2020 年度　卒　業　論　文

リズムゲームにおけるノーツ数を考慮した譜面の自動生成に関する研究

指導教員：渡辺 大地 教授

メディア学部 ゲームサイエンスプロジェクト
学籍番号　M0117090
川口　雄大

2021 年 2 月

2020 年度　　卒　業　論　文　概　要

論文題目

リズムゲームにおけるノーツ数を考慮した譜面の自動生成に関する研究

メディア学部 **学籍番号：M0117090**	**氏名**	川口　雄大	**指導教員**	渡辺 大地 教授

キーワード	リズムゲーム、自動生成、難易度、MIDI

近年、アーケードゲームだけでなくスマートフォンのゲームアプリにもリズムゲームが増えている。また、年々配信や販売されている楽曲も増えている。しかし、その中でリズムゲームとして遊ぶことが出来る楽曲は一部の楽曲しかない。楽曲が仮にあったとしてもその曲に設定されている難易度が自分のプレイ技術とかけ離れており、楽しく遊べないということも多々ある。そのため、今のリズムゲームでは好きな楽曲の自分のプレイ技術に合った難易度の譜面というのは多くない。

多くの先行研究では機械学習を使用した譜面の自動生成を行っている。機械学習を利用する場合、学習データとして多くの譜面のデータを必要にすることや機械学習に使用しやすいようにデータを作成しなければならない。長く続いているリズムゲームではデータを集めやすいが新しくできたリズムゲームでは機械学習に十分な量の学習データを準備することは難しい。また、多くの先行研究では一曲ごとの難易度調整をする研究はあるがリズムゲーム全体での難易度調整は見かけない。

本研究では学習データを必要とする機械学習を使用せずに動画・音声データから譜面の自動生成を行い、同時に難易度に直結する要素のノーツ数を指定することができるリズムゲームの譜面の自動生成を目標とした。本研究の手法の流れとして動画データを MIDI 形式に変換、その MIDI データから雑音を削除する。残りのデータから同時に鳴る音が多い部分をノーツの配置地点候補とし、そのデータを使用し指定したノーツ数を抽出することで譜面を作成した。

フリーのゲームエンジンである Unity を用いて自作のリズムゲームの譜面作成ツールを作成し、既存のリズムゲーム譜面と実際に作成した譜面で評価実験を行った。実験では楽曲のリズムを損なっており、不快に感じる部分が多いという結果になった。

目 次

第 1 章

はじめに

1.1 研究背景と目的

近年、アーケードゲームだけでなくスマートフォンのゲームアプリにもリズムゲームが増えている。また、年々配信や販売されている楽曲も増えている。しかし、その中でリズムゲームとして遊ぶことが出来る楽曲は一部の楽曲しかない。そのため、配信されたばかりの楽曲やリミックス曲や改変を加えているカバー曲では遊びたくてもリズムゲームに存在していないということがある。楽曲が仮にあったとしてもその曲に設定されている難易度が自分のプレイ技術とかけ離れており、楽しく遊べないということも多々ある。そのため、今のリズムゲームでは好きな楽曲の自分のプレイ技術に合った難易度の譜面というのは多くない。

リズムゲームの譜面の自動生成の研究はいくつかある。多くの先行研究では機械学習を使用した譜面の自動生成を行っている。Donahue ら [1] の研究では画像処理技術に使用している CNN(Convolution Neutral Network) を用いてアーケードゲームの「DDR(Dance Dance Revolution)」の譜面の自動生成を行っている。この研究では動画、音声データからリズムの出現地点を抽出しプレイヤーが独自に制作した譜面のサンプルデータから機械学習によりノーツの配置を行なっている。機械学習を利用する場合、学習データとして多くの譜面のデータを必要に

することや機械学習に使用しやすいようにデータを作成しなければならない。長く続いているリズムゲームではデータを集めやすいが新しくできたリズムゲームでは機械学習に十分な量の学習データを準備することは難しい。また、楽曲がジャンルで分かれているように学習するデータもジャンルごとにどの学習データを用意するかという問題がある。ほかにも機械学習は使用する機器も十分なスペックのものを準備する必要がある。

辻野ら [2] の研究ではすでに作成した譜面データを学習データとして機械学習を使用し、譜面の自動生成を行なっている。この研究では学習データから学習のノイズになる譜面データを事前に除外している。他にも福永ら [3] の研究や辻野ら [4] の研究、Liang ら [5] の研究でも機械学習を用いてリズムゲーム譜面の自動生成を行なっており、どの研究でも事前に作成した譜面データを学習データとして必要としている。楽譜のデータや MIDI データ、実際のゲームの譜面データと言った特定の形のデータを必要とする研究では自分で使用するデータを準備する、または特定の形のデータがある楽曲しか使用することができないという問題がある。また、「歌ってみた」「演奏してみた」等の動画データでは楽譜のデータや歌詞は基本的に同じになるため、特定の形のデータがある楽曲であってもずれや元となる楽曲と全く同じ譜面になることが問題としてあがる。

リズムゲームにおいて難易度を考慮した研究では都丸 [6] の研究や紺野ら [7] の研究がある。都丸の研究ではリズムゲームの譜面の自動生成において MIDI データを使用したリズム取得の違いにより難易度表現を行っている。リズムゲームには楽曲ごとに複数の難易度の譜面が存在している。難易度はレベルという表記で分けられており、楽曲ごとの譜面はレベルという表記に加え、easy,normal,hard,expert の四段階の表記で表されていることが多い。そのため、ある曲では一番難しい難易度の譜面がほかの曲の二番目に難しい難易度の譜面と同じレベルであるということがある。また、一つの楽曲であっても一番難しい難易度の譜面と二番目の難易度の譜面では難易度に差があり、一番難しい譜面ではプレイが出来ないほど難しいが二番目の譜面では簡単で楽しめないといったことが起きる。特に自分の好きな曲の場合同じ曲を何度も行うため、難易度が自

分に合わないという状況は好ましくない。多くの先行研究では楽曲ごとの複数の難易度の譜面を作成する研究はあるがリズムゲームの譜面の難易度調整は見かけない。都丸の研究では楽曲ごとの複数の難易度の譜面作成を行っているが作成された譜面の難易度調整については研究をしていない。

本研究では学習データを必要とする機械学習を使用せずに動画・音声データから譜面の自動生成を行い、同時に難易度に直結する要素のノーツ数を指定することができるリズムゲームの譜面の自動生成を目標とした。本研究の手法の流れとして動画データを MIDI 形式に変換、その MIDI データから雑音を削除する。残りのデータから同時に鳴る音が多い部分をノーツの配置地点候補とし、そのデータを使用し指定したノーツ数を抽出することで譜面を作成した。

ほかにリズムゲームの面白さについての研究で坂本ら [8] の研究がある. この研究ではフロー理論を用いてリズムゲームのレーン数等が面白さに与える影響を分析している。本研究で作成する譜面は 1 レーンのみで作成するため、面白さへの影響について分析は出来ない。

