1　プログラム概要

この装置では単音のみからなるメロディを打ち込み、そのメロディを聞くことができる。以下のようなモードが存在する。なお初期状態は編集モードである。

1.1　編集モード

スティックでポインターを操作し、選択ボタンを押すことでポインターの位置にノートを置く (又は取り除く) ことができる。ポインターの色は`#FFFFFF` である。また、ボタンを押したまま横方向にポインターを動かす (縦に動いた場合は除く) と、動かした位置にもノートを置く (又は取り除く) ことができる。

スティックを長押しすることでポインターを複数コマ動かすことができる。最初に初めてスティックを倒したときに 1 つ、そこから 0.4 秒後に 1 つ、そこから 0.05 秒ごとに 1 つ移動するようになっている。斜めにポインターを動かすこともできるが、縦横の片方が移動できない場合は動ける方にのみ移動する。

選択ボタンを押した際にはポインターに対応した位置の音が 0.4 秒流れる。初期状態ではノートはどの位置にも存在しない。また、ノートの位置は以下のような意味を持つ。

横の位置は時間を表す。LED1 つ分が 32 分音符となっており、小節の境目 (正確には各小節の最初の列の LED) に薄緑色の線が表示されており、小節を 4 つに分けるように 1 小節あたり 3 本の白い線 (ポインターの色よりは薄い) が引かれている。小節数はプログラム内部の定数を変更することによって変えることができる。ポインターが画面端の 3 マスの地点に行こうとした時点で画面をポインターの位置がより画面中央に行くように画面を 1 小節分ずらす。編集不可能な領域は全ての LED が対応する音程の色で表示される。なお、表示される優先順位は高い順にポインター、ノート、縦線である。

縦の位置は音程を表す。音程は C3 から G5 までの 32 音 (半音含む) から成る。ただし、同じ時間に異なる音を 2 つ以上配置することはできず、既にノートのある位置に異なる音を配置しようとすると以前あった音は消える。縦の位置にはあらかじめ定められた色があり、同じ音程は同じ色で表される (例えば C3、C4、C5 は全て`#0078F0` の色で表示される)。編集不可能な領域に表示される色もこれに準じている。

1.2　演奏モード

演奏開始ボタンを押すと編集していたメロディが演奏される。演奏は全て矩形波によって行われ、隣接しているノートは全て 1 つの音としてまとめられる (例えば、同じ音の 4 分音符を 2 回続けて演奏しようとすると 2 分音符の音になる)。演奏中はポインターの代わりにどの地点を再生しているかを示す線 (ポインターと同色) が表示される。この線は演奏が進むにつれて動き、ポインターと同様に画面端の 3 マスの地点に行こうとした時点で画面をポインターの位置がより画面中央に行くように画面を 1 小節分ずらす。なお、BPM(曲のテンポ) はプログラム内部の定数を変更することによって変えることができる。演奏中に演奏開始ボタンを押した場合は演奏が一時停止され、もう一度演奏開始ボタンを押すと演奏が再開される。演奏を停止し、即座に編集モードに戻ることはできない。全ての演奏が終了した後は編集モードに戻り、再生前と全く同じ画面に戻る。

1.3　midi ファイルからプログラムを生成

`midiImport.py` を実行することによって midi ファイルを読み込み、初期状態が midi ファイルの通りになるようなプログラムを生成することができる。ユーザーはプログラムを実行した後、midi ファイルの名前 (拡張子`.mid` は除く) を入力することで midi ファイル名 (拡張子を除く) と同じ名前のディレクトリ内に同じ名前のプログラムファイル (拡張子は`.ino`) が生成される。midi ファイルにこの装置が表示できない音程の音が含まれていた場合、エラーが起こる。また、2 音以上が同時になっている場合は一番上の音が採用される。また、同じ音程で連なっているものは、midi ファイル上では異なる音として処理されていたとしてもこの装置では同一の音として扱われる。例えば、「夜に駆ける」のメロディを格納している midi ファイル `yorunikakeru.mid` に対してこのプログラムを使うと以下のようになる。

```
python MidiImport.py
Please input filename. ('.mid' is not necessary)
yorunikakeru
yorunikakeru/yorunikakeru.ino imported!
```

