目次

可変長テンプレートによる関数情報の取得

このは

1　はじめに

以前、スクリプト言語である AngelScript のラッパーライブラリを作成した際に、関数ポインタやメンバ関数ポインタから、関数定義を文字列として変換する経験をした。関数定義を文字列としてほしいときは中々ないが、関数情報を取得したいと思うことは多少ある。そういった時に役に立てば良いと思い、C++ の機能である可変長テンプレートやテンプレートを用いて、関数ポインタや、メンバ関数ポインタから、関数情報を取得する方法を示そうと思う。なお、以下に示すソースコード群は Visual Studio 2015 Update3 でのみ動作確認をしている。その他の環境では、うまく動かない場合があることをご容赦いただきたい。

2　可変長テンプレートについて

C++ の機能である可変長テンプレートとは、元々あった可変長引数を型安全にした機能であるといえる。まず、実引数がある可変長テンプレートについて説明しよう。可変長テンプレートを使用した与えられた引数をコンソールに出力し、最後の引数を出力した後改行を出力する `TemplatePrint` という関数と、実際に可変長テンプレートを展開する `TemplatePrint_impl` という関数は、以下のような定義となる。

実引数がある場合は 1 つずつ可変長テンプレートの引数から第 1 引数として取り出すことができる。そして、最終的に引数が void の関数へと帰着される。引数がどちらも void であるから、template 指定はいらなくなり、普通の関数定義として表される。この関数の展開は、コンパイル時に行われることに注意である。この関数群は、関数テンプレートに引数があるため、実引数ありの可変長テンプレートと言われる。

```
1  template<typename T, typename ...Arg>
2  void TemplatePrint_impl(T& Value, Arg&... arg){
3  std::cout << Value;
4  TemplatePrint_impl(arg...);
5  }
6
7  void TemplatePrint_impl(){
8
9  }
10
11 template<typename ...Arg>
12 void TemplatePrint(Arg&... arg){
13 TemplatePrint_impl(arg...);
14 std::cout << std::endl;
15 }
```

次に実引数のない可変長テンプレートを説明する。ここでは、可変長テンプレートとして渡された型の情報を出力する PrintType という関数を例に挙げる。紙面の都合上コンパイルエラーとなる改行が含まれているのはご容赦いただきたい。この関数では関数テンプレートに引数がないため、実引数のない可変長テンプレートと言われる。実引数のない可変長テンプレートの場合、実引数のある可変長テンプレートと違い第 1 引数だけでなく第 2 引数の要素が必要となる。そして、最終的に template 引数が 1 つだけの関数に帰着される。その関数では、ポインタや参照が付属していると is_class や is_const が正しく判定されないので、remove_pointer や remove_reference メタ関数でポインタや参照を削除して判定させている。typeid ではポインタや参照は出力されないので、こういった形で出力している。また、この型定義を使用して標準のメタ関数から様々な型特性について得ることができる。

```
1  template<typename First, typename Second, typename ...Arg>
2  void PrintType(){
```

```
3  PrintType<First>();
4          PrintType<Second,Arg...>();
5  }
6
7  template<typename T>
8  void PrintType(){
9  using remove_pr = std::remove_pointer
10                          <std::remove_reference<T>::type>::type;
11 std::cout << (std::is_const<remove_pr>::value ?
12                          "const " : "");
13 std::cout << typeid(remove_pr).name();
14 std::cout << (std::is_pointer<T>::value ? " *" :
15                          (std::is_reference<T>::value ? " &" : ""));
16 std::cout << std::endl;
17 }
```

3 関数ポインタ、メンバ関数ポインタから関数情報を出力する

関数ポインタやメンバ関数ポインタから、戻り値や引数情報を出力するのはおもったよりも簡単である。template を使用することで戻り値や引数の型を取得できる。それを使用して、関数情報をコンソールに出力する関数を示す。2 の可変長テンプレートで述べた関数を使用していることに注意である。以下のように関数のオーバーロードを使用することで、関数ポインタやメンバ関数ポインタから関数、メンバ関数、const 修飾されたメンバ関数の 3 つに割り振ることができる。これによって、関数に入った時点でメンバ関数か、const 修飾されたメンバ関数かという 2 点は取得することができる。また、示したように関数ポインタの定義や、メンバ関数ポインタの定義において、必要なところを template 指定することで一般的なものについて書くことができ、2 で説明した可変長テンプレートやテンプレートと組み合わせることで、型情報を出力することができる。このような、非常に短いコードで関数ポインタやメンバ関数ポインタから文字列として関数情報を出力することができる。また、今回は出力しているが、std::typeindex や多くの bool 変数で、型情報について汎用的に扱うことができ、以下の関数内でメタ関数を使用することで変数に代入する型

情報を取得することができるようになる。

3.1 秘伝のソースコード

```
1  template<typename Return, typename Arg...>
2  void OutputFunction(Return Func(Arg...)){
3          std::cout << "IsMemberFunction : false" << std::endl;
4          std::cout << "IsConstMemberFunction : false" << std::endl;
5  std::cout << "Result : ";
6  PrintType<Return>();
7  std::cout << "Argument : ";
8  PrintType<Arg...>();
9  }
10
11 template<typename Return, typename Class, typename Arg...>
12 void OutputFunction(Return (Class::*Func)(Arg...)){
13 std::cout << "Class : ";
14 PrintType<Class>();
15 std::cout << "IsMemberFunction : true" << std::endl;
16 std::cout << "IsConstMemberFunction : false" << std::endl;
17 std::cout << "Result : ";
18 PrintType<Return>();
19 std::cout << "Argument : ";
20 PrintType<Arg...>();
21 }
22
23 template<typename Return, typename Class, typename Arg...>
24 void OutputFunction(Return (Class::*Func)(Arg...)const){
25 std::cout << "Class : ";
26 PrintType<Class>();
27 std::cout << "IsMemberFunction : true" << std::endl;
28 std::cout << "IsConstMemberFunction : true" << std::endl;
29 std::cout << "Result : ";
30 PrintType<Return>();
31 std::cout << "Argument : ";
32 PrintType<Arg...>();
33 }
```

4　おわりに

可変長テンプレートを使用した関数ポインタやメンバ関数ポインタから関数情報の取得はどうだっただろうか。私はこれを書いた時思っていたよりも簡潔なコードとなり、C++ の機能を再確認した。以上に示したように `template` は非常に有用であり、コードを短くでき、複雑そうなことも容易に実装できる能力を持つ。また、コンパイル時に展開や計算が行われるため、実行時の処理が少なくすむといったメリットもある。関数情報の出力以外にも応用が広く効くため、ぜひ使いこなしてほしい。

twitter: @HunterKonoha

2016 年 11 月 23 日

8

Cubase のすすめ

トド

1　はじめに

音声処理に使われるソフトで私が使用している Cubase Pro8 についてオーディオファイルの編集を中心に紹介したいと思います。

2　入出力出来るファイルの種類

まず入出力出来るファイルは

- Wave ファイル (.wav)
- AIFC・AIFF ファイル (.aif)
- mp2 ファイル (.mp2) mp3 ファイル (.mp3)
- SD2 ファイル (.sd2)
- wma ファイル (.wma)
- flac ファイル (.flac)
- ogg ファイル (.ogg)
- REX2 ファイル (.rx2)
- Wave 64 ファイル (.w64)

と、普段使われてるファイル形式から化石のようなファイル形式まで入力できるようになっている

これらのファイルをプロジェクトに読み込むときに Wave ファイルにコピーしてプロジェクトファイルに保存できるので、元のファイルに何も影響を及ぼさない非破壊編集のために、コピーしておいた方がよい

3　基本的な編集方法

動画編集ソフトなどでもおなじみなオブジェクトを時間軸で設置することにより設置した音声ファイルを時間で再生することが出来ます

また、ツールを使うことで音声の音程を変えずに再生時間の伸縮ができ、これにより、テンポが違う楽曲でも同じテンポに合わせることができます。

4　ヒットポイント

読み込むファイルが音楽ファイルだった場合、ヒットポイントという楽器の音を出す部分を検知する機能がある

例えばドラムの音や、オーケストラの一拍ごとに音をだす部分を検知して曲のテンポを検出することが出来る

主に耳コピなどをするときにテンポが分かって便利である更にヒットポイントで音声を分割することが出来るので、一拍のみ音を取り出したい時など自動でやってくれ宇

5　VariAudio

歌や楽器の収録などで音程が上がりきらずに、少し音程がずれてしまう時など VariAudio を使うと音程を調整することが出来ます。

また音程の揺れなどを抑えてケロケロボイスなども作れることができます

6 標準 VST エフェクト

標準で EQ 処理が出来るエフェクトや、コンプレッサーなども付いており、マスタリングや正規化なども処理できるため楽曲ごとの音量差などを調整することができる

7 Groove Agent

Cubase に標準でついているドラム音源 Groove Agent は、Wav ファイルを鳴らす位置を midi トラックで制御することができるこれはドラム音源を midi 信号で操るための物だが、音程も設定できるため Wav のサンプラーとして扱うことができる

8 まとめ

今の時代だと無料で音声ファイルの編集などは出来るが、やはり自分的には有料ソフトを使った方が効率も上がりストレスがない編集作業が出来ると思う

さらに、音声処理と midi ファイルを扱える DAW は Cubase 以外にもいくつもあるので OS や PC スペックなどと相談して決めるのも良い

ここまで説明してきたが、エディタの影響で画像が挿入できず、さらに正直自分の文章が悪文のような気がするので、公式サイトから Cubase の体験版がリリースされているのでそれを使ってみてから便利さを体験するのもよいと思う

情弱自作 PC 記（C90 の記事の続き？）

いとしゅん

1　ごあいさつ

本誌を手に取って頂いた皆さんこんにちは。毎度ながらコンピュータ技術研究会の情報弱者担当及び、任期は残りわずかの平成２８年度東京都市大学文化団体連合会本部役員をやらせて頂いております、いとしゅんです。いつの間にか 2 年生の後半戦にもなりまして大学の授業や課題をこなすのも苦労する時期になってきました。最近は徹夜も解禁するようにもなってきて天パな頭の上だけでなく頭の中がぐちゃぐちゃになってきています。無理にでも時間を手に入れてガンプラでも作らないと精神死にそう。

今回も相変わらず読んでもプログラムのプの字も見当たらない、技術の技も無いような記事なのですがおつきあい頂けるとちょっと嬉しいです。（Minecraft の Mod 制作とか記事にしたかったけれど出せるほどの物が作れてないんですよね... プログラミング嫌いが現れる）

2　前回までのあらすじ

前回の夏 C90 では自作 PC 組んでみたよ（作成動機、パーツ選定などは友人の差し金）という内容で制作記を記事に書かせて頂きました。とりあえず使った部品を列

挙しますと（6 月時点ですのでパーツは当時買えたやつです)、

- CPU...Intel i5-6402P（i7 じゃないのは予算のしわ寄せ。）
- マザーボード...ASRock Z170M extreme4（ミニタワー型ではいい方のマザボ）
- CPU ファン...CoolerMaster HYPER 103 （予算内では最強クラスの CPU ファンらしい）
- SSD...Sandisk SSD plus 120GB × 2（RAID するために 2 つ）
- メモリ...TED48GM2400C16DC0 （こいつがスペックの足を引っ張ってるうえ個体値低い）
- ケース...AEOS USB3.0（僕の部屋狭いんでミニタワー型ですね）
- キーボード...BUFFALO BSKBC02BKF（普通の形過ぎてゲームでは使いづらい）
- 電源... 玄人志向　KRPW-L5-500W/80+（初心者なのに玄人志向）

その他ケースファン、配線や USB 有線マウス、win8.1 → win10、友人の食費（え？）

そして前回記事執筆時点では調整が済んでなくて紹介しそびれていましたが（友人作 PC 紹介動画（ニコ動で sm28579500）では紹介あったような・・・)、Mod マシマシマイクラやるのに重要パーツの GPU である

- GEFORCE GTX 970 （これ使うなら他のゲームもやれよ←時間と金が無い）

といった本来こんな組み方する奴いるのかとツッコミしか無い PC を作ってみました。前回の記事では完成してから一ヶ月弱だけ使った感想やここの調整が弱かったとか、オーバークロック値高くしすぎて今の設定じゃ不安定過ぎて使い物になって無いなあといったことを述べていました。とりあえずこいつを安定させて使える状態にして半年弱遊び倒してみたよというのが今回の記事になります。

3　半年使ってみた!

さて前置き長くて申し訳ありません、本題にまいりたいと思います。さて前回の時点ではなんとか起動してマイクラできるよやったねみたいなところで終わっていた

のですが、前回から流石に設定をいじらせていただきまして、レポート書きで一日付けっぱなしでも大丈夫なくらい（マイクラ何処行った）CPU とメモリの調整幅を下げて安定感を向上させてみました。

前回 6 月当時ではメモリ Frequency に関しては、DDR-2400 でスペック上いける（というかいってくれなきゃ困る）からこの値で固定していて、CPU 電圧などだけ合わせて安定させればいいかと、別にそこまで要求されるものをやりたくて自作 PC 作った訳ではないけれど（動機：マイクラ)、どうせなら性能下げたくないなとここは動かさずにいたのですが何回やっても、内部飛んじゃってすぐ再起動かかってしまいせっかくのマイクラ建築がーってなってしまいましてこんなんできる環境じゃないと思いまして妥協してメモリを DDR-2286 まで下げましたところなんかそれまでが嘘のように落ちなくなりまして、今の状態に至りました。前回では原因まではわからずなんか適当にいじってたら落ちなくなった、やったーという感じでメモリがどうやら原因というわけだと特定するまでには至らなかったのですが、曲がりなりにもコンピュータを学んでいる人間が自分カスタムの PC のことわかってないとか話にならないなということで原因を突き止めたわけです。カタログスペックよりも低い個体にあたってしまうと結構残念な感じになりますね。（メモリ返品交換してもなったんで諦め）これでも全然気にならないレベルなので今のところはこれでしばらくいくでしょうね。（友人曰く多分最初に部品交換するならメモリだな）

安定感を得たおかげで今現在うちの PC のなかでぶっちぎりに良い物になりました。ノートでは遅い Photoshop とか動かしても速い速い。小学校時代から家の PC がノートだった勢の僕にとっては感動ものですね。（あれこのセリフ前にも言ってたような…）この原稿もこの PC で書かせて頂きました。日々のレポートもはかどりますね…（出来が速くなったとは言ってない）

また GEFORCE GTX 970 を載せたので昨今の PC ゲームなら結構やれるグラフィックであるのですが未だにマイクラ以外のゲームに手を出してないですね…2 次元ロボット好きなので Figureheads というゲームをちょっとやってみましたが、PC のスペックは追いついてますが、家の回線がマンション回線なもんで若干遅いし、何よりも自分がゲームのスペックに追いつけない感じでしたね。ネット対戦するとすぐに狩られるいわゆる地雷でした。（ゲーミング PC 作っておきながらゲームあまり得意でないことを忘れる）アクションゲーム以外でも結構やりたいゲームはあるので

これから先が楽しみですね（現実逃避）

せっかくの高スペ（？）のパーツも使ってますし、ゲームやる以外にもクリエイター側としてなんか作ってみたいなと思ってはいますが、なんせ作りたい形にできるようになる前にどれも飽きてしまってる（ガンプラは説明書通り組めばカッコイイのできますし改造もある程度好きなようにできるので例外ですがパソコン作業になるとちょっとコンピュータで表現するまでの壁というか情弱ゆえにやり方を知らないとか…）ので、そこをぜひ治せるような人物になるのが今後の課題ですね。お絵かきとか DTM とか興味はあっても、作品作るセンスとできるまで勉強し続ける根気がないから駄目ですわ。

4 元々マイクラがやりたくてこの PC 作ったんじゃ

ここで、この自作パソコン出来たら絶対にやるぞという悲願であった Minecraft を実際にやってみた感想を少々。最初 GPU なしの家のノート PC では素の状態いわゆるバニラの環境ならまだやれる、mod 追加したら死ぬ。といった、なぜそれでやろうと思ったしという感じでした。

で、初めてこの PC を使ってやった時、スポーンして F3 押した瞬間 fps をみてびっくりしました。（図 1 は mod が約 30 個ほど入った Ver1.7.10 でアンプリファイドワールド生成し、影 Mod は Chocapic13 V5 Extreme を使用した時のスクリーンショット。58fps を確認。）このモニタ 60fps までしか対応してないのに 80fps とか何だよってなってしまいテンションが上がりました。影 mod 入れましても 60fps 足りてるしアンプリファイド地形生成問題なし、mod もゲリラ mod やアルスマギカが動くぞおおおって感じ（工業 mod は冬休みの楽しみにまだとっておいてる）で、夏休みずっと Minecraft に没頭してましたね…念願の友人とのマルチも出来ましたし、なんだか万々歳って感じです。（大学始まってからほぼ課題やるくらいにしか使って無いのがほんとに惜しい）

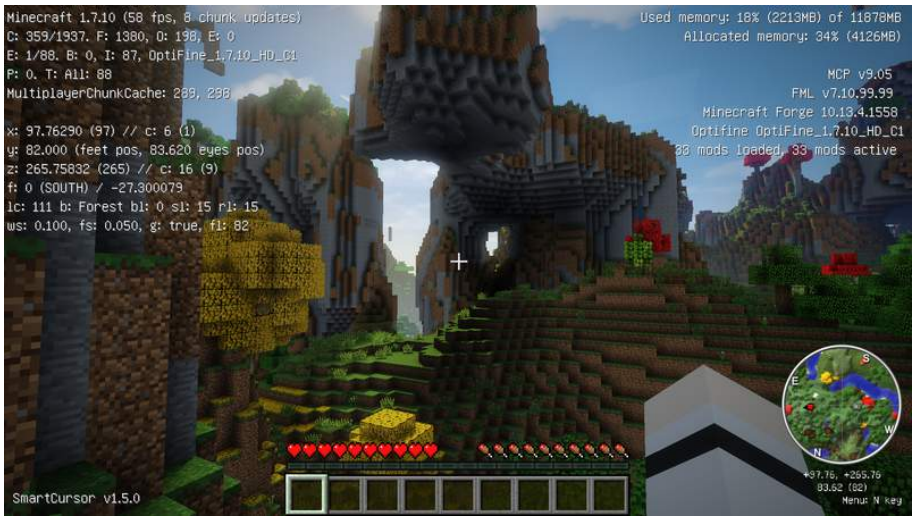


図 1: 30 個 Mod 入りアンプリファイド影 Mod 入れても 55〜60fps くらい出るよという図

そしてせっかく mod が動く PC も作りましたし自分用の mod を作ってみるかと思い立ちまして、現在進度が絶賛停止中ですが、mod の制作の勉強も始めてみました。まだオリジナルの機能を持ったものは作れておらず、ちょっと性能をいじった道具、ポーションのデバフを食べると与える食べ物程度しか作れてはいないのですけれどね。ゲームやアニメに出てくるような兵器や武器みたいなのを作ってみたいのですがなかなかうまくいかないです。（射撃武器の作り方誰か教えて下さい）

