Charamaker2 取扱説明書

By With Ball

Charamaker2 とは Charamaker2 およびその他付属のやーつでなんとなく 2D ゲームが作れるエンジンと言って差し支えないものである。こんな文書よりソースコードのコメントみた方がいいのであしからず。

hyojiman

表示マンは画面の表示および、音声の出力をしてくれる画面全般のオブジェクトである。drawings の配列、haikeidraws,effects のリストを持ち、これらに継承オブジェクトを入れ、hyoji()を呼び出すことで画面を描画する。他にも、カメラ座標 camx,背景色 HR,G,B などを持つ。bairitu をいじることでズームインアウトができる。とにかく覚えてほしい手順は以下である。

fileman.makehyojiman()で hyojiman を作成する。
new picture などで描画オブジェクトを作成し、picture.add(hyojiman)で追加する。
hyoji()を呼び出し描画する。
また、描画の順番は Z の値の大きさで決定される(背景は除く)。
背景はスクロール割合でソートされ、後ろの物ほど背景色が加算され、薄く見えるようになる。
hyojiman.playoto(),playbgm()で wav ファイルを読み込み、音を鳴らすことができる。調整できるのは音量だけで、再生速度などはいじることができない。fileman.playoto()でも同様に音はならせるが、hyojiman で音を鳴らす場合、hyojiman.hyoji をしない限り、音は再生されないので保存とかしとく時に便利。

線などの簡単な図形の描画はサポートしていない。ただ追加することは可能だろう。
picture はビットマップを読み込み、(0,254,254)をデフォルト透明色とする。サイズを自由に変形し、回転を加えることもできるので十分だと思うよ。

effects には effectchara というオブジェクトが入る。これは後述するがモーションを適用できる picture 群であり、時間経過で自動的に消える。逆に言えばゲーム内時間と関係なく、

描画する度に形が変化する。あまり使わないほうが良い。

fileman
ファイル入出力や、画面表示リソースの提供をする要すぎる人。他にも乱数 r を格納していて確率をあつかうこともできる。static なクラスである。

覚えてほしい関数は以下。

絶対に

fileman.setingup()を呼び出す。
この時に、画質の設定もする。画面の縦横比は settingup を呼び出した瞬間のものを維持する。画面のサイズはいつでも自由に変えることができるが、hyojiman の座標軸は画面のサイズごときでは変化しない。例えば、formsize が(1000,1000)で settingup が呼び出された場合、hyojiman の倍率,カメラを変更しなければ、hyojiman には xy 共に 0~1000 の物体を描画することになる。
また、loadfiletoka()で
.¥tex¥　(テクスチャー(.bmp)を置くと決められてるフォルダ)
.¥motion¥　(キャラクターのモーション(.c2m)を置くと決められてるフォルダ)
.¥character¥　(キャラクター(.c2c)を置くと決められてるフォルダ)
.¥oto¥　(音(.wav)を置くと決められてるフォルダ(oto¥bgm¥が bgm を置く場所))
のファイルを全てロードすることができる。
普段はその bmp が使用されることになってから読み込むのでゲーム開始時にロードしておくとフリーズしなくなる。
このロードを行ってる場合、ウィンドウはフリーズ状態になるのでタイトル画面を表示させておこう。

ちなみに、どのファイルにしても仕様上拡張子は書く必要がない。

リソースがなくても使用できるテクスチャがいくつかある。
redbit,bluebit,greenbit,(以下 bit 省略)
white,gray,black,yellow,cyan,purple,aqua,brown,crimson,pink,orange,indigo
である。

ランダム関数に必要なのは以下のとおりである。
setrandomseed()リセットする

percentin()なんとなく引数%の確率で true を出す。
whrandhani()0~引数までの整数を返す。r.Next()%(w+1)とほぼ同義
r.next()ランダムな整数をくれる。
r.nextdouble()0~1 の double をくれる。

Character

その名の通り、これを作るためにこのエンジンは生まれた。picture の集合体である

Chracter.onepicturchara()によって一枚の picture を持つキャラクターを簡単に作成することができる。
キャラクターの扱い方は以下の手順のとおりである
1.作る。
fileman.loadcharacter(“”)で読み込む、new character(),Character.onepicturechara()であ s く制するなどする。
2.表示する
character.resethyoji(hyojiman)で picture を hyojiman に登録する。character.sinu()で消すことができる。
3.モーションとか追加するかも
モーションは new motion()で箱を作り、
motion.addmoves()で移動するや、拡大する、回転するなどの動きを追加する。
そして character.addmoton(motion)でつい追加する。また、fileman.ldmotion()で読み込むこともできる。
4.動かす
character.frame()でキャラクターを動かす。effectchara であれば、2.のステップは（基本）不要で、hyojiman.hyoji()で動く。