3 行目の”yorunikakeru”のみユーザーによる入力である。正常にプログラムが生成されると 4 行目のようなメッセージが出力される。

2　ハードウェア回路の実装

ハードウェアの実装は以下の図 1 のようになった。

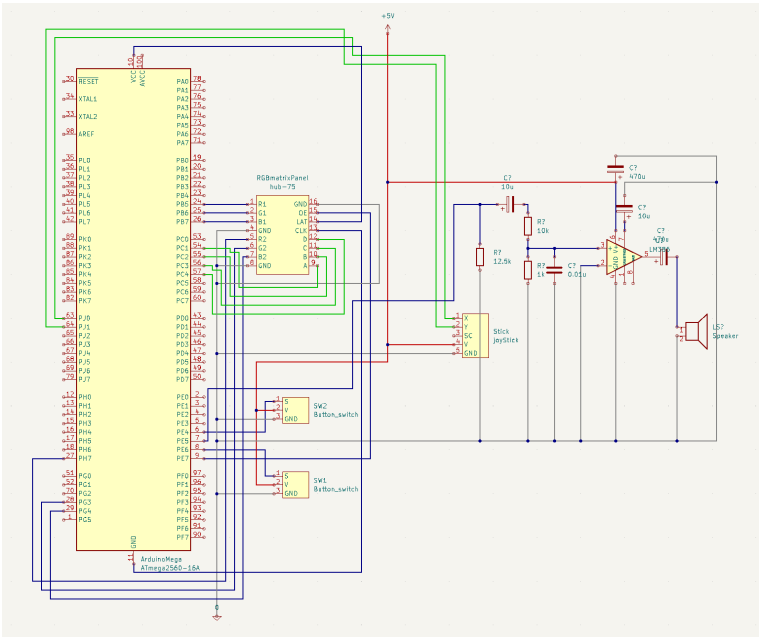

図 1: 装置の回路図

一番左のシンボルはは Arduino Mega を表しているが、正確に Arduino Mega を表すシンボルがソフトに存在しなかったため、各接続先にはピン番号のみが対応している。(例えば、11 番ピンは

Arduino Mega の 11 番ピンを意味しており、シンボルに書いているような GND としての役割はない。GND は回路図では Arduino と繋げず、、GND としてのシンボルとつなげることで表している。5V も同様である。) また、62 番ピンが存在しなかったため、62、63 番ピンはそれぞれ 63、64 番ピンに対応する。

図の線の色は以下のような意味を持つ。

- 赤は 5V と直接接続
- 灰は GND と直接接続
- 緑は Analog ピンと直接接続
- 青はその他

SW1 は選択ボタン、SW2 は演奏開始ボタンである。

回路図には記述されていないが HUB-75 には Arduino とは別の電源 (5V) を接続している。Arduino の 5V と接続した場合はうまくパネルが表示されない。

3　制御プログラムの解説

3.1　演奏開始ボタンの実装

図 2 は演奏開始ボタンによる状態遷移図である。分かりやすさのため状態遷移図を用いているが実際の実装はステートマシンではない。

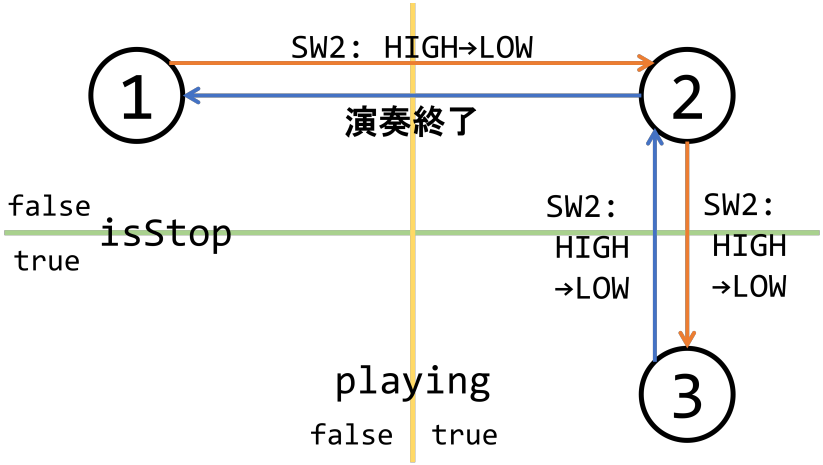

図 2: 演奏開始ボタンによる状態遷移図

状態 1 は編集モードである。ここから演奏開始ボタンが押される、すなわち SW2 が HIGH から LOW になるとき (ループの一つ前の SW2 の状態を持つことで検知できる) に状態 2(演奏モード) に遷移する。つまり、playing を true にする。

