

이어로이드: 청음 훈련 안드로이드 어플리케이션

(Earoid: An Ear Training Android Application)

지도교수 : 엄현상

이 보고서를 공학학사 학위 논문

대체 보고서로 제출함.

2015년 1월 22일

서울대학교 음악대학

작곡과 작곡전공

박 수 철

2015년 2월

초 록

이어로이드는 음악대학의 입시과목인 청음을 훈련할 수 있도록 만들어진 안드로이드용 애플리케이션이다. 기존에 사람에 의해서 이루어지던 청음 교육 방법에서 탈피해 컴퓨터를 이용한 알고리즘적 작곡 방식을 채택하여 혼자서도 스스로 연습을 할 수 있도록 하였다. 이전에 나와 있던 문제은행 방식의 청음 연습 어플리케이션은 학습자의 수준에 맞는 문제의 양이 많지 않거나, 사람의 음악적 경향성에 의한 선율을 만들기 때문에 추측에 의한 답안 작성이 가능하다는 단점이 있었다. 그리고 무작위적으로 음을 생성하는 방식의 어플리케이션들은 리듬이 생략되어 실제 청음 시험의 방식과 거리가 멀다는 단점이 있었다. 이러한 문제점을 극복하기 위해 음고 선택에 있어서는 제한된 무작위의 방식을 이용하고, 리듬과 화성에 있어서는 주어진 데이터 활용하여 최소한의 음악적인 진행을 마련하는 방법을 선택하였다. 이러한 과정에 의해서 실제 청음 시험의 유형에 맞는 즉, 다양한 리듬, 선율, 화성 패턴을 보여주면서도 음악적 맥락을 유지하는 악곡을 결과로 낼 수 있었다. 이러한 장점들 덕분에 실제 청음 시험을 대비하고 있는 입시생들 사이에서 많이 활용되고 있으며, 알고리즘을 이용한 작곡 방식의 좋은 쓰임새 중 하나로 볼 수 있겠다.

주요어 : 안드로이드, 어플리케이션, 청음, 알고리즘 작곡

목 차

제 1 장 서론	1
제 2 장 관련 배경지식	3
2.1 청음 훈련 방법	3
2.2 알고리즘 작곡의 역사적 배경	5
제 3 장 이어로이드 어플리케이션 설계	8
3.1 악곡 구성 설계 개요	8
3.2 리듬 구성 설계	8
3.3 화성 구성 설계	12
3.4 선율 구성 설계	14
제 4 장 이어로이드 어플리케이션 구현	18
4.1 악곡 구성 구현 개요	18
4.2 리듬 구성 구현	21
4.3 화성 구성 구현	25
4.4 선율 구성 구현	29
제 5 장 다른 어플리케이션과 비교 및 통계 자료	34
5.1 다른 어플리케이션과 비교	34
5.2 통계 자료	38
제 6 장 결 론	41
참고자료: 사용자 설명서	42

그림 목 차

〈그림 1〉 모짜르트 “음악의 주사위 놀이”	5
〈그림 2〉 바하 평균율 피아노곡집 1권 중 C minor 푸가 도입 부분	6
〈그림 3〉 음가 4/4의 리듬 유닛들	8
〈그림 4〉 음가 3/4의 리듬 유닛들	9
〈그림 5〉 음가 2/4의 리듬 유닛들	9
〈그림 6〉 음가 1/4의 리듬 유닛들	9
〈그림 7〉 전체 악곡의 음가 배분을 표기한 악보	10
〈그림 8〉 음가 1/4의 리듬 유닛들의 복잡도	11
〈그림 9〉 완성된 전체 악곡의 리듬 패턴	12
〈그림 10〉 완성된 전체 악곡의 화성 진행	14
〈그림 11〉 음역과 방향성의 설정	15
〈그림 12〉 음역과 방향성의 설정 후에 만들어진 선율	15
〈그림 13〉 음의 해결 과정을 거친 선율	16
〈그림 14〉 비화성음이 적용된 선율	16
〈그림 15〉 불임줄이 들어간 선율	17
〈그림 16〉 실제 악보의 리듬	22
〈그림 17〉 리듬 유닛들의 배열	22
〈그림 18〉 화성 진행의 예	28
〈그림 19〉 안드로이드용 청음 어플 “조성 음악 청음”	35
〈그림 20〉 안드로이드용 청음 어플 “Perfect Ear”	35
〈그림 21〉 웹 사이트 “Ear-Training”에서의 문제 풀이 화면	36
〈그림 22〉 총 사용자 설치 수	38
〈그림 23〉 현재 사용자 설치 수	38
〈그림 24〉 일일 사용자 설치 수	38
〈그림 25〉 사용자 평가 점수	39
〈그림 26〉 높은 평점을 준 리뷰	39

〈그림 27〉 낮은 평점을 준 리뷰.....	39
〈그림 28〉 이어로이드 플레이스토어 검색.....	42
〈그림 29〉 시작화면.....	43
〈그림 30〉 문제 생성 화면.....	44
〈그림 31〉 문제보기 화면아이콘 설명	44
〈그림 32〉 문제보기 화면	45
〈그림 33〉 좌-구매화면, 우-중급단계 예제화면.....	45
〈그림 34〉 좌-고급 단계 예제 화면, 우-전문가 단계 예제 화면.....	46

표 목 차

〈표 1〉 전체 악곡의 음가 배분.....	10
〈표 2〉 전체 악곡 리듬 유닛 할당.....	11
〈표 3〉 화성 진행의 예	13
〈표 5〉 MusicXML 계층 구조.....	24
〈표 6〉 화성 진행 XML 파일 목록.....	26

제 1 장 서론

기존 어플리케이션들의 문제

단순히 음감을 훈련하기 위한 어플리케이션 뿐만 아니라 청음 시험을 위해서 선율이나 화성 등을 들려주는 어플리케이션은 이미 몇 가지 존재한다. 하지만 이러한 어플리케이션들은 대부분 문제 은행 형식으로 설계되어 있다. 즉, 이미 출제 되어있는 문제들에서 무작위로 뽑아서 보여주는 방식이다. 이러한 문제들을 모아서 출판된 책들도 시중에 다수 있으며, 웹 사이트를 통한 방식이나 패키지 소프트웨어를 통해서 제공하기도 한다. 이러한 방식에는 몇 가지 문제점이 있다.

첫째로, 문제의 양이 많지가 않다는 것이다. 시중에 나와 있는 책들에는 100여개 안팎의 문제들을 담고 있다. 하지만 이는 기초적인 문제부터 심화된 문제까지의 수이기 때문에 실제로 도움이 되게 풀어 볼 수 있는 문제의 개수는 20개 안팎이라고 볼 수 있다. 적어도 하루에 한 두 문제 정도씩 풀어본다고 했을 때, 한 달이 되지 않아서 소모될 수 있는 분량이다. 풀었던 문제를 다시 풀어 보는 것도 도움이 되긴 하나 아무래도 새로운 문제만 하지는 못하다.

둘째로, 사람이 만든 문제들은 음악적인 경향성이 있어서 추측에 의한 답안 작성이 가능하다는 것이다. 아이러니 하게도 청음 음악교육을 위해서는 비음악적인 선율에 대해서도 익숙해지는 것이 필요한데, 사람이 만든 선율은 출제자가 경험한 기존의 음악을 토대로 만드는 습성이 남아 있어서 다양한 문제를 제시하지 못할 뿐만 아니라, 음악에 대한 감각에 의해서 추측을 해볼 수 있는 선율을 만든다는 단점이 있다.

이러한 문제점들은 컴퓨터의 무작위 음 생성을 통해서 충분히 해결될 수 있다. 특정한 알고리즘 내에서 사람이 느끼기엔 거의 무한대에 가까운 문제들을 만들어 낼 수 있고, 또한 사람의 감각에 의존하지 않으므로 경향성이 배제된 순전히 논리 과정에 의한 문제들이 만들어진다.

어플 제작 방법

어플리케이션은 모바일 디바이스가 보편된 시대의 흐름에 맞추어 안드로이드 디바이스를 대상으로 하여 개발하였다. 안드로이드 디바이스는 Linux 커널을 기반으로 만들어진 Android OS 위에서 돌아가는 JVM (Java Virtual Machine)에 의해 어플리케이션 단계의 프로그램들이 작동한다. 이를

위해서 Java SDK와 Android SDK를 사용하였다.

선율을 컴퓨터를 통해서 자동으로 만들어 내는 방법은 여러가지가 있다. 먼저 생각할 수 있는 방법으로는 리듬, 음고, 화성과 같은 음악적 요소들을 분리해 이에 대한 데이터를 구축하고 특정한 조건 내에서 제한된 무작위 선정의 방법으로 선택된 요소들의 조합으로 전체의 선율을 구성하는 것이다. 두 번째로 생각할 수 있는 방법은 앞서 말한 음악적 요소들의 특징들을 분석하고 이에 대한 논리적 구조를 규명한 다음 그 원리를 코드화하는 방법이다.

이 두 가지 방법 모두 실제 작곡에 있어서의 방법론과 맞아 떨어진다. 선율을 만들어내는 작곡가의 머리 속에는 그가 경험했던 수 많은 음악적 내용들이 담겨 있으며, 이는 일종의 데이터를 이룬다고 할 수 있다. 알게 모르게 데이터화된 이 바탕 안에서 영감의 발생이라고 하는 몇 가지 조합을 생각해내고 이를 토대로 작업을 시작해나간다. 그러면서도 대위법, 화성법, 동기 전개법과 같은 방법론적 수단들이나 시대 또는 지역성으로 결정지어지는 양식 및 형식을 적용하여 최종적인 결과를 얻어낸다.

이러한 방법들을 고려하여 실제 어플의 선율 알고리즘에 적용하였다. 어떤 요소와 어느 범위 까지를 데이터화할 것인가 하는 문제가 관건이었는데, 이는 다양한 선율을 만들어내기 위한 목적과 효율성에 중점을 두고 진행하였다. 결과적으로 리듬과 화성은 최대한 데이터화하고 선율은 그 진행원리를 코드화하는 방식으로 개발하였다. 이는 리듬과 화성에 양식적인 구속력을 강화하면서도 청음 문제의 특성인 음고의 다양함을 추구할 수 있는 방법이다.

청음 문제 생성에 대한 음악적인 기반 지식들과 청음 시험 과정을 반영하기 위한 정보들은 음악대학 작곡과 및 기악과 학생들의 경험을 통해 얻었다. 작곡과 입시에서는 다양한 박자, 음자리표, 조성을 가진 매우 어려운 문제까지 다루고 있고, 기악과 입시에서는 상대적으로 간단한 음악적 요소들을 가진 쉬운 문제들이 출제된다. 이러한 여러 난이도의 범위를 다루기 위해서 음악적 요소들을 알고리즘으로 어떻게 변환시키느냐가 관건이었다. 이제 본론으로 들어가서 이러한 요소들을 어떻게 기계적인 알고리즘으로 변환하고 작동하는지 알아보도록 하겠다.

제 2 장 관련 배경지식

2.1 청음 훈련 방법

한국 뿐만 아니라 전세계 모든 나라의 음악대학의 입시에 청음 시험이 필수적으로 이루어진다. 청음 시험은 음악을 들려주고 이를 받아 적게 하여 음을 인지하는 능력을 평가하는 것이다. 주로 성부가 하나인 단성의 선율을 들려주고, 때에 따라서 리듬이나 2성부 선율, 3성, 4성부 화성 진행이 추가되기도 한다.

이러한 시험을 보기 위해서는 음감이라고 하는 음악적인 감각이 필요한데, 음감은 절대음감과 상대음감으로 나뉘어진다. 절대음감은 다른 음에 의존하지 않고 독립적인 음만을 듣고도 그 음이 어떤 음인지 파악하는 능력이다. 예를 들면, 조성이나 기준이 되는 한 음을 들려주지 않은 상태에서 어떤 음의 음이름을 알 수 있으면 절대음감이다. 때때로 절대음감이라는 용어가 음악성과 관련해서 뛰어난 음악적 재능을 대변하는 뜻으로 쓰이거나, 미분음 또는 헤르쯔(Hz) 단위까지 구별하는 능력을 일컫는 뜻으로 쓰이곤 하는데 이는 완전히 잘못된 사용법이다. 실제로 작곡가, 연주자들 대부분이 절대음감이 아니며, 오히려 절대음감 보다는 상대음감이 음악 활동을 함에 있어서 유리하다는 말들이 있다. 그리고 미분음 또는 헤르쯔 단위까지 구별하는 능력은 절대음감에서의 예민함의 정도의 차이이지 절대음감이라면 꼭 그래야 한다는 당위성은 존재하지 않는다.

절대음감과 대비되는 말로 쓰이는 상대음감이란 용어가 있으나 이 또한 역시 꼭 절대음감과 배타적인 용어는 아니다. 절대음감이면서 부분적으로 상대음감적 능력이 조금 보이는 경우도 있고, 상대음감이면서 부분적으로 절대음감의 능력을 가진 경우도 있다. 상대음감의 뜻은 여러 음들을 들었을 때, 각 음들의 높낮이 관계를 파악하는 능력이다. 예를 들면, 선율을 듣고서 해당 선율의 음들을 이동도법의 계이름으로 알 수 있거나, 특정 음의 음이름을 들려 주었을 때, 나머지 음들의 음이름을 파악할 수 있을 때, 상대음감의 능력을 지녔다고 볼 수 있다.

절대음감과 상대음감 모두 음악적 환경 내에서 오랜 시간 노출되거나, 음감에 대한 교육을 특별히 받는 것으로 길러질 수 있다. 이는 마치 언어를 배우는 것과 같다. 어린 시절 모국어를 자연스럽게 배울 수 있듯이 피아노나 바이올린 같은 악기를 배우면서 음감이 길러지게 된다. 하지만 이런 과정을 거치더라도 음감이 생기지 않는 경우도 있다. 이러한 경우에도 외국어 교육을 받듯이 음감 교육을

따로 받으면 능력이 계발 될 수 있다.

음감 교육의 기본적인 방법은 음을 듣고 해당 음의 음이름이나 계이름을 맞추는 것이다. 음이름으로 맞추는 경우는 절대음감을 기르는 것에 해당하며 계이름을 맞추는 경우는 상대음감을 기르는 것에 해당하겠다. 이러한 교육은 기존에는 선생과 학생이 일대일로 직접 음을 들려주고 맞추는 형식으로 진행되었다. 선생은 피아노를 연주하여 음을 내고, 학생은 이에 대한 답을 말하거나 종이에 써서 맞추는 식이다. 혼자서는 연습하는 것은 그다지 도움이 되지 않는데 왜냐하면 피아노나 바이올린 같은 악기를 연주를 하면 무슨 음인지 이미 아는 상태가 되고, 따라서 맞추는 것이 의미가 없어지기 때문이다.

컴퓨터와 같이 인간을 대신할 수 있는 기계가 개발됨에 따라 선생의 역할을 대신할 수 있는 방법이 생겨났다. 즉, 피아노를 통해 음을 내는 역할을 컴퓨터가 대신하여 진행하는 것이다. 게다가 인간이 내는 방법은 평소 연주 경험에 의한 경향성이 있어 무작위적이지 않으므로 추측에 의한 맞추기가 가능했으나, 컴퓨터로는 거의 무작위에 가까운 (pseudo-random) 음을 생성해낼 수 있으므로 어떻게 보면 더 나은 학습을 진행할 수 있게 된 것이다. 이러한 장점 덕분에 컴퓨터를 이용한 음감 교육 소프트웨어가 예전부터 많이 있어 왔다.

