그러기 위해 패턴이란 무엇이고, 지금까지 플레이어들의 선호하는 패턴은 어떤 특징을 가지고 있는지 알아본다. 그리고 BMS 파일의 구조에 대해 간략하게 소개한 뒤, 작성자의 개인적인 수동 패턴 제작법을 알고리즘으로 구현하여 패턴을 자동으로 제작하는 프로그램을 만든다.

1. 패턴이란?

BMS 파일 자체는 음악 파일이 재생되는 시각만 정해준다 하였다. 그런데 이 음악 파일은 오브젝트에 할당되어 건반부의 입력 없이도 자동으로 재생되는 배경음과, 건반부의 입력을 통해 재생되는 키음으로 나눌 수 있다. BMS 파일의 구조를 좀 더 자세히 설명하면 아래와 같다.

곡을 MIDI 혹은 DAW로 작성시 곡을 트랙으로 나눌 수 있으며, 이 트랙을 분할하여 키음 파일을 .ogg나 .wav 포맷으로 얻을 수 있다. BMS 파일 내에서 오브젝트를 생성하면, 이 오브젝트에 키음 파일을 할당할 수 있다.

오브젝트는 할당 위치에 따라 크게 배경음과 키음 둘로 나눌 수 있다. 패턴 실행 시 배경음은 자동으로 연주되는 부분이며, 키음은 실제로 연주해야 하는 부분이다. 일반적으로 키음에 할당된 오브젝트를 노트라고 한다.

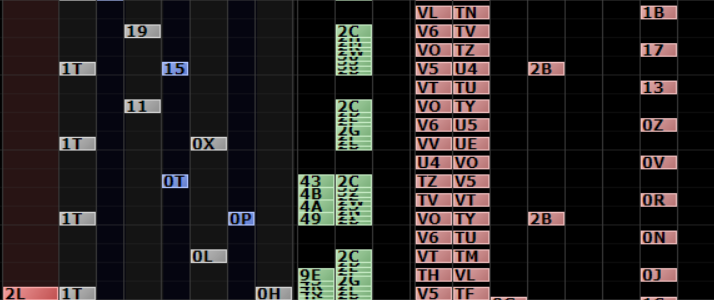
따라서 키음으로 할당된 오브젝트가 연주감을 결정한다. 무슨 오브젝트를 어디에 할당할 것이냐는 1998년 BMS 포맷 개발 이후부터 꾸준하게 패턴 제작자들의 과제였다. 이번 글에서는 이의 자동화에 대해서 다룬다.

패턴의 예시를 들기 위해 최초로 등록된 (기존 게임 데이터를 모방한 것이 아닌) BMS 패턴을 소개한다. dai가 제작한 Rainy Heart라는 곡으로, 유튜브 링크를 최하단 비고에 기재하였다. 피아노, 드럼, 신스음 등의 키음 파일 46개를 사용했으며 노트는 273개이다. 곡이 1분 20초 정도이므로 1초에 평균 3-4개의 노트를 연주해야 한다. 1998년 공개 당시에는 이는 ‘어려운’ 패턴이었다.

그러나 20년 이상이 흐른 지금은 1초에 노트 20개 이상이 나와야 어렵다고 평가받는다. 현재는 입문자와 숙련자의 차이가 매우 커졌기에 1초에 노트 한 개 이하를 연주하는 쉬운 패턴부터, 70개를 연주하는 극한의 패턴까지 모두 수요가 있는 상황이다. 따라서 데이터 제작자 한 명서는 이러한 수요를 모두 충족시킬 수는 없다. 그러므로 BMS에서는 공개된 곡에 같이 첨부된 패턴을 기반으로 다른 패턴을 제작하는 것이 활성화 되어있다. 작곡가가 제약하지 않는 한 모두가 패턴을 제작할 수 있다. 이 다른 패턴을 차분(差分)이라 하며, BMS 난이도표에 등재된 패턴은 대부분이 차분으로 구성된다.

2. 좋은 패턴이란?

BMS를 즐기는 사람이 많은 만큼 재미있는 패턴에 대한 관점도 다양하다. 각종 BMS 난이도표에도 곡의 수만큼 다양한 패턴이 소개되며 대체로 좋은 평가를 받는다. 따라서 간략하게 좋은 패턴에 대한 가이드라인만 제시한다. 그리고 이 가이드라인을 지키지 않는 패턴들을, 현재 가장 많이 플레이되는 7키 BMS에 초점을 맞추어, 예시를 들어 소개한다.