5　おわりに

ここまでただ自作パソコン作って、使った感想を述べてもう終わるのかよ、本当に技術面語ってねえな。となる記事でしたがここまで読んでいただきありがとうございました。正直記事が前回とほぼ同じものとなってしまいまして本当に申し訳ありません。先程も少し触れさせていただきましたが、次回何かしらのご縁がありましたら今度こそは Minecraft の Mod 制作などで自分の何かしらが語れるものを用意しておきたいと思います。（レポート書き以外でなんか形が作れるものをこのパソコンでやりたいですし）

最後となりますがこのコンピュータ技術研究会会誌を手にとっていただき誠に感謝します。

twitter: `@itus_in_stellar`
普段はあんまりつぶやいていないうえコンピュータ関係ない話しかしてないですが... よろしければ
2016 年 12 月某日

Atom というエディタ

まくら

1　はじめに

今最高にイケてるエディタを紹介する。現にこの原稿は tex で書いてるのだが、そのエディタでコンパイルやら pdf ファイルを出力している。そのエディタとは「Atom」である。

Atom とは Git Hub が開発したオープンソースのテキストエディタである。開発テーマは「A hackable text editor for the 21st Century」(20 世紀に向けたハック可能なテキストエディタ) とされている。らしい。

Windows、Mac、Linux の環境で無料で使えるうえに、GitHub 製であるため信頼性も高いのである。

2　十万馬力だ、鉄腕 Atom

2.1　インストール方法

Atom の入れ方を紹介する。以下の URL から exe ファイル AtomSetup.exe をダウンロードし、実行するだけである。非常に簡単である。

https: //atom.io

2.2 パッケージ

ここが私が一番お勧めしたい部分である。まずパッケージの導入が楽なのだ。よって日本語化がインストールしてから数分でできるのである。最高！

具体的なパッケージの入れ方はメニューバーにある File から Settings を選ぶ。そして Settings 項目内にある Install を選択し、欲しいパッケージを検索し、インストールボタンを押すだけである。

パッケージはすでに膨大な数があり、様々なパッケージが用意されている。POV-ray や Verilog などの言語に対応したパッケージもあり、様々な開発者にも柔軟に対応できる。さらに、Atom 内でコンパイル、実行も出来るパッケージもあるので新しい言語を覚えるのも非常に捗るのである。自分が使ったものでいえば python や tex、ruby などがコンパイルできるパッケージなどがある。ただし、Path を設定しないとコンパイル、実行できないので注意が必要だ。

パッケージには pdf や png、jpg などの画像ファイルも見れるようにするパッケージもある。さらに twitter を見れるパッケージまでも存在する。

さらには vim や emacs、sublime text などの他のテキストエディタに似せることがパッケージも存在するので使ってみることをお勧めする。

2.3 Git との連携

GitHub 製だということもあり、Git と連携が出来るパッケージも豊富である。comit や push も Atom 上で出来るパッケージも存在する。

また、README などで使うマークダウンをリアルタイムでプレビューしながら書け、なおかつ HTML の出力もできるのである。これはパッケージも入れずに標準でできる機能が備わっており、Git を良く使う人にもオススメできる。

2.4 テーマ

Atom はカラースキームなどのテーマの設定も簡単に変えることができる。パッケージをインストールする Setting のパッケージ画面で欲しいテーマを検索しインス

トール、さらに Setting のテーマで使いたいテーマを選ぶだけである。これにより自分好みのエディタにすることもできる。

2.5　キーバインドの設定

Atom ではキーバインドが設定できるのも魅力だ。キーバインドが変更できることに好きなテーマと合わせてさらに自分好みのエディタに変えることができるのである。

2.6　難点

Atom がかなりお勧めできるエディタであることは今まで示した通りだが、難点があることも上げておこうと思う。

それは、少し重いという点である。最初の立ち上がりが遅く起動するのが多少億劫になってしまうのだ。しかし、欠点らしい欠点はそれくらいである。

3　最後に

ここまで Atom をお勧めしてきたが個人的には使っていて楽しく捗るエディタであることは確かである。一回試してみても損はないはずだ。これが Atom を使うきっかけになってくれれば幸いである。

2016 年 12 月 7 日

まくら

声優の声帯ってどんな声帯だ？ケプストラム分析編

フミノ

1　はじめに

オタクなら誰しもあぁ＾〜耳が幸せなんじゃ＾〜っという体験をしているはず。ではその耳が幸せな音はどこから来るか？だいたいは声優の喉から来ています。じゃあ同じ音を自分で出そうと思っても、そんなに世の中上手くないです汚い音が出るだけです。ではそんな幸せになる音を出す声優と出せないオタクでは何が違うか、そう声帯です。その声帯、一体どんな構造になってるか気になりますよね。そんな声帯の特性について解析する手段があるので今回はその手法を元に解析していこうと思います。

2　ケプストラム分析

2.1　音声のモデル

周波数域においてのある音声を $S(\omega)$ とした時、$S(\omega)$ を音源と声道フィルタというモデルに分けることが出来ます。声道フィルタのモデルはさらに声道の伝達関数と口唇からの放射特性に分けられます [1]。しかし今回は音声 $S(\omega)$ を音源と声道の二つのモデルであると考えます。

音源とは、今どれ位の高さの音が出ているかというピッチ周波数などを指し、これを $G(\omega)$ とします。声道フィルタとは人の声道における共振や反共振特性、つまりはどの周波数域は強くどの周波数域は弱いかというスペクトルの包絡を指し、これを $H(\omega)$ とします。これら $S(\omega)$,$G(\omega)$,$H(\omega)$ には式 1 の関係があります。

$$S(\omega) = G(\omega) \cdot H(\omega) \tag{1}$$

これは音声は肺からでた音源が声道というフィルタを掛けられて音声になるということです。

2.2　ケプストラム

私達が観測できるのは音声 $S(\omega)$ だけです。ですので $G(\omega)$ と $H(\omega)$ は一般的に推定できません。しかし式 1 をある性質を使い分離します。それは対数を取るということです。式 1 に対数をとり変形したものを式 2 に示します。

$$\begin{aligned} \log|S(\omega)| &= \log|G(\omega) \cdot H(\omega)| \\ &= \log|G(\omega)| + \log|H(\omega)| \end{aligned} \tag{2}$$

さらに対数をとった対数スペクトル $\log|S(\omega)|$ を逆フーリエ変換します。この対数スペクトルの逆フーリエ変換をケプストラムと言います。逆フーリエ変換した結果を c_n とおき式を式 3 に示します。

$$\begin{aligned} c_n &= \frac{1}{2\pi}\int_{-\pi}^{\pi} \log|S(\omega)| e^{jn\omega} d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} (\log|G(\omega)| + \log|H(\omega)|) e^{jn\omega} d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} \log|G(\omega)| e^{jn\omega} d\omega + \frac{1}{2\pi}\int_{-\pi}^{\pi} \log|H(\omega)| e^{jn\omega} d\omega \end{aligned} \tag{3}$$

音源のケプストラムを c_{G_n}、声道フィルタのケプストラムを c_{H_n} とおき式 3 を式 4 のようにまとめます。

$$c_n = c_{G_n} + c_{H_n} \tag{4}$$

式 4 で音声のケプストラムを音源と声道フィルタのケプストラムの和に落とし込めました。

ここでケプストラムについてのお話を。ケプストラムは対数スペクトル $S(\omega)$ を逆フーリエ変換しましたが、少し視点を変えてみましょう。$S(\omega)$ を時間信号と思いフーリエ変換したと考えてみてください。そうするとフーリエ変換の結果としてでてきた c_{G_n} と c_{H_n} は周波数域のものと考えられます。スペクトル包絡である $H(\omega)$ は声道に依存します。声道は声を発する度に急激に形が変わるなどのことは起こりません。つまり変化が緩やかなもの、低周波数域のものと考えることが出来ます。また、音源である $G(\omega)$ は声の高さに依存するためこちらは急激に変化しうるものです。そのため変化が急なもの、高周波数域のものと考えることが出来ます。ここで注意していただきたいのは、c_n は時間域のものです。しかし、意味合い的には周波数域として読み取れるものと思ってください。

次にケプストラムという名前についてですが、これは深い意味はない造語です。スペクトル (Spectrum) の Spec を反転させてケプストラム (Cepstrum) に、ただコレだけです。ちなみにスペクトルでは横軸が周波数 (Frequency) ですがこれもケプストラムではケフレンシー (Quefrency) といい、スペクトルにかけるフィルタ (Filter) もケプストラムではリフタ (Lifter) と呼びます。

閑話休題。ではでは実際にある音声で c_n を計算し出力してみました。出力結果を図 1 に示します。

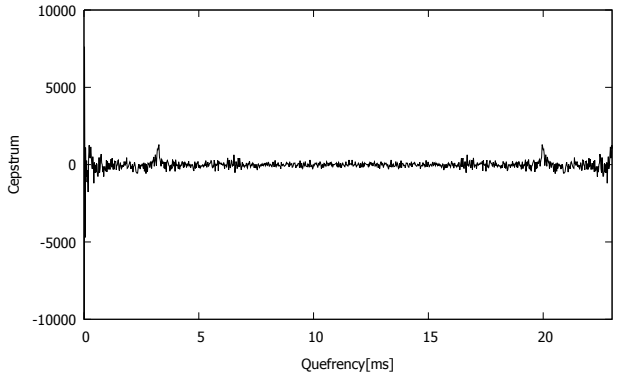


図 1: ケプストラムの結果

ケプストラムもフーリエ変換同様、左右対称の結果となります。前述した、この c_n を周波数域とみた時に低周波数域が声道フィルタ $H(\omega)$ に対応する c_{H_n}、高周波数域が音源 $G(\omega)$ に対応する c_{G_n} となります。実際には明示的に分離できませんが、ほぼそう解釈して良いものとなります。c_{H_n} だけを取り出すために c_n にリフタを掛けます。低周波数域だけを残すのでローパスになります、結果を図 2 に示します。

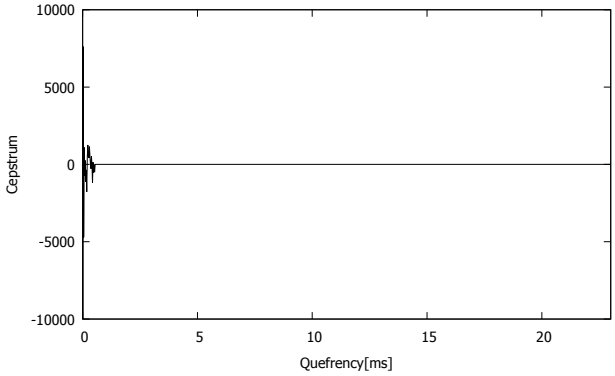


図 2: リフタを掛けた結果

低周波数域だけをのこした c_n に再度フーリエ変換をします。低周波数域と言っていますが c_n は本来時間域です。これで声道フィルタ $H(\omega)$ の周波数特性、ケプストラム包絡が導き出せます。先程のローパスのリフタを掛けた c_n をフーリエ変換した結果を図 3 に示します。

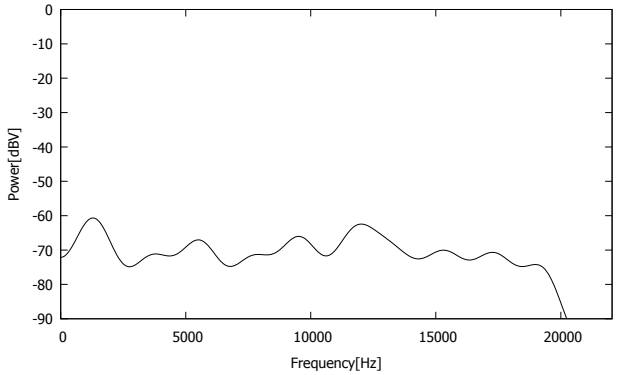


図 3: ケプストラム包絡

これでどの周波数域がどれだけ出ているかというのが出力できました。このケプストラム包絡ですが、ローパスのリフタでどれだけ低周波数域を残すかで包絡の数が変わります。この低周波数域を残す数を分析次数と言います。図 3 では分析次数を 24 でローパスのリフタを掛けました。分析次数を 46 でローパスのリフタを掛けたケプストラム包絡を図 4 に示します

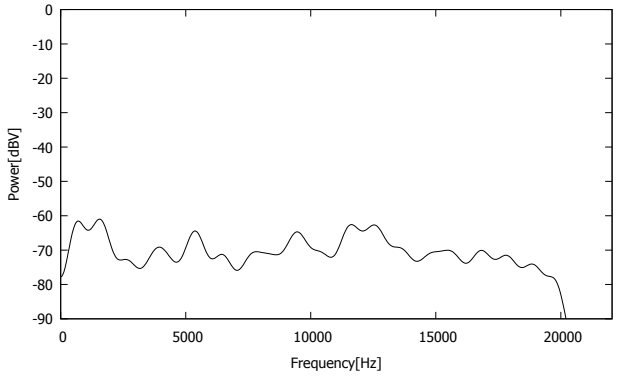


図 4: 分析次数 46 でのケプストラム包絡

図 4 から見れる通り包絡の数が増えました。この分析次数の数は、サンプリング周波数や何を分析したいかで変わります。音声の場合、フォルマントが約 1kHz に 1 つなのでサンプリング周波数から 1000 割った数を分析次数に。ですがこれに次数を 2

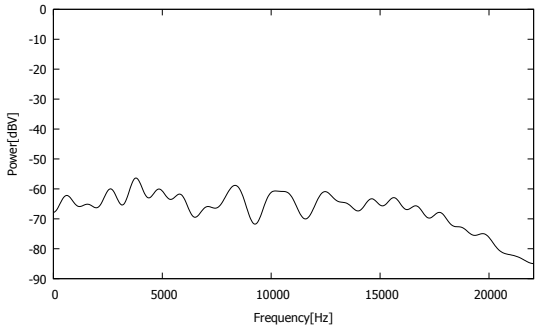
つ加えると良いらしいので結果的に分析次数は、$\frac{\text{サンプリング周波数}}{1000}+2$ となります。なぜ 2 加えるのかは、自分でもよくわかっていません。そうした方がより良い結果が出るという経験則からくる話だそうです。

3　実際に解析しよう

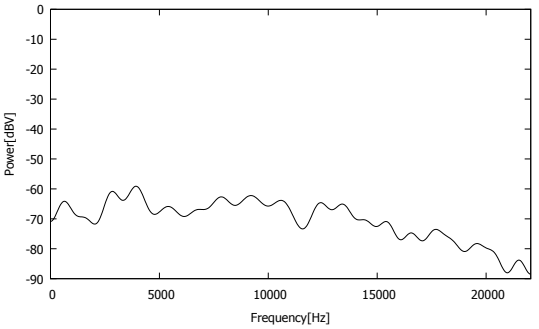
ここからが本命です、声優の声道解析に行きましょう。式 1 で示したように音声は音源と声道フィルタによって成り立ちます。声道フィルタをより詳細に解析・比較をしたいのならば音源を同じにしなければいけません。しかし、音源は様々な情報が含まれます。ピッチ周波数はもちろん、音韻も影響するかもしれません。じゃあ複数の声優が同じ音の高さ・音韻のものがあるの・・・・？ありました「THE IDOLM@STER CINDERELLA GIRLS 4thLIVE TriCastle Story - Starlight Castle-」[*1]です。そう、ソロリミックスという、うってつけのものがあります。歌であれば同じ音の高さ・音韻に限りなく寄せることが出来ます。バックグラウンドで流れる演奏も、同じフレームでの解析なら全ての音に等しく加わるだけなので比較への影響を最小限にできます。バックグラウンドの影響は最小限にできますが、それでもやはりバックグラウンドがない箇所での比較がしたいですよね。そこで今回はバックグラウンドが比較的ない箇所があった『Tulip』を解析の対象とします。

解析対象人物は LiPPS の五人、速水奏・塩見周子・城ヶ崎美嘉・宮本フレデリカ・一ノ瀬志希です。本来なら中の人の声道フィルタを解析すると断言したいですが、色々な役を演じている声優さんが、果たして役を変えても声道フィルタが変わらないのかという所まで調べきれてないので、今回は歌っているアイドルの声道フィルタということにします。解析箇所としては Tulip のラスサビの”なんてね”における”て”の部分のケプストラム包絡を見ていきたいと思います。分析次数は 44.1kHz サンプリングなので 46 で解析を行いました。解析結果を図 5 に示します。

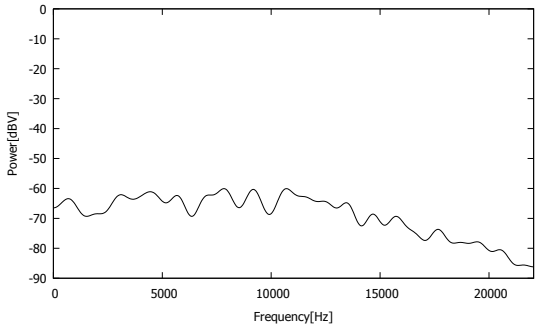
[*1] THE IDOLM@STER CINDERELLA GIRLS 4thLIVE TriCastle Story における神戸公演、StarLight Castle で発売となった CD。今回使用した『Tulip』の他に『Snow Wings』・『ハイファイ☆デイズ』が収録されている。また SSA(さいたまスーパーアリーナ) 公演では『絶対特権主張しますっ！』・『パステルピンクな恋』・『オルゴールの小箱』のソロリミックスが収録された「絶対ピンクな小箱」が発売された。