무작위로 생성한 음을 들려주고 맞추는 방식은 음감을 기르는데 매우 효과적이다. 대부분의 소프트웨어들은 몇 개의 음을 개별적으로 들려주고 피아노 모양의 버튼이나 음정이 적힌 버튼을 누르는 인터페이스로써 답을 할 수 있게 되어 있다. 하지만 이러한 방식으로 대학에서의 입시가 치뤄지지 않는다. 음감 자체를 기르는데는 효과적이거나 그것이 바로 입시에서 효과를 보기에는 부족함이 있다는 말이다. 이는 마치 교과서를 통해서 공부를 많이 한 것만으로는 대학 수학 능력 시험을 치는데 부족함이 있는 것과 같다. 문제집을 많이 풀어보고 시간을 재고 시험의 분위기에 익숙해지는 과정 역시 중요하다.

음감의 능력 자체가 청음 시험을 치르는데 어느 정도 큰 요소로 작용하지만, 이 못지 않게 실제로 문제를 듣고 답안을 적어 보는 연습 또한 필수적이다. 음감에 자신 있는 많은 학생들 중 적지 않은 수가 실제 청음 시험에서는 좋은 결과를 얻지 못하는데 이는 시험에 대한 연습 과정이 부족했던 것이라 판단한다. 이에 따라 음감 자체를 기르는 훈련에 관련된 소프트웨어 뿐만 아니라 청음 시험을

준비하기 위한 소프트웨어의 필요성이 대두되었다. 이러한 맥락이 본 어플리케이션인 이어로이드를 개발하게된 계기이다.

2.2 알고리즘 작곡의 역사적 배경

20세기 중반 컴퓨터의 등장으로 예술 분야에 또한 지대한 영향을 미쳤고, 그 일환으로 알고리즘을 통하여 작곡을 하는 사람들이 생겨났다. 하지만 이렇게 컴퓨터와 연관된 알고리즘이라는 용어를 사용하기 이전에도 방법론적으로는 같다고 볼 수 있는 작곡의 기법들이 있었다. 대표적으로 알레아토릭(Aleatorik)이라는 기법이 있는데 이는 작곡을 함에 있어서 처음부터 끝까지 정형화된 형식을 갖게 하는 것이 아닌 부분 또는 전체적으로 우연성의 요소를 삽입함으로써 항상 다른 음악이 나오게 하는 것이다. 이를 사용한 대표적인 예로는 모짜르트의 “음악의 주사위 놀이”라는 곡을 들 수 있다.

	A	B	C	D	E	F	G	H
2	96	22	141	41	106	122	11	30
3	32	6	128	63	146	46	134	81
4	69	95	158	13	153	55	110	24
5	40	17	113	85	161	2	159	100
6	148	74	163	43	80	97	36	107
7	104	157	27	167	154	68	118	91
8	162	60	171	53	99	133	21	127
9	119	94	114	50	140	86	169	94
10	98	142	42	156	75	129	62	123
11	3	87	165	61	135	47	147	33
12	54	130	10	103	28	97	106	5

〈그림 1〉 모짜르트 “음악의 주사위 놀이”¹

모짜르트의 작품에서 인용한 위 그림의 예를 보면 숫자들로 이루어진 표와 해당 숫자들이 적힌 악보를 볼 수 있다. 즉, 주사위를 굴려 무작위로 수를 얻고 그 수에 해당하는 악보를 연주하여 음악을 만드는 방식이다. 이러한 방식을 컴퓨터에 이식 시킬 경우, 숫자들이 적힌 악보는 인덱스가 붙은 선을 데이터로, 숫자들의 표는 이 데이터를 인덱스를 통해서 참조하는 배열로 만들어 볼 수 있다. 시간에 의존한 의사 난수 (pseudo-random)를 기반으로 마치 주사위를 굴리듯 수를 선택하고 이 값이

¹ Bonn: N.Simrock, n.d.(1793). Plate 48. 에서 인용, imslp.org에서 재인용

참조하는 선율을 데이터에서 찾는 식을 반복하여 원하는 만큼의 전체 선율을 얻는다.

위의 예에서는 우연성이라는 요소를 도입하여 항상 다양한 음악이 나올 수 있는 가능성을 열었을 뿐 사실 알고리즘적인 방법은 음악에 있어서 불가분의 관계에 있었다. “작곡”이라는 용어는 영어로 “구성하다”라는 뜻을 가진 “composition”이며, 이를 통해 서양 예술 음악에서 작곡에 대한 행위를 음을 소재로 하여 작곡가 자신이 부여한 일정한 규약과 기준들을 토대로 구성하는 예술이라고 여겼음을 알 수 있다. 실제로 서양 예술 음악에서는 일정한 양식과 정해진 형식을 항상 가지며 그 안에서 음들을 구성해왔다. 예를 들면 푸가(fugue)라고 하는 형식을 들 수 있는데 이는 아래의 그림에서 볼 수 있듯이 주제와 응답의 선율로 시작하며, 이 선율을 통해서 곡 전체를 지배한다.²



〈그림 2〉 바하 평균율 피아노곡집 1권 중 C minor 푸가 도입 부분

으뜸음조로 연주되는 선율인 주제가 끝나자마자 딸림음조나 으뜸음조의 딸림화음권 내에서 이 주제가 변형된 선율인 응답이 나온다. 이러한 주제-응답의 구조는 ‘푸가’라고 이름이 붙은 모든 곡에서 똑같이 해당되는 형식이며 마치 정해진 함수를 통과한 후 변환되는 것처럼 일률적이다. 굳이 알고리즘이라는 개념을 차용하지 않아도 이미 많은 음악 내에서 알고리즘적인 과정을 사용하고 있었고 컴퓨터가 등장함에 따라 단지 이 과정을 기계에 맡겼을 뿐이다.

알고리즘의 적용

이어로이드에도 위와 같은 원리가 그대로 적용된다. 즉, 데이터를 구축하거나 무작위로 생성한 선율을 일정한 방법을 이용하여 전체의 구조에 이용한다. 다만 위에서 보인 예와 다른 점은 데이터화하는 음악적 요소와 전체의 구조에 적용하는 방법이 다르다는 것이다. 위의 모짜르트

² [3] Kent Kennan 저. 나인용 역. 대위법 (18세기 양식). 1991. 세광음악출판사 151-158쪽 참조

예에서는 선율, 리듬, 화성 모두 데이터화되었다. 이 데이터 안에서 항목들을 고르기 위해서 난수를 발생시키는 과정 외에는 우연적 요소가 없다. 하지만 이어로이드에서는 리듬과 화성만을 데이터화하였고, 선율은 데이터에서 뽑아낸 화성 진행과 일정한 방향성에 근거하여 제한된 무작위로 구성하는 방식을 취하였다. 이는 위에서 보인 푸가의 구조처럼 함수를 통해 변환되는 모습과 같다.

이러한 두 가지 방법 모두 장단점을 가지고 있다. 데이터화하는 경우의 장점은 이를 통해서 생겨는 결과물을 어느 정도 가늠할 수 있기 때문에 결과물의 질을 쉽게 손볼 수 있다는 것이다. 하지만 이는 장점이자 단점이 되어 데이터화된 자료 이상의 다양한 결과를 내놓지 못한다는 단점도 되어버린다. 데이터에 의존하지 않고 구조에 대한 알고리즘을 만드는 방식의 장점은 이와 반대이다. 즉, 매우 다양한 결과를 내놓을 수 있다는 점이다. 하지만 마찬가지로 구조로 설정한 제약 조건들을 세심하게 마련해놓지 않으면 생각지도 못한 질이 낮은 결과물들을 내놓을 수도 있다는 단점이 있다.

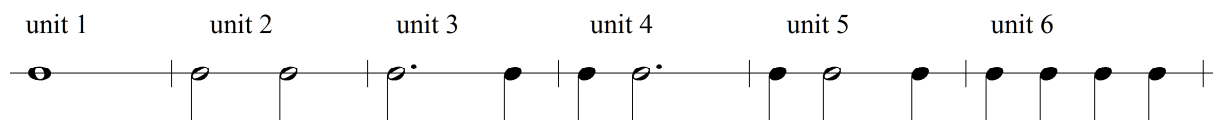
제 3 장 이어로이드 어플리케이션 설계

3.1 악곡 구성 설계 개요

보통의 청음 시험에서는 8마디 내외의 문제를 낸다. 이에 따라 많은 청음 교재들이 8마디의 문제를 제공하고 있다. 이어로이드에서도 이러한 기준을 따라 8마디에 해당하는 악곡을 구성하는 것으로 정했다. 음악을 구성하는 요소로는 리듬, 선율, 화성을 들 수 있다. 리듬은 음을 포함한 각종 음악적 요소들의 규칙적인 배열로써 생겨난다. 선율은 리듬을 포함하는 개념이며, 리듬 위에 음고를 얹으면 선율이 된다. 화성은 선율을 포함하는 개념이며 다수의 선율이 동시에 울림을 통해서 발생된다. 이어로이드에서는 이러한 음악적 3요소에 입각하여 각 요소를 분리시켜 리듬과 화성의 경우에는 데이터화하고, 선율의 경우에는 코드로써 생성원리를 표현하였다. 이러한 각 요소들이 어떻게 설계되었는지 알아보도록 하겠다.

3.2 리듬 구성 설계

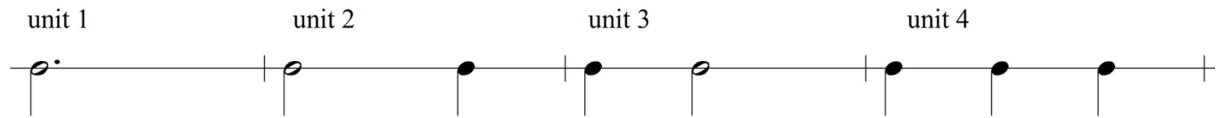
리듬은 미리 정해놓은 패턴들의 조합으로 전체 악곡의 리듬이 구성되는 방식이다. 정해놓은 리듬 패턴의 최소 단위를 ‘리듬 유닛(rhythm unit)’이라고 부른다. 이어로이드에서 지원하는 박자는 3/8, 6/8, 9/8, 2/4, 3/4, 4/4인데 기준 박이 되는 8분음표와 4분음표 별로 리듬 유닛들이 나뉘어진다. 즉, 3/8, 6/8, 9/8끼리 사용할 수 있는 리듬 유닛을 공유하고, 2/4, 3/4, 4/4끼리 리듬 유닛을 공유한다. 각 리듬 유닛들은 음가(note value)를 갖는다. 4분음표 계열의 리듬 유닛들은 4, 3, 2, 1의 음가를 가지며, 8분음표 계열의 리듬 유닛들은 3의 음가를 가진다. 예를 들어, 4분음표 계열의 음가 4를 갖는 리듬 유닛들은 다음 <그림 3>과 같다.



<그림 3> 음가 4/4의 리듬 유닛들

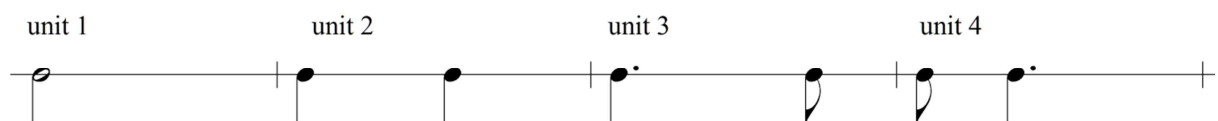
4분음표를 기준으로 한 음가 4만큼의 박을 온음표, 2분음표, 점 2분음표, 4분음표 등의 조합으로 나타낼 수 있는 모든 리듬을 형성한다. 온십표, 2분십표, 점 2분십표, 4분십표도 들어갈 수 있는데 이는 음표가 표기된 후에 프레이즈의 끝에 넣거나, 복잡한 리듬을 만들기 위해 음표 사이에 무작위적으로 들

어 갈 수 있다. 다음 <그림 4>는 4분음표 기준으로 음가 3를 갖는 리듬 유닛들이다.



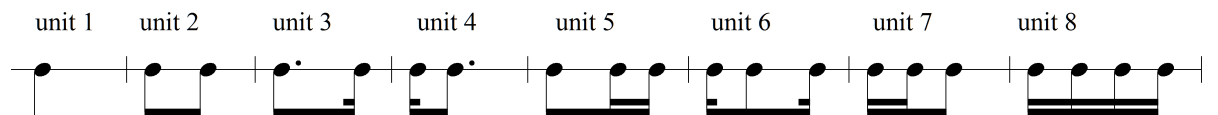
<그림 4> 음가 3/4의 리듬 유닛들

이 유닛들도 마찬가지로 음가 3을 채워 넣을 수 있는 음표들의 조합들로 이루어져 있다. 온음표는 들어갈 수 없으므로 생략되고, 점 2분음표, 2분음표, 4분음표의 조합들로 채워진다. 다음은 4분음표 기준 음가 2를 갖는 리듬 유닛들이다.



<그림 5> 음가 2/4의 리듬 유닛들

음가 2의 유닛들에서는 추가적으로 점 4분음표와 8분음표가 추가된다. 8분음표 4개의 조합으로 음가 2를 구성하는 것도 생각할 수 있으나 이는 밑에서 보일 음가 1을 갖는 리듬 유닛 중에서 두 번째 유닛 두 개의 조합으로 만들어질 수 있으므로 생각하였다. 아래 <그림 6>은 음가 1을 갖는 리듬 유닛들이다.



<그림 6> 음가 1/4의 리듬 유닛들

음가 1을 갖는 유닛들은 4분음표, 8분음표, 점 8분음표, 16분음표들의 조합으로 구성된다. 이제 위에서 보인 음가 4, 3, 2, 1를 갖는 유닛들을 토대로 어떻게 전체 악곡의 리듬을 구성하는가 살펴보기로 하자.