音声データを MIDI データに変換する土村ら [9] の研究がある。本研究では土村らの研究とは違い演奏するためのデータではなく、ノーツの位置をとるためのデータであるので目的および必要データを得るためには不十分と判断した。まーてぃラボ [10] の動画には動画データを歌詞や複数の楽器の音を含めたまま MIDI 形式に変換すると存在していない歌詞が聞こえるというものがある。本研究ではこの動画を参考に動画データを歌詞や複数の楽器の音を含めたまま MIDI 形式に変換して使用した。

MIDI データに関する研究として香川ら [11] の研究や高田ら [12] の研究がある。香川らの研究では MIDI データから音楽の重要な部分を抽出し、音楽ゲームの譜面の自動生成を行っている。高田らの研究では MIDI データからサビの検出等、MIDI データの楽曲の解析を行っている。それぞれ使用している MIDI データは本研究で使用した動画・音声データを変換した MIDI データとは大きく異なるため、本研究での MIDI データの解析は難しいと判断した。

フリーのゲームエンジンである Unity[13] を用いて自作のリズムゲームの譜面作成ツールを作成し、既存のリズムゲーム譜面と実際に作成した譜面で評価実験を行った。実験では楽曲のリズムを損なっており、不快に感じる部分が多いという結果になった。

1.2 論文構成

本論文の構成を説明する。第 2 章では本研究におけるリズムゲームの譜面の自動生成の詳しい手法の説明について述べる。第 3 章では本研究の手法が有効であるか、生成した譜面を使用して行った実験とその考察について述べる。第 4 章では終わりとして、本研究を通しての総括を述べる。

第 2 章

提案手法

第 2 章では本研究の提案手法について述べる。本研究では学習データを必要とする機械学習を使用せずに動画・音声データから譜面の自動生成を行い、同時に難易度に直結する要素のノーツ数を指定することができるリズムゲームの譜面の自動生成を目標とした。本研究のために音楽ゲーム用譜面生成ツールを独自に作成し、これを用いてノーツ数を指定した譜面の生成を行った。

本研究で想定するリズムゲーム [14] はアーケードゲームの「チュウニズム」[15] とした。アプリゲームの場合は似たシステムである「バンドリ」[16] を想定する。入力データは動画データも含めた音声データのみを使用した。ノーツの流れるレーンは 1 レーンとし、ノーツの種類はタップのみで同時押しや長押しを含めないものとした。本研究では二つの理由から動画、音声データを入力データとした。第一に音声データが音楽ゲームで使用するため譜面の作成をする、しない関係なく必要なデータとなる。第二に動画、音声データはインターネット上から簡単に用意することが出来る。入力データとする動画データも含めた音声データは Youtube 等の動画データを想定する。これには音楽以外の動画も対象とする。また、音楽動画は楽器の音だけではなく歌詞がついている楽曲を想定し、「歌ってみた」「演奏してみた」「替え歌」等の楽曲もそのまま音声データとして使用することを想定する。また、動画データの映像部分は本研究では使用しない。映像部分ではアニメ・ミュージック・ビデオ（Anime Music Video, AMV）と言った楽曲とは関係な

い映像を使用した動画もあるため、 入力データとして使用するデータに適さない。

本研究の手法の流れとして動画データを MIDI 形式に変換、その MIDI データから雑音を削除する。残りのデータから同時に鳴る音が多い部分をノーツの配置地点候補とし、そのデータを使用し指定したノーツ数を抽出することで譜面を作成した。

2.1 MIDI データの加工について

初めに音声データを MIDI 形式に変換、その後 MIDI データを使用しやすい形に変換し、MIDI データから雑音を削除した。

音声データからの MIDI データへの変換にはファイルの形式を変換する bearaudio[17] 等のサイトや Audacity[18] 等のアプリケーションがあるため、これを利用した。図 2.1 は音声データを変換した MIDI データを可視化した図である。赤色の部分が音のデータであり、縦軸が音階、横軸が音の鳴っている時間をそれぞれ表している。変換したデータは同時に五個以上の音が鳴ることや一瞬しか鳴らない音が常に鳴り続ける状態になる。また、音声データの時には存在していない雑音が変換した MIDI データには存在している。この時点での MIDI データはあくまで曲と認識できるレベルのもので実際の曲と遜色がある。また MIDI データは元の音声データの形式により一定音階以上、一定音階以下のデータは存在しないものになる。入力データの音声を波形のまま使用するという方法も存在するが波形の解析では機械学習を用いる研究でやっており、機械学習を使用しない方法では機械学習を使用する方法と同等の解析は難しいため、本研究では波形の解析はしないものとした。