状態 2 から演奏開始ボタンが押されるとき、状態 3(一時停止) に遷移する。つまり、isStop を true にする。そこからもう一度演奏開始ボタンが押されるとき、状態 2 に遷移する。つまり、isStop を false にする。

3.2　選択ボタンの実装

基本的には演奏開始ボタンと同様に SW1 が HIGH から LOW になるのを検知し、処理を行う。また、スティックを動かしながらの選択操作を実現するため、スティックを動かした際にも SW1 が HIGH か LOW のどちらであるかを確認する (LOW の場合に選択処理)。

3.3　スティックの実装

スティックの入力は X、Y 共に 0 から 1023 までの整数で表され、512 が中央として扱われる。しかし、傾きが 0 であれば必ずしも入力が 512 になるわけではないため (ずれが多少生じる)、整数定数 a を用いて `abs([入力値] - 512)` が a より大きくなればその方向にスティックを倒していると判断する。今回この a は 200 としている。

スティックの状態は図 3 のようになっている。こちらも分かりやすさのため状態遷移図を用いているが実際の実装はステートマシンではない。緑の矢印はスティックの入力内容が変化したことによ

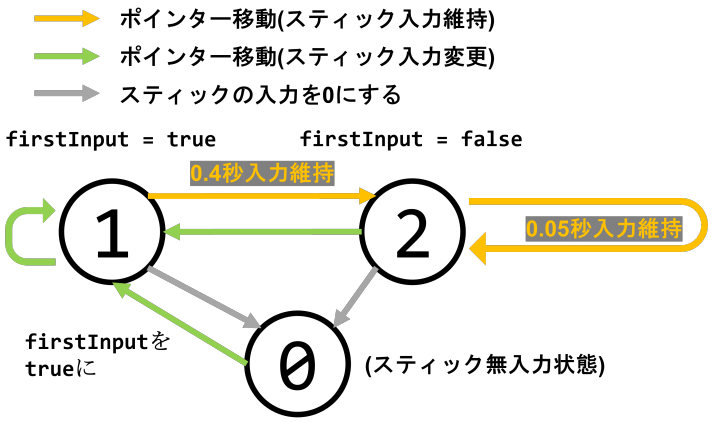

図 3: スティックの状態遷移図

よるポインター移動が起こった際の状態遷移である。また、黄色の矢印はスティックの入力維持が一定時間行われたときに発生するポインター移動が起こった際の状態遷移、灰色の矢印はスティックの入力維持が 0 になった際の状態遷移である。

状態 0 はスティックの入力が 0 の状態、状態 1 は最初のポインター移動からスティックの入力が維持され続けている状態である。状態 2 は 2 回目以降のポインター移動からスティックの入力が維持され続けている状態である。

この状態の決定はスティックの入力と `firstInput` によって行う。スティックの入力が 0 であれば状態は 0 であり、そうでなければ `firstInput` が true であれば状態 1、false であれば状態 2 である。なお、状態が 0 から 1 になる際に `firstInput` を true にしているため、状態 0 では firstInput は false である。

3.4 LED パネルへの表示機構の実装

LED マトリクスへの表示は RGBmatrixPanel というライブラリを用いる [1]。RGBmatrixPanel のインスタンスを生成できる。このプログラムで用いるメソッドは以下である。

- Color888 ... RGB カラーの値 (各色 0〜255) を用いて色を表す整数を生成することができる。
- begin ... LED の点灯開始ができる。
- drawPixel ... LED の位置、Color888 メソッドで生成した色を指定してその位置をその色で点灯させることができる。
- fillScreen ... 全ての LED を指定した色で光らせることができる。

3.4.1 楽譜の管理

楽譜の管理は 1 次元配列 notes によって行われる。この配列の大きさは小節数の 32 倍に等しい。つまり、32 分音符を最小単位として管理を行っている。この配列のインデックス i の要素が j のとき、最初から 32 分音符が i+1 個目の位置に LED の j 行目に対応する音が存在することを意味する。例えば、図 4 のようになっている。