먼저, 악곡 전체의 부분들에 어떠한 음가의 유닛들을 놓을 것인가 결정해야 한다. 이 결정은 무작위 선정으로 이루어진다. 4분의 4박자 악곡의 예를 들어보자. 4분의 4박자 악곡의 경우에는 4분음표 음가가 4개 들어 갈 수 있는 한 마디가 8개 존재한다. 처음에 1마디의 첫째 박에 놓일 음가를 4, 3, 2, 1 중에서 무작위로 결정한다. 만약 결정된 음가가 4보다 작다면 마디가 모두 채워지지 않았으므로 무작위 선정을 다시 한 번 한다. 이 때, 1마디 안에서 남은 음가를 채울 수 있는 음가 중에서만 선택해야 하므로 처음에 선정되었던 음가가를 x 라고 한다면 $4 - x$ 와 같거나 작은 음가 중 골라야 할 것이다. 예를 들어, 처음에 2를 골랐다면 $4 - 2 = 2$ 이므로 2, 1 중에서 다시 선택이 이루어져야 할 것이고, 처음에 3을 골랐다면 그 다음은 100%의 확률로 1을 선택하게 될 것이다. 이러한 과정을

의사코드(pseudo code)로 나타내면 다음과 같다.

```

for measure := 1 to 8 {
  filledValues := 0; index := 1; restValues = 4;
  while (values < 4) {
    values[measure][index] := choose(restValues) // choose에서는 restValues 이하의
                                                    값들을 무작위로 선택하여 리턴한다.
    filledValues := filledValues + values[measure][index]
    index = index + 1
  }
}

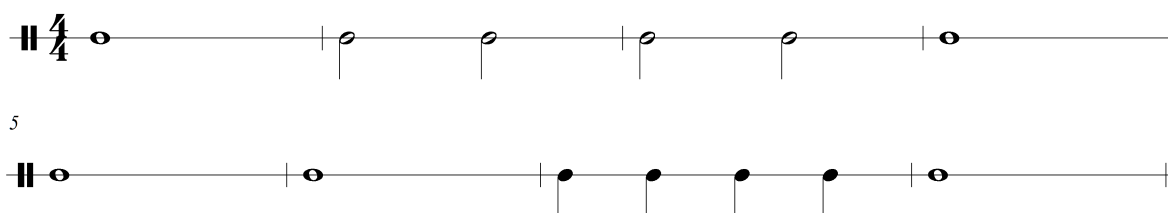
```

. 위 코드에서 measure는 마디 번호, filledValues는 한 마디 안에서 현재 채워진 음가, index는 한 마디 안에서 현재 음가에 해당하는 인덱스, restValues는 한 마디 안에서 남은 음가를 가리킨다. 1부터 8마디까지 진행되며, choose 함수를 통해서 음가가 선택되고 values 이차원 배열 안에 각각 기록된다. 예를 들어 위 알고리즘을 통해서 다음과 같이 values 배열이 채워 졌다고 하자.

	index 1	index 2	index 3	index 4	index 5	index 6	index 7	index 8
index 1	4	2	2	4	4	4	1	4
index 2	/	2	2	/	/	/	1	/
index 3	/	/	/	/	/	/	1	/
index 4	/	/	/	/	/	/	1	/

〈표 1〉 전체 악곡의 음가 배분

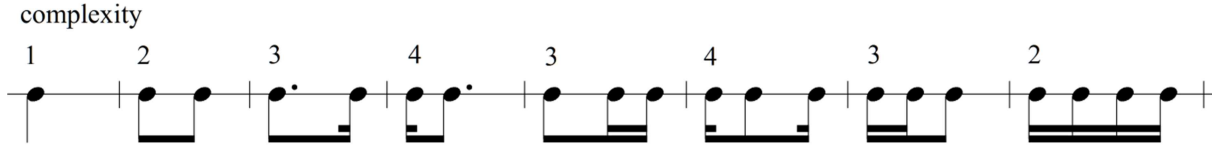
위 표에서 가로축의 인덱스들은 마디 번호를 가리키는 values의 1차 배열의 인덱스이고, 세로축은 각 마디 안에서의 음가 값을 가리키는 values의 2차 배열의 인덱스이다. ‘/’ 문자는 값이 배정되지 않았음을 의미한다. 위와 같이 배열이 만들어진 것을 악보 상으로 표기하면 다음과 같다.



〈그림 7〉 전체 악곡의 음가 배분을 표기한 악보

〈표 1〉에서 values[1][1]의 값이 4이므로 〈그림 7〉의 1마디에는 음가 4를 갖는 온음표가 나온다. 한 마디를 모두 차지 했으므로 values[2]로 넘어가서 2마디를 위와 같이 2분음표 두 개로 채운다. 나머지 마디도 이런식으로 해석 할 수 있다. 이 다음에 할 일은 음가가 배분된 각 부분에 리듬 유닛을 채워 넣는 일이다.

리듬 유닛을 고르는 과정 또한 무작위의 방법을 사용한다. 단, 사용자가 설정한 단계에 따라서 리듬의 난이도를 달리하기 위해 리듬 유닛 마다 정해놓은 복잡도를 고려하여 선택한다.



〈그림 8〉 음가 1/4의 리듬 유닛들의 복잡도

```
for i := 1 to length(values)
  for j := 1 to length(values[i]) {
    complexity := getComplexity(level) // 사용자가 정한 level에 따라 확률적으로
                                         complexity 값을 얻는다.
    rhythmUnits[i][j] := getRhythmUnit(complexity) // complexity에 따라서 무작
                                                    위로 리듬 유닛을 얻는다.
  }
```

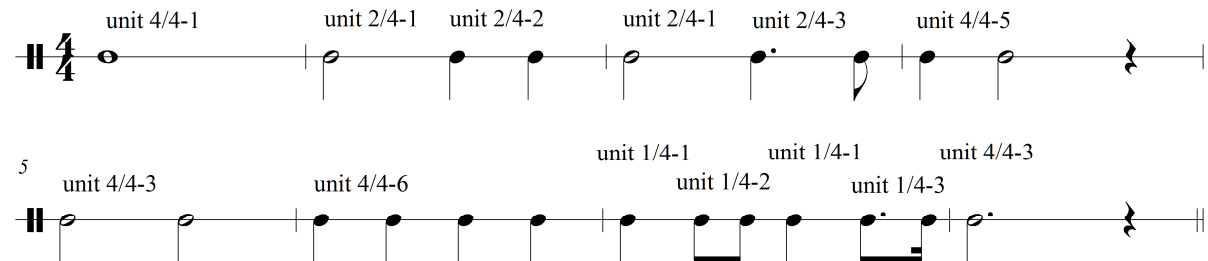
위 코드에서 `length(values)`는 `values` 1차 배열의 길이이고, `length(values[i])`는 `values[i]`에 해당하는 2차 배열의 길이이다. `for`문을 이중으로 중첩하여 `values` 배열을 탐색하면서 리듬 유닛 오브젝트들의 2차원 배열인 `rhythmUnits`에 `complexity`에 따라 무작위로 선택된 리듬 유닛들을 넣는다. 위와 같은 알고리즘을 통해서 `rhythmUnits`에 다음 〈표 2〉와 같은 데이터를 넣을 수 있다.

	index 1	index 2	index 3	index 4	index 5	index 6	index 7	index 8
index 1	unit4/4-1	unit2/4-1	unit2/4-1	unit4/4-5	unit4/4-3	unit4/4-6	unit1/4-1	unit4/4-3
index 2	/	unit2/4-2	unit2/4-3	/	/	/	unit1/4-2	/
index 3	/	/	/	/	/	/	unit1/4-1	/
index 4	/	/	/	/	/	/	unit1/4-3	/

〈표 2〉 전체 악곡 리듬 유닛 할당

위 표에서 가로축의 인덱스들은 마디 번호를 가리키는 `rhythmUnits`의 1차 배열의 인덱스이고, 세로축은 각 마디 안에서의 각 리듬 유닛을 가리키는 `values`의 2차 배열의 인덱스이다. ‘/’ 문자는

값이 배정되지 않았음을 의미한다. 가로축 index 1, 세로축 index 1 즉, 1마디 첫 번째 리듬 유닛에 해당하는 unit4/4-1는 4분음표 기준 음가 4를 갖는 리듬 유닛들 중에서 첫 번째 것을 말한다. 마찬가지로 가로축 index 2 즉, 마디 2에서는 4분음표 기준 음가 2를 갖는 unit2/4-1과 unit2/4-2를 갖는다. 이것을 토대로 악보 상에 표기하면 다음과 같다.



〈그림 9〉 완성된 전체 악곡의 리듬 패턴

〈그림 9〉에서 마디 4와 마디 8에 있는 쉼표는 악곡의 프레이즈 구성을 위해서 음표를 대체한 것이다. 화성 구성 설계 부분에서 다시 보겠지만 악곡은 4마디의 화성 진행의 조합으로 이루어진다. 이 4마디의 화성 진행은 반종지나 정종지로 이루어져 있으므로 마지막에 쉼표를 넣어 주는 것이 음악적 진행에 도움이 된다.

위의 과정을 다시 정리해보자. 처음에 무작위의 과정을 거쳐서 〈표 1〉 또는 〈그림 7〉와 같이 전체 음가 배분 구조를 얻는다. 그런 다음 해당 배분 구조를 기반으로 확률에 기반한 무작위 과정을 거쳐서 해당하는 리듬 유닛을 〈표 2〉 또는 〈그림 9〉와 같이 얻는다. 이렇게 만든 리듬 구조 위에 화성 구조를 쌓아서 선율을 만들 준비를 할 것이다. 이러한 화성 구조를 만드는 과정을 다음절에서 설명한다.

3.3 화성 구성 설계

화성도 리듬과 마찬가지로 미리 정해놓은 패턴들의 조합으로 전체 악곡의 화성 진행이 결정된다. 하지만 리듬과 같이 작은 단위로 나뉘어지지는 않고, 전체 악곡을 이루는 8마디를 두 부분으로 나누어, 4마디 화성 진행의 쌍으로 이루어진다. 4마디 화성 진행은 반종지와 정종지로 끝을 맺으며, 반종지로 끝을 맺는 화성 진행과 정종지로 끝을 맺는 화성 진행이 하나의 쌍을 이루어 전체 악곡의 화성 진행을 형성한다. 화성 진행을 이루는 각 화성들은 음도 이론에 의한 표기와 화성적 리듬에 해당하는 음가를 가진다. 조는 기본적으로 C를 중심으로 표기되나, 사용자가 문제 생성시에 지정한 조로

조음김된다. 또한 화성 진행은 3/8, 6/8, 9/8, 2/4, 3/4, 4/4 등의 박자를 속성으로 갖는다. 예를 들어서 다음과 같은 4/4박자의 장음계, 반중지에서 나올 수 있는 화성 진행을 만들어 볼 수 있다.

$C : I(4) - V7/a(4) - vi(2) - ii56(2) - V(4)$

위의 화성 진행은 C조를 중심으로 5개의 화성의 진행을 나타낸다. 각 화성 진행은 음도 이론으로 표기된 도수 즉, I, II, III, IV, V, VI, VII 등을 가지며, 숫자 저음으로 전위표기를 한다. 괄호 안의 숫자는 화성적 리듬 즉, 음가를 나타낸다. 예를 들어 화성 진행의 첫 화성인 I(4)는 4박을 갖는 I화음을 뜻하며, 네 번째 ii56(2)는 2박을 갖는 ii 7화음의 1전위 형태를 나타낸다. 마지막의 화성이 V화음인 것으로 반중지를 하는 것을 알 수 있다.

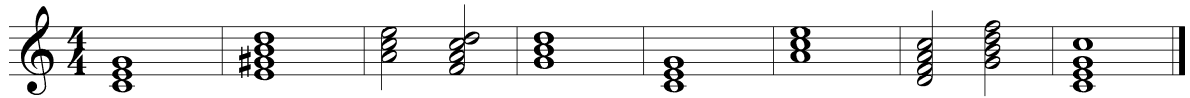
프로그램에서는 위와 같은 화성 진행을 여러 개 미리 준비해놓고 그 중에서 무작위로 선택하여 전체의 화성진행을 조합해낸다. 예를 들면 아래와 같은 여러 화성 진행 리스트를 만들어 놓는다.

번호	중지 유형	화성 진행의 예
1	반중지	$C : I(4) - V7/a(4) - vi(2) - ii56(2) - V(4)$
2		$C : I(4) - V7/a(4) - vi(2) - V7/F(2) - V(4)$
3		$C : I(4) - V7(2) - vi(2) - IV(2) - ii56(2) - V(4)$
4		$C : I(4) - V7(2) - vi(2) - IV(2) - V7/F(2) - V(4)$
5	정중지	$C : I(4) - vi(4) - ii7(2) - V7(2) - I(4)$
6		$C : I(4) - V7/a(2) - vi(2) - ii7(2) - V7(2) - I(4)$
7		$C : I(4) - vi(2) - V7/d(2) - ii7(2) - V7(2) - I(4)$
8		$C : I(2) - V7/a(2) - vi(2) - V7/d(2) - ii7(2) - V7(2) - I(4)$

〈표 3〉 화성 진행의 예

. 위의 표에서는 8개의 화성 진행을 보여준다. 번호 1, 2, 3, 4은 반중지의 화성 진행이고, 5, 6, 7, 8은 정중지의 화성진행이다. 반중지의 화성 진행은 보는 것 같이 V화음으로 마무리되며, 정중지의 화성 진행은 I화음으로 마무리된다. 전체 악곡의 화성 진행은 반중지와 정중지의 화성 진행을 결합하여 이루어지므로 1, 2, 3, 4 중에서 하나, 5, 6, 7, 8 중에서 하나를 무작위로 선택하여 연결 시키면 된다. 표에 나온 모든 화성 진행은 그 자체로 완전한 악절을 형성하므로 그 앞이나 뒤로 어떠한 악절이 나와도 상관이 없다. 예를 들어 1번과 5번의 화성 진행을 결합하면 다음과 같은 전체 악곡의 화성 진행을 얻을 수 있다.

위의 화성 진행을 악보에 옮겨보면 다음과 같다.



〈그림 10〉 완성된 전체 악곡의 화성 진행

위 표에서 나타난 화성 진행은 7음의 해결, 이쑈름 해결과 같은 성부의 진행 상의 문제는 고려하지 않은 채 화성의 구성음과 전위 상태만을 나타내었다. 이 화성 진행 상의 구성음들을 토대로 선율을 구성하며 이 때 성부의 진행 문제에 대해 고려하게 된다. 또한 단선율의 경우 사실상 전위 상태는 무시된다. 이러한 선율 구성 방법에 대해서 다음 절에서 알아보겠다.

3.4 선율 구성 설계

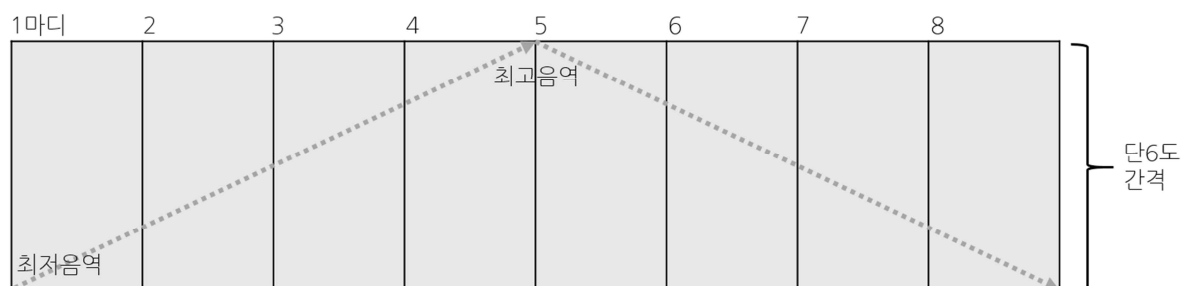
선율은 화성적 배경을 내포한 음고들의 집합에 리듬이 붙여진 음악적 요소라고 할 수 있다. 그렇기 때문에 선율은 앞서 설명한 리듬을 포함하는 용어이고, 또한 화성을 암시하기도 한다. 따라서 앞서 만들었던 리듬과 화성 다음에는 정확히 말하자면 음고들의 집합이 설정되어야 한다. 그러나 좁은 의미에서 선율은 역시 이 음고들의 집합을 의미하기도 한다. 따라서 이 절에서 설명하는 선율은 이러한 좁은 의미로 주로 사용될 것이다.

조성 선율의 생성

선율을 생성하는 과정을 차례대로 설명하자면 다음과 같다.

1. 선율의 방향성과 음역을 설정한다. 음역은 난이도에 따라서 결정된다. 반음 간격을 1로 두고 난이도1에서는 8, 난이도 2에서는 9, 난이도3에서는 10, 난이도4에서는 12, 난이도5에서는 14, 난이도 6에서는 17, 난이도 7에서는 20, 난이도 8에서는 24로 정해진다. 예를 들면 난이도1에서는 최고음과 최저음의 반음의 개수가 8개에 해당하는 단6도 차이 난다는 뜻이다. 난이도 8에서는 마찬가지로 계산하여 두 옥타브의 차이가 나게 된다. 방향성은 최저음역에서 시작하여 5마디 째에서 최고 음역에

달하고 다시 8마디까지 최저음역으로 떨어지는 구조를 만들었다. 난이도1의 예를 도식화하면 다음과 같다.