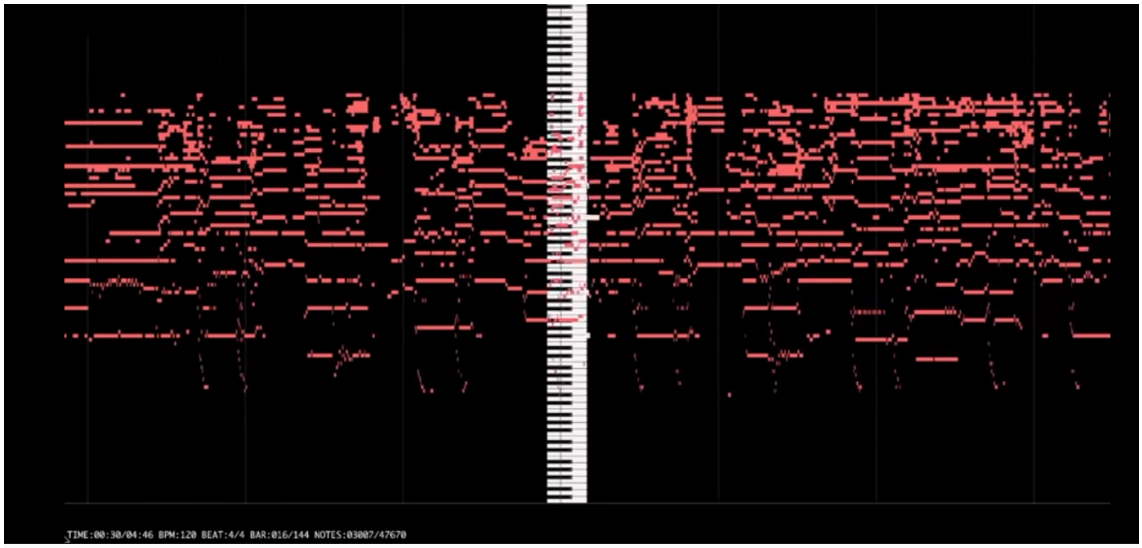

図 2.1　変換した MIDI データを可視化した画像

https://www.youtube.com/watch?v=UooSKhLPJ48

次に MIDI データから雑音を削除するために使用しやすい形に変換を行った。そのために必要なデータはどの時間から鳴り始めたか、どれだけ鳴り続けたかの二つである。

MIDI データ [19] とは音声データではなく演奏情報である。演奏情報にはイベント、トラックデータ、経過時間のデータを一つとし、このデータを複数持っている。イベントはトラックデータが何のデータかを格納している。具体的には音を鳴らす、音を止めると言った基本的なデータから楽曲のタイトル、BPM と言ったデータがある。トラックデータにはイベントによって格納されているデータは異なり、データの大きさも違う。例として音を鳴らすというイベントの場合は音階と音量の二つのデータを格納している。経過時間は楽曲の初めからの時間ではなく、そのデータの次のデータまでの経過時間が格納している。経過時間は MIDI データの分解能によって異なり、全音符の分解能が 480 だった場合経過時間 120「四分音符分の時間 (の後)」を意味する。MIDI データでは格納している順に時間が経過していくため、一部のデータを見ただけでは初めからどのくらい時間が経ったのか、音を鳴らす場合いつ止まるのか、反対に音を止める場合いつ鳴り始めたのかという情報がわからないため、そのまま加工、使用することは難しい。加工を行

う時に参照する必要があるイベントは音を鳴らす、音を止めるの二つのみである。

加工時に保持する必要があるデータは三つある。第一にすべての音に対してそれぞれ今音が鳴っているかという情報である。音階は MIDI データでは 0 から 127 に割り振られているため、最大 128 のデータを格納する。第二にすべての音に対してそれぞれ音が鳴っている場合今どれだけの時間鳴っていたかという情報である。第三に初めからどれだけの時間が経過しているかの情報である。

加工手順としては MIDI データを初めのデータから順に確認する。この時にそのデータのイベントを確認する。音が鳴るというイベントの時にどの音かを確認し、どの音が鳴ったかを記録する。また、初めからどれだけの時間が経過しているかを保持する。加工後に必要なデータは鳴り始めの時間と鳴り続けた時間だけだが確認等のために鳴り始めの時間と共にどの音が鳴っているか、どの位の音量かの情報も保持する。音を止めるというイベントの時に音が止まったという情報を記録し、この音がどれだけの時間鳴っていたかの情報を初期化、0 にする。また、初期化前に必要なデータであるどれだけの時間鳴っていたかを保存する。この時、鳴り始めの時間と鳴っていた時間の単位は秒でも MIDI データ基準の時間（デルタタイム）でもどちらでもよい。今回は変換する必要がないため MIDI データ基準の時間で保存する。最後にそのデータの経過時間を初めからどれだけの時間が経過したかの情報とすべての音に対してそれぞれ音が鳴っている場合どれだけの時間鳴っているかの情報に加算を行う。加算はすべてのイベントで行う。確認する必要があるトラックデータは音を鳴らす、音を止めるの二つのイベントのみであるが経過時間はあくまで次のデータまでの経過時間であるため、イベント関係なく加算を行わなければならない。また、加算をするときはイベントを確認した後にしなければならない。経過時間は次のデータまでの時間なので今確認しているデータは加算前のイベントであるため、加算は確認後に行う。ここまでを繰り返し行い、MIDI データを最後まで確認をする。図 2.2 は MIDI データからノーツのデータを作成した図である。図の同じ色の部分が変換されたデータである。作成後に加工時に保

持しているすべての音が今鳴っているかという情報を確認する。MIDI データでは音が鳴るというイベントと音が止まるというイベントは同数であるため、最後まで確認した後に一つでも音が鳴っているという状態の場合は MIDI データが破損している、正常ではないということとなる。

MIDIデータ

番号	イベント	トラックデータ	経過時間
1	楽曲名	・・・	0
2	BPM	120	0
3	音を鳴らす	音階：ド 音量：50	3
4	音を止める	音階：ド 音量：0	2
5	音を鳴らす	音階：ミ 音量：60	0
6	音を鳴らす	音階：ソ 音量：80	1
7	音を止める	音階：ミ 音量：0	3
8	音を鳴らす	音階：レ 音量：30	0
9	音を止める	音階：ソ 音量：0	2
…			

ノーツのデータ

時間	音階	音量	鳴っている時間
0	ド	50	3
5	ミ	60	1
5	ソ	80	4
9	レ	30	5
11	ソ	50	1
15	シ	40	2
17	ド	70	4
20	ド	80	1
21	レ	60	1
…			

図 2.2　MIDI データからノーツのデータを作成した図

その後この新しく作成したデータから雑音になっている音を抽出し、MIDI データから削除した。ここでは動画データを MIDI データに変換したときにできる一定時間以下しか鳴っていない部分を削除した。本研究では MIDI データ基準の時間で 1 以下を削除する。削除する一定時間以下は MIDI データ基準の時間で 1 以下としているがあくまで雑音になる音を削除したいためであり、1 以下である必要はないが本研究では最低限の 1 以下とする。また、聞こえない一定音量以下を削除するという方法や一定時間以下と一定音量以下を削除するという方法もある。しかし、本研究では最低限の雑音に聞こえる音を削除することを考え、特に雑音として聞こえる一定時間以下のみを削除するという方法とした。MIDI データから削除する時に削除する音の音が鳴るというイベントを含めたデータと音を止めるというイベントを含めたデータをそれぞれ削除する。また、それぞれを削除する時にその削除するデータの一つ前のデータに削除するデータの経過時

間を加算する。経過時間は次のデータまでの時間であるため、削除する時にひとつ前のデータの経過時間に加算を行わないで削除するとその分だけ全体の時間にずれが生じる。図 2.3 は MIDI データから削除を行ったノーツのデータから MIDI データを加工した図である。図の赤色の部分はそれぞれ削除を行ったデータである。図の灰色の部分は削除を行った結果、変更を行ったデータである。

ノーツのデータ

時間	音階	音量	鳴っている時間
0	ド	50	3
5	ミ	60	1
5	ソ	80	4
9	レ	30	5
11	ソ	50	1
15	シ	40	2
17	ド	70	4
20	ド	80	1
21	レ	60	1
…			

MIDIデータ

番号	イベント	トラックデータ	経過時間
1	楽曲名	・・・	0
2	BPM	120	0
3	音を鳴らす	音階：ド 音量：50	3
4	音を止める	音階：ド 音量：0	2
5	音を鳴らす	音階：ミ 音量：60	0
5	音を鳴らす	音階：ソ 音量：80	4
7	音を止める	音階：ミ 音量：0	3
6	音を鳴らす	音階：レ 音量：30	0
7	音を止める	音階：ソ 音量：0	2
…			

図 2.3　削除を行ったノーツのデータから MIDI データを加工した図

2.2　ノーツ配置候補の作成について

次に MIDI データから同時に鳴る音を抽出し、ノーツ配置の候補を作成した。

ノーツの配置候補は楽曲の音の鳴る瞬間をリズムの出現位置と考え、候補地点とした。MIDI データでは音の鳴る瞬間というのはイベントの一つとして存在するが元とする動画データでは音の波形データであり、連続したデータであるため音の鳴る瞬間を取ることは難しい。加工した MIDI データでは音声データをそのまま変換したデータである。そのため、楽譜として使用するデータと違い同時にいくつもの音が鳴ることがあるため、音が鳴るというイベントをそのままリ