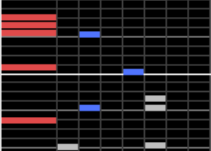
 그림은 7키 BMS를 BMS 파일 편집기 uBMSC로 시각화한 것이다. 왼쪽부터 붉은 오브젝트가 스크래치, 하얀색 및 파란색 오브젝트가 건반부, 초록색 오브젝트는 그래픽, 오른쪽의 붉은 오브젝트는 배경음 오브젝트에 해당된다.

먼저 건반부의 패턴에 대해 다루며, 스크래치 레인은 고려하지 않는다. 관습적으로 스크래치 사용은 음량이나 주파수가 다른 키음과 뚜렷히 구별되는 키음에만 적게 사용한다. 이러한 관습적인 패턴 제작법은 E. 에서 다룬다.

A. 음의 높이 측면

대부분의 BMS 구동기 및 입력장치에서 백건-흑건 입력부 UI를 채택한다. 그림과 같이 피아노의 건반 배열의 일부를 따왔다. 그러므로 패턴을 작성할 때 피아노와 같이 낮은 음은 왼쪽으로, 높은 음은 오른쪽으로 배치한다. 건반부는 7건반이지만 거의 모든 곡에는 7개 이상 키음 파일이 사용되므로 이를 일대일 대응시키는 것은 어렵다. 따라서 상황에 따라 이러한 규칙을 다소 위반하더라도, 유연하게 대처하는 것 또한 중요하다.

하지만 피아노 아르페지오와 같이 빠르게 높이가 변화하는 음을 키음으로 사용할 때, 변화와 반대되는 방향으로 패턴을 작성하는 것은 분명 좋지 않다. 또는 음높이를 완전히 무시한 채 무작위로 키음을 배치하는 것은, 연주하는 음이 명백한 경우, 추천되지 않는다.

그림은 피아노음으로 구성된 한 패턴의 일부이다. (음의 높낮이를 고려하지 않은) 무작위 같은 오브젝트의 배열과, 관습적으로 기피되는 스크래치의 적극적 사용이 인상적이다. 그러나 이와 같은 패턴을 다른 곡의 차분으로 제작한다 하면 추천할 수 없다. 그림은 The Grape Song의 동봉 패턴에서 발췌하였다.

B. 분배의 측면

만약 8개의 음으로 구성된 화음의 모든 음을 연주하게 하고 싶다면 제시된 그림과 같이 배치할 수밖에 없다. 이는 시간에 따른 화음의 변화를 패턴으로 전혀 느낄 수 없기에 기피되는 패턴이다. 다시 말해, 시간에 따른 음의 변화를 패턴의 변화로 나타내지 못한다면 좋지 않다. 그림은 The Clock Song의 동봉 패턴에서 발췌하였다.

따라서 어떤 키음을 연주하게 오브젝트를 배경음과 키음으로 나누는 것 또한 중요하다. 곡의 변화가 느껴지도록 나눠야 하며, 목표하는 난이도와 부합하도록 키음으로 사용하는 오브젝트의 개수 및 밀도를 조절해야 할 것이다.

C. 연타의 측면

연타의 사용은 프로그램 작성에서 가장 중요한 부분이며, 연주감을 증폭시킬 수도 혹은 완전히 망쳐버릴 수도 있는 요소이다. 연타의 정량적인 정의는 없지만 일반적으로는 리듬게임에서 BPM이 (BPM 120-240) 4분박 또는 16비트 (125-62.5ms) 이하의 간격으로 연속적으로 처리해야 하는 배치를 연타라고 한다. BMS 편집기로 볼 때 연타는 세로 간격이 가까운 두 개 이상의 노트이다.

연타는 같은 음을 잇달아 연주하는 경우에만 넣는 것이 좋다. 그렇지 않을 경우 연타의 배치를 최소화하는 것이 일반적이다. B.의 과정에서 너무 많은 오브젝트를 키음으로 사용하고자 할 때 연타가 필연적으로 생길 수밖에 없는데, 상황에 따라 연타를 유지하거나 키음을 일부 배경음으로 전환할 것인지 결정하는 능력이 있어야 좋은 패턴 제작자이다.