〈그림 11〉 음역과 방향성의 설정

2. 설정한 방향성과 음역을 고려하고, 주어진 화성 진행에 따라 음고를 생성한다. 〈그림 9〉에서의 리듬과 〈그림 10〉에서의 화성을 사용하면 〈그림 12〉와 같은 결과를 얻을 수 있다. 그림을 보면 대략적인 방향성을 〈그림 11〉과 맞아떨어지는 것 같지만, 세부적으로는 들쭉날쭉 하여 가지런하지 않은 것을 알 수 있다. 이는 전체적인 방향성을 유지하면서도 세부적으로는 확률을 이용한 무작위 선택 방법을 이용하였기 때문이다. 음고가 단조 증가하거나 단조 감소하는 선율보다는 진동하면서 올라가고 내려가는 선율이 보다 음악적이다. 또한 최고음과 최저음의 차이가 단6도 내가 아니라 옥타브가 나는 것을 볼 수 있다. 5마디에서는 I도 화음이 나와 하고, 단6도 이내의 음이 선택된다면 ‘술’음이 선택 되어야할 것이다. 하지만 이런 경직된 제한성은 선율을 음악적으로 만들지 못하므로 이 또한 확률을 통해서 거시적인 선율의 방향성을 유지하는 가운데 음이 선택되도록 한 결과이다.



〈그림 12〉 음역과 방향성의 설정 후에 만들어진 선율

3. 위와 같이 화성음들을 이용하여 선율을 만들고 나면 음의 해결 과정이 진행된다. 조성음악에서

이름³의 해결과 7음⁴의 해결은 필수적이다. 하지만 위의 악곡에서 2마디에서의 ‘솔#’음과 7마디에서의 ‘시’음이 해결되지 않고 있다. 그러므로 처음부터 끝까지 화성 진행과 음의 해결상태를 체크하고 해결이 되지 않은 경우 이를 바로 잡아주는 과정이 필요하다. 하지만 7마디의 ‘도’음과 같이 자연해결 되는 경우는 확률에 의해 부분적으로 해결한다. 이러한 결과를 <그림 13>에서 볼 수 있다.



<그림 13> 음의 해결 과정을 거친 선율

4. 그 다음에는 비화성음을 설정하게 된다. 비화성음은 경과음, 전타음, 보조음, 선행음, 계류음 등⁵이 있으며, 이미 만들어진 선율의 상태에 따라 부분적으로 적용된다. 예를 들면 <그림 14>에서 마디 3에서 기존의 ‘레’음이 ‘솔’음으로 바뀌어 경과적 비화성음으로 된 것을 볼 수 있다. 또한 마디 6에서 셋째 박의 ‘미’가 솔#으로 바뀌어 보조적 비화성음이 되었다. 그러나 전타음, 선행음, 계류음 등은 마땅히 들어갈 자리가 없기 때문에 추가되지 않은 것을 확인할 수 있다.



<그림 14> 비화성음이 적용된 선율

³ V화음에서의 3음, 으뜸음과 단2도 차이 나는 음계에서의 7음을 말한다. 이는 I화음의 근음 즉, 으뜸음으로 가려는 성질을 지닌다.

⁴ 7화음에서의 7음을 말한다. 이는 2도 하행하여 해결되려는 성질을 지닌다.

⁵ [1] 백병동, 화성학 (증보판), 2008, 수문당 7장 참고

5. 붙임줄이 들어갈 수 있는 가능성을 체크하고 정해놓은 확률에 따라 붙임줄을 붙인다. 난이도가 상승할수록 붙임줄이 붙여질 확률이 올라간다. 다음 <그림 15>에서 6마디와 7마디 사이에 ‘라’음을 공유하므로 붙임줄이 들어갈 수 있고, 이를 적용한 것을 볼 수 있다.



<그림 15> 붙임줄이 들어간 선율

위에 열거한 과정들을 거쳐서 최종적으로 선율이 완성되었다. 완성된 선율은 문제듣기 화면을 통해서 볼 수 있다.

제 4 장 이어로이드 어플리케이션 구현

4.1 악곡 구성 구현 개요

앞 4장에서 살펴본 악곡의 설계를 토대로 실제 알고리즘을 구현한다. 앞서 본 바와 같이 화성과 리듬은 미리 정해놓은 데이터들을 이용하고, 선율은 설계된 내용에 따라 알고리즘을 구현하여 생성한다. 프로그램에서 사용자는 문제 생성 인터페이스를 통해서 자신이 원하는 음악적 요소들을 선택한다. 그런 다음 모든 악곡 구성 과정이 전적으로 프로그램 내부적으로 이루어져 최종 결과를 사용자에게 보여준다. 이러한 실제 구현 과정이 어떻게 되는지 살펴보도록 하겠다.

문제 매개변수 전달

문제를 생성하면 무조성의 문제는 SeriesMelody 클래스에 조성의 문제는 TonalMelody 클래스의 인스턴스를 각각 만들어낸다. 이 때, 아규먼트로 선택한 내용들 즉, 단계, 박자, 조표, 음자리표, 조성, 빠르기 등이 전달된다. 다음은 SeriesMelody 클래스와 TonalMelody 클래스의 생성자이다.

```
public TonalMelody(UnitDB rDB, HarmonicDB hDB, int beats, int beat_type, int level, int tempo, int fifth, int mode, String clef)
```

```
public SeriesMelody(UnitDB unitDB, HarmonicDB hDB, int beats, int beat_type, int level, int tempo, String clef)
```

아규먼트로 주어지는 UnitDB는 리듬 유닛 데이터 클래스이며 2.1.3절에서 상세히 설명한다. HarmonicDB는 화성 데이터 클래스이며 2.1.5절에서 상세히 설명한다. beats는 박자에서 분자에 해당하는 박의 수이고 2, 3, 6, 9 등의 integer 값을 가진다. beat_type은 박자에서 분모에 해당하는 기준이 되는 박이며 4, 8 등의 integer 값을 가진다. level은 단계의 값이며 0 - 7의 integer 값을 가진다. tempo는 선택한 빠르기를 템포로 변환한 값이다. andante의 경우 50 andantino는 60, moderato는 70, allegretto는 80, allegro는 90의 값을 갖게 된다. fifth는 조표의 개수를 의미한다. 5도권의 진행에 따라 조표가 없을 때는 0의 값을 갖고, sharp(#)이 하나일 때는 1, 두 개일 때는 2, 이런 식으로 1부터 7의 값을 채워나가고, 반대 방향으로 flat(b)이 하나일 때는 -1, 두 개일 때는 -2, 이런 식으로 -1부터 -7의 값을 채워나가서 결국 -7부터 7까지의 값 중 하나를 갖게 된다. mode는

장조인지 단조인지를 가리키며 장조의 경우 0의 값을, 단조의 경우 1의 값을 갖는다. clef는 음자리표를 나타내며 높은 음자리표의 경우 문자열 “G”를 갖고, 낮은 음자리표의 경우 문자열 “F”를 갖는다.

문제 생성 과정

문제를 생성하는 과정은 무조성의 경우와 조성의 경우 각각 다른 단계를 거친다. 조성의 경우가 무조성의 경우보다 훨씬 더 복잡하다. 왜냐하면 무조성의 경우엔 순전히 random 함수를 통해서 12음을 균등하게 나열할 수 있는데 반해, 조성의 경우에는 해당 조와 각 마디, 각 화성적 리듬 안에서의 화성에 맞게 음들을 선택하고, 이를 기준으로 다시 비화성음을 추가하고 해결하는 과정이 추가되기 때문이다. 먼저 조성의 경우, 다음과 같은 코드를 통해서 생성 과정의 윤곽을 볼 수 있다.

```
public TonalMelody(UnitDB rDB, HarmonicDB hDB, int beats, int beat_type, int level, int tempo, int fifth, int mode, String clef)
{
    ...필드 변수 초기화 과정 생략...
    setRhythm();
    setMelody();
    setSolvingTone();
    try {
        setNonHarmonic();
    } catch (InterruptedException e) {
        ...에러처리 과정 생략 ...
    }
    setCheckTie();
    setScoreData();
}
```

setRhythm, setMelody, setSolvingTone, setNonHarmonic, setCheckTie, setTieConsistent, setScoreData 이렇게 7가지의 과정을 거쳐서 조성 문제가 완성이 된다. 각 메소드들은 리듬과 선율을 만들고 그 데이터를 정해진 자료구조 형식에 맞게 저장한다. 리듬과 선율을 만드는 자세한 과정은 각각 2.1.2절과 2.1.3절에서 다룬다. 여기서는 각 메소드들의 기능을 언급하고 넘어가겠다.

setRhythm : 리듬 데이터로부터 얻은 리듬 유닛들에서 원하는 복잡도에 해당하는 리듬을 선별한 후 마디에 맞게 전체 리듬을 구성한다.

setMelody : setRhythm에서 만든 리듬의 총 개수를 파악하고, 음자리표와 난이도에 따라 음역을 제한시킨다. 그리고 화성 데이터로부터 화성 진행을 얻은 후 고려한 제약조건들에 맞춰 선율을 만들어낸다.

setSolvingTone : setMelody에서 만든 선율은 정해진 화성 내에서 무작위로 음을 생성한 것이므로 조성음악에서 필수적으로 필요한 이퀄음 해결이 안되어 있다. 해당 조의 V화음과 VII화음의 경우에 이퀄음이 있는 경우 다음의 화성적 리듬 안에서 반드시 해결되도록 음을 수정한다.

setNonHarmonic : 위 과정까지 만들어낸 결과는 모든 음이 화성음이다. 선율을 더욱 매끄럽게 만들기 위해 몇몇 화성음들을 알맞은 비화성음으로 교체해야 한다. 이 때, 비화성음은 조성음악에서 사용하는 경과음, 전타음, 보조음, 선행음, 계류음 등이 적용된다.

setCheckTie : 설정한 난이도에 따라 붙임줄을 추가한다. 붙임줄은 멜로디를 더욱 복잡하게 하는 요소가 된다.

setScoreData : ScoreData 클래스의 인스턴스를 만든다. ScoreData 클래스는 실제로 화면에 악보를 보여주기 위해 필요한 음자리표, 음표, 쉼표 등의 위치정보들을 담고 있는 클래스이다.

무조성 문제의 경우, 위에 보인 조성의 경우보다 훨씬 간결하다. 다음 코드는 무조성 문제를 만들어내는 SeriesMelody 클래스의 생성자 소스이다.

```
public SeriesMelody(UnitDB unitDB, HarmonicDB hDB, int beats, int beat_type, int level,
int tempo, String clef)
{
    ...필드 변수 초기화 과정 생략...
    deleteacci();
    addMelody();
}
```

deleteacci : 무조성의 경우 조표가 붙지 않으므로, 각 노트의 환경변수로 지정된 조표들을 모두 지워주는 작업이 우선적으로 들어간다.

addMelody : 무조성을 대표하는 선율 작곡법 중 하나인 12음렬 기법에 의해서 멜로디를 만든다. 난이도에 따라 음역대를 설정하고 12음렬을 무작위로 설정한 후, Original, Retrograde, Inversion, Retrograde-Inversion 쌍을 이어 붙인다.

4.2 리듬 구성 구현

리듬 데이터

이 프로그램에서 리듬은 기본적으로 리듬 데이터에 이미 저장되어 있는 리듬 유닛⁶들의 조합으로 구성된다. 리듬 데이터는 MusicXML⁷ 형식으로 구성되어 있다.

XML을 데이터의 가장 상위 요소는 <unit>이다. <unit>요소는 속성으로 beat_type, value, division, type를 갖는다. beat_type은 박자에서 분모에 해당하며 박을 세는 기준에 되는 박을 뜻하고 4, 8 등의 값을 갖게 된다. complexity는 난이도이다. 리듬 유닛의 난이도를 1부터 6까지로 분류해 놓았으며 문제의 난이도에 따라서 선택하게 된다. division은 beat_type의 박을 기준으로 리듬 유닛의 박 수를 표시한 것이다. 예를 들어, beat_type이 4이고 division이 4이면 4/4박자 기준으로 온음표에 해당하는 음가를 가지게된다. type은 리듬의 종류를 가리키는데 “start”, “normal”, “end”의 문자열 값을 가질 수 있다. “start”의 경우 악곡을 시작하기에 용이한 리듬이다. 즉, 해당 박자를 비교적 정확하게 파악할 수 있는 박절적 리듬⁸을 가진 리듬 유닛에 해당한다. “normal”은 곡 중간에 어디에 놓여도 좋은 리듬을 말한다. “end”는 곡 마지막에 놓이면 좋은 리듬이다. 종지를 위해서 길게 끄는 리듬을 “end” 종류로 분류하였다.

<note> 요소의 하위 요소로 <note>들이 들어간다. 원하는 리듬을 만들기 위해 여러 개를 넣을 수 있는 요소이다. note는 실제 악보상의 음표 하나에 대응한다. note는 별다른 속성 값을 가지지 않으며 하위 요소들을 통해 하나의 음표를 구현한다. <note>의 하위 요소로는 <type>, , <dot> 등이 들어간다. <type>은 음가(note value)를 나타낸다. 즉 온음표를 나타내는 “whole”, 2분음표를 나타내는 “half”, 4분음표를 나타내는 “quarter”, 8분음표를 나타내는 “8th”, 16분음표를 나타내는 “16th” 등의 문자열 값을 가진다. 은 강박에 놓인 음표를 표시하며 이는 후에 멜로디를 만들 때 비화성음을 구성할 때 사용된다. 예를 들어, 경과음은 강박과 강박 사이의 약박에 놓이는

⁶ 한 박이나 두 박 등의 주어진 화성적 리듬 안에서 표현 가능한 리듬들의 집합을 나타낸 것이다. 일반적인 4분음표, 8분음표 등에서부터 붓점 리듬 같이 복잡한 리듬들도 표현된다.

⁷ 악보를 XML 형식으로 표기하는 방법이며 finale, sibelius를 비롯한 대부분의 악보 작업용 소프트웨어에서 지원한다.

⁸ 약박이 강조되거나 쉼표로 시작하는 리듬, 또는 음표에 점이 많이 쉼인 리듬들은 박절을 파악하기 쉽지 않다. 강박이 강조되고 쉼표가 뒤에 놓이며, 점이 없는 음표들은 상대적으로 박절을 잘 파악할 수 있다.

비화성음이고, 전타음은 강박에 놓이는 비화성음, 선행음은 강박의 바로 앞 약박에 놓이는 비화성음이다. 비화성음의 구성에 있어서 이러한 박절 관계의 고려가 필수적이므로 리듬을 형성하는 때부터 강박과 약박에 대한 속성을 부여하였다. <dot>은 음표에 붙는 점을 뜻한다. 한 음표에 여러 개가 붙을 수 있지만 보통 한 개나 두 개가 붙는다. 점4분음표를 표현하기 위해서는 <type>에 “quarter” 값을 대입하고, <dot> 요소를 하나 추가시켜서 만들 수 있다.

실제 악보의 리듬이 MusicXML로 어떻게 변환되는지 예를 들어 보자. 다음 <그림 16>와 같이 4분음표 2개를 갖는 리듬을 생각해보면, <note> 요소가 두 개가 필요하고, 각 <note> 요소의 <type>에 문자열 값 “quarter”가 필요하다.