ズムの出現地点として使用することが出来ない。本研究では MIDI データで同じ時間に鳴らし始める音が多い時間を楽曲での音の鳴る瞬間とした。加工した MIDI データから音を鳴らすというイベントのみを抽出し、どの時間にいくつの音がなったかを取得した。ほかにも音の鳴り始めではなく、いくつの音が同時に鳴っているかを取るという方法もある。しかし、音が同時に多く鳴っている部分をリズムの出現地点として使用する場合、実際には聞こえないような音量の音が含まれてしまうことや音の鳴っている間というデータであるため、長押し以外のノーツ配置候補として適していない。そのため、本研究では音が鳴っている、聞こえている部分よりも音の鳴る瞬間をリズムの出現地点とした。

また、同時に鳴る音の抽出する時間はそのままノーツとノーツの間隔になる。例として 1 秒ごとに同時に鳴る音を抽出する場合、作成される譜面のノーツとノーツの間隔は 1 秒ごととなる。MIDI データの経過時間で 1 単位ずつで抽出するとリズムゲームをすることができないほど短い間隔になる。そのため、MIDI データの経過時間を 16 分音符単位でデータを取ることでよりリズムゲームとして使用しやすいデータとなる。この単位を 8 分音符にする等、値を大きくすることでノーツごとの距離が離れるため、比較的初心者向けの譜面も作成することができると考えた。ただし、単位を大きくするほどノーツの配置候補が減るため、同時押しを考慮しないノーツ数の指定をすると 16 分音符ごとに常にノーツが配置している譜面になる可能性がある。本研究ではあくまである程度リズムゲームに慣れた上級者を想定し、単位は可変で自由に設定できるものとした。

ノーツの配置候補のデータはどの時間、いくつの音が鳴ったかの二つの情報を抽出した。それぞれの時間ごとに MIDI データからその時間の間の音を鳴らすというイベントを検索し、その数を保存した。その時間の間に音を鳴らすというイベントがない場合では 0 という数を保存した。ノーツの配置候補のデータのいくつの音が鳴ったかという情報は数が大きいほどリズムの出現地点として有力として優先的にノーツの配置とした。

2.3　ノーツ配置候補から指定数のノーツの抽出について

最後に作成したノーツの配置候補のデータから指定したノーツ数に調整し、抽出した。

抽出するノーツ数は指定数と同数ではなく、ある程度上下するものとした。理由として第一に一曲のノーツ数は少なくとも 100 以上になるため、一つや二つノーツ数が増えたとしても難易度に大きな影響がない。第二に指定数に 1 足りない、多い時に無理に増減した結果リズムにあまり合わない位置に追加することになる、またはリズムに合っているのに減らすことになる場合がある。そのため、本研究では指定数からある程度上下したノーツ数を抽出した。

ノーツの抽出をいくつかの手順に分けた。初めに指定数以下のある程度の数のノーツ位置を抽出した。次に残りの数のノーツ位置を抽出をした。

指定数以下のある程度の数のノーツ位置の抽出ではノーツの配置候補から同時に鳴る音が多い順にノーツの配置位置とした。例として同時に鳴る音の最大値が 10 の場合、同時に鳴る音が 10 の配置候補の数が指定数以下なら 10 の配置候補をすべてノーツの出現地点とする。その後、同時に鳴る音が 9 の配置候補の数が指定数から 10 の配置候補の数を引いた数以下なら 9 の配置候補を全てノーツの出現地点とする。これを繰り返し、指定数超えない同時に鳴る音の数までをノーツの出現地点として確定する。同時に鳴る音の数が 4 の配置候補までが指定数以下で 3 の配置候補の数を追加すると指定数を超える場合、同時に鳴る音の数が 4 以上の配置候補はノーツの出現地点として確定し 3 以下の配置候補は含まないとなる。この段階でデータによる配置候補、音が鳴っている位置が指定数以下の場合は考えないものとした。基本的にリズムゲームではロングノーツや同時押し等を含めて 3000 から 4000 ノーツが最大値である。データの段階で楽曲にもよるが 1 分半の長さの楽曲を MIDI データの 1 単位ごとで考えるとノーツの配置候補は 8000 から 9000 は存在した。そのため、ロングノーツや同時押しを含めない 3 分ほどのリズムゲームの楽曲であればノーツの配置候補の数が指定数以下にならない、または指定数が極端に大きい場合はリ

ズムゲームとして遊ぶことができないほどノーツの間隔が短いこととなる。この段階で指定数のノーツ数になっている場合は譜面の作成を終了した。

最後に指定数までの残りのノーツ数まで調整をした。上記の段階では指定数以下の配置が確定したノーツのデータがある。このデータを使用しノーツの配置候補から残りのノーツを抽出した。

残りのノーツは譜面全体のノーツ量のばらつきを減らすため、すでに確定したノーツのデータからノーツ量が少ない場所を抽出しノーツを配置した。手順としては初めにすでに確定したノーツのデータから時間ごとのノーツ量のデータを作成する。作成するデータは時間と数値の二つの情報を保存する。時間ごとにノーツが近くにあるほど値が大きくする。確定したノーツごとにそのノーツの時間の上下の時間に数値を加算する。本研究では確定したノーツの時間には 10 を加算し、時間が離れるごとに 9、8、7・・・と 1 減らした数値を加算した。

次に作成したノーツ量のデータから値の低い時間、場所を抽出する。数値が低い順に優先的にノーツの配置位置とした。その後、抽出したデータを使用して残りのノーツの抽出をするために三つの動作から一つの動作を行う。第一に最も値が低い時間、場所の数が残りのノーツ数と同じであった場合、その最も低い値の時間、場所をノーツの配置位置と確定し、譜面の作成を終了する。第二に最も値が低い時間、場所の数が残りのノーツ数より少ない場合、その最も低い値の時間、場所をノーツの配置位置として確定する。その後、確定したノーツのデータを含めたノーツ量のデータをもう一度作成する。そのデータを使用してもう一度、三つの動作から一つの動作を行う。第三に最も値が低い時間、場所の数が残りのノーツ数より多い場合、ノーツ量のデータを圧縮する。ノーツとノーツの間が極端に離れている場合、ノーツ量のデータは連続して最も低い値になる。この連続して最も低い値である部分に優先的にノーツの配置を行う。ノーツ量のデータの圧縮は最初の時間、場所からそのデータと次のデータのノーツ量の値を加算し、そのデータの時間と保存する。その次のデータは時間のデータとともに削除する。その後に作成するデータは圧縮前のデータの半分のデータ量になる。そして、そのデータをノーツ量のデータとして使用