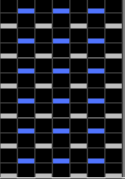
이는 연타가 배치된 패턴이 같은 밀도의 연타가 없는 패턴보다 어렵기 때문이다. 한 키에 대해 노트 사이의 간격이 좁을수록 정확한 타이밍에 연주하기가 어려운 것과 같다. 발광 BMS에 등재되었다가 삭제된 fuga (HELL Version)은 다른 키음을 가진 노트를 연타로 많이 배치하였다. 그림과 같이 연타의 간격이 아주 좁아 정확히 연주하기가 불가능하였으며 패턴의 평도 좋지 못하였다. 만약 이 패턴에서 노트를 다른 레인으로 옮겨 연타를 제거한다면 같은 밀도를 가지나 난이도는 훨씬 더 쉬운 패턴이 될 것이다.

그러나 연타가 반드시 기피되는 요소는 아니다. 언급했듯이 동일한 음에 대해 적절하게 배치하면 연주감을 대폭 살릴 수 있는데, 주로 곡에서 강조되는 악기의 음이 짧은 시간 후에 다시 재생되는 경우 연타로 배치한다. 그리고 느린 템포의 곡에서 높은 밀도의 패턴을 제작하기 위해서는 동시에 많은 음을 연주해야 되고 이 과정에서 연타가 필연적으로 들어가야 한다. 이러한 연타를 적절히 활용하는 것 역시 패턴 제작자의 능력이다.

D. 고난이도 난타 패턴에 대해서

B.의 분배 과정에서 연주감을 위해 잘 들리는 음을 키음으로 사용해야 하는 것은 명백하다. 그러나 높은 밀도의 고난이도 패턴을 위해서는 잘 안 들리는 음을 사용해야 한다. 이 때는 연주하는 음이 무슨 음인지 알아내기 어려우므로 음계의 요소는 지켜지지 않아도 나쁘지 않다. 이러한 경우 평가 요소는 B.에서 언급한 시간에 따른 음의 변화를 패턴의 변화로 나타냄의 여부이다.

C.에서 연타를 최소화하는 것이 좋다고 언급하였다. 그러므로 잘 안 들리는 오브젝트까지 모두 사용한다면, 곡에 따라 처리하기 어려운 연타 배치가 너무 많아져 좋지 않을 수 있다. 일반적으로 괜찮은 패턴은 연타가 없는 고밀도 패턴이며, 이를 난타로 부른다. 예를 들어, 패턴을 플레이한 사람 당 평균 플레이 횟수가 가장 높은 Air -GOD-은 난타로 구성되며 완급조절 또한 잘 되었기에 名패턴으로 불린다. 난타가 무엇인지 비고의 Air -GOD- 영상을 참고하면 알 수 있을 것이다.

 밀도를 최대화하면서 오브젝트를 균일하게 배치하는 방법은 4동시입력과 3동시입력의 반복이다. 이를 데님으로 부른다. 그러나 이러한 반복이 길어지면 연타를 넣지 않는 이상 곡의 변화를 반영할 수 없다. 그림과 같이 세로 눈금 두 칸 단위로 패턴이 반복되기 때문이다.

 번째 세로 눈금에 대해 필요한 동시입력의 수를 라 하면, 인 경우, 번째 세로 눈금에 연타가 배치될 수밖에 없다. 또한, 어떤 구간 의 모든 에 대해 이라면 이 큰 경우 반복적인 데님의 발생하므로 좋지 않다. 난타 패턴의 예시로, 혹은 의 형태를 가질 수 있다. 각각의 레인에 대해 노트의 밀도가 균일하도록, 패턴이 반복되지 않도록 한다. 그림은 인 경우 난타를 구성하는 예시이다.

물론 곡이나 목표 난이도에 따라 를 조절할 수 있으며, 이는 패턴 제작자가 판단할 일이다.

E. 관습 및 최근의 경향

아케이드 콘트롤러 기준으로 스크래치를 처리하기 위해서는 손이 크지 않은 이상 손이동이 발생한다. 건반부와 스크래치 입력 부 사이의 거리가 멀기 때문이다. 그러므로 스크래치의 사용은 관습적으로 턴테이블 음, 심벌즈 음, 보컬 샘플링 등의 강조하고 싶은 키음에 한정된다.

스크래치를 처리 중 손 이동을 줄이기 위해 손목으로 처리하는 방법도 생겼다. 이를 사용하면 스크래치 및 건반부 총 8개의 입력 장치와 손목, 손가락을 일대일 대응시킬 수 있다. BMS의 수많은 고난이도 스크래치를 처리하기 위해 이전부터 사용된 방법이나, 아직도 부정적으로 보는 시각이 존재한다. 따라서 스크래치를 다른 건반부와 같이 취급하여, 사실상 8키 패턴을 만드는 것은 좋지 않다.