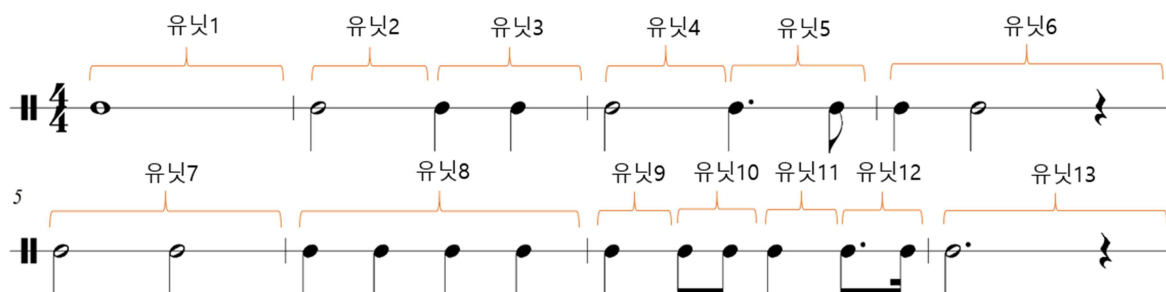
<그림 16> 실제 악보의 리듬

그래서 MusicXML로 변환된 결과는 다음과 같다.

```
<unit beat_type="4" value="2" complexity="1" divisions="1" type="end">
  <note>
    <type>quarter</type>
    <strong/>
  </note>
  <note>
    <rest/>
    <type>quarter</type>
  </note>
</unit>
```

실제 악곡 리듬의 구현

이러한 MusicXML 데이터를 이용하여 실제 악곡에 사용되는 리듬들은 다음에 보이는 <그림 17>와 같이 리듬 유닛들의 배열로 구성된다.



<그림 17> 리듬 유닛들의 배열

4분음표가 기준 박이 되는 4/4박자의 악곡이므로, 모든 유닛들의 beat_type 속성은 4이다. 유닛1, 6, 7, 13의 경우 기준 박에 대한 음가가 4이므로 value 속성 값은 4를 갖는다. 유닛2, 3, 4, 5 등은 value 값으로 2를 갖고, 유닛 9, 10, 11, 12는 value 값으로 1를 갖는다. 유닛 1, 2, 4, 9, 11은 음표가 하나이므로 division 값으로 1을 갖고, 유닛 3, 5, 7, 10, 12, 13은 division 값으로 2를, 유닛 6은 division 값으로 3을, 유닛 8은 division 값으로 4를 갖는다. 유닛 1은 악곡을 시작하는 리듬이므로 type으로 문자열 값 “start”을 갖고, 유닛 13은 “end”를, 그 밖에 나머지들은 “normal”을 갖는다. 이런 식으로 하여 악곡 전체의 XML 계층 구조를 표로 정리하면 다음과 같다.

<unit>계층	beat_type	value	division	type	<note>계층	<type>	기타 요소
유닛1	4	4	1	“start”	note1	“whole”	<stong>
유닛2	4	2	1	“normal”	note1	“half”	
유닛3	4	2	2	“normal”	note1	“quarter”	
					note2	“quarter”	
유닛4	4	2	1	“normal”	note1	“half”	
유닛5	4	2	2	“normal”	note1	“quarter”	<dot>
							
유닛6	4	4	3	“normal”	note1	“quarter”	
					note2	“half”	
					rest	“quarter”	
유닛7	4	4	2	“normal”	note1	“half”	<stong>
					note2	“half”	
유닛8	4	4	4	“normal”	note1	“quarter”	
					note2	“quarter”	
					note3	“quarter”	
					note4	“quarter”	
유닛9	4	1	1	“normal”	note1	“quarter”	
유닛10	4	1	2	“normal”	note1	“8th”	
					note2	“8th”	
유닛11	4	1	1	“normal”	note1	“quarter”	
유닛12	4	1	2	“normal”	note1	“quarter”	

					note2	"8th"	<dot>
유닛13	4	4	2	"end"	note1	"half"	<dot>
							
					rest	"quarter"	

〈표 4〉 MusicXML 계층 구조

4.3 화성 구성 구현

리듬을 생성한 후 화성 데이터를 참조하여 난이도에 따라 알맞은 화성 진행을 선택한다. 리듬 데이터와 마찬가지로 화성 데이터도 XML 형식으로 짜여 있지만 MusicXML과 같은 표준화 또는 상용화되어 있는 방식이 아닌 어플리케이션을 위해 임의로 만든 XML 형식을 갖는다. 화성 진행을 결정하는 음악적 요소에는 박자, 조성 등이 들어간다. 또한 화성 진행에 따라서 선율의 난이도도 결정되므로 역으로 난이도에 따라서 화성 진행이 선택되어야 한다. 예를 들어, 기본적인 주3화음인 I, IV, V 화음을 이용한 화성 진행 보다는 7화음인 I7, IV7, V7 화음을 이용한 화성 진행이나 부3화음인 II, III, VI 화음을 이용한 화성 진행이 더 높은 난이도로 책정할 수 있으며, 부속화음을 이용한 화성 진행 즉, V7/IV, V7/V V7/VI를 이용한 화성 진행 등은 더욱 높은 난이도로 생각할 수 있다. 조성이 달라짐에 따라서 코드(chord)가 변하지만 이는 상대적인 음정 계산에 의해 하나로 통합할 수 있는 부분이다. 예를 들자면, C Major의 코드진행 C-F-G와 D Major의 코드진행 D-G-A는 각각 다르지만 음도이론(stufentheorie) ⁹ 에 따라서 똑같이 I-IV-V로 표기할 수 있다. 적절한 이도(transposition) 기능을 만든다면 C Major와 D Major를 비롯한 모든 12개조로 다시 변환할 수 있다. 지금까지 설명한 내용을 토대로 화성 진행을 결정짓는 요소는 박자, 난이도로 모아짐을 알 수 있다. 이 프로그램에서 지원하는 박자는 3/8, 6/8, 9/8, 2/4, 3/4, 4/4로 6가지가 있고, 난이도는 8단계까지 있으므로 그 조합으로 48가지의 화성 진행 카테고리가 나올 수 있다. 하지만 실제로는 이것을 조금 단순화하여 난이도를 두 단계씩 묶어서 4단계로 줄이고 24가지 카테고리로 만들었다. 한 카테고리 안에 묶여진 두 단계의 난이도 차이는 화성 진행에서가 아니라 리듬의 복잡도 및 선율의 복잡도가 높아지는 것으로 반영되게 하였다. 지금까지 내용을 바탕으로 만든 화성 진행 XML파일들은 다음과 같다.

	1단계	2단계	3단계	4단계
3/8	h138major.xml	h238major.xml	h338major.xml	h438major.xml
	h138minor.xml	h238minor.xml	h338minor.xml	h438minor.xml
6/8	h168major.xml	h268major.xml	h368major.xml	h468major.xml

⁹ 화성을 표기하는 방식은 음악을 향유해온 시대적, 지역적, 장르적 환경에 따라 다르다. 대표적으로 바로크 시대에 유행한 숫자저음(figured bass), 클래식 전반에 걸쳐 사용되는 음도이론, 낭만화성부터 근,현대에 이르는 곡들을 분석하기 위한 기능화성(functional harmony), 20C에 켄트를 비롯한 대중음악에 쓰이는 코드 표기법 등이 있다.

	h168minor.xml	h268minor.xml	h368minor.xml	h468minor.xml
9/8	h198major.xml	h298major.xml	h398major.xml	h498major.xml
	h198minor.xml	h298minor.xml	h398minor.xml	h498minor.xml
2/4	h124major.xml	h224major.xml	h324major.xml	h424major.xml
	h124minor.xml	h224minor.xml	h324minor.xml	h424minor.xml
3/4	h134major.xml	h234major.xml	h334major.xml	h434major.xml
	h134minor.xml	h234minor.xml	h334minor.xml	h434minor.xml
4/4	h144major.xml	h244major.xml	h344major.xml	h444major.xml
	h144minor.xml	h244minor.xml	h344minor.xml	h444minor.xml

〈표 5〉 화성 진행 XML 파일 목록

〈표 6〉에서 보는 바와 같이 단계와 박자별로 XML파일을 만들고 화성 진행을 분류하였다. 파일이름에는 단계, 박자, 조성이 명시되어있다. 예를 들어, h234major.xml 파일은 2단계, 3/4박자, major 조성의 화성 진행들을 담고 있고, h338minor.xml 파일은 3단계, 3/8박자, minor 조성의 화성 진행을 담고 있다. 이미 언급했던 바와 같이 각 XML 파일들의 형식은 임의로 만든 요소 이름들과

```

<harmonyDB>
  <hProgress beat="3/4" level="2" cadence="half" mode="Major">
    <hUnit figure="I" key="C" beats="3"/>
    <hUnit figure="V7" key="a" beats="3"/>
    <hUnit figure="vi" key="C" beats="2"/>
    <hUnit figure="ii56" key="C" beats="1"/>
    <hUnit figure="V" key="C" beats="3"/>
  </hProgress>
  <hProgress beat="3/4" level="2" cadence="auth" mode="Major">
    <hUnit figure="I" key="C" beats="3"/>
    <hUnit figure="vi" key="C" beats="3"/>
    <hUnit figure="ii7" key="G" beats="2"/>
    <hUnit figure="V7" key="C" beats="1"/>
    <hUnit figure="I" key="C" beats="3"/>
  </hProgress>
  ... 중략 ...
</harmonyDB>

```

속성들을 가지고 있다. 예를 들기 위해 h234major.xml의 파일의 내용 중 일부를 보자.

가장 상위 요소인 〈harmonyDB〉로 파일이 시작되고 끝을 맺는다. 〈harmonyDB〉의 하위 요소로 〈hProgress〉들이 여러 개 나열된다. 〈hProgress〉는 실제 화성 진행을 담고 있는 요소이다. 속성으로

beat, level, candence, mode를 가지며, 하위 요소로 <hUnit>들을 갖고 있다. beat는 박자에 해당하며 익히 알고 있듯이 이 프로그램에서 사용되는 박자인 “3/8”, “6/8”, “9/8”, “2/4”, “3/4”, “4/4” 중에 하나를 가지고 있다. level은 난이도를 뜻하며 위에서 언급한 바와 같이 화성 진행의 난이도는 4단계로 분류되므로 1, 2, 3, 4 중에서 하나의 값을 가지고 있다. 전체 문제의 악곡은 8마디 인데 반해 <hProgress>는 4마디의 화성 진행 정보를 갖는다. 이 4마디 화성 진행은 반종지(half cadence)¹⁰로 끝나거나 정종지(authentic cadence)¹¹로 끝이 난다. 그렇기 때문에 앞의 4마디는 반종지의 악절로, 나머지 뒤의 4마디는 정종지로 채울 수 있다. <hProgress>의 속성인 candence는 이렇게 악절이 반종지인지 정종지인지를 구분하는 역할을 하며, 반종지인 경우 “half” 문자열 값을 갖고, 정종지인 경우 “auth” 문자열 값을 갖는다. 위 코드에서 확인 할 수 있듯이 “half” 값을 갖는 경우 마지막 <hUnit>의 figure가 V화음을 뜻하는 “V” 문자열 값을 갖고, “auth”값을 갖는 경우 마지막 <hUnit>의 figure 값으로 I화음을 뜻하는 “I” 문자열을 갖는다. mode 속성은 장조의 화성 진행인지 단조의 화성 진행인지를 구별한다. 장조인 경우 “Major” 문자열 값을 갖고, 단조인 경우 “minor” 문자열 값을 갖는다.

<hProgress> 요소는 화성 진행을 나타내고, 그 화성 진행을 이루는 각 화음들은 <hProgress>의 하위 요소인 <hUnit>에 의해 표현된다. <hUnit>은 하나의 독립적인 화성적 리듬 안에서의 모든 화성적 속성들을 담고 있다. figure 속성은 음도이론 내에서 표기되는 I, IV, V 등의 로마 숫자로 표현된 도수를 말한다. key 속성은 해당 도수의 상대적 조성을 C key를 중심으로 나타낸다. 이는 부속화음을 표현하기 위함인데, 예를 들어 IV에 대한 부속화음인 V7/V는 figure 값 “V7”과 key값 “F”로 표현된다. beat는 박자의 분모 값인 기준 박에 대한 화성적 리듬의 길이를 명시한다. 예를 들어 3/4 박자에서 beat 값을 2로 갖는다면 4분음표 2개에 해당하는 화성적 리듬의 길이를 갖게 되는 것이다. 이런 내용을 토대로 위 XML 데이터에서 표현된 화성 진행은 다음과 같이 악보에 변환시켜 적어볼 수 있다.

¹⁰ V로 종지하는 것, 완전한 종지가 아니며 악곡의 다음 악절로 넘어가게 되거나 불완전한 느낌으로 악곡을 끝맺게 한다.

¹¹ I로 종지하는 것, 완전한 종지가 이루어지며, 악곡이 마무리된다.



〈그림 18〉 화성 진행의 예

위 그림에서 보여주는 악곡에서의 선율은 XML에서 포함하고 있지 않고, 화성 진행의 예시를 보여주기 위해 알맞은 선율을 임의로 고른 것이다. 실제 선율은 다음 절에서 자세히 살펴보겠지만, 위와 같이 변환된 화성 진행을 토대로 음고를 고르고 비화성음들을 추가해서 만들어진다. 앞 절에서 보인 리듬 데이터에서 얻어진 리듬 유닛들의 조합과 선율의 조합으로 최종적으로 하나의 악곡이 완성된다.

4.4 선율 구성 구현

선율 구성 설계에서 보였듯이 선율의 구성은 미리 정해진 데이터에 의존하지 않고 알고리즘적으로 짜여진 선율 구성 원리에 따라 이루어진다. 앞서 5.1절에서 설명했듯이 선율을 생성하면 TonalMelody 클래스를 생성하고 setMelody 메소드를 실행한다. 이 메소드는 다시 HarmonicProgress의 setMelody 메소드를 실행하고, 그 다음 HarmonicUnit의 setMelody를 실행하여 최종적으로 HarmonicUnit 내의 Note 오브젝트의 리스트인 noteList를 구성한다. Note 클래스는 Position 오브젝트를 통해서 음고를 설정하는데, 이 Position 안에는 음고를 나타내는 value와 이를 옥타브, 도수, 변화도로 표현한 octave, step, alter 변수를 갖는다. 아래는 설명한 것에 해당하는 실제 코드를 보여준다.

```
public class TonalMelody {
    ...
    private ArrayList<HarmonicProgress> hProgressList;
    private void setMelody();
}

public class HarmonicProgress {
    ...
    public ArrayList<HarmonicUnit> harmonicUnitList;
    public void setMelody(int[][] refer, int begin, int end, int height,
                        String clef);
}

public class HarmonicUnit {
    ...
    public ArrayList<Note> noteList;
    public void setMelody(int[] refer, int height, String clef) {
    }
}

public class Note {
    ...
    public boolean note;
    public int duration;
    public String type;
    public int dotCount;
    public Position position;
    public boolean strong;
}

public class Position implements Comparable<Position> {
    public int octave;
    public int step;
    public int alter;
    private int value;
}
```

위에서 설명했듯이 TonalMelody, HarmonicProgress, HarmonicUnit은 setMelody 메소드를 가지고

있어서 연쇄적으로 호출한다. 이 메소드들의 기능과 Note 및 Position의 역할에 대해서 차례대로 설명하고자 한다.

TonalMelody 클래스의 setMelody 메소드

이 메소드에서는 전체 8마디 악곡의 선율의 방향성에 대해 결정하고, 각 4마디 악절에 이 방향성에 관한 변수를 보내어 각각에서 선율을 생성할 수 있게 한다. 알고리즘의 자세한 진행 사항은 다음과 같다.