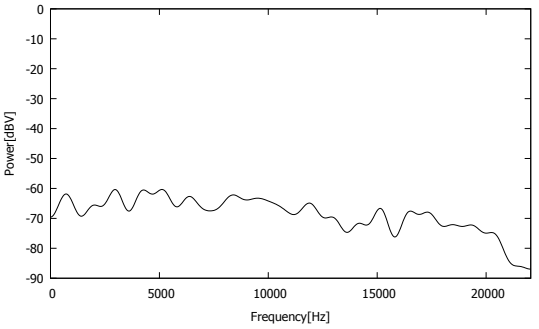
(a) 速水奏



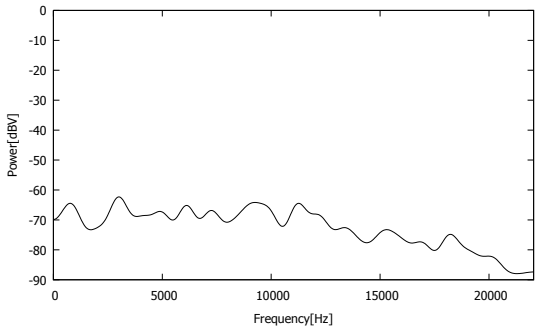
(b) 塩見周子



(c) 城ヶ崎美嘉



(d) 宮本フレデリカ



(e) 一ノ瀬志希

図 5: LiPPS のケプストラム包絡

似てる周波数域もあれば異なる周波数域もありますね。一番特徴があるとしたらフレデリカでしょうか、17kHz が LiPPS のメンバーと比べて減衰が少ないですね。このようにどの周波数域の強弱具合で個々人の声道というものが区別できます。

4　解析を踏まえて

今回はケプストラム分析で声道フィルタの違いというものを見ていきましたが、いかがでしたでしょうか。声優の声帯を解析したい～～っと冒頭で書きましたが、 今回は比較に視点を向けたので思ってたのと違うと思われたらゴメンナサイ許して。しかし、特定の人物のみの解析であれば「CINDERELLA MASTER」シリーズ[*2]のボーナストラックを使えば個人の声道フィルタの特徴がより詳しく得られるかと思います、是非解析して見てください。

ケプストラム分析ではスペクトル包絡が少しわかりにくいかもしれません。本当は LPC 分析でより詳しい声道フィルタの解析をと思いましたが時間が足りなかったので次に持ち越しとなります。次機会があるとしたら C92 と時間もあるのでそれまでに色々と手法を探っていきたいです。僕はライブ[*3]に行くんだ。

参考文献

[1] 荒井研究室「音源フィルタ理論 (Source-flter Theory)」, 2003, http://www.splab.net/acoustic-phonetics_demonstrations/G500/index-j.html ,2016 年 12 月 7 日閲覧

2016 年 12 月 7 日

[*2] アイドルマスター シンデレラガールズにおけるアイドルのシングル CD シリーズ。2016 年 12 月 7 日時点では No.45 まで発売中、No.48 までが発表となっている。

[*3] 2017 年 5 月からシンデレラガールズでツアーですよ同僚の皆さん。その前にミリオンの武道館を絶対行くぞ俺は勝つぞ。

MikuMikuDance のプラグインを作った話

おいがみ

1　はじめに

つい先日、Effekseer というエフェクト用ツールを直接 MikuMikuDance（以下 MMD と記す）で再生できるプラグインと保存された補間曲線を自動で挿入するプラグインを作成した。ここでは、その作成手法や思ったことについて書こうと思う。

2　MMDPlugin の作成

2.1　DLL インジェクションする

まずはじめに、プラグインを作るためには MMD の処理に割り込まなければならない。MMD にはそういった機能はないので、今回は DLL インジェクションという一般的な手法 [1] を用いて、割り込み処理を行った。対象の DLL は d3d9.dll で MikuMikuEffect（以下 MME と記す）でも同じ DLL を対象にしている。

実際の作り方に関しては「d3d9.dll インジェクション」と検索すれば詳細に出て来るのでそちらを参照してほしい。

2.2 生成するインスタンスを偽装する

前述した検索ワードで調べればこの部分に関しても書いてあるが、出来る限り C++ 準拠なコードを書くために、処理が異なっていたので興味があれば読んで貰いたい。

DLL インジェクションすることで任意のプログラムを実行できるようになったが、依然として MMD の描画部分にはアクセスできない。MMD の描画には DirectX9 を使用しているので、`Direct3DCreate9()` を呼び出して、IDirect3D9 インスタンスを生成しそこからデバイスを作り、描画を行っている。ゆえに `Direct3DCreate9()` で返すインスタンスを偽装すれば、描画処理が検知変更することができる。

IDirect3D9 は COM インターフェースという仕様に則って作られているがここでは細かい説明はせずに実装の話をする。また、C 言語で書くこともできるが、ここでは C++ として扱う。IDirect3D9 はすべて virtual 関数である。よって継承することで、処理を自由に変えることができる。そこで自分で IDirect3D9 を継承した MyDirect3D9 を作成し、virtual 関数をすべて実装する。しかし、実装する中身がわからないので、`Direct3DCreate9()` を呼び出し、帰ってきたインスタンスをそのまま呼び出す処理する中身のない実装になる。以下に擬似的なコードを示す。

```
1  class MyD3D9 : public IDirect3D9 {
2    IDirect3D9* base_d3d9;
3  public:
4    MyD3D9(UINT sdk_version) {
5      base_d3d9 = Direct3DCreate9(sdk_version);
6    }
7
8    HRESULT CreateDevice(UINT Adapter, D3DDEVTYPE DeviceType,
9                         HWND hFocusWindow, DWORD BehaviorFlags,
10                        D3DPRESENT_PARAMETERS*pPresentationParameters,
11                        IDirect3DDevice9** ppReturnedDeviceInterface
12                        ) override {
13     return base_d3d9->CreateDevice(Adapter, DeviceType, hFocusWindow,
14                                    BehaviorFlags,
15                                    pPresentationParameters,
16                                    ppReturnedDeviceInterface);
17   }
```

```
18
19    // このような感じで関数をすべて実装する
20    Result FuncXXX(Args... args) override {
21      return base_d3d9->FuncXXX(args...);
22    }
23    ...
24  };
```

こうしてできたクラスを MMD に返してあげることで、現在は中身のない実装になってる関数を書き換えることで、どのような処理でも追加できるようになる。描画は IDirect3DDevice9*が行うがこれも同様の手法で作ることができる。

しかし、実際にはこの手法には問題があり現在の実装では使われていない。このプラグイン単体で見れば、何も問題はないが MME と連携しようとするとエラーが発生してしまう。この実装の問題点は Direct3DCreate9() で生成したインスタンスとポインタが異なってしまうことにある。もちろん異なっても MMD が受け取るポインタ値は一つだけなので問題はないのだが、MME と連携しようとした場合、理由は分からないが何故か大本のポインタと継承後のポインタの 2 つが使われてしまう。もともとポインタ値でチェックしているのが問題なのだが、D3DX 関数の内部でそのような処理を行っているらしく、仕方なく現在は vtable を書き換える実装になっている。

2.3 実際に機能を追加する前に

ここまで読んで思うと思うが、とても面倒である。誰が作っても大体同じになり、わざわざプラグインを作るためにここまで準備をするのは作る気力を阻害させる。また、ほとんどのプラグイン d3d9.dll を書き換えているので競合する可能性や別のプラグインの面倒も見る必要がある。こういった事をなくすためにまずプラグインを読み込むプラグインを作成することにした。それが MMDPlugin である。

プラグインとして必要な機能はまず割り込み処理ができることである。そこで、IDirect3DDevice9 が持つ計 119 個の関数を再定義し、割り込みしたい関数だけを定義したクラスを作るだけで処理を割り込めるようにした。この割り込みでは既存の MMD の処理をキャンセルや書き換えをすることはできない。なぜなら、今後追

加されるだろうプラグインに大きな影響を与えてしまうためだ。しかし今となってはこの仕様を後悔している。そもそも既存の処理にないことをするので、何らかの影響を与えてしまうのは当然で、この仕様はただ無意味に制限を課しているだけになってしまっている。

かくして MMDPluginDLL1 が作られた。余談だが、MMAccel というプラグインが昔からある。そのプラグインが MMDPlugin に対応してもらい、今でもこのバージョン 1 が MMAccel に使われている。最も、描画をするプラグインではないため、用意してある割り込み処理は一切使われていないのだが。

3 EffekseerForMMD の作成

EffekseerForMMD はオープンソースプロジェクトなので、興味があるひとは見てもらいたい。

`oigami/EffekseerForMMD ( https://github.com/oigami/EffekseerForMMD )`

3.1 Effekseer を画面に描画させる

ようやく割り込む準備が整ったので機能を追加していく。ここではそんなに書くこともなく、ただ Effekseer ライブラリを呼ぶだけで良い。ただ注意点として、デバイスロスト処理がある。`Renderer` だけでなく `Effekseer::Effect` もリソースを開放する必要がある。また、デバイスロストを割り込ませる処理も問題だった。現在用意されている割り込み用関数は既存の処理が呼ばれる前に割り込みが入る。よって、`Reset()` が呼ばれる前にプラグインの `plugin->Reset()` が呼ばれる。このタイミングでリソースを解放するのは大丈夫だが、問題は復帰処理である。`Reset()` 後には再度リソースを読み込まなければならない。しかし、当時そのような既存の処理をしたあとに割り込む関数は用意されておらず、実現するにはすべての割り込み用関数の先頭にもし `plugin->Reset()` がされていたら、リソースを読み込むという処理を書くしかなかった。しかし、これもよく考えると分かるがうまくいかない。この理由は単純で、`Reset()` が実行されている途中で別の関数を呼び、割り込み用関数が呼ばれてしまうからである。

MMDPluginDll1 ではこのように不自由な部分があった。そこで、すべての割り込み用関数に PostXXX() という既存の処理を実行した後に呼ばれる関数を定義した。これにより、plugin->Reset();Reset();plugin->PostReset(); という直感的で楽な処理ができるようになった。

また、Effekseer は基本的に D3DDevice の設定を保存して描画終了時にもとに戻してくれるが、一部戻さない物があるので下記コードのように自前で処理する必要がある。

```
1  // 現在の状態を保存
2  float constant_data[256 * 4]; // 要素数は適当
3  device->GetVertexShaderConstantF(0, constant_data,
4                                   sizeof(constant_data)
5                                     / sizeof(float) / 4);
6  UINT stride;
7  IDirect3DVertexBuffer9* stream_data;
8  UINT offset;
9  device->GetStreamSource(0, &stream_data, &offset, &stride);
10 IDirect3DIndexBuffer9* index_data;
11 device->GetIndices(&index_data);
12 if ( g_renderer->BeginRendering() )
13   // 描画処理
14   ...
15   g_renderer->EndRendering();
16 }
17 // 保存した状態をセット
18 device->SetStreamSource(0, stream_data, offset, stride);
19 if ( stream_data ) stream_data->Release();
20 device->SetIndices(index_data);
21 if ( index_data ) index_data->Release();
22 device->SetVertexShaderConstantF(0, constant_data,
23                                  sizeof(constant_data)
24                                    / sizeof(float) / 4);
```

3.2 MMD との連携

描画ができたので、実際に MMD との連携処理を書いていく。最低限の機能として efk ファイルを読み込めなければならない。MMD ではメニューの「開く」とドラッグアンドドロップの 2 種類のファイル読み込み方法があるが、使いやすそうな

ドラッグアンドドロップの方を対応することにした。MMDPlugin は描画周りの割り込みはできるが、その他の割り込みは対応していなかった。そこで Win32 API をフックする関数を作成した。といっても調べたら出てくるので貼り付けただけである。そしてドラッグアンドドロップで呼ばれる関数である `DragQueryFileW()` をフックした。MMD はファイル名に末尾問わずどこかに「.pmd」と入っているものを pmd ファイルとして認識するので、誤検出防止のために、「.pmd.efk」という文字列を `DragQueryFileW()` で得られた文字列の後ろに付け加えることで pmd ファイルに偽装させた。しかし、このままでは存在しないファイルを MMD が pmd ファイルとして読み込もうとしてエラーになるだけである。そこで次に、`CreateFileW()` をフックする。この関数は名前とは裏腹にファイルの読み込みにも使用する。この関数でファイルを開くとき、末尾に「.pmd.efk」がついていたら 99.9% 間違いなくドラッグアンドドロップで読み込まれた efk ファイルである。よって、この条件を満たした時、予め用意しておいた pmd ファイルを開くことで MMD 側では通常の pmd ファイルを読み込んでるとして処理される。肝心の efk ファイルを読み込んでいないが、それは描画時に行う。MMD にプラグイン機構はないが、幾つかプラグイン用に関数が用意されている。その中にファイル名を取得できる関数がある。そこには当然「.pmd.efk」がついているので、まだ読み込んでいない場合読み込みをする。また、「ファイル名」、「MMD 内部の PMD に割り振られた ID」、「現在描画しようとしているオブジェクト番号」が対応付けされていて、関数で確認することができるので、その情報をもとにどのエフェクトを描画するかを選択することができる。

4 あとがき

今回は MMDPLugin と EffekseerForMMD について書いたが、次回書くときは MMDUtility のことを書こうと思う。現在 MMDUtility には「補間曲線パレット」と「画面分割」の 2 つ機能があるが、補間曲線パレットを実装するにあたってのメモリサーチ手法と逆アセンブル手法を紹介したいと思っている。

MMD のプラグインを作るのが大変だと感じたかもしれないが、MMDPlugin が割り込み処理をほとんど担っているので、これから作る際にはとても楽だと思う。現に私はすでに 3 つ機能をついかしたが大変だと思ったのは初回だけだった。もし、興

味を持った人がいたら、ぜひとも作ってみてもらいたい。そして MMD の発展に繋がったら、とても嬉しく思う。

参考文献

[1] VPVP wiki - MMD 周辺ライブラリ
(https://www6.atwiki.jp/vpvpwiki/pages/288.html#id_c428f21b)

twitter: @oigami013

2016 年 12 月 8 日

dotPeek を用いた逆コンパイル

pandemo

1　目的

C#で作成された EXE ファイルを逆コンパイラの dotPeek を用いて作り直す。

2　方法

1. C#で作成された EXE ファイルを用意する。今回はボタンを押すと画像が表示されるだけの簡単なものを自作した。

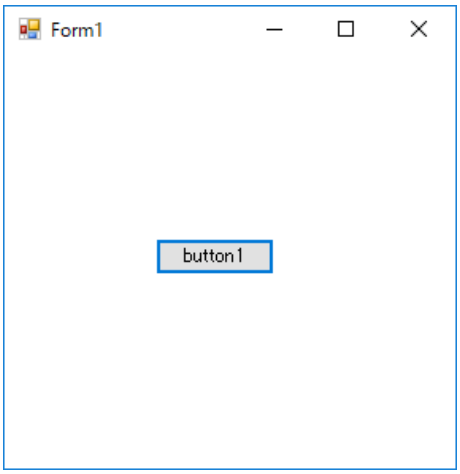


図 1: 起動時

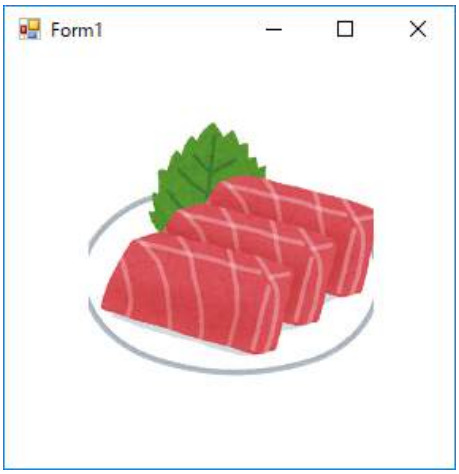


図 2: ボタンを押した後

2. dotPeek をインストールし、1 で用意した EXE ファイルを開く

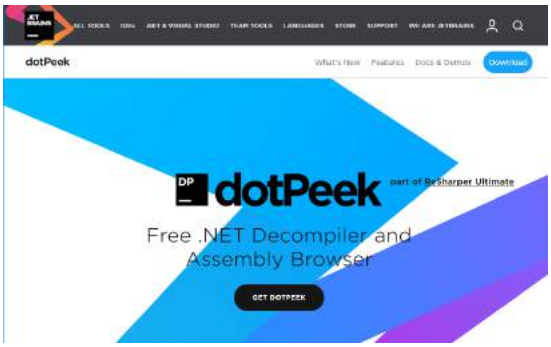


図 3: dotPeek ホームページ

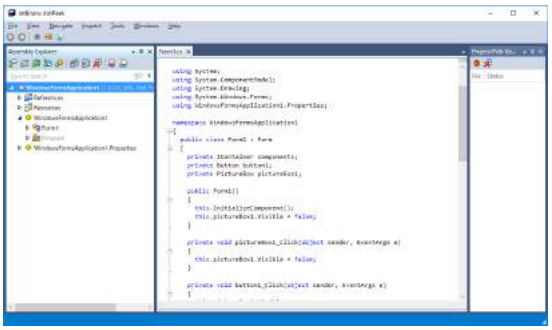


図 4: dotPeek 実行画面

3. 表示されたフォルダを右クリックし、「Export to Project」を押し、「Open project in Visual Studio」にチェックを入れ「Export」。