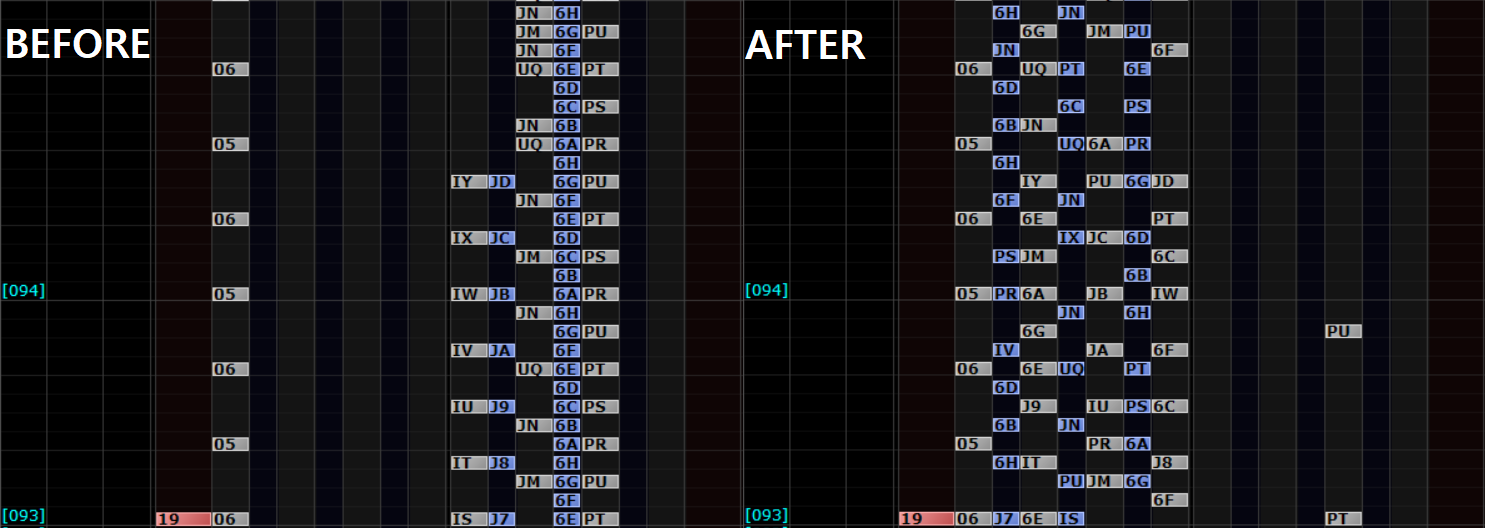
드럼음을 키음으로 사용하는 경우 관습적으로 음이 낮은 베이스 드럼음을 1번에 배치하고 높은 하이햇이나 라이드 음을 7번에 배치한다. 이는 A.의 연장선으로 해석된다. 이러한 경우 킥음이 배치된 1번 키에 다른 키음을 배치할 시 연주감이 상당히 이질적이기에, 이는 추천되지 않는다.

곡에서 준비된 오브젝트로 고난이도 패턴을 만들 수 없는 경우, 키음 파일이 할당되지 않은 無키음 오브젝트를 이용하여 오브젝트의 개수를 늘리기도 한다. 물론 이는 신중하게 사용해야 하는데, 타이밍에 맞춰 키음을 연주하는 리듬게임의 목적과 맞지 않기 때문이다. 주로 2000년대에 작성된 초기 BMS에서 고난이도 차분을 작성할 때 사용한다. 이는 초기 BMS 포맷은 사용할 수 있는 키음 파일의 개수가 지금보다 적었으며, 용량 및 로딩 시간의 문제로 오브젝트의 개수가 적은 경우가 대부분이었기 때문이다.

이상이 패턴을 수동으로 혹은 알고리즘을 통해 작성 시 고려해야 할 요소이다. 물론 좋은 패턴을 만들어야 하는 의무는 없으며, 누군가에게 재미없는 패턴도 다른 사람에게는 분명 재밌을 수 있다. 작곡가나 패턴 제작자가 의도적으로 특정 구간을 강조하기 위해 위를 무시하는 경우도 많다. 오픈소스 포맷인만큼 창작의 자유는 완전히 허용되기에 이를 지키지 않았다고 비난해서는 안 되며 오히려 그러한 패턴이 호평 받는 경우도 몇 존재한다. 위 가이드라인은 다만 이하 작성할 프로그램에 대한 지침 역할을 할 수 있는 정도다.