1. 음을 결정하는데 있어서 중심이 되는 음고를 고르기 위해 refer 2차원 배열의 값을 설정한다. 이는 미리 주어진 방향성에 대한 배열을 토대로 음자리표와 레벨에 의해서 이도(transpose)된 후 결정된다.

```
int[][][] funcarr0 =
{{{2,5}, {3,6}, {4,7}, {5,8}, {9,7}, {8,5}, {7,4}, {6,3}}, // 2/4, 3/8 박자
 {{0,5}, {1,6}, {2,7}, {3,8}, {4,9}, {10,5}, {8,3}, {6,1}}, // 3/4, 4/4, 6/8 박자
 {{0,6}, {1,7}, {2,8}, {4,10}, {12,6}, {10,4}, {8,2}, {6,0}} // 9/8 박자
};

int[][][] funcarr1 =
{{{2,6}, {4,8}, {5,9}, {7,13}, {10,16}, {14,8}, {10,5}, {8,3}}, // 2/4, 3/8 박자
 {{0,6}, {2,7}, {4,9}, {6,11}, {8,14}, {14,6}, {10,4}, {8,2}}, // 3/4, 4/4, 6/8 박자
 {{0,7}, {1,8}, {3,10}, {5,12}, {10,14}, {14,10}, {13,7}, {10,5}} // 9/8 박자
};
```

위 코드에 있는 funcarr0 변수는 3차원 배열로써 refer에 들어가는 2차원 배열들의 배열이다. 난이도에 따라서 funcarr0, funcarr1, funcarr2, funcarr3 중에서 하나를 선택하고, 박자에 따라 인덱스를 결정해 2차원 배열을 refer로 넣는다.

2. 1에서 결정한 refer 변수를 중심으로 선율이 얼마나 위 아래로 벌어질 수 있는가를 결정하는 height 변수의 값을 설정한다. 이 값은 미리 주어진 배열에서 난이도에 따라 선택하여 결정된다.

```
int[] heightarr = {8,9, 10,12, 14,17, 20,24};
```

위 코드는 height에 들어갈 값의 배열로써 난이도에 따라 인덱스를 결정해 값을 넣는다.

3. 4마디를 가진 두 악절에 해당하는 hProgressList의 두 항목의 setMelody를 호출한다. 이 때, 1에서 정한 refer 변수와, 2에서 정한 height 변수, 음자리표를 나타내는 clef 등이 매개변수로 전달된다.

HarmonicProgress 클래스의 setMelody 메소드

이 메소드에서는 tonalMelody의 setMelody에서 넘겨 받은 refer, height를 기준으로 하여 화성적 리듬에 해당하는 HarmonicUnit 오브젝트들의 선율을 설정한다. 이에 대한 자세한 진행 사항은 다음과 같다.

1. 넘겨 받은 refer 변수에 의해서, 각 HarmonicUnit들이 가지는 기준 음고인 2차원 배열 refer2를 설정한다. 예를 들어 refer 변수로 다음과 같은 2차원 배열을 넘겨받았다고 하자.

<code>{{0,5}, {1,6}, {2,7}, {3,8}, {4,9}, {10,5}, {8,3}, {6,1}}</code>
--

이는 악곡 전체에 대한 마디별 선율의 방향성이므로 이것을 다시 화성적 리듬별 선율의 방향성으로 나눌 필요가 있다. 첫 번째 4마디에 해당하는 HarmonicProgress를 예로 들면, 위 배열에서 처음 네 항목 즉, {0,5}, {1,6}, {2,7}, {3,8}을 취하게 되고 이 방향성을 다시 화성적 리듬의 길이에 맞게 잘라낸다. 예를 들어 4/4박자에서 1마디의 화성적 리듬이 2분음표 두 박씩으로 나뉜다면, {0,5}의 방향성은 {0,2.5}, {2.5,5}로 쪼개어질 수 있다. 실제로는 integer 배열로 저장하므로 반올림을 하여 {0,3}, {3,5}와 같이 나뉘어진다.

2. 각 HarmonicUnit들의 setMelody를 호출한다. 이 때, 1에서 설정한 refer2 변수와 TonalMelody 클래스로부터 넘겨 받았던 height와 clef를 전달한다.

HarmonicUnit 클래스의 setMelody 메소드

이 메소드에서는 HarmonicProgress의 setMelody에서 넘겨 받은 refer2 변수와 height를 기준으로 음표에 해당하는 Note 오브젝트들의 선율을 설정한다. 이에 대한 자세한 진행 사항은 다음과 같다.

1. 넘겨 받은 refer2 변수에 의해서, 각 Note들이 가지는 기준 음고인 refer3를 설정한다. 이 과정은 위에서 본 HarmonicProgress에서 refer2를 만들어내는 과정과 유사하다. 예를 들어 첫 마디에 넘겨받는 화성적 리듬의 선율 방향성이 {0,3}이고, 해당 화성적 리듬 구역 안에 4개의 음이 존재한다면, 이를 4등분하여 {0,0.75}, {0.75,1.5}, {1.5,2.25}, {2.25,3}으로 나누게 된다. 물론, 위에서와 같이 integer 배열로 저장되므로 반올림하여 실제로는 {0,1}, {1,2}, {2,2}, {2,3}의 배열을 얻게 된다.

2. HarmonicUnit에서 주어진 화성을 기반으로 refer3과, height에 의해서 무작위로 음을 선택한다. 화성은 TonalMelody 클래스의 setMelody가 실행되기 전에 TonalMelody의 생성자에서 이미 결정되어 있다. 이 화성과 위 1에서 살펴본 refer3의 배열과 넘겨 받은 height에 의존한다. 예를 들어,

화성이 V화음이고 refer3이 위에서 예를 든대로 {0,1}, {1,2}, {2,2}, {2,3}, height가 2라면 먼저 refer3에 height를 적용하여 {-1,2}, {0,3}, {1,3}, {1,4}의 배열을 얻는다. 이는 즉, 각 배열의 요소에서 최솟값에 height의 반에 해당하는 값을 빼고, 최댓값에 height의 반에 해당하는 값을 더하는 것이다. 만들어야 할 네 음은 각각 {-1,2}, {0,3}, {1,3}, {1,4}의 범위 안에서 생성된다. V화음은 구성음이 솔, 시, 레인데 이에 해당하는 값은 7, 11, 2 또는 옥타브를 이도시켜 -5, -1, -10이다. 위 배열의 범위 안에서 가질 수 있는 값들을 정리해보면 첫 번째 배열 안에서 -1, 2, 두 번째 배열 안에서 2, 세 번째 배열 안에서 2, 네 번째 배열 안에서 2가 해당된다. 이 값들 중에서 무작위로 값을 고르면 -1, 2, 2, 2의 배열이나 2, 2, 2, 2의 배열을 얻게 된다. 이는 곧 시, 레, 레, 레와 레, 레, 레, 레 해당하는 선율이 된다. 예로 든 것은 기준 음고를 정하는 refer, refer2, refer3 배열들과 높이를 나타내는 height 값이 작아서 폭넓은 선율을 만들지 못했지만, 이 범위를 늘리면 큰 도약이 있는 선율을 만들 수 있게 된다.

Note 클래스

이 클래스는 실제 음표에 대응하는 클래스이다. 위 코드에서 필드 변수 note는 음표인지 쉼표인지를 나타내고, duration은 지속시간, type은 온음표, 2분음표, 4분음표 등의 음가, dotCount는 음표에 붙는 점의 개수, position은 position 클래스에 의한 음고 표기, strong은 강박인지 아닌지를 나타낸다.

Position 클래스

조성의 음고 표기는 생각보다 까다롭게 표기된다. 왜냐하면 같은 음이라 할지라도 그 기능성¹²에 따라 다르게 표기될 수 있기 때문이다. 예를 들어, MIDI 음고 기준으로 61에 해당하는 조성에 따라서 C#4음은 Db4음으로도 여길 수 있다. 이러한 표현을 모두 반영하기 위해서, 순전히 음고에 해당하는 값인 value 이외에도, 옥타브를 나타내는 octave, 도수를 나타내는 step, 변화도를 나타내는 alter로 병행해서 표기한다. value와 octave, step, alter의 관계는 다음과 같다.¹³

$$\text{value} = \text{octave} * 12 + (\text{step} > 2 ? \text{step} * 2 - 1 : \text{step} * 2) + \text{alter}$$

예를 들어 C#4음은 4옥타브를 가지므로 octave 값 4와 C에 대응하는 step 값 0, #에 의해 올림되었으므로 alter 값 1을 갖는다. C#4의 value 값은 따라서 $4 * 12 + 0 * 2 + 1$ 로 계산하여 49이다.

¹² 기능화성과 해당 조에 의한 각 음의 성질

¹³ 식에서 $A ? B : C$ 의 형식으로 표현된 것은 삼항 연산자로서 진리 값 A가 맞으면 B의 값을, 틀리면 C의 값을 갖는다.

이에 반해 Db4음은 4옥타브를 가지므로 octave 값 4와 D에 대응하는 step 값 1, b에 의해 내림되었으므로 alter 값 -1을 값는다. Db4의 value 값은 따라서 $4 * 12 + 1 * 2 - 1$ 로 계산하여 C#4와 마찬가지로 49를 갖는다. 이처럼 동명이음의 경우에 value의 값은 같지만 octave, step, alter의 값은 달라질 수 있다.

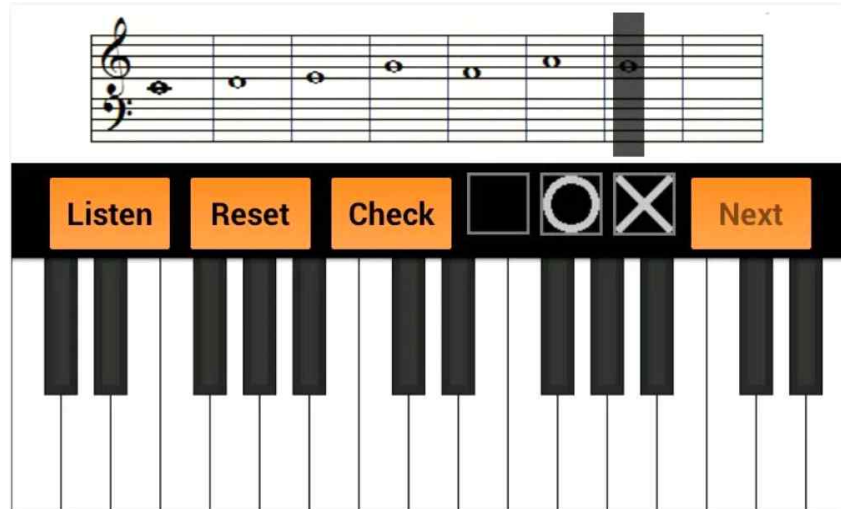
제 5 장 다른 어플리케이션과 비교 및 통계 자료

5.1 다른 어플리케이션과 비교

이제 시중에 나와있는 다른 어플들과 이어로이드 어플을 비교해보고자 한다. 개인용 컴퓨터가 등장한 때부터 줄곧 무작위로 음을 발생시켜 음을 맞추는 프로그램들이 있어 왔다. 하지만 이는 항상 컴퓨터가 있어야 가능했고, 실내에서만 이루어질 수 있었으므로 별다른 효용성을 가져다주지 못했다. 컴퓨터가 있는 실내에서는 실제 피아노 또는 전자 피아노, 기타 다른 악기들을 통해서도 청음 훈련을 할 수 있었기 때문이다. 그러나 모바일 기기가 보편화됨에 따라서 이동 중에도 충분히 연습을 할 수 있게 하는 어플들이 나오기 시작했다. 또한 애플의 앱스토어나 구글의 플레이스토어를 통해서 유료로 어플리케이션을 구매하는 방법이 편리해지면서 좋은 기능을 가진 어플리케이션이 더 많이 나오고 있는 실정이다. 플레이스토어에 올라온 몇 가지 청음 어플을 예로 들고 비교해보겠다.

청음 어플 예1 : 조성 음악 청음

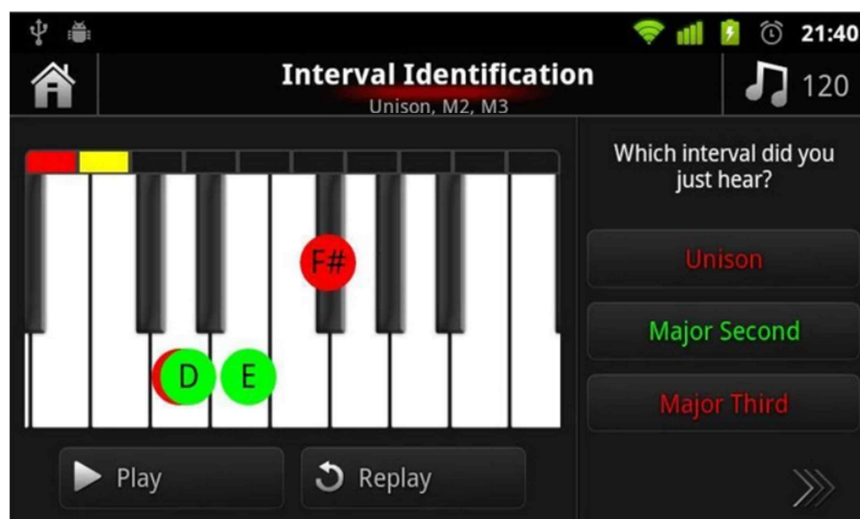
〈그림 19〉에 보이는 화면은 안드로이드용 청음 어플 “조성 음악 청음”의 작동화면이다. 무작위로 임의의 조성내의 8개의 음을 생성해내고 건반 버튼을 눌러 차례로 맞춰가는 방식이다. 완전한 무작위의 무조성이 아닌 조성 내에서의 음들을 생성한다는 것을 특징으로 하고 있다. 건반을 눌러 맞춘음들은 상단의 악보에 기보된다. 모바일 기기를 통해서 최대한 실제 청음 시험에 가깝게 구현되어 있다는 장점이 있으나, 리듬에 대해서는 전혀 고려하고 있지 않고, 난이도가 너무 낮다는 단점을 가지고 있다. 리듬은 모두 일정한 간격의 온음표를 가지며, 조성 문제에 집중하다 보니 임시표가 붙는 높은 난이도의 문제들은 다룰 수 없다. 이에 반해 이어로이드에서는 리듬과 무조에 가까운 문제들도 다룰 수 있다.