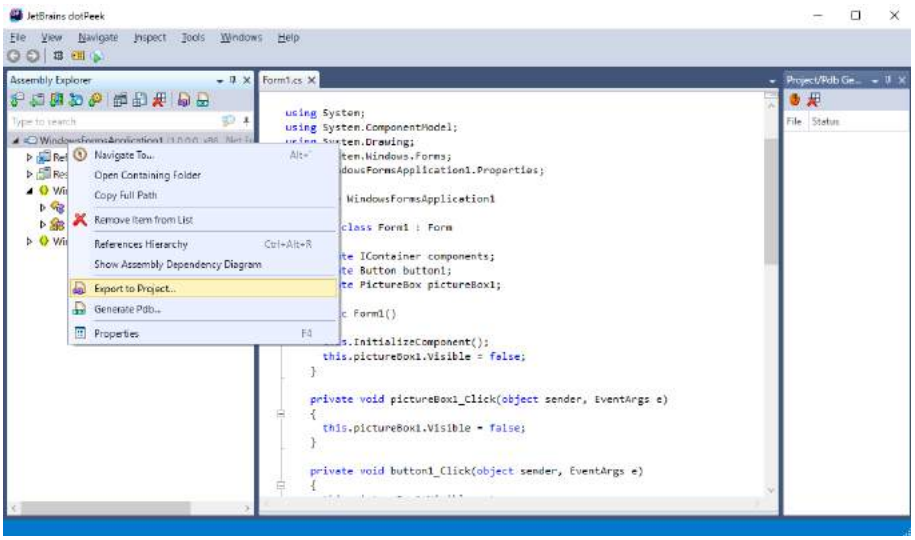


図 5: dotPeek 実行画面

4. sln ファイルが出力され、Visual Studio が開くのでコードを書き換えたり、画像を差し替えたりする。

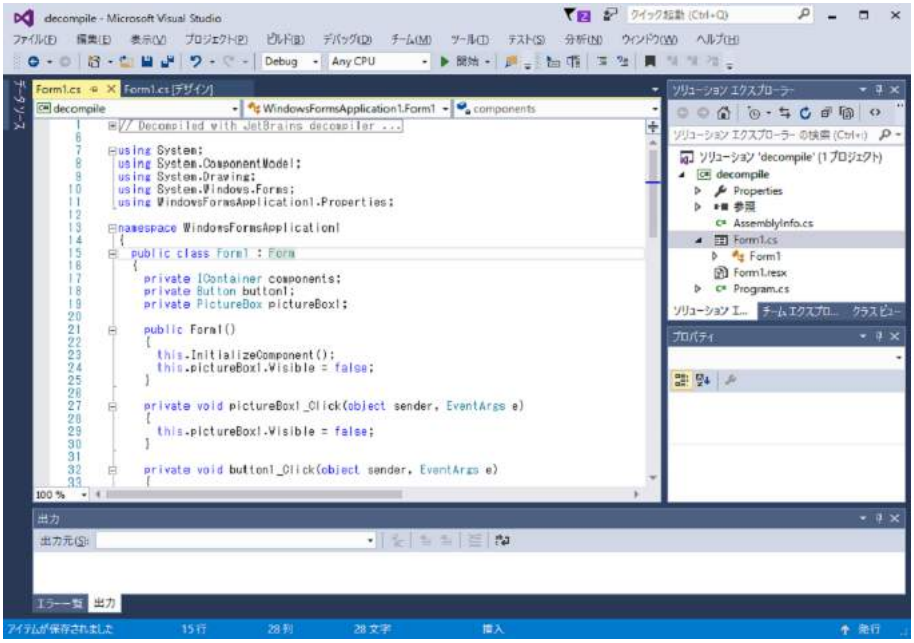


図 6: Visual Studio 実行画面

5. リリースビルドをし、EXE ファイルを出力して完成。

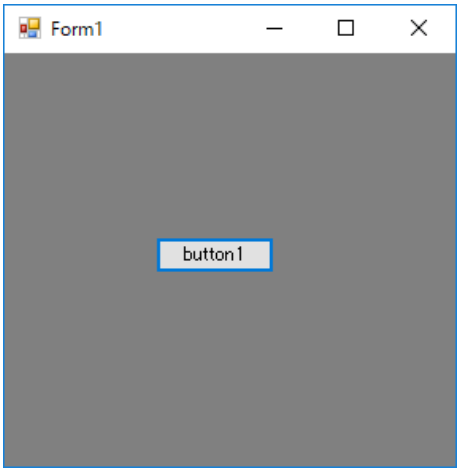


図 7: 起動時

図 8: ボタンを押したあと

3 おわりに

dotPeek では逆コンパイルした結果を Visual Studio のプロジェクトとして保存することができるため他の逆コンパイラよりも使いやすく感じた。また、ソースコードを公開しているアプリケーションを試したところ、かなりオリジナルに近い形が出てきたので、ソースコードを外部に知られたくない場合はコードの難読化などを行い、解析対策を施す必要があるだろう。

2016 年 12 月 07 日

Java の Deflater 等で PNG 画像を作ってみる

yamacken

1　はじめに

PNG は圧縮に Deflate という可逆圧縮アルゴリズムを使っています。そして Java では Deflater というクラスで Deflate 圧縮が標準で利用できるようなっています。

今回はこのクラスを利用して PNG の画素データを圧縮して、保存したものが PNG として画像ビューワソフトに読み込んでもらえるか確かめてみます。

1.1　CRC32

PNG には CRC という手法の誤り検出符号をつける必要があり、正しくないと破損しているファイルとみなされてしまいます。Java には CRC32 という名前のクラスもあり、PNG の 32bit の CRC がこのクラスで計算できるのか確認します。

1.2　java.util.zip

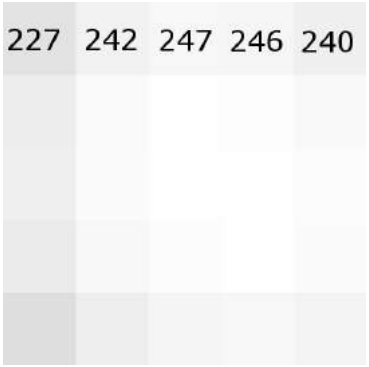
Deflater や Inflater（後述）、CRC32 のクラスは標準クラスライブラリの `java.util.zip` というパッケージ内に存在しています。一般的な圧縮形式の `.zip` 形式ファイルは圧縮メソッドのひとつとして Deflate を用いています。

2 PNG の構造

今回は Deflater と CRC32 のクラスの機能で PNG が作成できるか、が目的のため詳細は省きます。PNG の仕様書が一番詳しくファイルフォーマットの解説をしています。

3 まずは読み込んでみる

作るとはいったもののせっかく世の中には PNG 画像を扱えるアプリケーションが無数にあるので、まずはそれらで保存した PNG 画像の Deflate 圧縮された画素データを見てみましょう。用意した画像は GIMP2.8.16 で作成した以下のような 5 × 5 のものです。色情報はカラーですが、簡単のため RGB ともに同じ値にしてあります。一行目の画素にだけ RGB の値を記しました。



Inflate

Deflate で可逆圧縮されたデータを復元する操作は zlib で Inflate と名付けられています。そして Java では Inflater というクラスで利用できます。

3.1　Inflater で復元

画素データ

復元したデータを見る前に、元々の画素データはどう並んでいたか見ましょう。

行の先頭	画素の列の数							行の先頭	画素の列の数			
フィルタ方式	RGB 情報			RGB 情報			…	フィルタ方式	RGB 情報			…
?	R	G	B	R	G	B	…	?	R	G	B	…

復元したデータがこういう並びなら成功したといえるでしょう。フィルタ方式の byte が 1 のとき、RGB のデータは一つ左の画素の RGB との差だ、とだけ今は覚えておきましょう。

復元したデータ

GIMP で作成した画像の、画素データが含まれる箇所を byte 配列として Inflater（のインスタンス）に渡し、復元させた結果をみてみましょう。

```
1  68[8, -41, 5, -63, -63, 13, 0, 32, 8, 3, 64, 104, 49, 49, 124, -36 …
2  80[1, -29, -29, -29, 15, 15, 15, 5, 5, 5, -1, -1, -1, -6, -6, -6 …
```

一行目が復元前、二行目が復元後の画素データの byte 配列の中身です。配列の要素数を先頭に、次に `Arrays.toString()` で配列の中身を表示させました。

二行目についてみていきましょう。一つ目の byte はその行のフィルタ方式を表していました。この場合は 1 です。この RGB の並びと、実際の画像の色情報を見比べると、たしかに元の画像の色が復元できそうです。

要素数を比べると、損失なくデータ量を圧縮できていたことがわかります。

3.2　CRC32 を計算

今度は誤り検出符号を CRC32 クラスに計算させてみましょう。そのまえに、今更ですが PNG がチャンクという塊で画像の情報を保持していることを説明します。

チャンク

n (データ量)				チャンクタイプ				データ (n byte)								CRC			
4 byte				4 byte				n byte								4 byte			
0	1	2	3	0	1	2	3	0	1	2	3	4	…	…	n	0	1	2	3

チャンクタイプの部分で、このチャンクというデータブロックに何の情報が入っているのかを表します。これには様々な種類があり、画素データを保持するチャンクもその一つです。CRC の計算はチャンクタイプによらずみんな同じに、チャンクタイプの 4byte とデータの n byte を並べた、この表でいえば真ん中の二つをそのまま抜き出したような、連続した byte 列に対して CRC の計算をすることで決定します。

Java の CRC32 クラスでは、チャンクタイプとデータを byte 配列にして渡せば準備完了です。データを与えたインスタンスの`.getvalue()` で CRC を計算して返してくれます。

実行結果をここに書くことはしませんが、チャンクの最後尾の CRC の値と同じものが返ってきます。値が違ったときは、データに誤りがあったと判断できます。

4 PNG 画像を作る

ここではチャンクのデータをどう作っていくのかを書きます。例として、画素データのチャンクを作っていきましょう。

データをつくる

まずどのチャンクタイプなのか、そしてそのチャンクのデータを決定する必要があります。この例での画素データは、読み込みの時に使った 5 × 5 の画像を構成するようなものを目指したいと思います。フィルタ方式については 0、フィルタリングせず RGB をそのまま入れる方式をすべての行で行うこととします。そのときの byte 配列は、以下のようになります。

```
1 [0, -29, -29, -29, -14, -14, -14, -9, -9, -9, -10, -10 …
```

このデータ配列は、もちろんチャンクタイプごとにそれぞれ異なります。PNG の仕様に合わせて正しく構成しなければいけません。

データを圧縮する必要のあるチャンクタイプでは Deflate 圧縮してからチャンクのデータの部分としなければいけません。画素データは圧縮するチャンクでした。データ配列はできているので、Deflater に渡して圧縮した配列を作成します。本当なら Deflate するデータは、一度に圧縮する範囲が決まっていたりしますが、詳しくは仕様書等を読みましょう。

CRC は前述の通りの計算なので、作成した（かつ圧縮した）データがあれば決定できます。

データ量 n はチャンクのデータ配列の（圧縮した後の）長さを入れます。

こうして画像に必要なすべての情報をチャンクとして作り上げ、ファイルに書き込めば、ビューワで読めるはずです。ただしチャンクが必須であるかや、チャンクの並び順などにも決まりがあります。

4.1　読み込めたのか

結果として、期待した画像がビューワに表示されました。Deflater と CRC32 を使って自力で PNG 画像が作れたということです。

win10

win10 標準の画像ビューワの「フォト」は CRC が誤っていても気にせず表示するので、CRC をチェックしていないようです。

5　おわりに

バイナリファイルの読み書きの練習としてやったことをまとめてみました。

2016 年 12 月 7 日

クソザコが CTF やる

うろん

1　はじめに

CTF という言葉をご存じでしょうか？

CTF とは Capture The Flag の略で、直訳すると「その旗を取れ!」とわけがわからいですが。コンピュータセキュリティ技術の競技であります。ルールは出題されるデータからフラッグを見つけてそれ見つける競技です。出題されるデータにはさまざまなものがあり、簡単な問題では暗号を解読するものだったり、音声データだったりします。今回は、私がそこまで問題を解いていない初心者なので、今まで解いた問題で使ったツールについて書いていきたいと思います。

2　どのサイトの CTF を解いたか？

ksnctf　http://ksnctf.sweetduet.info/

akictf http://ctf.katsudon.org/

これら 2 つの常設 CTF サイトの問題を解きました。

3　今まで解いた問題で使ったおもなツール

3.1　WireShark

通信のパケットを解析できる便利なツール。最近、日本語化したやつにアップデートしました。さらにプロトコル別に色を分けてくれて、ユーザに優しい。

3.2　Bz と Stirling

ファイルのバイナリを解析できるツール。どちらか使うかは自由。(ちなみに私は状況に合わせて使ってます)

3.3　Tera Term

SSH クライアント。ほかのやつつかったことないです。GoogleChrome で SSH クライアントのアプリがあるらしいですがよくわからないです。

3.4　Visual Studio 2015

ご存じマイクロソフト製の IDE です。

4　まとめ

ほかにも、いろいろ使いましたが、問題をみたらバイナリを見るようにしています。これからも頑張っていきたいです。

うろん　2016 年 12 月 06 日

フルレンジスピーカーのエンクロージャーの設計と製作

ミンクス

1　はじめに

共立エレショップでスピーカーユニットを見ていてなんとなく「作ってみたい」と思い、スピーカーのエンクロージャーの設計と製作しました。特によく調べずに設計して作成したために、あまり納得のいくものになりませんでしたがせっかくなので記事にしようと思いました。

2　設計段階

今回のエンクロージャーの作成で設計するときに考えた要項は以下のとおり

- 小さいユニットでなるべく広いレンジで音が出る
- でも容量が多いほうが低音出そうだし箱は大きめで
- バスレフがメジャーらしいしバスレフでいこう

といった感じで「ぼくのかんがえたすごい○○」のような調子で設計していきました。また、具体的な数値には”http://www.asahi-net.or.jp/~ ab6s-med/NORTH/SP/bassreff.htm”の設計ソフトを用いた。スピーカーユニットにはFOSTEX の「FF85WK」を用いることにしました。理由は最低共振周波数 f_0 が他

の 8cm ユニットに比べ低めだったからでした。

こうしてエンクロージャーの寸法を決定し、材料調達を始めました。

図 1: FF85WK のスピーカーユニット

2.1　作成

エンクロージャーの材木は加工しやすいと言われてる MDF 材を使うことにしました。ダクトには FOSTEX が販売している P49P を用いりました。他にはハタガネ、木工用ボンド、木ビスなどを購入しました。また、木材の指定するサイズを間違えてしまい、一部の部分が別の材木になってしまいました。パネルの穴あけには友人に協力してもらいました。

スピーカーを組み立て始め、まず接合面周りの部分にマスキングテープで覆い、はみ出したボンドを拭き取れるようにしました。その後、ハタガネをで固定し一日置いておきました。

図 2: 予算の都合でハタガネのサイズが…

さらに中に吸音材をはりました。吸音材を吸音材を張り詰める意味は、中音や高音を吸収してダクトからノイズとして出てくるのを軽減することが狙いです。これに背面のパネルを取り付けて、スピーカーとダクトなどを取り付け、木ビスで補強して完成です。

図 3: 量や貼り方についてはあまり考えていない

図 4: これで完成

3　評価

実際にならしてみたところ、あまり低音に関しては出てないように感じました。また、遠くに 3M ほど離れるとベースの音が聞こえなくなってしまいました。

これを音響大好きな顧問の先生の元に持っていくと、「ダクトを短くしてユニットの f_0 辺りを肉厚して低音出てるように錯覚させるといいかもね。あとアンプのパワーが高くないと大きい箱は響かせられないよ」ということを言われました。また、音響関連の先生に見せたところ「ダクトを塞いでみては？」と言われました。

更に、大学の音響実験室をお借りして周波数特性を計測しました。

図 5: 無響室、置いてあるスピーカーは研究室のもの

周波数特性は以下のようになった.

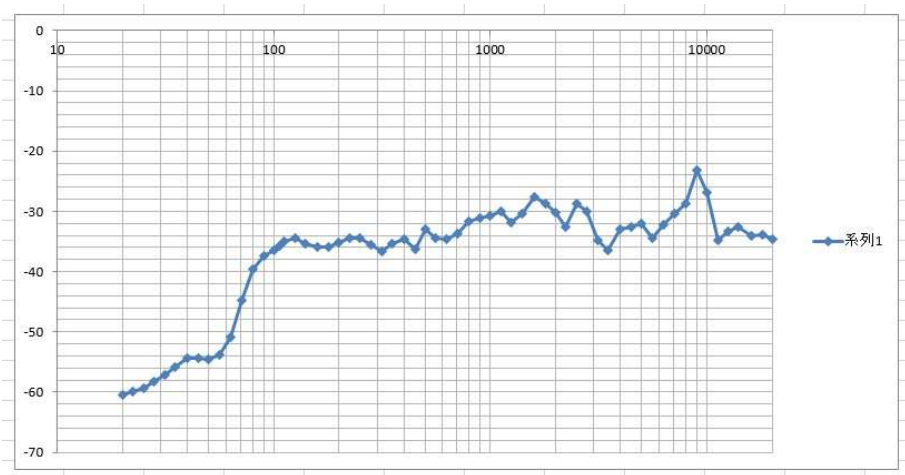


図 6: 周波数特性グラフ

4　まとめ

今回そもそもバスレフのしくみを全く調べず製作し、この記事の原稿を書いてるときに知るという愚行を犯してしまいました。音響大好きな顧問の先生に「これからいっぱい作ろうな」って言われたり、今回あまり納得のいく出来でなかったので、また作ろうと思います。

ミンクス

2016 年 12 月 7 日

Brainfuck で文字列を出力するソースコード

しまどり

1　はじめに

みんな大好き Brainfuck！... ですが文字列を出力するだけでも大変で困っている方も多いのではないでしょうか。そこで私は Brainfuck で文字列を表示するソースコードを出力するプログラムを C++ で書いてみました。

2　Brainfuck とは

開発者 Urban Mller がコンパイラがなるべく小さくなる言語として考案した。
Brainfuck プログラムは、以下の 8 個の実行可能な命令から成る。

- 「>」ポインタを一つ進める（インクリメントする）。
- 「<」ポインタを一つ戻す（デクリメントする）。
- 「.」現在ポインタが指している値を出力する。
- 「+」現在ポインタが指している値を 1 増やす（インクリメントする）。
- 「-」現在ポインタが指している値を 1 減らす（デクリメントする）。
- 「,」1 バイトを入力し、現在ポインタが指している値に代入する。
- 「[」現在ポインタが指している値が 0 であれば対応する「]」にジャンプする。