3. 사용 오브젝트 수동 분배 시 패턴 자동 제작 알고리즘

패턴 제작 시 오브젝트를 키음과 배경음으로 나눠야 한다. 우선, 이 과정을 수동으로 하는 경우를 가정한다. BMS 편집기는 14키 모드인 더블 플레이를 지원하므로, 더블 플레이 모드를 활성화한다. 활성화 시 건반부 영역에서 왼쪽 절반이 1P, 오른쪽으로 확장된 절반이 2P이다. 2P 레인에 패턴에 사용할 우선순위대로 (1-7) 오브젝트를 옮긴다. 목표를 시각적으로 나타내면 그림과 같다.



그림과 같이 기존 7키 패턴에 사용하는 1P 레인에 패턴을 미리 짤 수 있으며, 이 때 1P 레인의 오브젝트는 이동하지 않는다. 예를 들면, 1P 1번 키 레인에 베이스 드럼 음을 할당하고 1번 키 레인에 추가적인 변화가 없게끔 제한조건을 둘 수 있다. 그리고 그림 오른쪽의 2P 영역에 남은 오브젝트는 1P 영역으로 옮길 시 연타가 발생하므로 남겨둔 것이다.

A. 음 높이의 측면

이제 오브젝트에 할당된 키음 파일의 음의 높이를 알아야 한다. FFT (Fast Fourier transform)를 사용한다. 44.1kHZ 샘플링 레이트를 가진 스테레오 wav 파일을 기준으로 한다. 20Hz보다 정밀한 resolution을 원한다면 4096차 FFT를 사용해야 한다. 즉, 키음 파일에서 처음의 4096개의 sample에 대해 4096차 FFT를 하며, 이는 음악 파일의 처음 46.4ms에 해당된다. 이보다 길이가 짧은 음악 파일은 키음에 보통 사용하지 않을 것이며, 사용한다면 예외처리를 할 것이다.

소절(마디)별로 레인에 할당한 키음의 트랙이 다를 수 있으니 수동으로 패턴 제작을 할 구간을 소절의 범위로 입력한다. 각 구간마다 필요한 제한조건이 다를 수 있으니, 각 구간을 순차적으로 수동 입력 후 패턴을 각각 작성한다.

알고리즘 작성 시 우선적으로 BMS 파일에서 사용하는 모든 키음 파일을 FFT한다. 이후 각 키음에 대해 되는 peak frequency와 그 frequency에서의 amplitude를 array\_FFT로 저장한다.

한 구간에 대해 한 레인에 있는 오브젝트에 할당된 모든 키음에 대해 각각의 peak frequency를 array\_FFT로 불러온다. 제일 낮은 peak frequency를 가진 오브젝트를 0으로 하여 그 순서대로 오브젝트에 번호를 부여한 후 이를 array\_temp로 저장한다.

이제 오브젝트를 기존의 7키에서 사용하는 1P 영역으로 옮긴다. 옮기는 레인은 array\_temp에 할당된 번호의 mod 7값을 사용한다. (제한조건에 따라 7은 달라질 수 있음) 만약 옮기고자 하는 레인에 이미 오브젝트가 존재하거나, 제한조건이 있어 사용할 수 없다면 할당된 번호를 1씩 증가시킨다. (혹은 1씩 감소시킨다.) 만약 이 과정을 7번 반복하여도 오브젝트를 옮길 수 없다면 그 오브젝트는 패턴에 사용하지 않는다. 이를 재생시각 순으로 모든 오브젝트에 대해 반복한다.

각 레인에 대해 이를 우선순위대로 반복한다. 사용할 수 없는 오브젝트는 2P 영역에 남아있으므로 이를 확인 후 해결하고, 스크래치를 추가적으로 넣거나 만족스럽지 않은 구간을 정리하는 등의 수동 tuning을 거치면 7키 패턴이 완성된다.

B. 연타의 측면

그러나 단순히 A. 과정만을 거친 패턴은 부자연스러운 연타로 가득한 패턴이 될 가능성이 높다. 이러한 패턴을 원한다면 여기서 끝내도 되겠지만, 고난이도 난타 패턴을 만들기 위해서는 사용하는 오브젝트를 기준으로 전후에 어떤 오브젝트가 어떤 시간차로 배치되어 있는지 조사하여 연타배치를 피해야 한다.

경우에 따라 연타 배치를 원하는 경우도 있을 테니, 실행 전 제한 조건으로 몇 분박 혹은 몇 ms 이하부터 연타로 볼 것인지 정의하고, 얼마나 긴 연타까지 허용할 것인지 제어할 수 있게 한다.