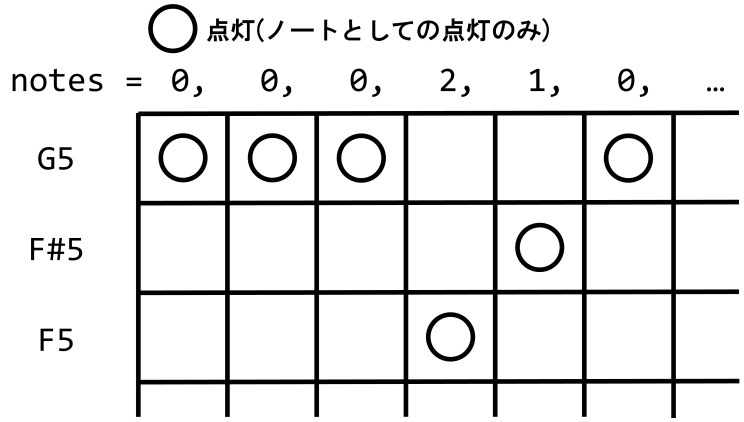

図 4: notes の要素と LED の点灯状況

3.4.2 ノートの表示

ポインターが移動する際に元あった場所にノートが存在すればそれを表示する、という処理を実装する。なお、ノートが追加される際には必ずその位置にポインターが存在するので、この処理が実装されていればノートの追加時には notes を更新するだけでよい。

3.4.3 画面移動の実装

この処理は関数 `wholeMove` で実装されている。この関数の中では、関数 `drawVeLine`、`decideVLine` を用いている。

まず、fillScreen メソッドで一度画面を全て消灯 (黒色に) する。次に、画面左端の位置を変更し、縦線を描画する (drawVeLine、decideVLine を用いる)。その次にノートの描画を行い、最後にポインターの描画を行う。この順に描画を行うことで優先順位を使用の通りにすることができる。

3.4.4 初期表示の実装

この処理は関数 firstView で実装されている。この関数はプログラム開始時だけでなく、演奏モードと編集モードの切り替え時にも使用されている。

編集モードの時は画面移動と同様に縦線を描画し、ノートの描画を行い、最後にポインターの描画を行う。ただし、画面左端は変更しない。また、演奏モードの時は最後にポインターではなく 0 列に再生地点を表す線を描画する。

3.5 演奏モードの実装

演奏には tone 関数を用いず、TimerThree ライブラリ [2] を用いて矩形波を出力させている。このライブラリでは Timer3 を用いて割り込み処理を行うことができる。Timer1 は既に LED マトリクスによって使用されているため使用できず、Timer2 は 8bit タイマーであるため [3]、音程を表現するためには不十分である。また、Timer3 以降は Arduino UNO には存在しないため、この装置では Arduino Mega を使用している。

Timer3.initialize でマイクロ秒単位で割り込みの時間を設定でき、Timer3.attachInterrupt で割り込み処理で実行される関数を指定できる。音が変わるたびに音の波形の周期を計算し、その 1/2 を割り込みの時間として設定し、割り込み処理で出力の HIGH/LOW を反転させることで矩形波の出力を実現させている。

32 分音符分の時間が過ぎると再生位置の線を 1 つ動かす。この時、ポインターと同じように画面を 1 小節分ずらす。

3.6 ノートの試聴機能の実装

音の鳴らせ方は演奏モードと同様である。選択ボタンを押した際に波形の出力を開始、同時にその時の millis 関数の出力を得る。millis 関数を用いて押下から 0.4 秒経ったかを調べ、経っていれば波形の出力を止める。

3.7 midi 読み込みスクリプトの実装

midi ファイルを扱うための python ライブラリの mido を用いる [4]。このファイルで midi ファイルを読み込み、notes と同様の形式の配列に変換した後、notes の変数宣言文に変換する。また、小節数も midi ファイルから読み取り、定数 measureSize の宣言文に変換する。この 2 つを baseProgram.ino に追加したものを新しいファイル (名前は拡張子を除き midi ファイルと同名) として生成する。

別のプロジェクトとして認識されるために midi ファイルと同名 (拡張子はなし) のディレクトリを作成し (既に存在する場合は作成しない。)、その中にファイルを作る。