- 「]」現在ポインタが指している値が 0 でなければ対応する「[」にジャンプする。

3 プログラム

プログラムをいくつか書いてみて基本である HelloWorld を表示するコードを出力し、比較していくことにしてみました。

3.1 単純式

```
#include <iostream>
#include <string>
using namespace std;
int main() {
        string s;
        int ptr=0;
        cin >> s;
        for (auto&& a : s) {
                while (ptr != (int)a) {
                        if (ptr < (int)a) {
                                cout << "+";
                                ptr++;
                        }
                        else {
                                cout << "-";
                                ptr--;
                        }
                }
                cout << ".";
        }
        return 0;
}
```

HelloWorld の出力です。

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++.++++++++++++++++++++++++++++++.+++++++..+++.--------
----------------.++++++++++++++++++++++++.+++.------.--------.
```

最初は単純にポインタが指す値を増やしたり減らしたりして出力するプログラムにしてみました。言うまでもなくクソコードですね。原因は 1 つのポインタしか使っていないからです。非効率極まりないです。

複数のポインタを使ってみることにしましょう。ループも組み込んでみることにします。

3.2 改良型

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #define N 12
5  using namespace std;
6  int main() {
7          string s;
8          vector<int> p_num;
9          vector<int> ptr;
10         bool flag;
11         int c_num;
12         int ptr_now = 0;
13         cin >> s;
14
15         //グループ分け
16         for (auto&& n : s) {
17                 flag = 0;
18                 for (auto&& p : p_num) {
19                         if ((int)n/N == p) {
20                                 flag = 1;
21                                 break;
22                         }
23                 }
24                 if (flag == 0) {
25                         p_num.push_back((int)n / N);
26                         ptr.push_back(0);
27                 }
28         }
29
30         //ループ部分
31         for (int i = 0; i < N; i++) {
32                 cout << "+";
```

```
33          }
34          cout << "[";
35          for (int i = 0; i < ptr.size();i++) {
36                  cout << ">";
37                  for (int j = 0; j < p_num[i];j++) {
38                          cout << "+";
39                          ptr[i]++;
40                  }
41          }
42          for (auto && p : ptr) {
43                  p *= N;
44                  cout << "<";
45          }
46          cout << "-]>";
47
48          for (auto && n : s) {
49                  for (int i = 0; i < p_num.size();i++) {
50                          if ((int)n / N == p_num[i]) {
51                                  c_num = i;
52                                  break;
53                          }
54                  }
55                  while (ptr_now != c_num) {
56                          if (ptr_now < c_num) {
57                                  cout << ">";
58                                  ptr_now++;
59                          }
60                          else {
61                                  cout << "<";
62                                  ptr_now--;
63                          }
64                  }
65                  while (ptr[c_num] != (int)n) {
66                          if (ptr[c_num] < (int)n) {
67                                  cout << "+";
68                                  ptr[c_num]++;
69                          }
70                          else {
71                                  cout << "-";
72                                  ptr[c_num]--;
73                          }
74                  }
75                  cout << ".";
```

```
76          }
77          return 0;
78  }
```

HelloWorld の出力

```
1  ++++++++++++[>++++++>++++++++>+++++++++>+++++++
2  <<<<-]>.>+++++.>..+++.>+++.<.+++.-------.<-.
```

解説をしますと、文字コードが近いもので分けたグループを作り、それぞれに対応した値をとるポインタを作る、そして出力したい文字コードに近いポインタの値を上下させて出力させていきます。

具体的には HelloWorld の文字コードを 12 で割ると H が 6、W が 7、e と d が 8、その他は 9 になります。6、7、8、9 の 4 つのグループを作って、対応するグループのポインタで出力していくわけです。12 はどこから来たかと言いますと、一通り試してみて 12 のときに HelloWorld のコードが一番短くなっただけです。

もちろんまだまだ改良の余地はありますが、最初と比べてかなり短くなりました。

4 おわりに

Brainfuck は最低限の 8 個の命令を組み合わせる言語です。最低限の命令ということは縛られるものがなく自由度が高いということです。HelloWorld を出力するだけでも様々な方法があり、どれだけコードを短くできるかなどの楽しみ方などがあります。

正直に言ってこのプログラムはネタで作り始めましたが、なんだかんだで楽しかったです。パズルみたいなものなので意外と作ってみると面白いものです。実用性は皆無ですがぜひやってみて下さい。楽しい！

2016 年 12 月 07 日

ボタンはまりには気を付けよう！

トロマグロドラゴン

1　はじめに

突然ですが私今年の 8 月あたりから beatmania っていう DJ シュミレーションゲーム（音ゲー）にはまりまして、生活費をどっぷり捧げてます。これ過去に家庭版がでていて家庭用コントローラーが存在しているんですけど（これ以降は専コンと呼びます）これ結構ゲーセンのやつといろいろ違ってて（後述）改造とかできるんです。

今回は専コンのボタンはまりを防止する改造をしたのでそれについて話します。

2　本文

2.1　専コンの仕組み

専コンの仕組みはアーケードコントローラーと違ってバネが入ってません。ボタンを押してボタンの底（？）についてる金属部分が基盤に触れることで反応します。単純すぎて専コン開いたときめっちゃびっくりしました。

2.2 ボタンはまりには気をつけよう！

個人的な話をするんですけど僕高校生時代に「恋と選挙とチョコレート」(PSP 版) にはまっててずーーーっとやってたんですけど 2 √終わったあたりで O ボタンが埋まってました。ゲームのボタンってのはすぐはまるんです（これがいいたかった)。

2.3 準備するものとやること

やりかたとかは調べればすぐでるので実際にやる方はそっち参考にしたほうがいいと思います (諸事情) 必要なものは、カッターナイフ、ハサミ、あとゴム板 (2cm)、ティッシュの箱、両面テープです。ゴム板は 5mm × 5mm に切って基盤にはりつけてティッシュ箱は 1cm × 1cm に切ってボタンの底に貼り付けます。これで基盤とボタンが底上げできてボタンがはまりにくくなります。これで終わり。

2.4 やってみて

カットとかは割と雑にやっても平気でした。あとよくよく考えればティッシュ箱貼るのとゴム板貼るのは両方やらないとダメです。私はこれで時間とられて何度もネジ回す羽目になりました、、、

3 おわりに

普段は一切ネジとかで何かする人間でもないので雑な解説しかできないのは許してください。最近音声解析とか興味あるんですけど今とってる講義でソフトつかうとこまでいってないので次にでも。

twitter: @torotoroleen

2016 年 12 月 7 日

Siv3D と Unity でゲームを作ってみたー

bay leaf

1　はじめに

こんにちは。このトピックでは、私がこの春から本格的に C を勉強し、Siv3D と Unity でゲームを作ってみた経験を、皆さんとの情報共有として紹介しています。ゲーム製作や、プログラミングに興味ある方は是非ご一読ください。

2　ゲーム製作にあたって

Visual Studio という統合開発環境のライブラリである”Siv3D”を使ってゲームを作り、その後”Unity”で同じゲームを作ろうとした際、作り方に差があったので、今後ゲーム作る際にはどちらがいいのかを検討していきます。先に結論を言うと、Siv3D はプログラマー向き、対して Unity はデザイナー向きです。

3　プログラマー向け、Siv3D

先ず Visual Studio で 0 からゲームを作るなら、基本は C++ の勉強から始めましょう。簡単な関数等が使えるようになってから、リファレンスを見て、ゲーム画面を作り始めるといいです。（クラスやポインタが分かるようになると作りやすくなる）

私はシューティングゲームが作りたかったので、自機、弾、障害物などのオブジェクトを先に設置、その後座標を 1 フレームごとに計算させ、移動させました。あたり判定は `object.intersects(shot)`(object は障害物、shot は弾のオブジェクト) で簡単に取得できるので、条件を if で繋げ、デバッグしながら動かして確認していました。常にフローチャートを意識し、このボタンを押したらこうなり、次にああなる、というのを確認しながらゲームシーンを製作しました。Siv3D の欠点（というかプログラム全般に言えますが)、言ったことしかやってくれません。しかし、裏を返せば言ったことはやりますし、それ以上の事はしません。バグが発生したらそれは 100 %自分のせいです。

実際のゲームシーン製作画面

```
# include <Siv3D.hpp>

void Main() {
        int playerx = Window::Width() / 2; //x座標
        int shotx = -10, shoty = -10;
        bool shoot = 0;  // 弾を発射しているかどうか
        int speed = 7; //弾速

        while (System::Update())
        {
                Circle player(playerx, 440, 30); // プレイヤー
                Circle shot(shotx, shoty, 10); // 弾

                if (Input::KeyRight.pressed &&
                Window::Width() - 30 >= playerx)
                {
                        playerx += 3; //右移動
                }
                if (Input::KeyLeft.pressed && playerx >= 30)
                {
                        playerx -= 3; //左移動
                }
                if (Input::KeySpace.clicked && shoot != 1)
                        //スペースで発射
                {
                        shoot = 1; //弾発射
                        //fire.playMulti(0.4); (発射時の音声ファイル)
```

```
28
29                  }
30
31                  if (shoot == 0) //位置同期
32                  {
33                          shotx = player.x;
34                          shoty = player.y;
35
36                  }
37                  else {
38                          shoty -= speed; //射出
39                  }
40
41                  player.draw();
42                  shot.draw(shoot ? Palette::White : Palette::Black);
43                  if (shoty < 0) {
44                          shoot = 0;
45                  }
46          }
47  }
```

このコードでは、最低限自機を動かして遊ぶ事が出来ます。ここに、オブジェクトを追加してあたり判定を付ければ、ゲームがほぼ完成します。

C++ を使うこのやり方では、ゲームシーンの殆どは条件式で成り立っているので、フローチャートを上手く考えながら作る必要があります。

4　デザイナー向け、Unity

Unity で 0 からゲームシーンを作るなら、先ずインターフェイスに慣れなければなりません。どこでオブジェクトを作り、どこでアニメーションを付けて、どこでスクリプト書いてオブジェクトを動かすか・・・。プログラム経験しかなかった私としては、少してこずった場面でもあります。

シューティングゲームを作るには、オブジェクトを作るまでは Siv3D と同じですが、オブジェクトの移動、衝突判定等はオブジェクト自体に組み込むので、大筋のフローチャートがありません。ここが私にはとっかかりにくい所でした。これは、

Unity がサポートしている言語が C#や JavaScript 等のオブジェクト指向を扱う物だからです。全てを条件的に判断して進めていくことはできませんが、Unity はインターフェイスにさえ慣れれば、やりたい事をすぐに表現できて、簡単にビジュアル面が綺麗に見えるように作れるのが強みです。そして、Unity にはアセットストアというものがあり、無料で簡単に素材を手に入れる事が出来ます。

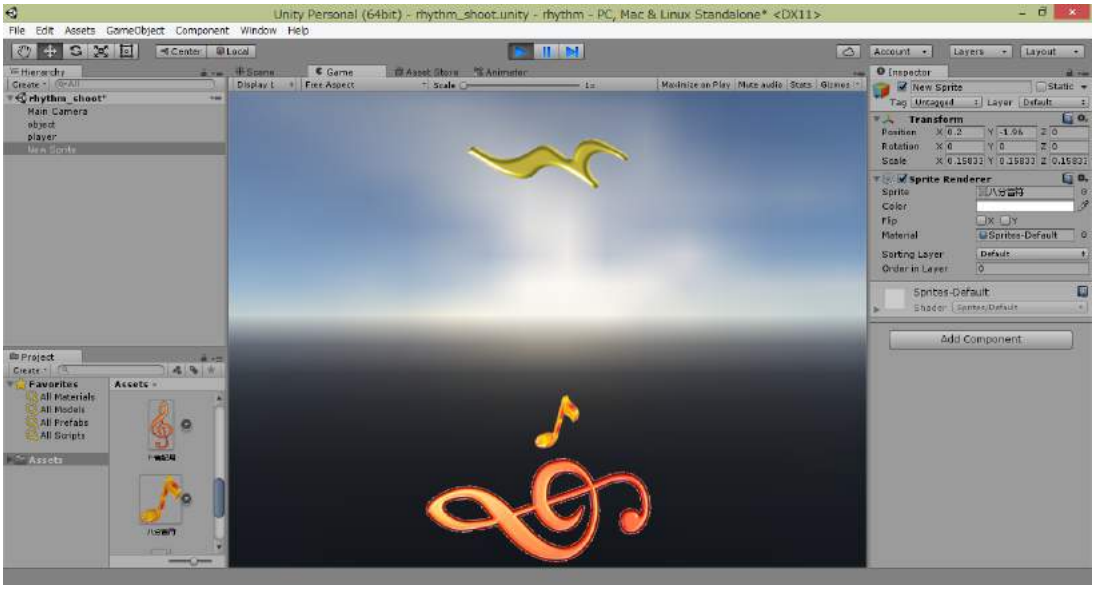


図 1: Unity の開発画面

こうしてみると、Visual Stuio の開発画面よりもゲームを作っている気分を感じられるので、モチベーションは上がります。すぐにテストして挙動を見る事も出来るので、フィードバックも簡単です。

5 結論

こうして、二つの方法で作った結果、Siv3D だとプログラミングがしやすく、凝ったゲームシステムが作りやすいので、パズルゲームや簡単なアクションゲームを作る時に便利だと思います。また、Unity ではよりよいテクスチャを使い直感的に作る事ができ、物理エンジンも搭載しているので、RPG やシミュレーションゲームを作る

のに向いている、という結果に至りました。

取り敢えずプログラミングは少し知ってる！又は知らないけど興味ある！という方は Siv3D を触ってみるといいかも知れません。ゲームを本気で作りたい！よさげなビジュアルでかっこよく見せたい！となると Unity の方が圧倒的に感覚で作れますし、実際のゲーム企業でも使用しているところは多いです。勿論、どちらも自分で面白いゲームシステムを考える必要はありますよ。

6　おわりに

今回は、どちらも 3D の表現が出来るにもかかわらず、私の技術不足で 3D ゲームを作る事が出来ませんでした。次は 3D のゲーム作りにも挑戦し、また Siv3D と Unity の違いを比較・検証してみようと思います。

ここまで読んでくださってありがとうございます。もし、これを読んでゲーム作りに興味が出た、またはますます作りたくなった、と感じて頂けるととても嬉しいです。Visual Studio も Siv3D も、Unity でさえも今は無料で誰でも扱う事が出来るので、少しでも気になったものがあれば、すぐに検索してインストールしてみることをお勧めします。あなただけの面白いゲーム、作ってみませんか？

2016 年 12 月 07 日

-lm 忘れてね？

にも

1　はじめに

「コンパイルできない。課題終わらない。」「-lm 忘れてね？」「あぁ…」というようなやり取りを課題の度にしています。にもです。私はパソコン、プログラミングの知識を持たないままこのサークルに入ってしまいました。ここで語れるような知識を全く持ち合わせていないので、大学生活の中で気になったことについて書いていきたいと思います。

2　なんでやねん

私はプログラミングの課題のコンパイラに GNU Compiler Collection、俗に言う GCC を使用しています。（というかこれしかわからない）プログラム内で sin、べき乗などの数学的な関数を使用するために、`#include<math.h>`を記述しますよね。その際、GCC でコンパイルする場合、gcc work.c -lm のようにファイル名の後に-lm と付け加えなければなりません。私はいつもそれを忘れてしまいます。なんで！`<stdio.h>`とか`<stdlib.h>`はいらないのに！面倒くさい！-lm ってなに…

3 -lm

-lm というのは GCC を拡張するためのコマンド、いわゆるオプションです。-l と m にはそれぞれ意味があります。-l はライブラリのリンク、m はライブラリの名前、ここでは libm.a を表しています。つまり、<math.h>は libm.a というライブラリをリンクしなければ使えないということになりますね。ここで、知識のない私に疑問が生まれました。ライブラリとかリンクってなに…？

4 ライブラリとリンク

ライブラリとは、既にコンパイルされたいくつかの関数の集合のことです。libm.a というライブラリは sin、べき乗などの数学的な関数が既にコンパイルされた状態となっているものの集合、ということになりますね。また、ライブラリは自作することもできるので自分仕様の便利なライブラリを作成することもできます。

リンクとは、ライブラリの関数をプログラム内で使用できるようにすることです。まあ、そんな感じはしてましたよね。

ちなみに、<stdio.h>などのライブラリを取り込まなくてもよいものは、標準ライブラリと言います。この標準ライブラリは例外としてオプションをつける必要はありません。<math.h>が面倒なわけではなく、<stdio.h>とかが便利すぎただけだったんだね！

このライブラリとリンクについてはまだまだ説明するべきことがたくさんあります。静的ライブラリや共有ライブラリの話や、自作ライブラリの話などがありますが、私の知識では説明できないのでここまでとさせていただきます。（この無能め！）

5 おわりに

役に立つような内容を書けず申し訳ありませんでした。今後、皆さんの役に立つような内容を書けるよう努力したいと思います。また、これを通してライブラリ等に興味を持ったような気がしなくもないので、機会があればライブラリについての深い内

容のものが書けたらなと思います。このサークルに入った以上パソコン、プログラミングの知識について深め、よりよいものが書けるようになりたいと思います。最後まで読んでいただきありがとうございました。

2016 年 12 月 7 日

Siv3D を用いたゲーム作成

koi

1　はじめに

私は今年の 9 月に開催された学園祭でゲームを展示した。自機を左右キーで左右に動かすが、上下には自動で動き、を回避するゲームである。このゲームは Siv3D を使用して作成した。