4. BMS 파일을 텍스트 단위로 접근 (Optional)

이상 3. 에서 pseudo-code로 자동 패턴 제작을 논의했다. 오브젝트를 옮긴다 함은 BMS 편집기에서 시각화된 BMS 파일과 그 오브젝트를 간단한 조작으로 같은 시각의 다른 레인으로 재생되도록 할 수 있기 때문에 옮긴다는 표현을 사용하였다. 그러나 BMS 파일은 본질적으로 텍스트 파일이기에 텍스트로 표현된 오브젝트를 전문화된 BMS 편집기 없이 간단하게 옮길 수는 없다. 낙원에 투고한 글을 이해하였다면 메모장으로 BMS를 편집하는 것은 서론에서 언급한 인력 소모 행위라고 할 수 있다.

하지만 이 인력 소모를 컴퓨터가 대신해준다면 다르다. 그러기 위해 A.에서 BMS 파일 구조를 간단하게 언급하고 B.에서 오브젝트 조작 과정 일부를 python code로 구현해본다. 실제 code는 비고에 기재된 github 링크를 참고하시오.

A. BMS 파일 구조

BMS 파일 내부에서는 먼저 사용할 키음 파일을 키음 슬롯에 할당한다. 키음 슬롯은 01-ZZ까지 1295개이다. 만약 F3번 슬롯에 foon.wav를 할당하고 싶으면 #WAVF3 foon.wav로 기입한다.

이후 오브젝트의 할당은 마디 번호와 출력 레인을 선언한 후, 해당 마디 및 레인에서의 재생 시각을 정의한다. 43번째 마디의 1/4 지점에 foon.wav를 출력하고 싶으면 #043xy:00F30000으로 기입한다. xy는 출력 레인 번호이고, ‘:’ 이후는 두 문자씩 처리한다. 00은 아무 오브젝트가 할당되지 않음을, F3은 foon.wav가 할당된 키음 슬롯을 의미한다.

따라서 패턴은 특정 시각에 특정 레인에서 재생되는 오브젝트를 다른 레인으로 옮겨 제작된다. 단, 옮기는 레인은 키음의 peak freq를 고려하고, 오브젝트가 다른 오브젝트와 겹치는지 혹은 연타를 발생하는지 확인 후 오브젝트를 옮겨야 할 것이다.

코드로 구현 시 오브젝트가 이동 전에 있던 레인에서 키음 슬롯 번호가 있던 자리를 00으로 바꾸고, 오브젝트가 이동한 레인에서 오브젝트 시각에 해당하는 00 문자를 키음 슬롯 번호를 바꾸면 된다.

B. python code로 구현

먼저, 마디와 레인 별로 ‘:’ 이후의 (문자 길이)/2, 이하 res가 다르기 때문에 이를 마디별로 각 레인의 res의 최소공배수로 정의한다. 예를 들어, 특정 마디의 1P 1번 레인에서는 1/2 지점에 할당된 오브젝트를 표시하기 위해 ‘:’ 뒤를 001A로 (res=2), 2P 2번 레인에서 2/3 지점의 오브젝트를 0000BZ로 (res=3) 표기했다면 res를 6으로 정의한다. 이 작업은 마디별로 시행해서 res\_list의 n번째 element가 n번째 마디의 res가 되도록 한다.

그런 다음 각 오브젝트의 정보를 list내의 tuple로 나타낸다. 1P 혹은 2P 오브젝트 정보 전체의 리스트 (각각 op1, op2)는 레인별로 sublist를 가지고 있으며, 이 sublist는 마디 별로 sub\_sublist를 가진다. sub\_sublist 내의 오브젝트를 (재생지점, 키음 슬롯 번호)로 저장한다. 이때 재생지점 m/res에서 m만 저장하는데 res는 res\_list에 이미 저장 되어있기 때문이다. 예시로 #04311:001A는 1P 오브젝트 정보의 1번 레인 sublist 내의 43번째 sub\_sublist에서 (1, ‘1A’)로 저장된다.

이제 패턴 제작에 사용할 오브젝트를 2P에 정렬했으면 이를 1P 영역으로 옮겨야 한다. op2의 sublist에 대해 재생지점이 지정해준 범위 내에 있는 모든 오브젝트 정보를 읽어 3.A. 에서 언급한 array\_temp를 만든다. 오브젝트 별로 array\_temp의 value를 읽어 value의 mod 7값에 해당하는 1P 레인으로 오브젝트 이동을 시도한다.