〈그림 19〉 안드로이드용 청음 어플 “조성 음악 청음”

청음 어플 예2 : Perfect Ear

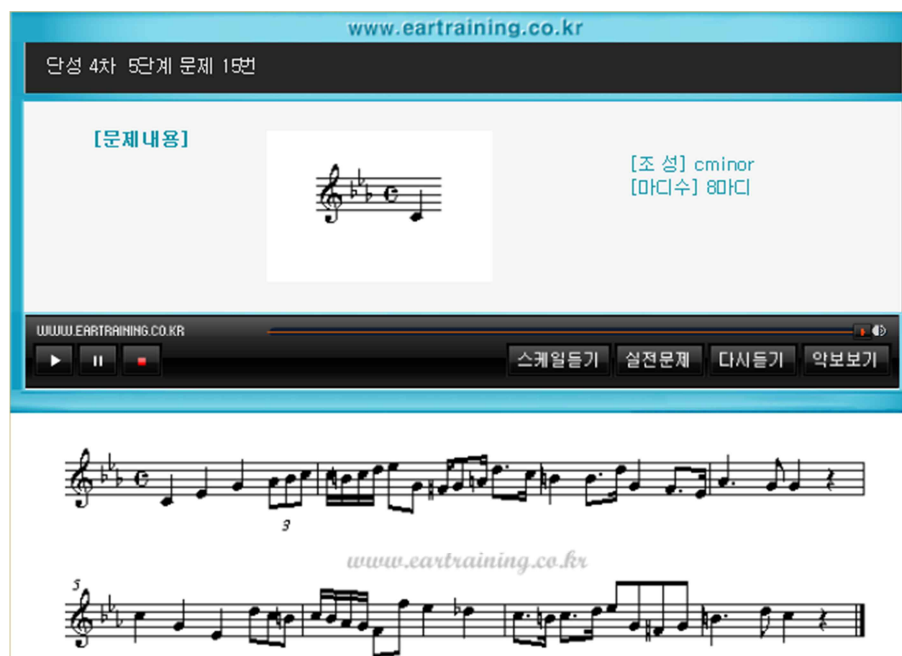
“Perfect Ear”는 EDuckAppSV라는 곳에서 만든 안드로이드용 청음 어플이다. 〈그림 20〉에서 보듯이 건반과 답안 버튼 위주의 인터페이스를 보여준다. 음정(Interval)이나, 선율(Melody), 화음(Harmony) 등의 연습을 할 수 있고 최근에 업데이트된 2버전에서는 리듬 기능과 시창 기능도 추가되었다. 직관적인 인터페이스로 바로 문제에 응답할 수 있는 특징을 가지고 있다. 하지만 이 역시 실제 청음 시험과 같은 리듬이 포함된 선율을 완성된 악곡의 형태로 들려주는 기능은 제공하지 않고 있다.



〈그림 20〉 안드로이드용 청음 어플 “Perfect Ear”

청음 어플 예3 : Ear-Training 웹 사이트¹⁴

Ear-Training 웹 사이트는 청음 연습 프로그램을 제공하고 있으며, 작곡과 입시생들 사이에서 꽤 인지도가 높다. 하지만 모바일 기기를 통해서 청음 연습을 하는 솔루션을 제공하고 있지는 않다. 사실, 이어로이드를 만들면서 이 Ear-Training. 사이트의 문제 제공 기능에 모바일 디바이스의 이점과 랜덤 문제 생성의 이점을 더하는 것에 중점을 두었다. 한국에서 만들어진 사이트인 만큼 한국의 작곡과 내지 청음 시험을 요구하는 학과의 요구사항에 특화되어 개발되었다. 이어로이드와 같이 문제를 재생하고 답을 확인할 수 있다. 단점이 있다면, 항상 컴퓨터를 통해 웹 사이트에서 실행된다는 점과 문제를 일일이 만든 것이므로 같은 문제를 접하게 된다는 것이다. 문제 수를 굉장히 많이 늘려서 이러한 단점을 극복하고 있는 상태이다. 아래 <그림 21>에서 이 웹 사이트에서의 문제 풀이 화면을 보여준다. 화면 상단에서 문제에 관련된 정보화면과 문제를 재생하는데 필요한 인터페이스들을 보여주고 있고, 화면 하단에서 문제를 악보에 표기하여 보여주고 있다. 이러한 기본적인 인터페이스의 원리는 이어로이드와 크게 다르지 않다.



<그림 21> 웹 사이트 “Ear-Training”에서의 문제 풀이 화면

지금까지 시중에 나온 몇 가지 청음 훈련 어플리케이션들을 살펴보았다. 어플들 모두 각자의 특징과 장점을 가지고 있다. “조성 음악 청음” 어플은 무작위적인 음이 아닌 조성 내의 음을 만드는 것과

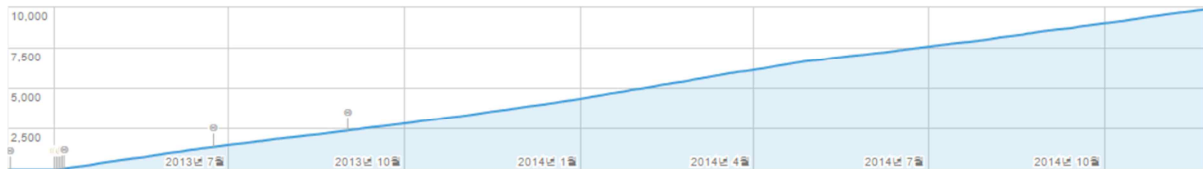
¹⁴ <http://www.eartraining.co.kr>을 통해 접속 할 수 있다.

답을 고르는데 직관적인 인터페이스를 갖고 있는 것을 특징으로 한다. 그리고 “Perfect Ear”는 음정, 화음, 리듬 등을 각각 연습할 수 있는 특징과 “조성 음악 청음” 어플과 같이 건반으로 답을 입력할 수 있는 즉각적인 인터페이스를 제공하는 장점을 가지고 있다. 마지막으로 본 “Ear-Training”은 다른 두 어플 보다 이어로이드에 더 가까운 인터페이스를 제공하며, 즉각적인 반응을 요구하지 않고 문제를 직접 종이 위에 풀어보게끔 만든다. 이는 입시에서 요구하는 문제 방식에 가장 최적화된 시스템이라고 할 수 있다. 이어로이드는 “조성 음악 청음”의 무작위로 조성 문제를 만들 수 있다는 장점과 “Perfect Ear”의 다양한 음정, 화음, 리듬을 내장하고 있다는 장점과 입시에 가장 최적화되어 있는 “Ear-Training”의 장점을 각각 따서 만든 어플이다.

5.2 통계 자료

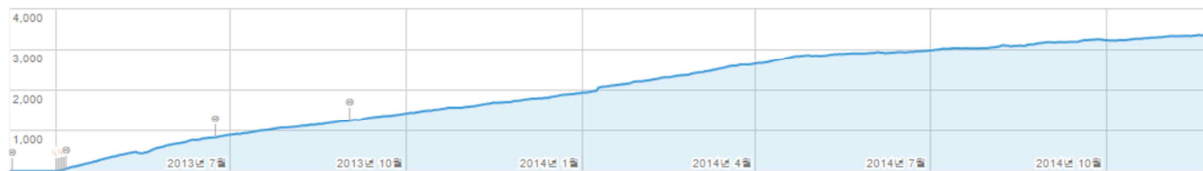
사용자 설치 수

2013년 3월 8일 플레이스토어 등록 이후 2014년 9월 5일까지의 어플리케이션 통계자료는 다음과 같다.



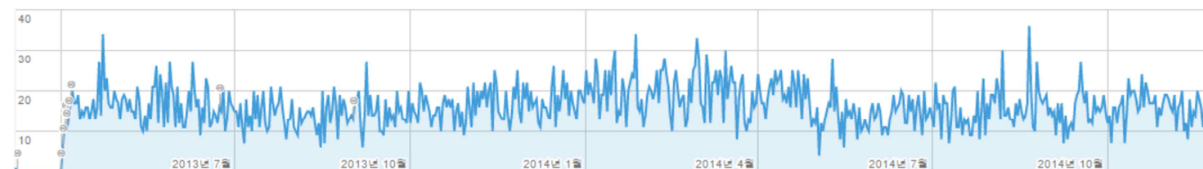
〈그림 22〉 총 사용자 설치 수

총 사용자 설치 수(어플리케이션을 설치했던 사용자의 수)는 현재 8,590명으로, 어플리케이션 등록 이후 꾸준한 증가세를 보이고 있다.



〈그림 23〉 현재 사용자 설치 수

현재 사용자 설치 수(어플리케이션을 현재 설치하여 보유하고 있는 사용자 수)는 3,296명으로, 총 사용자 설치 수와 마찬가지로 어플리케이션 등록 이후 꾸준하게 증가하는 것을 볼 수 있다.



〈그림 24〉 일일 사용자 설치 수

일일 사용자 설치 수는 위 그래프와 같으며 하루 평균 15.70명 정도가 설치한 것으로 나타난다.

이어로이드는 입시 준비를 하는 학생들을 대상으로 한 프로그램이다. 그러므로 매년 프로그램에 대한 수요량의 변동이 크지 않다. 이는 위와 같이 꾸준히 상승하는 총 사용자 설치 수와 현재 사용자 설치 수를 통해서도 확인할 수 있다.

사용자 평점

어플리케이션을 사용하는 사용자들이 평가한 평점의 결과는 다음과 같다.



〈그림 25〉 사용자 평가 점수

대체적으로 높은 평점을 받았지만 최저평점을 받은 경우도 많아서 리뷰를 보고 이유를 분석해보았다. 먼저 좋은 평점을 준 리뷰들은 다음과 같다.



〈그림 26〉 높은 평점을 준 리뷰

높은 평점을 준 리뷰들을 살펴보면 대체로 프로그램이 가진 기능에 주목하는 것을 볼 수 있다. 실제 청음 시험처럼 연습할 수 있고, 청음실력에 따라 난이도 별로 연습할 수 있는 기능 등에 사용자가 좋은 평가를 주고 있으며 그 외에 부가적으로 필요한 기능들을 요구하고 있다. 위의 리뷰에 따르면 문제시작 전에 해당조의 음계(scale)를 들려주는 기능이나, 여러 문제를 한꺼번에 만들어주는 기능들이다. 음계를 들려주는 것은 몇몇 학교입시에서 그렇게 하고 있으므로 업데이트 때 반영해볼 사항이고, 한꺼번에 많은 문제를 만드는 기능은 불필요한 것을 보인다. 프로그램 사용에 있어서 한 문제를 푸는 시간은 5분에서 10분 정도이고, 대체적으로 하루에 두세 개 문제, 많아야 열 개의 문제 정도만을 풀기 때문에 한번에 한 개의 문제를 만드는 것만으로도 충분하다고 본다. 다음은 낮은 평점을 준 리뷰들이다.

★★★★★
앱 버전 1.17
GT-I9100M(GT-I9100M)

Ilian Tchervensky(2013. 12. 7. 오전 10:45)

자동 번역됨(원본 언어: 영어)

루트 액세스 검사기? 그것은 뿌리 전화를 확인하고 휴대 전화가 뿌리를 발견하면 - 종료! 왜? developer는 이메일에 응답하지 않습니다. 그것은 UNROOTED 전화에 잘 작동합니다. [원본 리뷰 숨기기](#)

원본 리뷰

Root access check?!? It checks for rooted phone and if it finds that the phone is rooted - shuts down! Why? The developer does not respond to emails. It works fine on unrooted phone.

★★★★★
앱 버전 1.17
Galaxy Mega 6.3(meliusitespr)

Jacob Faturechi(2014. 1. 14. 오전 1:48)

자동 번역됨(원본 언어: 영어)

루트 액세스 루트 액세스를 시도합니다. 매우 의심스러운. 조심하십시오. [원본 리뷰 숨기기](#)

원본 리뷰

Root access Attempts to gain root access. Highly suspicious. Beware.

★★★★★
앱 버전 1.17
Galaxy Tab 7.0 Plus(GT-P6200)

Serene Cheam(2014. 4. 23. 오전 2:29)

자동 번역됨(원본 언어: 영어)

그 정말 나쁜, 그 음악성 하지 디자인 등. 자동이 응용 프로그램에서 생성 음악 노트는 학생들에게 아주 나쁜 느낌을 제공합니다. 그것뿐만 아니라 모든 구문을 무시합니다. [원본 리뷰 숨기기](#)

원본 리뷰

its really bad, as its not musicality design. music notes that auto generate from this apps give a very bad sense for students. it ignore all the phrases as well.

낮은 평점을 준 리뷰들을 살펴보면 대체로 루트 액세스 검사와 비음악적인 청음 문제의 선율에 대해 언급하고 있는 것을 볼 수 있다. 루팅된 핸드폰의 경우 결제 크래킹의 가능성이 있어서 막아놓기 위해 프로그램 실행 초기에 루트 액세스 검사를 하여 프로그램을 중지시킨다. 대체로 외국에서 루팅된 기기로 프로그램을 이용하려는 시도가 많은 것 같고, 루트 액세스 검사 자체를 의심스러운 프로그램 작동으로 여겨서 낮은 평점을 준 경우가 많다. 그러나 루팅 액세스 검사는 결제 보안 문제에 있어서 근본적으로 차단할 수 있는 좋은 방법이므로 이것을 해제할 필요는 없다고 본다. 비음악적인 선율 문제에 대해서 언급하자면, 한국의 입시에서는 학생들간의 변별력을 위해 무조적 혹은 고의적으로 비음악적으로 나타난 선율을 이용하고 있고, 프로그램은 이러한 경향에 맞추어 제작한 것이므로, 상황이 다른 외국에서는 다른 호응도를 보일 수 있다고 생각한다.

제 6 장 결 론

지금까지 안드로이드용 청음 훈련 어플리케이션 이어로이드에 대한 사용법 및 작동원리, 사용자 평가 등을 살펴보았다. 이어로이드는 청음 입시를 준비하는 학생들이 손쉽게 혼자서 학습할 수 있도록 만들어진 어플리케이션이다. 청음 입시 문제에 적합한 악곡을 컴퓨터의 기능을 활용하여 만들어주고 재생해준다. 악곡을 구성하는데 필요한 난이도, 박자, 조성, 음자리표, 빠르기 등을 지정하면 그에 맞추어 컴퓨터 내부에 구현된 알고리즘을 통하여 문제를 만들어준다. 리듬과 화성의 경우에는 미리 컴퓨터의 데이터로 변환해 놓은 자료들을 토대로 무작위적으로 생성해내고, 선율의 경우에는 화성법 및 대위법에 입각한 진행 방법을 코드화 시켜서 제한된 무작위적 수법을 통해서 생성해낸다.

이렇게 하여 기존에 있던 문제은행 방식의 청음 훈련 프로그램이나 문제집들에서 있었던 문제량이 불충분하다는 단점을 극복할 수 있었다. 이어로이드 프로그램은 실제적으로는 거의 무한대라고 표현할 수 있는 양의 문제들을 생성할 수 있기 때문이다. 또한 전적으로 무작위적인 방식을 선택한 프로그램의 단점인 입시유형에 맞지 않다는 점과 비음악적인 선율, 리듬이 고려되지 않은 선율을 극복할 수 있었다. 이어로이드는 8 마디의 규격화된 악곡을 제시하기 때문에 매우 입시유형에 적합한 선율을 만들어내며, 음악적으로 매우 제한된 무작위적 수법을 이용하므로 어느 정도 선율의 음악적 흐름을 확보한다. 이러한 장점들을 계기로 현재 많은 수의 사용자들이 사용하고 있으며, 입시의 유형이 변하지 않는 한 꾸준히 애용될 것으로 예상된다.