1.1　秘伝のソースコード

```
1   # include <Siv3D.hpp>
2
3  void Main()
4  {
5          Font font(30);
6          double x = 300;
7          double y = 200;
8          double a = 0;
9          double b = 250;
10         double h = -50;
11         double i = -50;
12         double d = 1;
13         int32 score =  0;
14         int32 hantei = 0;
15         int32 j = 0;
16 while (System::Update())
17         {
```

```
18              Line(0, 400, 650, 400).draw(5);
19              Line(0, 100, 650, 100).draw(5);
20              Circle(x, y, 20).draw(Palette::Red);
21              Circle(a, b, 20).draw(Palette::Blue);
22              Circle(h, i, 20).draw(Palette::Blue);
23              y = y + 12 * d;
24              if (y <= 375) { d = d*(-1); }
25              if (y >= 120) { d = d*(-1); }
26              if (x <= 0) { x = 650; }
27              else if (x >= 650) { x = 0; }
28              if (Input::KeyRight.pressed)
29              {
30                      x = x + 7;
31              }
32              if (Input::KeyLeft.pressed)
33              {
34                      x = x - 7;
35              }
36              font(score).draw(300, 400);
37              if (hantei == 0) {
38                      score++;
39              }
40      if (160 > score && score > 100) {
41                      a = a + 12;
42              }
43              else if (200 > score && score > 160) {
44                      a = 0;
45                      b = 200;
46              }
47              else if (260 > score && score > 201) {
48                      a = a + 12;
49              }
50              else if (300 > score && score > 261) {
51                      a = 0;
52                      b = 300;
53              }
54              else if (360 > score && score > 301) {
55                      a = a + 12;
56              }
57              else if (400 > score && score > 361) {
58                      a = 635;
59                      b = 250;
60              }
```

```
61  else if (460 > score && score > 401) {
62                          a = a - 12;
63                  }
64                  else if (500 > score && score > 461) {
65                          a = 635;
66                          b = 200;
67                  }
68
69          if (fabs(a - x)*fabs(a - x) + fabs(b - y)*fabs(b - y) <= 900) {
70                          hantei = 1;
71                  }
72                  else if (fabs(h - x)*fabs(h - x) + fabs(i - y)*fabs(i - y)
73                          hantei = 1;
74                  }
75                  if (score >= 500) {
76                          hantei = 2;
77                  }
78                  if(hantei==1){
79                  font(L"Gameover").draw(225, 200);
80          }
81                  if (hantei == 2) {
82                          font(L"Gameclear").draw(210, 200);
83                  }
84                  if (Input::KeyR.pressed)
85                  {
86                          score = 0;
87                          x = 300;
88                          y = 200;
89                          a = 0;
90                          b = 250;
91                          hantei = 0;
92                          h = -50;
93                          i = -50;
94                          d = 1;
95                  }
96          }
97  }
```

1.2 制作過程

まず、自機の赤い玉と白線を描画する。2 本の白線の間を自機が上下するようにできている。左右のキーを押すとその方向に動く。スコアは画面下部に表示され、時間の経過で点数が上昇する。次に青い玉の挙動を入力する。今回は score=500 までの挙動を表記したが、実際のプログラムでは score=5000 まで入力した。赤い玉が青い玉に接触するとゲームオーバーの表示がされ、スコアの上昇が停止する。R キーを押すとスコア（を含む変数）がリセットされ、再スタートとなる。スコアが一定量に達すると、ゲームクリアの表示がされる。

2 おわりに

今回初めてゲームを製作して、プログラムには膨大な行数を必要とし、製作に必要な時間の長さを体感した。この記事から興味を持った人はぜひともゲームを作ってみて欲しい。

2016 年 12 月 7 日

学園祭で展示した自作ミニゲームについて

ばこやし

1　はじめに

10 月末の学園祭で作品を展示することになった私は、先輩方の助言を受け、Siv3D を用いたミニゲームを制作した。内容は、黄色い球を十字キーで操作し、ランダムに動く赤い球 (CPU) から一定時間逃げ続けるというシンプルなものである。簡単ではあるが、そのゲームのソースコードと解説を以下に記す。

2　自機の動作

[→] キーを押したとき (他の 3 方向も同様)

```
1  if (player.x < 639)
2  {
3          if (Input::KeyRight.pressed)
4          {
5                  for (int32 i = 0; i < 3; ++i)
6                          ++player.x;
7          }
8  }
```

画面外に出ないように

```
if (player.x < 639)
```

で動作可能な位置を制限。

```
if (Input::KeyRight.pressed)
  ++player.x;
```

というコードでテストプレイをしたところ、自機の動きが極端に遅かったため、

```
if (Input::KeyRight.pressed)
  for (int32 i = 0; i < 3; ++i)
    ++player.x;
```

上記のように for 文を用いることで、3 倍の速度を出すことに成功した。

3 CPU の行動パターン

```
1  if (0 < enemy.x && enemy.x < 639 && 0 < enemy.y && enemy.y < 479)
2  {
3          for (int32 i = 0; i < Random(1, 100); ++i)
4                  ++enemy.y;
5          for (int32 i = 0; i < Random(1, 100); ++i)
6                  ++enemy.x;
7          for (int32 i = 0; i < Random(1, 100); ++i)
8                  --enemy.y;
9          for (int32 i = 0; i < Random(1, 100); ++i)
10                 --enemy.x;
11 }
```

自機と同様に、画面外に出ないように if 文で制限したところ、(当たり前だが)CPU が画面端で静止するようになった。まったくの偶然だったが、これを利用して、全ての CPU の動きが静止するまで逃げ続けることをクリアの条件とした。CPU の動きには Random を用い、上下左右移動をそれぞれ 1〜100 回繰り返すことで予測不能な動きを実現させたが、ランダムであるが故にプレイ時間がその時次第でまちまちになるという問題点が残ってしまった。

4　衝突時のエフェクト

```
1  const bool e1 = player1.intersects(enemy1);
2  player1.draw(e1 ? Palette::Red : Palette::Yellow);
```

ここでは割愛したが、画面四端に当たり判定のある赤い壁を設けている。赤い壁あるいは赤い玉 (CPU) に衝突すると即座に自機の色が黄から赤に変化する仕掛けだ。本来であればシーン遷移でゲームオーバー画面を表示すべきところなのだが、残念ながら知識不足で想定通りには行かなかった。完成品では黄から赤に変化したらゲームオーバーということにしたが、プレイヤーの困惑は避けられなかっただろう。今回もっとも残念だった箇所である。

5　おわりに

プログラミング経験の浅さ故、完成した作品は極めて簡素な出来になってしまった。それでも、自分でゲームを作るという体験は、もの作りの楽しさ、プログラミングの難しさを私に教えてくれた。また、CPU の行動パターンの改良やシーン遷移の実装など、今後の課題を知る良い機会だったようにも思う。今後もプログラミングを学ぶうえで様々なことを吸収し、ゲーム等の作品制作に生かせるようにしていきたい。

2016 年 12 月 07 日

Unity を用いた地形作成

しゃりん

1　はじめに

太古から、人間にとって芸術は自身の行き場のない感情をぶつける場であった。

平安・鎌倉時代の貴族たちは嬉しいとき、悲しいときに句を読んできたし、中世の音楽家、画家達は自らの恋や愛するものを失うような激情の度に後世に残る作品を産み出している。

その中でもとりわけ風景画は特別で、古代エジプトやクレタ文明の頃から図式的な風景の表現は存在しており、5 世紀ごろから中国では山水画と呼ばれる墨で描いた風景画が存在してるのだ。

風景という人類に対して壮大なものに芸術を求め、絵に起こしてきた歴史は非常に長いのである。

しかし当然、大多数の一般人は壮大な風景を書き表すほどの画力がないことは言うまでもない。

誰もが皆、ダ・ヴィンチやミケランジェロやルーベンスやゴッホや葛飾北斎やレンブラントのように絵が上手い訳ではないからである。

しかし芸術性はともかく、最新の技術を駆使すれば自身の思い通りに風景を表現することは可能な時代となったのかもしれない。

というわけで今回私はゲームエンジン「Unity」の機能の一つ、誰でも手軽に簡単に 3D 空間を作ることができる「Terrain Engine」を紹介したいと思う。

もしこの機能を使いこなすことができれば、あなた自身で自分で歩き回れる風景を

作ることすら出来てしまうかもしれない、そんな面白い機能である。

またコーディングは得意じゃないけどゲームエンジンを触ってみたいという方や、Unity でどんなことができるのか気になる方はぜひ読んでみてほしい。

それでは始めていきましょう。

2 Unity に関して

まず、意外と知らない方が多いのかと考えて、Unity というゲームエンジンに関する基礎事項をまとめてみました。

Unity とは、統合開発環境を内蔵し、様々なプラットフォームに対応することを目的としたゲームエンジンです。Unity の最大の特徴としては Unity 一つで iOS、Android といったスマートフォン、PlayStation や Xbox といったコンシューマゲーム機に対応できることにあります。

近年は大手のゲームエンジンを個人が無償で利用できるような環境の整備と高性能化によって商用から個人のゲームまで幅広く、特にスマートフォンのゲームでは多く採用されています。

今回紹介する Terrain Engine は、Unity で 3D ゲームを作成する際に地形を作ることを目的として搭載された機能であり、絵の才能や技術に関係なく誰でも簡単に 3D の地形を作ることができるというものです。

今回はその Terrain Engine の最も基本的な使い方を、誰にでもなるべくわかりやすく説明したいと思います。

2.1 注釈

以下の記述はすべて Unity5.3.4 のバージョンにおいてのものであるため、他のバージョン、特に大幅なバージョンアップがあった際にはこの文章内での説明とは大きく操作が変更されることがあるので念頭に置いて頂きたい。

また、この記事と情報が異なる際やさらなる情報がほしい方はぜひ Unity の公式マニュアル (https://docs.unity3d.com/ja/current/Manual/index.html) を覗いてみてほしい。

むしろページ数の都合で実際の画面を載せることができなかったので公式マニュアルと照らし合わせながら読んでもらうことをオススメします。

マニュアルは公式が出す正確な情報をわかりやすく端的に得られる為、積極的に利用していきましょう (自己啓発)。

3 Terrain へようこそ

3.1 インストールと準備

まず Unity のインストール方法だが、公式マニュアルに綺麗にまとめられているので (https://docs.unity3d.com/jp/current/Manual/InstallingUnity.html) 早速ながら割愛させてもらいます。

まず、インストールされている Unity を起動し、適当な名前とディレクトリに保存します。

その後、Unity にゲームのオブジェクトを表示する領域であるシーンに地形を作成する Terrain オブジェクトを追加するため、メニューから GameObject → 3D Object → Terrain と選択します。画面に平坦で広い地面が表示されたら成功です。

この状態になると左側の Hierarchy メニュー内に Terrain というオブジェクトが生成されていると思うので、クリックします。

この状態で右側の Inspector に Terrain のメニューが表示されていると思います。Terrain による地形生成では、このメニューで主な操作を行っていくことになります。

3.2 地形作成

では地形を作っていきましょう。

まずは平らな地形に凹凸を作っていきます。Inspector に並んでいる 7 つのアイコンの一番左にある、「Raise / Lower Terrain」というアイコンを選択して地形をクリックしてみてください。地形に山ができたかと思います。上げ過ぎた際は Shift キーを押しながらクリックすることで地形を下げることができると思います。

この機能では選択したブラシの種類とサイズを選択し地形の凹凸を定めていきます。

主にこの機能を使って凹凸の設計をしていきます。

この機能を使っていると、例えば平面の地形から谷を作る時などに苦労すると思います。

そんな時に利用するのが 2 番目のアイコン、「Paint Height」です。このツールでは設定で Height を操作し、「Flatten」のボタンを押すことで Terrain 全体の高さを設定した値に合わせることが出来ます。

もう一つの地形を組み立てるのに役立つツールに「Smooth Height」があります。このツールは名前の通り、地形の凹凸を緩やかにしてくれます。

この 3 つの機能で地形の概形を作っていきます。

3.3 テクスチャを張り森林を作る

次に真っ白な地面にテクスチャを貼っていきます。

まず貼り付けるテクスチャの準備をします。今回は標準の Unity が提供してくれている Standard Asset に入っているテクスチャを利用します。

素材のインポートは Unity ウインドウ上部のメニューバーから、Assets → Import Package → Environment を選択することでインポートすることができます。

インポートした後、Terrain の Inspector の 7 つのアイコンの中央にある「Paint Texture」を選択し、Texture という項目内の Edit Texture を選択します。

そこで出てきた Add Terrain Texture ウインドウの左中央にある、RGB Smoothless という四角形の中にある select ボタンを押します。

そうすると先ほどインポートしたアセットが表示されると思うので、適当なテクスチャを選択し、add をクリックします。(よくわからない人は grass hill albedo Texture を選択してみてください。)

白一色だった地形に選択したテクスチャが書き込まれたら成功です。

最後に木を植えてみましょう。Inspector の右から 3 番目のアイコン、「Place Trees」を選択し、Trees という項目内の Edit Trees を選択します。

こちらも先ほどインポートしたアセットが表示されてると思うので、適当な木を選択し、add をクリックします。(よくわからない人は Broadleaf_desktop を選択してみてください)

お察しの通り、テクスチャ選択と同じ動作です。

インポートした木を選択した状態で地形を右クリックすると地形に木を生やすことができます。

以上で Unity Terrain の基礎的な話を終了します。カメラを動かしてウインドウ上部のプレビューボタンで綺麗な風景を映せるようにして遊んでみましょう。

4 あとがき

今回はページ数の都合もあり、紹介程度の内容ですが、光源を増やしたり、水面を作ったり、驚くほど多くの機能があります。公式のマニュアルや検索を駆使し、あなたの思い描く風景を作っていきましょう。

また Unity には Asset Store という、ほかの方が作ったアセットを貰ったり購入したりする機能があります。

利用することで自身では作ることが難しい機能を実装することができます。

ゲームエンジンは便利ですが、専用の知識が多く始めにくいと思います。そんな人がゲームエンジンを使うきっかけになってくれたらいいなと思います。

Twitter の bot を作ってみた話

ポケトレ

1　はじめに

Twitter 上には自動的に tweet する bot と呼ばれるアカウントがたくさん存在しています。簡単に bot を作ることができるサービスとして twittbot などがありますが、今回は PHP 言語を使って定期的にツイートする bot を作ってみたいと思います。

2　アプリケーションの登録

Twitter の開発者サイトからアプリケーションを登録します。登録後、ConsumerKey と ConsumerSecret をメモしておきます。また、アクセストークンを発行して AccessToken と AccessTokenSecret もメモしておきます。

アプリケーションの登録と実際に運用したい bot のアカウントが違う場合は AccessToken と AccessTokenSecret は別の方法で取得しなければならないので注意しましょう。

3　PHP スクリプトの作成

次に、PHP のスクリプトを作成します。TwitterOAuth という OAuth を簡単にしてくれるライブラリがあるのでこれを使います。たくさんのライブラリを入れた

い場合は Composer を使ってライブラリを管理するのもいいかもしれません。

4 今回作成したソースコード

```
<?php
require_once(__DIR__ . '/vendor/autoload.php');

define('CONSUMER_KEY','YOUR_CONSUMER_KEY');
define('CONSUMER_SECRET','YOUR_CONSUMER_SECRET');
define('ACCESS_TOKEN','YOUR_ACCESS_TOKEN');
define('ACCESS_TOKEN_SECRET','YOUR_ACCESS_SECRET');

use Abraham\TwitterOAuth\TwitterOAuth;

$connection = new TwitterOAuth(
  CONSUMER_KEY,
  CONSUMER_SECRET,
  ACCESS_TOKEN,
  ACCESS_TOKEN_SECRET);

//日付を取得します
$day = date( "Y年m月d日H時i分s秒" );

//ツイート内容を時間によって決定します
switch (date(H)) {
        case '00':
                $tweet_mes = "そろそろ寝ないと遅刻しちゃうよ、兄さん。";
                break;
        case '07':
                $tweet_mes = "兄さん、おはようございます。";
                break;
        case '09':
                $tweet_mes = "そろそろ授業がはじまりますよ、兄さん。";
                break;
        case '13':
                $tweet_mes = "今日はなにを食べようかな？";
                break;
        default:
                // ファイルの行をランダムに抽出
                $filelist = file('list.txt');
                if( shuffle($filelist) ){
```

```
38                  $tweet_mes = $filelist[0];
39                  }
40                  break;
41  }
42  //ツイート処理
43  $res = $connection->post("statuses/update",[
44    'status' => "$tweet_mes\n$day"
45  ]);
46  //エラー処理は省略
47  ?>
48
49
```

21 行目からの switch 文、case 文によってツイートする内容である変数`$tweet_mes`を決定しています。date(H) によって取得した今何時であるかという情報によって場合分けし、default では、予め用意しておいた list.txt の行をランダムで抽出し、`$tweet_mes` に代入しています。

43 行目では Twitter REST API を呼び出しています。今回はツイート処理を実装したいので POST statuses/update という API を使います。

5　サーバーへのアップロード

WinSCP などの FTP クライアントを使ってサーバーへファイルをアップロードします。自動実行の仕様上、実行される PHP ファイルの permission は 755 などにしておきましょう。

6　自動実行の設定

スクリプトを定期的に実行するために、cron コマンドを使う場合が多いですが今回は GoogleAppsScript のスクリプトトリガーで代用します。このようなスクリプトを一時間ごとに実行するように設定します。

```
1  function myFunction() {
2    UrlFetchApp.fetch("http://???????/bot.php");
3  }
```

GoogleAppsScript は JavaScript 互換のサーバー上で動くスクリプト言語です。JavaScript の関数と同じように書くことができ、Google のさまざまなサービスにアクセスすることができます。今回は詳しく言及しませんが、ちょっとした Web サービスを作ることもできます。

7 おわりに

Twitter を彩る bot 達を見ていて、実際に作ってみたくなったので自作してみることにしました。リプライの機能が実装されていないのでそこを次回への課題としようと思います。今回気がついた点として、インターネット上の情報は鮮度に気を使わないといけないと実感しました。（特にプログラミングなどの日々変わっていくもの）コードが動かない原因がバージョンが古かったなんてことがあったからです。自作することで自由度の高い bot を作ることができます。bot を自作して、あなたも快適な TwitterLife を過ごしてみませんか？

2016 年 11 月 28

初めてのゲーム作成

リトマス試験紙

1　はじめに

自分は過去に RPG ツクールを使ってゲーム作成したことがありました。しかし、その時は C 言語も DX ライブラリも勉強してなかったのでマップを作成する時はどのような過程でマップが作られているのか全く意識せずにオブジェクトの設置をしていました。

その結果、今年になってツクール以外でゲームを作り始めて、マップ作成に苦戦しました。そこで今回は、DX ライブラリと C 言語を使って、簡単なマップを作成し、キャラを動かすプログラムを記載します。