이 때 이 시도가 성공적이기 위해서는, 같은 재생 지점을 갖는 오브젝트가 해당 레인에 없어야 되고, 미리 정의해준 연타에 대해 원하지 않는 연타가 발생하지 않아야 한다. 이는 오브젝트 재생지점 근처의 해당 1P 레인의 오브젝트를 op1에서 읽어 처리할 수 있다. 만약 1/16마디 이하의 간격을 갖는 연타를 원하지 않으면 해당 오브젝트 재생 지점 전후로 1/16마디에 해당하는 범위를 읽으면 된다.

시도가 성공적이라면 해당 오브젝트를 op2에서 remove하고, op1에 insert하여 재생지점 순으로 정렬한다. 이를 모든 op2 오브젝트에 대해 반복하여 최종 패턴 정보가 담긴 op1과 op2를 얻은 뒤, op1과 op2를 기반으로 BMS 파일을 작성하여 폴더 내에 write한다.

실제 python 코드는 변수명이 글의 설명과 다를 수 있고, 아직 초기 버전이라 알고리즘을 개선할 필요가 있다. 하지만 큰 틀은 위와 같으며 이를 기반으로 github에 공개, 업데이트할 예정이다.

5. 사용 오브젝트 자동 분배 알고리즘

그러나 사용할 오브젝트를 2P에 분배하는 과정도 번거로울 수 있다. 따라서 사용할 오브젝트를 2P에 자동으로 분배하는 알고리즘이 필요하다 느낄 수 있다. 이는 아직 github에 공개한 코드에 구현되지는 않았다. 알고리즘의 결합으로 곡과 맞지 않는 오브젝트 분배가 발생할 수도 있지만, 코드 실행 한 번으로 패턴이 생성된다면 기존 차분 제작 관행과 상당히 다르기에 공개에 조심스럽기도 하다.

알고리즘의 아이디어만 간단히 제시하도록 한다. 먼저, 곡을 악절 단위로 나눠야 한다. 그러나 곡의 형식이 다양하듯 bms의 구성 또한 매우 다양하기에 악절을 8마디 등으로 정의해서 이를 일괄적으로 처리하면 악절로의 분할이 부자연스럽게 된다. 따라서 길이가 길거나 peak freq가 높고 amplitude가 큰 키음이 할당된 오브젝트를 기준으로 악절 분할을 시도한다. 보통 bms에서는 악절 별로 길이가 긴 배경음을 뽑거나, peak freq가 높고 amplitude가 큰 심벌즈나 효과음을 사용하기에 악절의 일방적 정의보다는 유연하게 대처할 수 있을 것으로 생각된다.

사실 악절은 자동으로 분할하는 것보다 직접 지정하는 것이 낫다. 예를 들면 사용하지 않는 2P스크래치 레인에, 악절간 경계에 키음이 할당되지 않은 오브젝트를 배치해 지정할 수 있다. 하지만 이조차 ‘번거롭다고’ 느끼는 사용자를 위해 굳이 알고리즘을 구현한다면 위와 같이 될 것이다.

이제 분할한 악절 별로 길이가 너무 짧지도 길지도 않으면서도 amplitude가 적절한 키음 파일이 할당된 오브젝트를 찾는다. 각 요소를 종합해서 점수로 나타낸 뒤 점수가 threshold보다 높은 오브젝트를 2P 영역에 할당한다. 이 과정 역시 FFT 결과를 이용하면 된다. 다만 같은 키음을 악절이 바뀌어도 계속 사용하는 것은 지루할 수 있기에, 이전 악절에 사용 빈도가 높은 키음에 대해 점수를 보정하는 식으로 진행한다.

이 과정은 사용자가 원하는 총 노트 수와 노트 밀도를 설정하여 유연하게 조절하도록 한다. 만약 사용자가 초당 노트 10개 정도가 나오는 패턴을 원하는데, 코드가 일괄적으로 초당 노트가 30개씩 나오는 패턴을 출력한다면 곤란할 것이다. 따라서, 몇몇 변수의 수동 설정 요소를 도입하여 사용자의 요구에 맞춘 다양한 패턴이 나올 수 있도록 한다.