し、もう一度三つの動作から一つの動作を行う。データの圧縮を行う場合、半分の時間のデータは削除されるため、ノーツの配置候補のノーツ配置間隔が初めの状態の半分になる。ここまでの動作をノーツ数が指定したノーツ数になるまで繰り返し行う。図 2.4 は残りのノーツ数を抽出する手順の流れの図である。矢印の順に残りのノーツの抽出を行う。

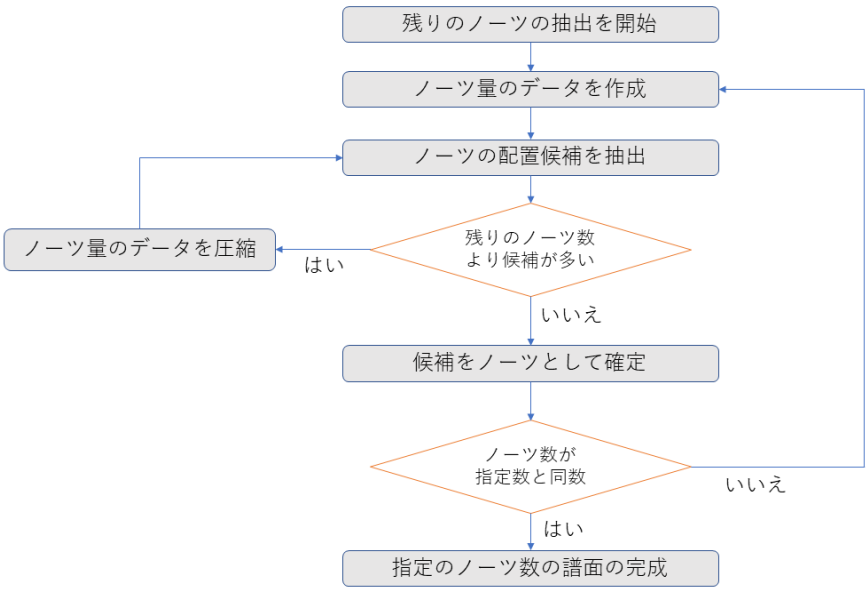

図 2.4　残りのノーツ数を抽出する手順の流れの図

残りのノーツを抽出する手順の中でノーツ量のデータの中で最も低い値を使用したがこれは常に 0 ではない。残りのノーツ数より少ない場合、繰り返し行うと少しずつノーツ量の少ない場所を埋めていくためある程度離れてはいるが 0 になるほど離れてはいないということが起こる。この場合は最も低い値、1 を参照することになる。また、残りのノーツ数より多い場合でも圧縮した結果、ノーツ量が 0 の場所がなくなることがある。

この残りのノーツ数の抽出は抽出が終わるまで何度も同じ手順を繰り返す。この繰り返しはどの楽曲、楽曲の長さであっても 500 回以下で終わる。繰り返しが長く続く場合は楽曲のデータによる問題ではなく、手順通りに動いていない可能性が高い。

以上の手法で指定したノーツ数のリズムゲームの譜面のデータを作成した。

2.4 実装について

実装にはフリーのゲームエンジンである Unity を使用して作成した。リズムゲーム部分はいくつかのリズムゲームの作成方法が掲載されているサイトを参考に作成した。レーンは一レーンのみで上から下にノーツが流れる。操作は Space キーで行い、タップ時にノーツとの距離から評価をした。譜面の作成部分では本手法通りに作成をした。また、Unity 上では音声データの MIDI 形式への変換を作成出来なかったため、アプリケーションで変換した MIDI データを事前に用意した。変換した MIDI データの BPM、分解能を使用して作成を行っていたが計測・計算した結果、どの楽曲でも同じ BPM になっており 16 分音符の長さの計算が出来なかった。そのため、MIDI データの経過時間単位 10 ごとにノーツの配置候補を抽出した。譜面データは楽曲名、BPM、楽曲の長さ、ノーツのデータを保存した。ノーツのデータには出現時間、流れるレーン、アクションを保存した。保存には Json 形式 [20] を使用し、手書きで作成することも可能とした。

第 3 章

評価と分析

第 3 章では実際に本研究の手法を用いて自動生成をした譜面を実際に遊んで評価してもらい、譜面生成について既存リズムゲームと同程度の譜面であるかを検証した。

3.1　実験方法

本手法がリズムゲームの譜面の自動生成において既存リズムゲームの譜面と同ノーツ数の譜面が出来ているか、作成した譜面がリズムゲームとしてどう感じるかを評価実験した。実験方法としてアーケードゲームの「チュウニズム」の譜面の動画を視聴し、本手法で作成した譜面を実際に遊んでもらい評価をした。実験では 1 人につき、別の楽曲を二つそれぞれで実験を行う。

本実験で使用した楽曲は三曲用意した。曲名は「夏祭り」、「六兆年と一夜物語」、「ナイトオブナイツ」である。「夏祭り」は歌手が歌っている楽曲として選んだ。「六兆年と一夜物語」はボーカロイドが歌っている楽曲として選んだ。「ナイトオブナイツ」は歌詞がない楽曲として選んだ。全ての楽曲はアーケードゲームの「チュウニズム」に存在し、動画サイト等で譜面確認用動画がある楽曲である。また、全ての楽曲は近年の楽曲ではなく、ある程度知られている楽曲であることと「チュウニズム」だけでなく多くのアーケードゲーム、アプリゲームで使用されている楽曲であることから選曲した。

実験用として用意した楽曲はそれぞれ 1 分半ほどの楽曲の長さである。指定するノーツ数はそれぞれ「チュウニズム」の同じ楽曲の譜面からノーツの配置位置のみを抽出し、ノーツ数をカウントした値とした。カウントを行った「チュウニズム」の譜面の難易度は上級者を想定しているため、二番目に難しい hard の譜面を使用した。「チュウニズム」でのリズムゲーム全体の難易度では「夏祭り」、「六兆年と一夜物語」、「ナイトオブナイツ」の順に難易度が難しくなっている。「夏祭り」では 175 ノーツ、「六兆年と一夜物語」では 250 ノーツ、「ナイトオブナイツ」では 300 ノーツをノーツ数として指定した。作成された譜面のノーツ数はそれぞれ「夏祭り」では 174 ノーツ、「六兆年と一夜物語」では 245 ノーツ、「ナイトオブナイツ」では 295 ノーツとなった。表 3.1 は楽曲ごとの譜面のデータを表にしたものである。

表 3.1　楽曲一覧

楽曲名	楽曲の長さ	指定するノーツ数	作成されたノーツ数
夏祭り	1:16	175 ノーツ	174 ノーツ
六兆年と一夜物語	1:16	250 ノーツ	245 ノーツ
ナイトオブナイツ	1:20	300 ノーツ	295 ノーツ