2　マップ作成

2.1　プログラミングの説明

まず使用する画像は media フォルダに入っていて、サイズは横縦ともに 50 ピクセルと仮定したうえで話を進めます。

主人公キャラはサンタとします。理由は今年文化祭で提出した作品のコードの一部を用いているからです。

2.2　gamemain.cpp

```
1  #include "loading.h"
2  #include<DxLib.h>
3  #define IMG_CHIPSIZE 50
4  #define MAP_WIDTH 5
5  #define MAP_HEIGHT 5
6  extern int g_mapdata[MAP_HEIGHT][MAP_WIDTH];
7  /*マップデータ縦横*/
8  enum MapItem {
9          MPITEM_NO, MPITEM_RED
10 };
11 struct StageData {
12         int santax, santay;
13 };
14 extern StageData g_stagedata;
15 void GameMain();
16 void DrawMap();
17 int g_mapdata[MAP_HEIGHT][MAP_WIDTH] = {
18         /*0=歩ける床,1=通れない壁,*/
19         /*0 1 2 3 4 */
20         { 1,1,1,1,1 },/*1行*/
21         { 1,0,0,0,1 },/*2行*/
22         { 1,0,0,0,1 },/*3行*/
23         { 1,0,0,0,1 },/*4行*/
24         { 1,1,1,1,1 } /*5行*/
25 };
26 int x, y;/*マス*/
27 int v=1;
28 int g_x = g_stagedata.santax;/*主人公のx座標*/
29 int g_y = g_stagedata.santay;/*主人公のy座標*/
30 StageData g_stagedata;
31 int santamovecounter;
32 int g_mv = 1;/*主人公の進むマス数*/
33 void GameMain() {
34 DrawMap();
35 SetDrawScreen(DX_SCREEN_BACK);
36 int me=LoadGraph("media\\me.png");
37 /*主人公の初期位置*/
38 g_stagedata.santax = 2;
39 g_stagedata.santay = 2;
40 while (ProcessMessage() == 0 &&
41 CheckHitKey(KEY_INPUT_ESCAPE) == 0) {
42 santamovecounter++;
```

```
43 if (santamovecounter == 10) {
44 santamovecounter = 0;
45 }
46 g_x = 0;
47 g_y = 0;
48 ClearDrawScreen();
49 DrawMap();
50 int key = GetJoypadInputState(DX_INPUT_KEY_PAD1);
51 if (santamovecounter == 0) {
52 if (key & PAD_INPUT_LEFT) {
53 g_mv = -1;
54 g_x += g_mv;
55                          }
56 if (key & PAD_INPUT_RIGHT) {
57 g_mv = 1;
58 g_x += g_mv;
59 }
60 if (key & PAD_INPUT_UP) {
61 ProcessMessage();
62 g_mv = -1;
63 g_y += g_mv;
64 }
65 if (key & PAD_INPUT_DOWN) {
66 g_mv = 1;
67 g_y += g_mv;
68 }
69 g_stagedata.santax = g_stagedata.santax + g_x;
70 /*通れない床の処理*/
71 /*x座標動かしてサンタの座標が本来通れない場所にいたら元に戻す*/
72 if (g_mapdata[g_stagedata.santay]
73 [g_stagedata.santax] != MPITEM_NO) {
74 g_stagedata.santax = g_stagedata.santax - g_x;
75 }
76 g_stagedata.santay = g_stagedata.santay + g_y;
77 /*通れない床の処理*/
78 /*y座標動かしてサンタの座標が本来通れない場所にいたら元に戻す*/
79 if (g_mapdata[g_stagedata.santay]
80 [g_stagedata.santax] != MPITEM_NO) {
81 g_stagedata.santay = g_stagedata.santay - g_y;
82 }
83 }
84 DrawGraph(g_stagedata.santax*IMG_CHIPSIZE,
85 g_stagedata.santay*IMG_CHIPSIZE, me, TRUE);
```

```
86  clsDx();
87  ScreenFlip();
88  }
89  }
90  void DrawMap() {
91  for (y = 0; y < MAP_HEIGHT; y++) {
92  for (x = 0; x < MAP_WIDTH; x++) {
93  DrawGraph(x*IMG_CHIPSIZE, y*IMG_CHIPSIZE,
94  g_imghandles.field, FALSE);
95  /*置物*/
96  int c = g_mapdata[y][x];
97  if (c > 0) {
98  DrawGraph(x*IMG_CHIPSIZE, y*IMG_CHIPSIZE,
99  g_imghandles.mapitems[c], TRUE);
100 }
101 }
102 }
103 }
```

2.3 loading.cpp

```
1   #include"loading.h"
2   ImageHandles g_imghandles;
3   BOOL LoadGameImage(){
4           g_imghandles.field = LoadGraph("media\\santa0.png");
5           if (g_imghandles.field == -1)return FALSE;
6           g_imghandles.mapitems[1]= LoadGraph("media\\santa1.png");
7           if (g_imghandles.mapitems[1] == -1)return FALSE;
8
9           return TRUE;
10  }
```

2.4 loading.h

```
1   #ifndef __LOADING_H__
2   #define __LOADING_H__
3   #include<DxLib.h>
4   struct ImageHandles {
5           int field;
```

```
6           int mapitems[2];
7   };
8   extern ImageHandles g_imghandles;
9
10  BOOL LoadGameImage();
11  #endif
```

3　今回のプログムを書き終えて・・・・

今回初めてプログラミング言語でゲームのマップ画面の作成、キャラを動かすところまで作成しました。アクションゲームを作成するときピクセル座標で当たり判定をどう作るのか頭を捻ることになりますが、このプログラムだと主人公と敵の座標が同じな時に当たり判定を発生させるだけで作れるので手軽だと思いました。

またプログラミング言語でゲームマップ作製をして、簡単なミニゲームを作るのならばRPGツクールが使いやすい物だと実感しました。

しかし何もないものからプログラミングで作り上げていく作業は非常に楽しかったです。

2016年12月7日

GAME の作り方

あーさん

1　はじめに

学祭で作ったブロック崩しっぽいものを参考に、次回 GAME を作成するときに役立つものを置いておきます。

2　GAME に必要なもの

GAME を作るにあたって最低限必要なものは以下の 4 つだと考えています。

- GAMETITlE 画面
- MAIN 画面
- GAMEOVER 画面
- GAMECLEAR 画面

3　作り方

では、上記のものを作るにはどうすればいいか？僕のソースコードをコピペしてください。

3.1 僕のソースコード

```
1  int main(){
2
3  enum class GameState
4          {
5                  GAMETITLE,
6                  MAIN,
7                  GAMEOVER,
8                  GAMECLEAR
9          };
10 GameState gameState = GameState::GAMETITLE;
11
12     auto gametitle = [&]()
13
14         {
15                 if (Input::AnyKeyClicked())
16                 {
17                         gameState = GameState::Main;
18                 };
19     }
20
21     auto Main = [&]()
22
23         {
24         if (Input::AnyKeyClicked())
25                 {
26                         gameState = GameState::GAMECLEAR;
27                 };
28     }
29
30     auto gameOver = [&]()
31
32     {
33         if (Input::AnyKeyClicked())
34         {
35         gameState = GameState::GAMETITLE;
36                 };
37     };
38
39     auto gameClear = [&]()
```

```
40          {
41
42                  if (Input::AnyKeyClicked())
43          {
44                          gameState = GameState::GAMETITLE;
45                  };
46
47      };
48
49      std::vector<std::function<void()>> sequences =
50          {
51                  gametitle,
52                  Main,
53                  gameOver,
54                  gameClear
55          };
56
57
58          while (System::Update())
59          {
60                  sequences[static_cast<int>(gameState)]();
61
62
63          };
64  }
65
66
67  }
```

4 おわりに

コピペは恥だが役に立つ。

twitter: @comp097

2016年12月7日

CC で遊んでる話

しょとー

1　はじめに

最近になってようやく ComputerCraft(以下 CC) に触れたので、とりあえずプログラムを実行できる程度まで記載してみたいと思います。CC とは、Minecraft の世界に lua で作られたプログラムで動くコンピュータ (Computer) やロボット (Turtle) を追加する mod[*1]です。正直言ってよりわかりやすいサイトが既にあるのはナイショの話。

2　導入

- Minecraft 1.7.10
- Minecraft Forge[*2]
- ComputerCraft 1.75

まずはじめに Minecraft 本体の Ver ですが、1.7.10 を推奨します。Forge 導入の問題と、CC が 1.8.9 までしか対応していないためです。1.7.10 では commands.getBlockInfos() という機能などが使えませんが、これはサバイバルでは使用しないので特に問題はありません。

[*1] ゲームにデータの追加などを行うファイル

[*2] いわゆる前提 mod と呼ばれるもので、必要な API を内包している。

次に Forge の導入ですが、申し訳ありませんがここでは省略させていただきます。わかりやすく解説しているサイトが多くあるので其方をどうぞ。

最後に CC の導入ですが、特に難しいことはありません。Forge 導入の際に作成したフォルダ下にある mods フォルダに CC 本体を投げ入れるだけです。これで起動に成功すれば、導入に関しては終了です。

3 プログラムの編集

次は実際にプログラムの編集をします。Turtle を右クリックして CUI が表示されたら edit (ファイル名).lua と入力、するとエディタが起動してプログラムを書き込むことが可能になります。書き終えたら Ctrl を押してメニューを開き、Save で Enter を押した後、メニューから Exit を選択すれば退出します。また、ここでは拡張子を付けていますが、別に付けていなくても問題ありません。

上記の CC のエディタ以外を使用することも可能で、一度 Save を行うと (saves¥ワールド名 ¥ computer¥コンピュータの ID) のフォルダ内に lua ファイルが生成されるので其方を各エディタで編集することができます。

ID とはセーブデータ内で Computer または Turtle ごとに自動で割り振られるものです。

4 仕様とか

ついでに Turtle の仕様について重要な何点かについてだけ触れておきたいと思います。まずは Computer/Turtle を破壊すると中身が初期化されてしまう点です。これに関しては label を使用することで設定を維持することが可能です。label set (ラベル名) を実行するだけなので、とりあえず label 名は付けておいた方が事故防止になるかもしれません。

次に移動には燃料を使用する点です。refuel コマンドでインベントリ内からの補給が可能で、勿論プログラム中に組み込むことで自動化が出来ます。下記の例のプログラムには、補給を組み込んでいないのでご注意ください。燃料を消費するのは移動する時だけで、向きを変えるだけでは消費しません。

5　実際に動かしてみる

プログラムが組めたら早速実行してみましょう。実行する場合は、CUI 上でファイル名を入力するだけです。

例としてサトウキビの収穫を自動化してみました。サトウキビの 3 段目に向かって Turtle を、その下にチェストを置くと一直線上のサトウキビの収穫を行います。

```
1  function homepoint(step)
2          for i=1,step do
3                  turtle.back()
4          end
5          for i=1,16 do
6                  turtle.dropDown(i)
7          end
8          sleep(1200) -- 収穫後の待機時間
9  end
10
11 step=0
12 while true do
13         turtle.dig()
14         turtle.forward()
15         step=step+1
16         success, data = turtle.inspectDown()
17         if "minecraft:reeds"~=data.name then
18                 homepoint(step)
19                 step=0
20         end
21 end
```

前方を収穫しては移動を繰り返して、サトウキビ畑の範囲外に出たら進んだ分だけ戻っているだけです。このように自身の要望に合った動きを組めるのが CC の魅力と言えるでしょう。

数が多いのでここでは紹介出来ませんが、上記の turtle.dig() のような turtle API は ComputerCraft Wiki に記載されています。また、CC がメジャーな mod なこともあり、探せばソースコードのサンプルが見つかるので、其方を参考にするのも良いと思います。

6 おわりに

書いてみたはいいけど、どう考えても解説足りてないし、わかりにくい！次の進捗はなんとかしたいところです。

2016 年 12 月 6 日

Siv3D のクラス SceneManager について

ソー

1　はじめに

皆さんは自分で初めてゲームを作るときシーン遷移はどのような方法でやりましたか？私はプログラミング初心者で初めて Siv3D でゲームを作るときシーン遷移に苦労したのですが、SceneManager というクラスが使いやすく楽にシーン遷移できたので紹介したいと思います。

2　SceneManager とは？

SeneManager とは Siv3D に付属している HamFrameWork のクラスの 1 つです。このクラスを使うことによって楽にシーン遷移が行えるようになります。

C++ において他に有名なシーン遷移というと switch 文がありますがシーンの数が増えたりすると switch 文を何度も書き直さなければならないなど不便な面もありますがその点で SceneManager はより使いやすいと思います。

3　ソースコードとその説明

```
1  # include <Siv3D.hpp>
```

```
2  # include <HamFramework.hpp>
3
4  class Title : public SceneManager<String>::Scene
5  {
6  public:
7      void init() override
8      {
9      }
10
11     void update() override
12     {
13     }
14
15     void draw() const override
16     {
17     }
18 };
19
20 class Game : public SceneManager<String>::Scene
21 {
22 public:
23     void init() override
24     {
25     }
26
27     void update() override
28     {
29     }
30
31     void draw() const override
32     {
33     }
34 };
35
36 class Result : public SceneManager<String>::Scene
37 {
38 public:
39     void init() override
40     {
41     }
42
43     void update() override
44     {
```

```
45      }
46
47      void draw() const override
48      {
49      }
50  };
51
52
53  void Main()
54  {
55      SceneManager<String> manager;
56
57          manager.add<Title>(L"Title");
58      manager.add<Game>(L"Game");
59      manager.add<Result>(L"Result");
60
61      while (System::Update())
62      {
63      manager.updateAndDraw();
64      }
65  }
```

上記のソースコードが SceneManager の基本的なソースコードです。この書き方だと 3 つのシーンを作ることができます。

シーン内でのやりたいことはそれぞれのクラス内の init,update,draw 関数に書きます。init 関数はシーンを切り替えた時に 1 度だけ呼ばれます。update 関数は毎フレームごとに呼ばれます。draw 関数は毎フレーム update 関数の後に呼ばれます。update 関数と draw 関数の違いは update は普通のメンバ関数で draw は const メンバ関数であるということです。なので、draw 関数内でメンバ変数の値の変更はできません。

シーンの登録は main 関数内の add 関数で設定することができます。上記のソースコードでは String 型で定義しているため L”” で設定できます。

4 おわりに

ここで紹介したのは SceneManager の本当に基本的なことでまだまだ応用がたくさんできると思います。もっと詳しく知りたい人はインターネット等で調べてみて

ください。この文章が少しでも役に立てば幸いです。

2016 年 12 月 7 日

図形とテクスチャ

PZOG

1　概要

今年から Siv3D を用いてゲームを作ってみようとしたときに「図形」と」「テクスチャ」の二つを使い機会があり、その際にいろいろと苦戦したので（初心者というのもあるが）いくつか自分なりに気づいたことをまとめてみました。

2　図形ついて

まずこのことについて論じるにあたり「図形」と「テクスチャ」について説明しなければなりません。Siv3D においては「図形」というのはその図形を出力するにあたっての特別な関数を持つもののことをさします。具体的な例としては、円を出力するために使われる `Circle(`（中心の x 座標），（中心の y 座標），（半径の長さ））や四角形を出力するための関数である `Rect`（(x 座標），（y 座標），（幅），（高さ））など一般的な図形にはそれぞれに独立した関数が与えられています。

3　テクスチャについて

テクスチャというのは画像ファイルのことです。Siv3D ではパソコンにある画像ファイルを読み込んで出力できる Texture という独自の関数が備わっていてこの関数に画像を出力する位置や色、大きさなどを指定して画面に出力できるのです。

4 互いの良いとこ

前の二つのセクションで大まかながら図形とテクスチャがそれぞれどの様なものかを説明しました。ではここからは二つそれぞれのプラスとマイナスの面を比較しつつ紹介します。

4.1 図形のプラスの点

図形を使うことによる利点は図形に対応した多くの関数の存在です。例えば、あたり判定を実装したいときテクスチャは座標の位置で逐一判定しなくてはいけないのですが、図形の場合には（図形の関数）.mouseover や（図形の関数）.leftpressed などを使うことで簡単かつソースコードをとても見やすい状態であたり判定を実装できるのです。しかし、図形に関しては図形の数の多さ故、先ほど紹介したものが実装されていないものもありますが、それでも十分すぎるほど当たり判定に関しては実装しやすくなっています。

4.2 テクスチャのプラスの点

テクスチャを使う利点というのは第一に表現の自由度にあります。自身が作成したりどこかからダウンロードしたものを使うわけですから当然、図形だけでは表現できないものを使えるため作成物がより鮮やかに彩られるでしょう。例えばパズルゲームを作る場合、図形を使ってしまうと絵柄が無いのでパズルとは言えなくなってしまいます。

5　互いの悪いところ

5.1　図形の悪いところ

図形の悪いところは、あまりないのですが強いてあげるとすると前述した関数が未実装でノものがあるところでしょうか。あまり作るもののレベルが高くないかもことも一因かもしれませんがゲームを作っている際に図形にを使っていて不自由はあまり感じることなくいろいろなものが実装することが出来ました。

5.2　テクスチャの悪いところ

テクスチャを使おうとすると図形で使えた関数の一部がテクスチャでは使うことができないのでかゆいところに手が届かないむずがゆさうぃ感じることが多々あった。またこれも自分の未熟さがいけないのかもしれないがテクスチャをループを組んで生成することができないなどのトラブルに見合われることがあったので、図形とは違い扱いが難しいのだと感じました。

6　おわりに

今回ゲームを作り、そこで感じたことを書くという珍しい機会を得ることができたので、少ない知識と経験で好き勝手書いては見ましたが、なかなか難しいですね。そんなことはさておき、もし次に書く機会があったらその時にはテクスチャの扱いも余裕ぶちかませるぐらいになりたいものです。最後になりますがこのカスみたいな文を読んでくれた人が少しでも「へえ〜」と思ってくれたら幸いです。

ctrl+Z

あるみに

1　はじめに

ある日のこと、コミケ原稿に着手したあるみには、初めて扱う「Cloud LaTeX」に苦戦していた。「cloud LaTeX」とはネット上で LaTeX を編集することのできるサービスのことである。あるみには自分が作ったゲームについて綴っていたが、どうもコンパイルエラーが発生してしまう。保存もままならない状態、あるみには一度数時間かけて書いた原稿をドラッグにて範囲選択し、他のテキストファイルに移そうとしたが.....