4　考察・工夫点

4.1　プログラムの高速化

演奏モードの際に画面更新に時間をかけすぎた場合、タイマーの割り込み時間変更が遅れて演奏が正常に行うことができなくなってしまうため、画面更新の際の実行時間をできるだけ小さくするよう工夫した。

例えば、全ての LED を消灯させたいときに drawPixel メソッドで 1 つずつ消すより fillScreen メソッドで消す方が早い。

また、1 音だけしか鳴らすことのできない制約のため、音が鳴っている音程だけ 1 で他が 0 となっているような 32 分音符数× 32 の 2 次元配列で楽譜を管理するよりも、各箇所の音程が記されている 32 分音符数の長さの 1 次元配列によって楽譜を管理する方が空間計算量を削減することができ、画面の更新の際の時間計算量も削減できる。私が課題 3 で実装したように音程と長さを管理する方法であればより空間計算量は削減できるが、LED の表示処理が複雑になり、処理に時間がかかると考えたため採用はしなかった。

4.2　midi データの読み込み

課題 3 までで作成したプログラムは Arduino 上で実行されるため、このプログラムから直接 PC のファイルにアクセスすることは難しい。そこで今回は midi データを読み込んでその midi に沿った初期状態になるプログラムを生成するスクリプトを実行することで midi の読み込みを間接的に行えるようにした。

4.3　縦線の描画

今回は任意の距離の画面移動に対応できるよう縦線の画面移動の描画時に縦線の位置を計算し直していたが、実質的に現在の使用では 1 小節分の画面移動しか存在しないため、縦線の位置は動かない。また、縦線描画後に編集不可範囲の描画処理を行うことで画面外に縦線が出現する心配をする必要はない。

4.4　同じ音の境目

今回の装置では音の開始と終了部分が繋がった場合 1 つの音として認識されてしまうという問題がある。今回は 32 分音符という小さい長さの音符を最小単位としユーザー側で音と音に間を作ってもらうという形を取ったが、音が最後の部分でフェードアウトするといったような波形の工夫を与えて境目で波形が異なるようにすれば、音の開始と終了部分が別の音として演奏されるようになる。ただし、この装置では音の開始部分や終了部分が区別できないような管理をしているので、この部分に工夫をして区別できるようにする必要がある。

5　感想

今回の課題では期限の都合もありできなかったが、最終的には midi の読み込みからプログラムの生成、書き込みまでを行えるアプリケーションが使えるようになるのが理想であると考えている。

最初に和音の実装について実験をして行き詰まり、授業 1 回分の時間を無駄にしてしまった。最初に少しの機能でもいいので全貌を完成させ、その後の拡張の一環として和音の実験は行うべきであったと考えている。

また、コードが課題 3 までとは異なり、かなり長く実装も複雑であった (課題 1G が 76 行、課題 2D が 139 行、課題 3E が 87 行に対し、課題 4 は Arduino のコードが 410 行、python コードが 57 行) ため、シリアル通信を多用してデバッグを行った。ただし、Arduino Mega のピンは多かったため、シリアル通信に影響を与えないようなピンの使い方を考えるのは容易であった。Arduino Uno で複雑なコードを書く場合はピンを開けるよう注意したい。

最初は LED マトリクスパネルの使い方がよくわからず、アドバイスを受けたままに Arduino Mega を使っていた (恐らく LED マトリクスパネルのみであれば Arduino Uno でも動作する) が、マトリクスパネルのライブラリが Timer1 を使っていて波の出力に Timer3 が必要であると気付いたのが製作の終盤であったため、幸運であった。特に Arduino Uno の場合タイマーの数はかなり制約が厳しいため、装置を作る際にはタイマーの個数や、空いているタイマーのビット数に注意すべきであると考えた。

参考文献

[1] RGB matrix Panel,
https://www.arduino.cc/reference/en/libraries/rgb-matrix-panel/

[2] TimerThree - Arduino Reference,
https://www.arduino.cc/reference/en/libraries/timerthree/

[3] Skill Builder: Advanced Arduino Sound Synthesis - Make: (makezine.com),
https://makezine.com/projects/advanced-arduino-sound-synthesis//

[4] Mido - MIDI Objects for Python — Mido 1.3.0 documentation,
https://mido.readthedocs.io/en/stable/