キャラクターはまず物理的なあたり判定に使用して欲しい枠があり、その下に関節である setu()のツリーがぶら下がっている。setu はそれぞれ setu のリスト、picture 一枚、親の setu からの距離を持っている。

また、キャラクターは kijyun というものを持っており、setkijyun により現在の姿が基準に登録される(作成時にも呼び出されている)。kijyun はその名の通り、基準でこれをもとにキャラクターの反転、拡大縮小などを行うことができる。
character.resettokijyun()で基準にリセット、
character.refrestokijyun()で角度をそのままにサイズ、透明度、テクスチャーの種類を基準

に戻すことができる。

そして、キャラクターはモーションクラスによって体系的に動かすことができる。関節のそれぞれの角度を直接変更することもできるが、それだと、ツリーの下にある関節に回転の影響を及ぼすことができないので変な動きになる。

キャラクターは Charamaker2.exe で編集することができる。
motion もソースコードで書くのとほぼ同じように作ることができる。
work.addmoves(new moveman());って感じで書く。

inputin

ボタンマウスの入力を管理する。説明もしたくないぐらい単純。
inputin.input(Key)で入力をし、(これは Form.Keydown のイベントと接続する。テンプレートプロジェクトを見ろ！)
inputin.setpointer()でマウスの hyojiman に対応した座標をセットする。(inputin.x で参照)
inputin.topre()で入力を過去のものとする。

inputin.ok(Key,itype)でキーが押されているか確認する。
itype には三種類ある。　down up ing だ。察してくれ。down は押された瞬間だけ True,up は……

IPC というもので入力を変換することができる。W キーを押したらジャンプだとする。コードでは
if(i.ok(Keys.W,itype.down))jump();
と書いておき、やっぱ Space でジャンプにして―なと思ったときは
Space を W に変換する IPC を作り、それを inputin.input(Key,IPC)にぶち込む。すると万事 OK になる。

Shape

これはあたり判定の時に使う奴でマジで説明しても意味ないので飛ばしたい。あえて言うなら三種類あるってことかな
Rectangle 長方形。

Circre　楕円
Triangle　三角形
それぞれ width,height で形を決定する。triangle が特殊で、長方形のどの辺を底辺とするか決めた後(-2,-1,1,2 で決定)頂点がどのあたりにあるかの割合を 0~1 で決定することになる(0.5 で二等辺、1or0 で直角三角形)一応全ての三角形を作れるはず。

Gameset1

こちら、ゲームを作るのに便利なものが付属している。

Scene

プロジェクトテンプレートを見て察して欲しいが、
scenemanager.s=new Scene()とし、

scenemanager.s.frame()とすればすべて解決する奴である。継承して上手くやんな。

new Scene(scenemanger);で作成し、
scene.start()でシーンマネージャの s を自分に書き換える。また、end()で次のシーンである Scene.Next を自動で start()させる。
それぞれ、start が正しくなっていれば onstart(),onend()を呼び出す。

例えばゲーム画面をポーズしたい場合は end()を呼び出さずに、こうする

```
var Paus=new Scene(scenemanager);
Paus.next=gamegamen;
Paus.start();
//ポーズ終わったら
Paus.end();
//このとき、gamegamen.onstart()は呼び出されない。
```

onend,onstart,frame を継承して使う。

標準で備え付けている物は hyojiman(new Scene()で作成される。),next(end()の時に呼び出される次のシーン)SceneManager ぐらいのものである。

FP

これまた static なクラス。ゲーム内で使用するキャラクターのパラメータ(バランス調整しやすくなる)や、文章(ローカライズができるかも！？)を別途 txt に保存する際に使用する。
始めに seting()でパラメータとして読みこむテキスト、文章として読み込むテキストファイルを措定する。
文章の場合
名前:内容
パラメータの場合
名前:1.00
てな感じで txt 内に書く。それぞれ行を変えることはできない。しかし、文章の場合は¥n を改行にしてくれる。
ゲーム内に読み込む場合は FP.GT(“名前”),FP.PR(“名前”)で取得する。
指定したパラメータが存在しない場合はエラー、指定した文章がない場合は "テキストがないよって昨日ママに言われたんだ……”になる。

あとは後述のエンテティの起動シミュレーションがある。

SD

セーブデータを使うクラス。継承してカスタマイズもできる。標準では音量や、キー入力のセッティングのセーブデータが反映されている。
SD.savesave でセーブ、SD.loadsave()でセーブデータをロードし、現在使用中のセーブデータとして SD.S にロードしたものを格納する。
SD.S はスタティックでパブリックなのでどこからでも参照できる。