実験後はアンケートを行う。アンケート内容は被験者についてと実験の楽曲ごとについてとした。被験者については「リズムゲームをしたことがあるか」、「ある場合、アプリ、アーケードゲームのどちらか」、「いくつのリズムゲームを遊んだことがあるか」、「そのリズムゲーム名」の四つについて質問を行った。質問の理由として被験者のリズムゲームの経験についてを知るためである。アプリゲーム、アーケードゲームについてはそれぞれ想定しているリズムゲームがアーケードゲームであるため質問を行った。実験の楽曲については「曲のリズムに合っているか」、「ノーツの流れてくる頻度についてどう感じるか」、「プレイ中に不快に感じたか、感じる部分があったか」、「従来のリズムゲームの譜面と比較して、異なると感じる点があれば答えてください」の四つについて質問を行った。

図 2.1 は実験用に作成したリズムゲーム画面である。

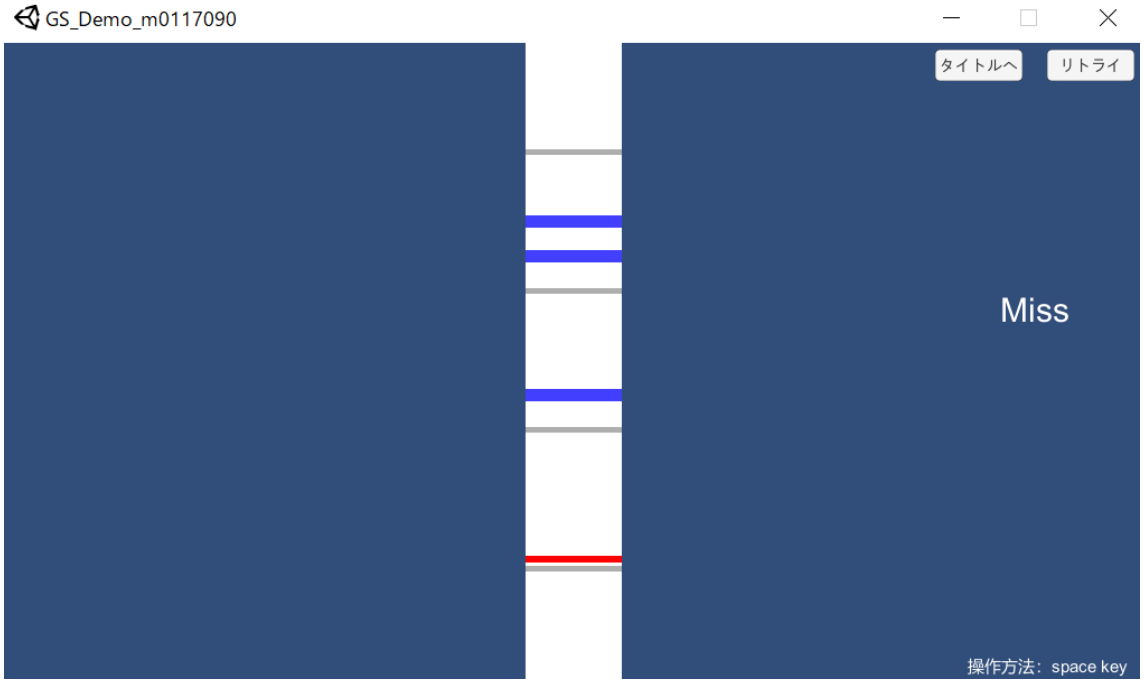

図 3.1　実験用に作成したリズムゲーム画面

3.2　実験結果

実験は 20 代 4 名に行った。被験者はそれぞれアプリゲーム、アーケードゲーム両方のリズムゲームを遊んだことがあった。

3.2.1　曲のリズムに合っているか

この項目ではリズムゲームとして特に重要なリズムに合っているかについて質問を行った。質問は 4 段階で 1 がリズムが合っていない、4 がリズムに合っているとした。図 3.2 は結果をグラフで表したものである。結果としてほとんどが被験者がどの曲であっても 1 を選択していた。曲ごとに見ると指定したノーツ数が多かった「六兆年と一夜物語」と「ナイトオブナイツ」は 1 のみであった。

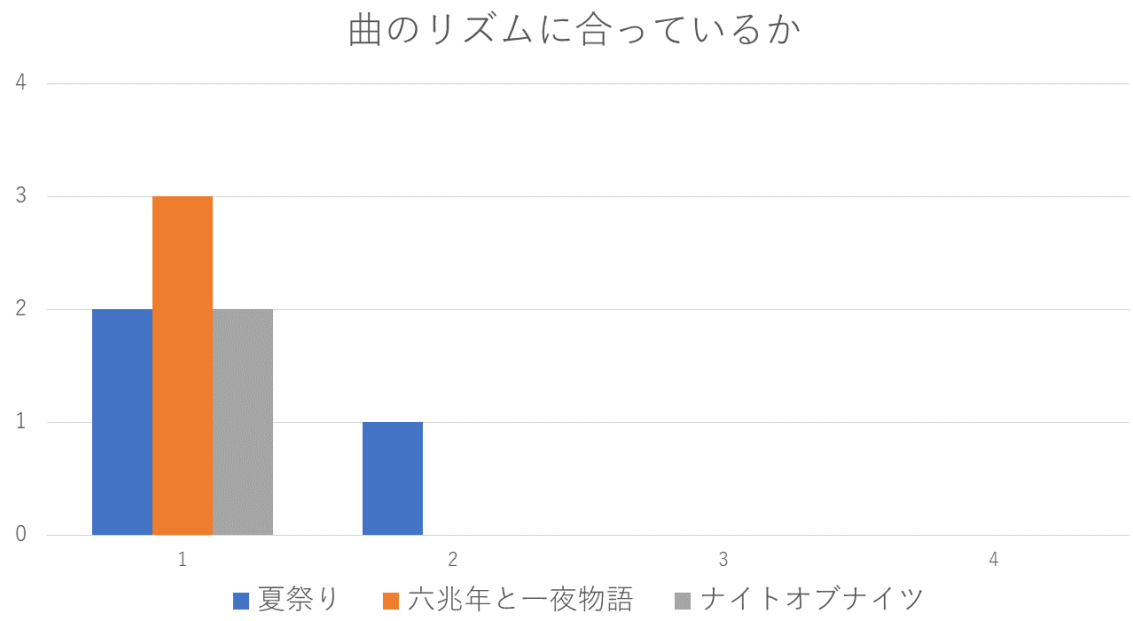

図 3.2　「曲のリズムに合っているか」の結果

3.2.2　ノーツの流れてくる頻度についてどう感じるか

この項目では譜面のノーツとノーツの間隔、流れてくる頻度がどうかについて質問を行った。ノーツとノーツの間が極端に開いている場所がある等のノーツ量が少ない部分がないかを質問した。質問では 4 段階で 1 が少ない、4 が多いとした。図 3.3 は結果をグラフで表したものである。結果として平均は 2 と少しノーツの流れてくる頻度が少ないとなった。曲ごとに見ると「夏祭り」は 1 が多く、「六兆年と一夜物語」は 2 が多く、「ナイトオブナイツ」は 3 が多いという結果となった。