참고자료: 사용자 설명서

1. 개요

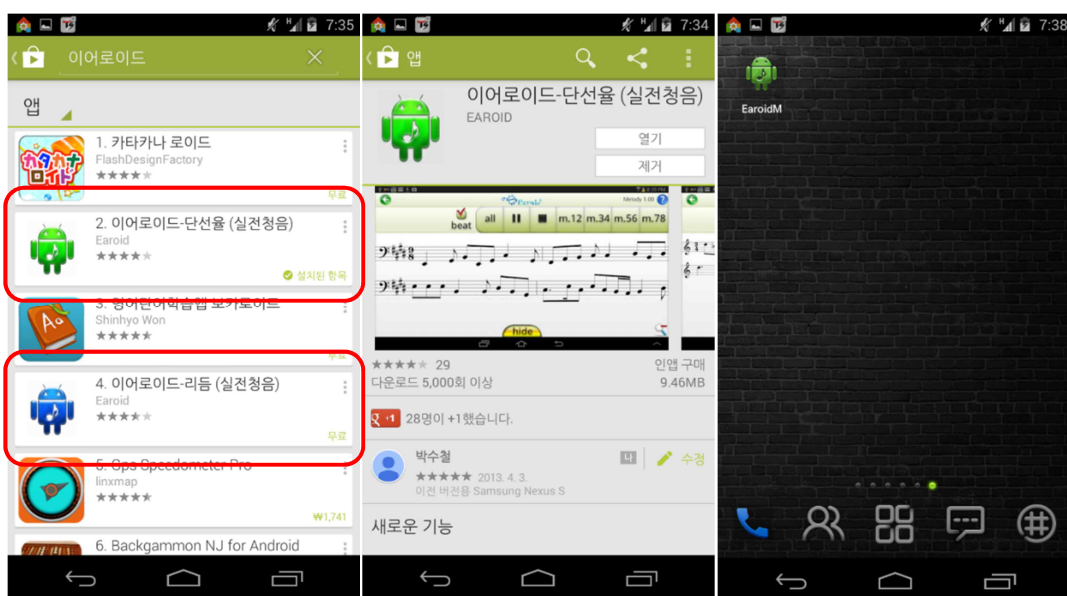
이어로이드는 작곡과를 비롯한 청음시험을 요구하는 음대 입시에 맞추어 개발된 청음훈련 프로그램이다. 기존에 나와 있는 청음 연습 프로그램은 무작위로 음을 생성하여 비음악적인 선율 패턴을 가지고 있거나, 리듬이 고려되지 않는 것이 대부분이었다. 또한 실제 입시를 위해 모의시험 형태로 문제를 만들어 주는 프로그램은 없었다. 이러한 점을 반영하여 실제 청음 시험에 나올 법한 음악적인 선율 패턴과, 문제들을 만들어 주는 알고리즘을 자체 제작하여 스마트폰을 통하여 언제 어디서든 편리하게 연습할 수 있도록 하였다.

1.1 특징

1. 단순한 음 나열이 아닌 리듬이 포함된 선율을 들려줌으로써 음악적인 청음교육을 할 수 있다.
2. 난이도를 지정할 수 있어 자신에게 적합한 연습을 할 수 있다.
3. 조성, 음역, 박자, 빠르기 등을 지정할 수 있어 보다 다양하게 선율을 만들 수 있다.
4. 상위권 대학들에서 요구하는 무조 선율을 만들어 준다.
5. 스마트폰 어플로 제공되므로 언제 어디서나 사용이 가능하며, 특히 청음 시험 직전에 귀를 익숙해지게 만드는데 용이하다.
6. 프로그램이 알아서 다양한 패턴의 선율을 랜덤하게 생성하기 때문에 비슷한 선율들로만 연습하는 것을 피할 수 있다.
7. 이미 존재하는 선율을 플레이하는 것이 아니라 매번 새로운 선율을 만들어 내기 때문에 연습하고 싶은 만큼 다양한 문제를 접할 수 있다.

1.2 다운로드 방법

이어로이드는 안드로이드 플레이스토어에 등록되어 있다. 아래 캡처화면 그림 중 왼쪽에 나타난 것과 같이 플레이스토어에서 '이어로이드'로 검색하면 관련 항목들 중에서 찾을 수 있다. '이어로이드-단선율' 버전과, '이어로이드-리듬'버전이 등록되어 있는 것을 볼 수 있다. '이어로이드-단선율'을 클릭하면 가운데 그림과 같이 다운로드를 할 수 있는 화면이 뜨며 다운로드가 완료되면 오른쪽 그림과 같이 핸드폰의 바탕화면에 이어로이드의 아이콘이 생성된다.



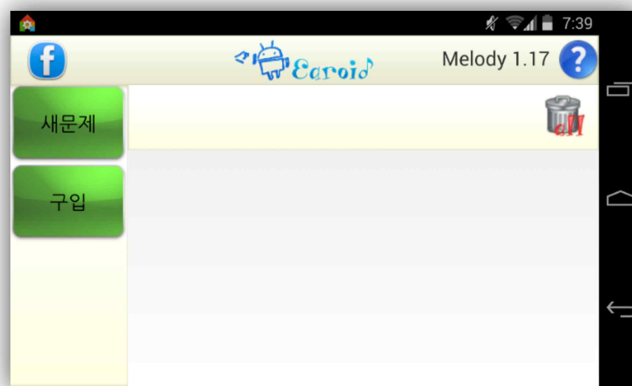
1.3 실행환경 및 권장사양

1. 이어로이드는 안드로이드 운영체제 버전 2.3(진저브레드, gingerbread) 이상에서 작동한다.
2. 프로그램 크기는 9.5MB이나 원활한 작동을 위해 20MB이상의 외장 메모리 공간 확보가 필요하다.

2. 사용법


2.1. 시작화면


이어로이드 어플을 실행하면 그림2와 같은 시작화면이 나온다. 좌측에 '새문제'를 통하여 새로운 청음 문제를 만들 수 있고, '구입'버튼을 통해 처음에 제공되는 '기본'단계 이외에 '중급', '고급', '전문가' 단계를 구입할 수 있다.



〈그림 29〉 시작화면

아이콘 설명

 이어로이드 공식 페이스북으로 링크한다.

 현재 시작화면에 대한 도움말을 볼 수 있다.

2.2. 새문제 화면

새문제 버튼을 클릭하면 새로운 팝업창이 뜨고, 음악적 요소들을 골라 문제를 만들 수 있다.

1. 단계 : 기본-1, 기본-2, 중급-1, 중급-2, 고급-1, 고급-2, 전문가-1, 전문가-2의 8단계 중에서 하나를 고를 수 있다. 기본-1, 기본-2의 난이도는 구매를 하지 않아도 처음부터 제공된다.
2. 박자 : 새로 만들 문제의 박자를 고를 수 있다. 2/4, 3/4, 4/4, 3/8, 6/8, 9/8 박자 중에서 선택한다.
3. 조표 : 조표를 고른다. Sharp 7개의 조부터 flat 7개의 조표까지 총 15개의 조표를 고를 수 있다.
4. 음자리표 : 높은음자리표와 낮은음자리표 중에서 하나를 고른다.
5. 조성 : 선택한 조표에서 가능한 조성, 즉 Major와 minor중에 한 조성을 고른다.
6. 빠르기 : allegro, allegretto, moderato, andantino, andante 중에서 하나를 고른다.
7. 무조 : 전문가 단계를 구입하면 무조를 선택할 수 있다.
8. 캐싱 : heap allocation에 실패하여 문제 생성시 오류가 있는 기기의 경우 캐싱 버튼을 해제하면 해결 할 수 있다. 캐싱 기능은 음원을 메모리에 올려놓아 더 빠르게 문제를 생성한다.
요소를 원하는 대로 고른 후 '만들기' 버튼을 누르면 〈그림3〉의 오른쪽 그림처럼 리스트에

새로운 문제가 추가되는 것을 볼 수 있다.



〈그림 30〉 문제 생성 화면

아이콘 설명



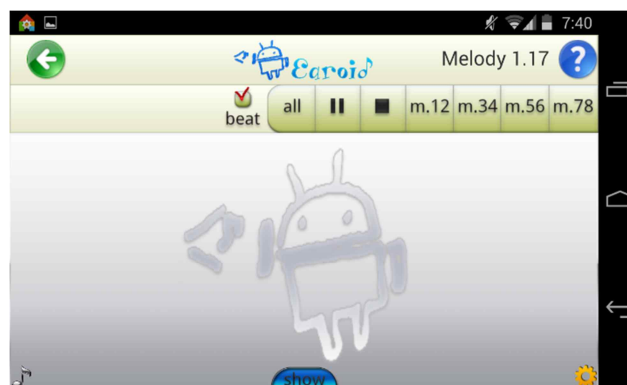
해당 줄의 문제를 삭제한다.



생성된 모든 문제를 삭제한다.

2.1. 문제듣기

만들어진 문제를 클릭하면 다음과 같은 문제듣기 창으로 넘어간다. 상단에는 문제를 재생할 수 있는 버튼들이 있고 중앙에는 악보가 나오게 될 화면이, 맨아래에는 첫음보기, 악보보기, 설정을 할 수 있는 아이콘이 놓여 있다.



〈그림 31〉 문제보기 화면아이콘 설명



전 화면으로 이동한다.



현재 문제듣기/보기 화면에 대한 도움말을 볼 수 있다.



선율을 들려주기 앞서 예비박을 들려줄지 결정한다.



모든 마디 (1-8 마디)를 재생한다.



현재 재생되고 있는 음악을 잠시 대기 시킨다.



현재 재생되고 있는 음악을 정지한다.

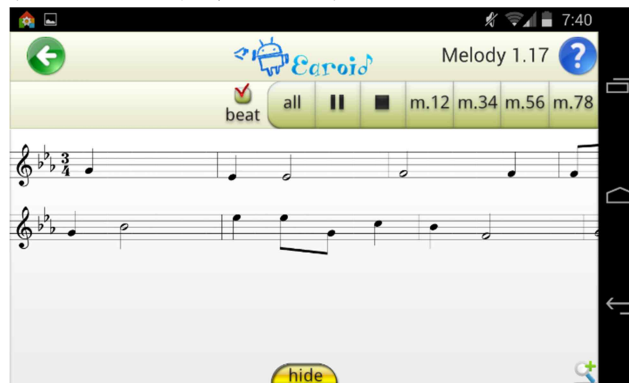
- m.12 1-2 마디를 재생한다.
- m.34 3-4 마디를 재생한다.
- m.56 5-6 마디를 재생한다.
- m.78 7-8 마디를 재생한다.

🎵 클릭하면 화면 중앙에 첫음이 표시 된다. 상대음감의 경우 유용하게 사용할 수 있다.

⚙️ 선율을 재생할 때의 구간별 반복 횟수, 구간 사이의 시간간격을 설정하는 팝업창이 뜬다.

2.1. 정답보기

화면하단의 show/hide 버튼을 클릭하여 화면 중앙의 악보를 통해 정답을 확인할 수 있다.



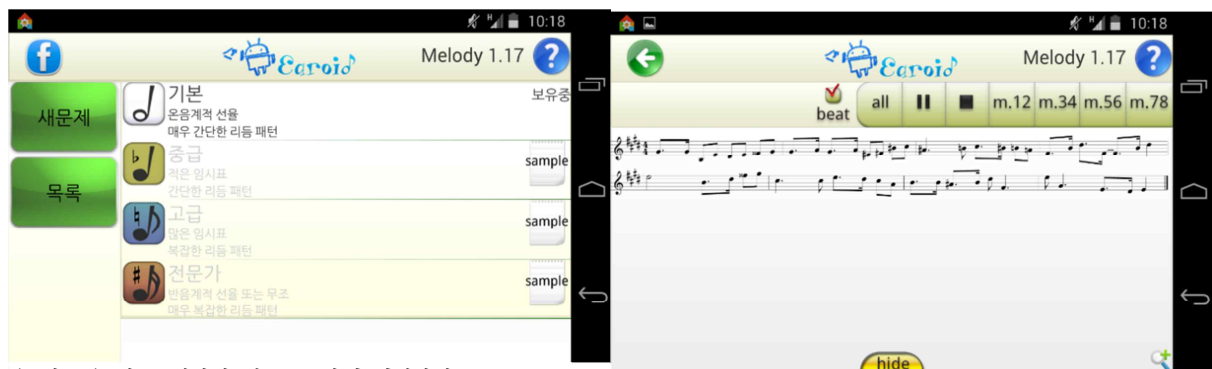
〈그림 32〉 문제보기 화면

아이콘 설명

🔍 악보를 확대 및 축소하여 보여준다.

3. 구매

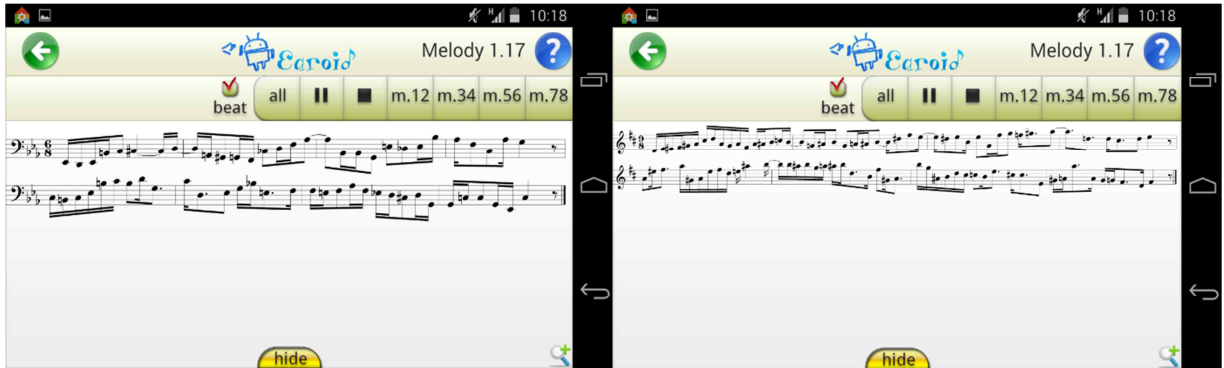
처음에 제공되는 기본-1, 2단계 외에 중급, 고급, 전문가 단계를 이용하기 위해서는 구매를 하여야 한다. 시작 화면에서 구매버튼을 클릭하면 〈그림5〉의 왼쪽과 같이 중급, 고급, 전문가 단계를 구입할 수 있는 화면이 뜬다. 원하는 단계를 클릭하면 결제 창이 뜨고 통신사 결제나, 신용카드를 통한 결제를 할 수 있다. 각 단계에 대한 설명은 다음과 같다.



〈그림 33〉 좌-구매화면, 우-중급단계 예제화면

🎵 중급 - 경과적·보조적·전타적 비화성음에 의한 임시표 등이 추가 된다.

벗점 등의 간단한 리듬 패턴이 추가 된다.



〈그림 34〉 좌-고급 단계 예제 화면, 우-전문가 단계 예제 화면



고급 - 부속화음에 의한 임시표 등이 추가된다.

16분음표 등에 의한 세분화된 리듬과 붙임줄에 의한 복잡한 리듬이 추가 된다.



전문가 - 부속화음과 비화성음에 의한 임시표 등의 빈도수가 더 많아지고,
역벗점과 같은 자주 쓰이지 않는 어려운 리듬들이 추가 된다.

또한, '무조' 옵션을 선택할 수 있게 되어 무조 선율을 생성하고 재생할 수 있게 된다.

4. 사용자 문의

제품을 사용하다가 버그나 문제점 또는 개선해야할 사항 등이 발견되면 아래 공식 페이스북 주소로 개발자와 연락할 수 있다.

<http://www.facebook.com/earoid>

참 고 문 헌

- [1] 백병동. 화성학 (증보판). 2008. 수문당
- [2] Diether de La Motte 저. 서정은 역. 화성학, 2005. 음악춘추사
- [3] Kent Kennan 저. 나인용 역. 대위법 (18세기 양식). 1991. 세광음악출판사
- [4] Diether de La Motte 저. 최우정 역. 대위법, 2013. 음악춘추사
- [5] Thomas Benjamin 저. 박재성 역. 대위법. 1993. 수문당
- [6] 김상형 저. 안드로이드 프로그래밍 정복 1. 2013. 한빛미디어
- [7] 김상형 저. 안드로이드 프로그래밍 정복 2. 2013. 한빛미디어
- [8] 고현철, 전호철 저. 안드로이드의 모든 것 NDK. 2012. 한빛미디어
- [9] Paul Deitel 저. Java How to Program. 2012. PearsonEducation
- [10] 이석원 저. 음악음향학. 2003. 심설당
- [11] A. 쇤베르크 저. 최동선 역. 작곡기법. 1991. 세광음악출판사