개인적으로 이 알고리즘을 코드로 구현하여도 공개를 해야 할지 의문이 든다. 문단 처음에 언급했듯이 코드 실행 한 번으로 분 단위로 패턴이 생성된다면, 패턴 양산을 시도할 것이며 양산된 패턴이 곡과 어울리든 어울리지 않든 그 책임은 공개한 사람에게 전가될 수 있기 때문이다. 곡과 어울려도 차분 제작 과정이 무성의하다며 불만을 표하는 플레이어가 생길 것이며, 곡과 어울리지 않다면 질 낮은 패턴을 양산하도록 단초를 제공한 셈이다. (장르가 다르긴 하지만, 20년 전 스도쿠 생성 및 해결 프로그램이 개발되어도 여전히 사람이 직접 손으로 만든 스도쿠를 좋아하는 마니아 계층이 있는 것처럼 말이다. 본인은 저러한 스도쿠를 10년 넘게 풀어도 차이를 알지 못한다.)

이상 공개 여부는 비고에 기재한 github 링크에 추후 게재할 것이다. 6.에서 소개할 예시는 due date 문제로 5.의 알고리즘을 python으로 구현 및 사용하지 않았기에 양해바란다.

6. 코드 실제 사용 예시

작곡가 Lime의 2019년에 공개한 8bit Voyager라는 BMS로 4.B. 에서 소개한 code로 패턴을 작성하였다. Youtube 영상 링크는 비고에 기재하였다. 수동 분배 및 fine tuning에 걸린 시간은 30분 정도였으나 코드 자체는 실행 후 30초 안에 패턴 작성을 완료하였다. 1000개가 넘는 키음 파일에 대해 3000개가 넘는 오브젝트를 30초 안에 배열하는 것은 놀랍다. 실제 패턴에서 1번 레인을 제외하고 16비트 이하 간격의 연타가 발생하지 않음을 알 수 있다. 수동으로 이를 배치했다면 1초에 한 개씩 배열해도 1시간 정도가 걸리며 실제로는 하루 단위로 걸릴 것이다. 이 패턴은 2020년 1월 10일에 공개하였으며, 일본의 BMS 차분 플레이 전문 유저 hex가 좋은 패턴으로 추천하였다.

또다른 사용 예시로 sun3이 2006년 공개한 星の器～STAR\_OF\_ANDROMEDA로 패턴을 작성하였다. 연타 생성 여부 확인 알고리즘이 잘 구현되었는지 확인하기 위해, 2연타는 허용하나 3연타는 허용하지 않도록 조건을 준 뒤 실행하였다. 실제로 조건에 맞게 출력되었기에, 기본적인 알고리즘이 python으로 잘 구현됨을 확인할 수 있다. 이외의 다른 제한 조건은 거의 주지 않았기에 코드 실행은 10초 안에 완료되었다. 영상은 비고에 기재한 유튜브 링크의 첫 2분 30초에 해당된다.

이 외에 다른 조건은 코드 내에서 다양하게 걸 수 있지만, 일단 이 두 개의 예시만을 소개한다.

7. 결론

패턴의 정의와 구성을 BMS를 재생 시의 시각적 요소와 BMS 파일 내부 구조의 측면에서 설명하였다. 또한 좋은 BMS 패턴의 조건을 예시를 들며 설명하고, 이 조건을 알고리즘으로 구현하기 위해 BMS 파일 내부 구조를 간략하게 소개하였다. 패턴 자동 제작을 위한 모든 기능을 작성하지는 않았지만, 오브젝트를 수동으로 할당하면 이를 패턴 영역으로 옮겨주는 코드를 작성하였다. 이 python 코드를 github에 공개하였고, 실제 사용 예시를 차분 업로드 사이트와 그 영상을 유튜브에 공개하였다. 추후 알고리즘의 개선과 기능의 추가로 수요에 맞는 다양하고 곡에 알맞은 좋은 패턴을 자동으로 작성할 수 있을 것이다.

8. 비고

코드 공개 github 링크 <https://fsrs.github.io/lab.html>

유튜브 링크

[BMS] dai - Rainy Heart (1998) <https://www.youtube.com/watch?v=X2hRnALlCBk>  
8bit Voyager [Auto Voyager] <https://www.youtube.com/watch?v=K_nrwQphP6Y>  
星の器～STAR OF ANDROMEDA (EZ?) <https://www.youtube.com/watch?v=qPi9Ce-eyFo>

회지에 참고한 BMS 패턴

fuga (HELL Version) 패턴 영상  
<https://www.youtube.com/watch?v=EfSg6zxjfAU>

Air -GOD- 패턴 영상  
<https://www.youtube.com/watch?v=35C8TX_KNDc>

도움을 준 사이트

BMS Score Viewer <http://www.ribbit.xyz/bms/score/>  
Hex BMS [https://bms.hexlataia.xyz](https://bms.hexlataia.xyz/tables/tanosi.html)