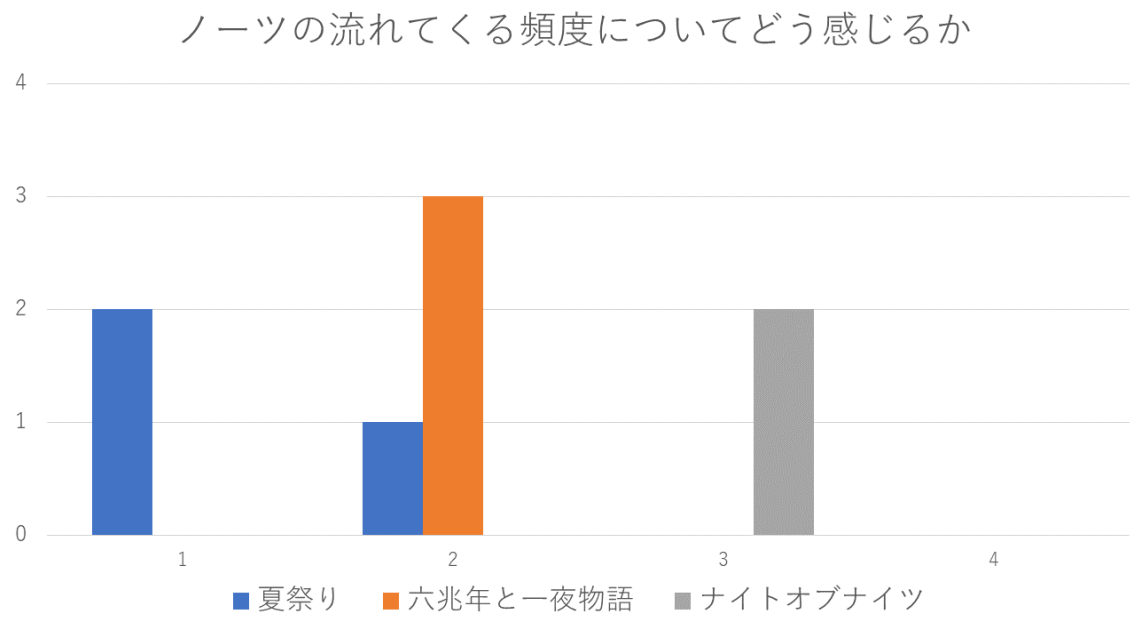

図 3.3　「曲のリズムに合っているか」の結果

3.2.3　プレイ中に不快に感じたか、感じる部分があったか

この項目ではリズムが取りずらいなどプレイ中に不快に感じる部分があったかを質問した。質問は 4 段階で 1 が不快に感じる部分がない、4 が不快に感じる部分があったとした。図 3.4 は結果をグラフで表したものである。結果として全ての楽曲で 3、4 といった不快に感じる部分があったとなった。

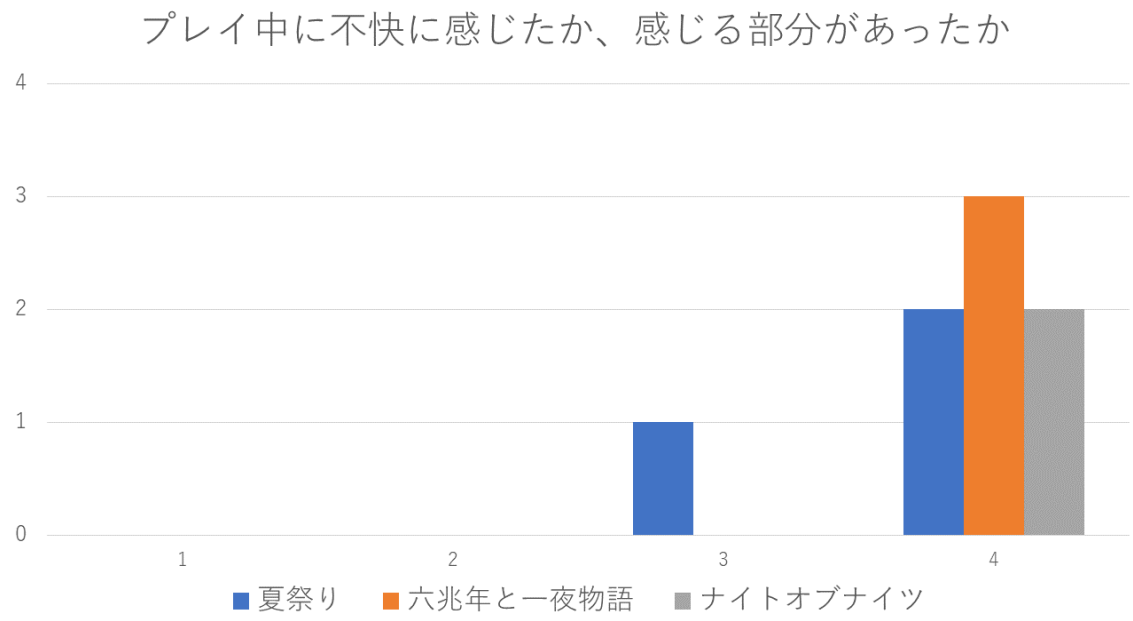

図 3.4　「曲のリズムに合っているか」の結果

3.2.4　従来のリズムゲームの譜面と比較して、異なると感じる点があれば答えてください

この項目では目標とした従来のリズムゲームの譜面と比較して作成した譜面の異なる部分を質問した。質問は自由記述で行った。結果として「リズムに合っていない」、「連続で押す部分がリズムとは関係ないように感じる」等の作成した譜面が従来のリズムゲームの譜面とは異なり、特にリズムが取れていないとなった。ノーツの連続する部分などただたたき続けるだけといった従来のリズムゲームではあまりないノーツの配置についての記述も多かった。

3.3　考察

ここからは実験の結果から本研究について考察を行う。実験結果から本手法で作成された譜面のノーツ数は概ね指定数通りに作成が出来た。しかし、アンケートの結果から本研究のリズムゲームの譜面の作成手法は不適切であったとなった。特にリズムゲームとして重要なリズムについて評価が良くない、悪いということからリズムゲームの譜面として使用することは出来ないと考え

られる。この理由として本手法では楽曲のデータを中心に考えてはいたが楽曲以外の動画データを含めていたため、楽曲としての盛り上がりやサビ、間奏といった部分を一切考慮していなかった。あくまで音の鳴る瞬間をリズムの位置としていたことから楽曲のリズムとは合っていなかったと考えられる。この問題を解決するには動画、音声データから楽曲の盛り上がりやサビ、間奏の抽出をすることや演奏される楽器を抽出する等の音声データとしてではなく、楽曲演奏されているものと考え音楽としての視点から音声データの解析を行う必要があると考える。