貼り付けられたのは、以前コピーしたコード。ふと編集中の原稿をみれば、書いた内容だけが、すっぽりと抜け落ちていた。半分寝ていた脳が一気に覚醒する。え？これヤバくない？えまｔｔってまｔｔｔってまってまｔってｔｔ mtmt ｔｔｔｔ

「睡眠削って書いた原稿が消滅した」疲弊したあるみにの心を折るには十分すぎた。。。。

そーーんな窮地を救ったのはググって出てきた「win キー +Z キー」。ショートカットキーの一つであり、その能力は「一つ前の処理に戻ること」。感謝に震えるあるみにの原稿内容は「ショートカットキーを紹介する」へ、大きく舵を切った。

2　-感謝を込めて-
あるみにが選ぶショートカットキー 10 選

幾度となくまとめられてきたであろう話題かもしれない。それでも紹介せずにはいられない。あなたの作業の助けになればなにより。

2.1　「win キー +Z キー」　/「ctrl キー +Z キー」
一回戻るは undo（アンドゥー）

あるみにを救った奇跡の技術。一発逆転のキー. undo（取り消し）ができる。数回押せば数回前の操作まで戻ることができる。エディターやエクセルなんかだと undo がツールバーに備わっていたりするが、ない場所でもパソコンの機能として備わっているので大体使うことができる。

2.2　「ctrl キー +C キー」と「ctrl キー +V キー」
コピーアンドペースト! ショートカットキーの王道

「ctrl キー +C キー」は選択した範囲をコピー可能。「ctrl キー +V キー」はコピーした内容をペーストできる。いちいち右クリックしてコピー、貼り付けなどしなくてもよい。有名だけど大事。

2.3　「F10 キー」確定する前に!

文字を入力しているとき、Enter などで確定する前に、「F10 キー」を押すと確定していない部分が半角、全角、小文字、大文字など変換できる。なんだか使えそうな気がする。全角状態のまま打ち込んでしまってもこのキーでゴリ押すことができなくもない。半角英数字と日本語入力、交互に行わねばならないとき、地味に良さそう。

2.4 （「Fn キー」/「Alt キー」/「win キー」）+ptr Scr スマホでおなじみスクリーンショット

あるみには「Fn キー」+「ptr　Scr キー」でした。言わずと知れたスクリーンショット機能。スマホと違って音もしないしエフェクトがほとんど発生しないので本当に取れているのか心配になる。このとき撮られたスクリーンショットはクリップボードに一時保存されおり、ペイントソフトなどに貼り付けを行うことで初めて出力される。「Alt キー」を用いるとその時開いているウィンドウのみに働く。

2.5 「Insert キー」と「Shift キー」+「Caps look キー」 絶対に許さない

前者は「入力の際に挿入（デフォルト）か上書きするかを切り替える」、後者は「英字入力の際のモードを小文字（デフォルト）を大文字に切り替えられる」だが、ホントに入力の邪魔しかしていないような気がする。プンプン。どちらももう一度同じようにキーを押すことで解除・切り替えができる。

2.6 「ctrl キー」+「A キー」　全選択！！

自分が開いているページなどに表示されている全ての文字を範囲選択する。いわゆる全選択で、まとめてコピーしたり消したいときにおすすめ。

2.7 「Shift キー」+「←↑↓→方向キー」

文字入力中ではこれによって範囲選択を自由に行うことができる。ドラッグするより精度が高くなると思う。あとはコピーとかにお任せ。

2.8 「ctrl キー」＋「H キー」と「ctrl キー」＋「I キー」 ネットのお供。

「H キー」の方はインターネット履歴を開き、「I キー」の方はお気に入り一覧を開く。ネットサーフィンがより円滑になる。

2.9 「ctrl キー」（＋「Shift キー」）押しながらドラッグ ファイルなどのコピーとショートカットの作成

簡単にファイルを分身させることができる。ドラッグの手間がかかるとはいえ楽。

2.10 「ctrl キー ＋Shift キー ＋ESC」 または 「ctrl キー ＋Alt キー ＋Delete キー

タスクマネージャーを開く最終奥義。プログラムの強制終了も出来てしまう。「Esc キー」で止まらなかったらこれを使ってみよう。なおやりすぎはパソコンによくない影響をあたえるとかなんとか。

3 まとめ

たくさんあるショートカットキーの中からとりあえず１０個選んで紹介してみました。いかがだったでしょうか。全部知っていましたか？大学ではパソコンに慣れればなれるほど、マウスは使わなくなると教わりました。ショートカットキーを使いこなせればマウスを使う機会はグッと減るでしょうし、キーボードから手が離れない分、作業も捗る..... そんな気がします！個人的にこの原稿を書いていて「F10 キー」はなかなか使い心地が良かったです。そういえば、うちのサークル名「CTRL」とも呼ぶけど、「ctrl キー」とかかってたりするんすかね？

参考文献

[1]「ショートカットキー一覧表」
http://www.geocities.jp/kagemusyamk1/index2.html
[2]「キーボードショートカットキーの一覧表—マイクロソフトアクセシビリティ」
https://cloudlatex.io/projects/43195/edit

2016 年 12 月 07 日

いらいらボールを作るにあたって

abwan

1　はじめに

ここでは私が秋に東京都市大学の文化祭で発表した、あるゲームを作った話をしたいと思います。簡単なゲームを Siv3D を使って作ってみたいかなぁと思って何をやればいいかわからない人（そんな人いるのか）に向けて書いたつもりです。当たり前にゲームを作っている人などには初心者はこんな気持ちで作っているんだなぁなんて思って見てくれると幸いです。

2　開発環境

Visual Studio 2015 , Siv3D（August2016v2）

3　導入

まず、私がどのようなゲームを作ったかというと、”いらいらボール”というタイトルのゲームです。簡単にゲームのやり方を説明すると、ボールをマウスでドラッグして操作し、ゴールまで持っていくなかで、途中で障害物や壁に触れるとスタート地点に戻らされるというゲームです。

このゲームを作ろうと思ったきっかけはもちろん文化祭に提出するためなのですが、このゲームにすると決めたのは拡張性が高いと思ったからです。なぜなら、ス

テージに作ろうと思えばいくらでもいろいろなアルゴリズムで動く障害物を生成することができ、なおかつ初心者の私でも自分のできうる範囲内での挑戦ができると考えたからです。さらに、やめたいとこでやめても文句を言われないとも思い”いらいらボール”を作ることに決めました。

4 本文

まずこのプログラムは簡単に言うとボール、障害物、タイマー、とメニュー画面によって作られています。ので、実はクラスを使うととても綺麗にプログラムが書けると思うのですが、私にはその能力がないのでそれは次回の課題にするとして、2 つの面を作り、だいたい 1000 行ぐらいのプログラムになりました。

そこで今回はこのゲームの核であるボールの生成方について説明したいと思います。これが私が書いたボールのプログラムはこれです。

```
#include <Siv3D.hpp>
void Main(){
//あとで必要で、座標を宣言これをしないとボールが動かなくなる
    Point pos(50, 50);
    while (System::Update()) {
//マウスの移動量を保存
    const Point delta = Mouse::Delta();
//マウスの当たり判定のための関数
    const Circle player(Mouse::Pos(), 0.001);
//Youというボールをpos(座標のこと)に大きさ30のボールを生成
    const Circle You(pos, 30);
//ボールとマウスとのあたり判定
    const bool I = player.intersects(You);
//マウスが押されてる時のみの移動
    if (Input::MouseL.pressed) {
        if (I == true)
            pos += delta;
        }
//ボールを描写
    Circle(pos, 30).draw(Palette::Yellow);
    }
}

```

このプログラムでマウスを使ってボールを動かすプログラムを生成しました。このプログラムの説明はプログラム内のコメントを読んでくれれば分かる程度にコメントをつけたつもりです。

このプログラムを作るにあたって、私が一番苦戦したところはマウスとボールの当たり判定でした。今はマウスとの当たり判定はそのためのがあることを知りましたが、この時のプログラムではマウスの先端にとても小さな円を作ることによって当たり判定を作っています。これがないとどこをクリックしてもボールが動いてしまい自分が思うような操作ができなかったのでした。

5　おわりに

ここまで読んでくださりありがとうございました。本当はもっとたくさん書きたいことがたくさんあるのですが、ページ数の問題でここまでになってしまいました。見返してみると内容がプログラムコードとそのコメントしか内容がないのですが、私が一番書きたかったのはここだったので自己満足することができました←

また会う機会がありましたらよろしくお願いします。

2016 年 12 月 7 日

はじめてのゲームプログラミング

もろきゅ～

1　ご挨拶

皆さん初めまして、現在 1 年生で今回初めて部誌を書かせて頂きますもろきゅ～と申します。主な内容は、私が前期のプログラミングの授業で得た知識を使ってちょっとしたゲームを作ったという話です。プログラミングがある程度出来る方にとっては知っていて当たり前のことばかりの上、大学に入学するまでプログラミングをほとんど知らなかった人間が書く内容なので期待はしないでください。

2　ゲームの作り方

実際に私がゲームを作った順序を説明いたします。

2.1　構想

まず初めにどんなゲームを作りたいのかと、開発環境等を考えます。私はホッケーの様なゲームが作りたかったので、Visual Studio 2015 Update3、言語は C++、ライブラリに Siv3D を用いました。

2.2　コードを書く

急にコードを書けと言われても初心者にはとても難しいので、様々なサンプルコードを検索し、そこからヒントを得て書き始めます。(私は Siv3D のリファレンスを主に参考にさせて頂きました。)

2.3　動かしてみる

先ほど書いたコードを一通り見直してビルドする。エラーが出た場合は 2.2 に戻り文法等が間違っていないかを確認して再び動かしてみます。

2.4　手直し

実行結果を元に気になった部分のコードを書き直したり、新しい要素を取り入れたりしてみます。

3　便利な文

ここでは C++ のゲーム作りにおいて役立った文を紹介したいと思います。

- if 文
  基本の文法ではあるが一番使った文。条件が正しいならば指定された文が処理されるというもので、壁、ボール、バーなどのあたり判定を作るのに役立ちました。
- switch 文
  if 文と同じ様に条件によって処理をする文法。前期の授業では習わなかったため、友人から教えていただきました。switch 文内の case の後の値によって処理を分けられるのでタイトル画面のシーン偏移において役立ちました。
- break 文
  処理の流れを強制的に終了させて、そのブロックから抜け出すという文。私は主に switch 文の中で用いました。

この他ににも便利は文はたくさんあると思いますが、私が理解していないので残念ながら紹介することが出来ません。困ったら本やネットに何かしらヒントがあると思いますので、その都度調べてみることをお勧めします。

4 おわりに

ここまで読んでくださりありがとうございました。薄っぺらい内容の上、拙い文章でありましたが、何か役に立つ情報をお届けできていたら幸いです。自分はまだまだプログラミング初心者ですが、これからコツコツ勉強してもっと良い物が作れたらなと思います。間違っている内容を書いていたらごめんなさい。それでは、次の機会にお会いしましょう！

2016 年 12 月 7 日

Windows のスケーリングで文字が米粒になる問題とその解決策

Toriatama

1　はじめに

近年 4K ディスプレイが大流行である。私も Apple 社の Retina ディスプレイを見てから、なんだかんだ言って 4K の美麗解像度に惚れたクチである。どうせ作業するならキレイな画面の方がいい。あとかっこいい。ところが当の Windows はこの新環境に適応したとは到底言い難い状況だ。アプリケーションによってはウィンドウサイズだけが大きくなって文字は米粒みたいな小ささのまま……といったケースが往々にしてある。

情報系の我々にとって目は仕事道具の一つである。せっかく高解像度ディスプレイを買ったのに、目を細めてディスプレイとのにらめっこに興じ続けるのはあまりに滑稽だ。

本稿ではまず Windows のスケーリング方式について簡単に述べる。次に、フォントサイズが極小で表示されてしまうアプリのスケーリングを、外部マニフェストを用いて強制変更し、それなりのサイズでレイアウトを崩さず表示させる方法を解説する。

2 Windows のスケーリング仕様

Windows は個人設定の「拡大率の設定」で自身の DPI を認識する。アプリケーションを起動する際、対象が持つ「スケーリング対応状況」を確認し、内容に応じたスケーリング処理を実施している。

2.1 アプリケーションのスケーリング対応状況

Windows アプリケーションは、スケーリング機能の対応状況によって以下の 3 種類に分類される。

- Not DPI Aware
  アプリケーションがスケーリング機能を搭載していないもの。
- System DPI Aware
  アプリケーションがスケーリング機能を搭載しているもの。
- Per monitor DPI Aware
  Vista より追加。アプリケーションがマルチディスプレイを介したスケーリングに対応しているもの。

2.2 Not DPI Aware

Not DPI Aware の代表的なアプリケーションに、Windows のデバイス　マネージャーなどがある。オフィシャルなアプリケーションですらスケーリングに対応していない状況に、Microsoft の Windows ネイティブアプリケーション移行を推し進めたい思想が見え隠れしている。

Not DPI Aware のアプリケーションは、高解像度で起動すると Windows の DPI に合わせて「そのまま」ウィンドウが拡大される。その結果非常にジャギーが目立ったボケ表示になる。

2.3　System DPI Aware と Per monitor DPI Aware

System DPI Aware のアプリケーションは、Windows の要求にあわせてアプリケーション側がスケーリングする機能を実装している。ただしその対応状況に関してはアプリケーションによって大きく異なっており、

- DPI に対応するフォントサイズが設定されていないため、文字が非常に小さく表示される
- フォントサイズはスケーリングされるが、UI に用いられている画像が非常に小さく表示される
- フォントが綺麗になるだけでその大きさも UI も小さいまま

などなど、一口に「対応」と言ってもその程度は様々であることがわかる。

またこのカテゴリにおいては、DPI の異なる複数のディスプレイを移動した際に、アプリの DPI がアプリを起動したディスプレイのもので固定される。例えば、フル HD のディスプレイで起動した Office 2016 アプリを拡大率 200% のディスプレイに移動させると、Not DPI Aware と同じ方法でスケーリングされてしまう。これに対応したものが Per monitor DPI Aware である。

3　System DPI Aware なアプリケーションのスケーリング方法を強制変更する

目的の話へと移る。だが前もって断っておくと、この方法ではスケーリングに完全対応したアプリケーションと同じ美麗さを保つことはできない。なぜなら今から紹介する方法は、中途半端な System DPI Aware アプリケーションの対応状況を Not DPI Aware に書き換え、件の「そのまんま拡大法」に切り替える方法だからである。

あくまで「このアプリが使いたいけどレイアウト崩れて使えたもんじゃない……ギザギザでいいから使いたい……」というアプリの為の応急処置だと捉えて欲しい。

3.1 マニフェスト

アプリケーションの DPI 対応状況は、実行ファイルに内蔵されているマニフェストと呼ばれるデータに記載される。マニフェストは主に、アプリケーションが使用する外部のアセンブリを特定のバージョン及び PATH で指定するために用いられるものである。Microsoft .NET Framework に触れたことがある諸兄ならば見たことがあるかもしれない。実はこのマニフェストには、アプリケーションの DPI スケーリング対応状況も書いておくことができるのだ。

以下に、外部のマニフェストを参照させることで Not DPI Aware 式のスケーリングを適用させる方法を示す。（ただし、元のファイルのマニフェストを参照しなくなるため、不具合が生じる場合もある）

3.2 レジストリを書き換えて外部マニフェストを読み込めるようにする

レジストリ エディタを起動し、以下のレジストリに移動する。

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide`

以下のデータを追加する。タイプ：DWORD(32 ビット) 値

名前：PreferExternalManifest

値：1

これで.exe と同じディレクトリに外部マニフェストが存在しているときに、そちらを読み込むようになる。

3.3 .manifest に書き込む

ファイル名は「(アプリケーション名).exe.manifest」

マニフェストは XML で記載されている。以下の内容を入力する。

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2
3 <assembly
4 xmlns="urn:schemas-microsoft-com:asm.v1"
```

```
5  manifestVersion="1.0"
6  xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
7
8  <asmv3:application>
9    <asmv3:windowsSettings
10   xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
11
12     <ms_windowsSettings:dpiAware xmlns:ms_windowsSettings=
13     "http://schemas.microsoft.com/SMI/2005/WindowsSettings">
14     false
15     </ms_windowsSettings:dpiAware>
16
17   </asmv3:windowsSettings>
18 </asmv3:application>
19
20 </assembly>
```

作成した.manifest ファイルを.exe ファイルと同一のディレクトリ内に置き、アプリケーションを起動する。すると、Not DPI Aware のレイアウトの崩れないスケーリングで立ち上がってくれるようになる。

4 おわりに

近年の高精細度化・マルチプラットフォーム環境下で、DPI スケーリングや UI のポジショニングは開発上もはや避けて通れない。せっかくの綺麗なディスプレイなのだから、アプリの UI もそれをリスペクトした綺麗なものであって然るべきだろう。我々が開発する際には、できるだけ以下の点に気を配ると良いはずだ。

- UI の画像サイズを複数用意しておき、DPI 設定に合わせて切り替える (主に iOS アプリケーションなどで用いられる)
- UI をベクター画像で作成する
- UI のスクリーン上の位置はできるだけ比率計算等を用いて算出する

twitter: @Toritama_TAR

2016 年 12 月 07 日

あとがき

東京都市大学コンピュータ技術研究会です。三年生が引退し一・二年生での活動となりました。一年生はまだまだこれから技術をつけ、二年生は各々分野が違うことに手を付けていっています。今年度は夏コミと冬コミ両方の参加となり締め切り前はいつも以上に慌ただしかったですが無事に部誌を発行できて一安心です。

来年度に向けてコンピュータ技術研究会では新たな活動を試みています。何かの機会に発表できる場があればと思っております。

本誌を読んでいただきありがとございました、面白いと感じたり興味を持った内容などがあれば幸いです。次回は学園祭、2017 年 6 月 10・11 日に開催される東京都市大学横浜キャンパスでの横浜祭です。ぜひお越しください！

発行者: 東京都市大学コンピュータ技術研究会
表紙: ねーつ
発行: 2016 年 12 月 9 日
印刷: 株式会社 栄光
Web: http://www.tcu-ctrl.jp/