継承しカスタマイズする手引きは

```
[Serializable]
class SDD:SD
{
 //追加したいデータを入れる
 public score;
 override public void resetIPC()
```

{
　/*好きなデフォルトの入力変換を書く(こちらで用意しているセッティングではデフォルト入力以外のボタンを割り当てれないよって言えばいいのか？)*/
}
static public new SDD.S{get {return (SDD)_S;}set{_S=value;}}
//これがないと継承した SDD として参照ができない。
}
を書く。このセーブデータは.¥save¥に格納され、checkyou と一緒に保存される。checkyou ってのは保存ができないセーブデータを作ってしまったときに書き込ませないための仕組みである。これがないため多くの悲劇が起こった。必要ないかもしれないが、伝統的に機能を置いてある。
class をシリアライズして保存しているので頑張ればセーブデータをいじることができるかもしれない。しかし、大変すぎる。

以下は無視してもいい。

Entity

キャラクターとあたり判定を一緒くたに扱うためのクラス。EntityManager に格納され、一斉に frame される。あたり判定を司る ataribinding,物理特性を司る buturiinfo で構成されている。他にもイベントなるものを司る機関もある。

var E=new Entity(character,ataribinding,buturiinfo);で宣言し、
E.add(Entitiymanager);で追加する。
この時、character 以外の引数はコピーされるため使いまわしができる。character も使いまわしたい場合は別途コピーしてくれ。

buturiinfo は物理を司る。速度、加速度、重さ、摩擦、空気抵抗、反発、あたり判定の分類を持つ。ものにぶつかったときの処理や、反発処理もここが担当しているため、継承して回転したぶつかったときに速度を生むようにすることとかも可能かもしれない。

ataribinding は ataribindingrecipie から作成される。character の関節と図形を結びつけ、あたり判定を形成する。その中でも結びつける関節が””つまりキャラクターそのものである場合、そのあたり判定は物理的動作に用いられる。逆に””と図形が関連付けられていなければ物理的な動き行われない。Entity のソースコードを見てもらえばわかるが、
ataribinding.frame()で現在のキャラクターとあたり判定が同期される。Entity は二つの

ataribinding を持っており、一つは 1 フレーム前のあたり判定を表している。

イベントを司る機関、EEV にイベントがたくさん宿っている。ここに一時的な関数を書いてカスタマイズしてもよい。

しかし、hp がなくなると死ぬような StatusedEntity を作りたい場合は継承をした方がいいような気がする。その手順としては、
Status クラスを作成(HP とか書く)
Status を持つ Sentity を Entity を継承して作る。その際、frame をオーバーライドし、hp が 0 になったときに this.remove()を呼び出すようにする。また Sentity は SEEVENTER を持つようにしたいので
override void setEEventer
{
 _EEV= new SEEVEnter(this);
}
public new SEEventer EEV{get{return _EEV;}}
を記述する。
あとはコピーするときのコンストラクタも書いたり。
EEventer を継承し SEEVENTER とする、新しくダメージを受けた時や、HP が 0 になったときなどのイベントを追加する。その際 EventArgs も Sentitiy に対応したものを作る。

Waza

Entitiy に一時的に特性を与えるためのクラス。
var www=new waza();で宣言し、
www.add(Entity)でついかする。
継承したのち frame とかをオーバーライドして作るわけだけど、
framed というイベントがあるので
framed+=(a,b)=>
{
 好きな文を書く。
}
こういった感じで一時的な関数を作ってカスタマイズもできる。

EntityManager

Entity を束ねているクラス。
EntityDatabese を持ち、これがエンテティを管理している。
EntityManager.moves,overweights,atarerus,atarens で重さがカンストしていないもの（動けるもの)、重さがカンストしている物(地形)、物理的に当たり判定を持つ者もたないものを取得できる。
他にも特定のタイプの Entity を getTypeEnts で取り出すこともできる。このタイプの分類は Entitymanager に追加された時点で分類され、istyped で Enittymanager に入っている Entity であれば型が何かというものを制限できる。
EntityManager.frame()を呼び出すことで全ての Entity の frame が行われる。あたり判定の処理も行ってくれるので Scene では EntityManager.frame();hyojiman.hyoji();がセットになるだろう。

Templette

ゲームにはなっていないが、なんとなくのテンプレートを用意した。参考にしてほしい。
特に SEtting の部分だ。
これはキーコンフィグの部分などはかなりめんどいのでこのテンプレートのまま使うのがいいだろう。UI 部分も作るのが面倒すぎてわけわかんなくなったのでこのまま使用して問題が発生するようであれば作り変えてくれ。使うためには用意したリソースを張り付けて使ってほしい。