また、ノーツの頻度、ノーツ量について本研究で作成された譜面が 1 レーンのみであり長押しが含まれないという制限により連続したノーツの配置等がただ押し続けると感じる要因になっていると考える。従来のリズムゲームでは連続したノーツの押し続けは長押しで押し続ける、レーンを常に変え同じレーンで押し続けないようにする、それぞれの手で別々のリズムを取るなどと言った方法で単調にならないようにしている。今回の譜面の作成方法および実験方法では作成した譜面がリズムが取れていないため不快に感じるのか、ほかのアクションがないため不快に感じるのかが明確には分からない。

ほかにもリズムゲームの譜面を作成する上でゲームである以上作成された譜面が面白い、楽しくなければならない。しかし、本研究では従来のリズムゲームの解析等は行っていないためどのような譜面にすれば面白い、楽しいと感じることが多いのか、反対にどのような譜面にはならないようにしたほうがいいのかといったことが分からない。ゲームに関する研究では特に問題であるどうすれば面白くなるのかと言った問題に対して何かしらのアプローチを行わないといけないが本研究ではあまり考えられていないため、リズムゲームとして面白くない、不快に感じるという結果が多くなってしまった。

本研究の手法で作成された譜面は動画・音声データから指定したノーツ数の譜面の作成をすることに成功した。しかし、作成した譜面はリズムが取れていない、ノーツが連続している部分がある等不快に感じる点が多くリズムゲームの譜面としては不適切であった。

第 4 章

まとめ

本研究では、音楽ゲーム譜面を自動生成する研究の改善策として、学習データを必要とする機械学習を使用せずに動画・音声データから譜面の自動生成を行い、同時に難易度に直結する要素のノーツ数を指定することが出来るリズムゲームの譜面の自動生成の手法を提案した。楽器以外の歌詞等を含めた音声データをそのまま MIDI データに変換を行い、その MIDI データから雑音になる音を削除する、その後残りのデータからリズムの出現位置としてノーツの配置候補を作成し、配置候補から指定数のノーツ数を抽出しリズムゲームの譜面を作成した。結果、指定数のノーツ数の譜面を作成することが出来た。しかし、作成した譜面はリズムを損なっており、不快に感じる部分が多いという結果であった。

今後の展望として抽出するデータや計算式を変えることでよりリズムの出現位置として適している場所を探し出すことを目指したい。また、動画・音声データをどのデータでも使用可能としていたため、楽曲の盛り上がりや間奏というのを関係なく使用した。そのため、リズムゲームの譜面として盛り上がりや間奏と言った部分を考慮した譜面の作成を目指したい。本研究ではリズムゲームにおけるノーツの配置のみ研究をした。しかし、譜面生成に関して多くの課題があり、ノーツの配置はもちろんのことノーツをどのレーンを流れるか、ノーツにどのようなアクションをさせるかなどによっても難易度は変わる。今後は、生成したノーツに関してどのような配置や

アクションが望ましいかについて考察や改善を行っていきたい。

謝辞

本論文を執筆するにあたり、指導していただいた渡辺大地教授、阿部雅樹実験助手、忙しい中実験に協力していただいた研究室の皆さま、そして支えていただいたすべての人に感謝いたします。本当にありがとうございました。

参考文献

[1] Chris Donahue, Zachary C. Lipton, and Julian McAuley. Dance dance convolution. *ICML 2017*, 2017.

[2] 辻野雄大, 山西良典, 山下洋一, 井本桂右. ダンスゲーム譜面の特性分析とクラスタリングに基づく特徴的な譜面の自動生成. エンタテインメントコンピューティングシンポジウム 2019 論文集, pp. 96–103, 2019.

[3] 福永 大輝越智 景子, 大淵康成. リズムアクションゲームにおけるキー音の自動推定. 芸術科学会論文誌, Vol. 18, No. 1, pp. 10–18, 2019.

[4] 辻野雄大, 山西 良典福本 淳一. 時系列深層学習に基づく難易度間関係モデルを用いたダンスゲーム譜面難易度の自動調整. 情報処理学会論文誌, Vol. 59, No. 11, pp. 1953–1964, 2018.

[5] Liang Yubin 池田 心. リズムゲームの上達を支援するコンテンツ自動生成法. 情報処理学会研究報告, Vol. 39, No. 11, 2018.

[6] 都丸英紀. 楽曲からのリズム取得の違いによる音楽ゲーム譜面の難易度表現に関する研究. 学部卒業論文, 東京工科大学メディア学部ゲームサイエンスプロジェクト, 2017.

[7] 紺野 凌西野 順二. 音楽ゲームの個人難易度ファジィモデル. 情報処理学会研究報告, Vol. 39, No. 9, 2018.

[8] 坂本 洸橋本 剛. フロー理論を用いた音楽ゲームの要素が面白さに与える影響の分析. 情報処

理学会研究報告, Vol. 183, No. 11, 2019.

[9] 土村貫太, 小松貴大. ディープラーニングによる自動採譜システムの開発. 第 82 回全国大会講演論文集, No. 1, pp. 367–368, 2020.

[10] まーてぃラボ / MartyLabo.【音の錯覚】ピアノ演奏だけなのに歌詞が聞こえる？/ auditory illusions: Hearing lyrics when only piano is played. https://www.youtube.com/watch?v=UooSKhLPJ48. 参照: 2020.12.25.

[11] 香川 俊宗稲葉 真理. 音楽の重要な構成要素の抽出の提案　—音楽ゲーム用譜面自動生成のために—. エンタテインメントコンピューティングシンポジウム, pp. 326–333, 2015.

[12] 高田 友則橋口 博樹. Midi に置けるメロディ情報を利用した繰り返し構造の検出. 埼玉大学紀要　工学部　第 39 号　2006, pp. 107–109, 2006.

[13] Unity. Unity technologis. https://unity3d.com/jp.. 参照: 2021.1.18.

[14] アイテルハック.【音ゲー用語集】音ゲー用語を徹底解説！https://eiter-hexe.com/music-game-glossary/. 参照: 2021.1.18.

[15] 株式会社セガ.「chunithm」公式 web サイト. https://chunithm.sega.jp/. 参照: 2021.1.17.

[16] BUSHIROAD.「bang dream!」公式 web サイト. https://bang-dream.com/. 参照: 2021.1.18.

[17] bearaudio. Mp3 から midi. https://www.bearaudiotool.com/jp/mp3-to-midi. 参照: 2021.1.18.

[18] softonic. Windows 用 audacity. https://audacity.softonic.jp/. 参照: 2021.1.18.

[19] わいやぎ. Smf (standard midi files) の構造. https://sites.google.com/site/yyagisite/profile?authuser=0. 参照: 2021.1.17.

[20] 株式会社システムインテグレータ. Json とは？データフォーマット（データ形式）について

学ぼう！https://products.sint.co.jp/topsic/blog/json#toc-2. 参照: 2021.1.18.