

OG2D ライブラリリファレンス

OGSystem.h

```
class Circle
```

円系オブジェクト用 class

```
Circle r,CenterX,CenterY;
```

内部入力方法

float*3,int*3,Circle の3つのコンストラクタが用意されている。

生成時に全値は0に初期化される。

```
int main()
```

```
{
```

```
    Circle a;
```

```
    Circle b = {5.0f,10.0f,3.0f}
```

```
}
```

```
a.CenterX = 0.0f,a.CenterY = 0.0f, a.r = 0.0f;
```

```
b.CenterX = 5.0f,b.CenterY = 10.0f b.r = 3.0f;
```

```
class Box2D
```

長方形系オブジェクト用 class

```
float x,y,w,h;
```

内部入力方法

flob.t*4,int*4,Box2D の3つのコンストラクタが用意されている。

生成時に全値は0に初期化される。

内部関数

Offset ←学内ライブラリ仕様、引数の値を x,y にプラスする。

OffsetSize ←w と h の値に x と y をプラスしてくれる。頂点を合わせる用

```
int main()
```

```
{
```

```
    Box2D a;
```

```
    Box2D b = {3.0f,5.0f,32.0f,32.0f};
```

```
    b.OffsetSize();
```

```
}
```

```
a.x = 0.0f,a.y = 0.0f,a.w = 0.0f,a.h = 0.0f;
```

```
b.x = 3.0f,b.y = 5.0f,b.w = 32.0f,b.h = 32.0f;
```

OffsetSize 後

```
b.x = 3.0f;b.y = 5.0f,b.w = 35.0f,b.h = 37.0f;
```

```
class Vec3
```

3次元ベクトル用 class

```
float x,y,z;
```

内部入力方法

flob.t*3,int*3 の2つのコンストラクタが用意されている。

生成時に全値は0に初期化される。

```
int main()
```

```
{
```

```
    Vec3 a;
```

```
    Vec3 b(1.0f,3.0f,5.0f);
```

```
}
```

```
a.x = 0.0f,a.y = 0.0f,a.z = 0.0f;
```

```
b.x = 1.0f,b.y = 3.0f,b.z = 5.0f;
```

```
class Vec2
```

2次元ベクトル用 class

```
float x,y;
```

内部入力方法

flob.t*2,int*2 の2つのコンストラクタが用意されている。

生成時に全値は0に初期化される。

```
int main()
```

```
{
```

```
    Vec2 a;
```

```
    Vec2 b(1.0f,3.0f);
```

```
}
```

```
a.x = 0.0f,a.y = 0.0f;
```

```
b.x = 1.0f,b.y = 3.0f;
```

※ここまでのものは= {}でも()どちらでも入力できる。

```
class Mat4
```

4*1の行列用配列。内部処理等は値を入れているだけなので割愛。

```
class Mat4x4
```

4*4の行列用配列。以下略。

class Window

GLFW の機能を使って Window を生成管理するための class。

void createWindow(int,int,char*,bool)で生成できる。

引数は1つ目から Window の横サイズ、Window の縦サイズ、

Window の名前、フルスクリーンかどうか。

このクラスは **OGTask.cpp** にて使用しているので書かずとも自動的に生成されるようになっている。変更する場合は **OGTask.cpp** の **Initialize** 内の値を変えることで可能である。

ここで生成しているのはキーボード入力に関係しているため、ほかの箇所での記述は保証しない。この class を使うことはほとんどないだろう。

class FPS

FPS の計測を行うための class

指定した FPS にフレームレート进行いじりたい場合にこの class を使用するが基本的に 60fps で動作するようになっている。

この二つはライブラリ側で使用しているものなので特にいじることはない。

class CollisionBox

四角形型のオブジェクトの当たり判定を行う class

Box2D hitBase 内に判定したい値をいれ、**Box** との判定を行う場合は **bool hitBox**、円との判定を行う場合は **bool hitCircle** を使い引数に判定を行いたいオブジェクトの **Collision** をいれる。接触している場合は **true** が返ってくる

回転を反映させる場合は判定を行う前に **void Rotate** で回転の値を引数にいれる。内部的にラジアン値に変更しているので角度を **float** 型で渡せば動く。

class CollisionCircle

円型オブジェクトの当たり判定を行う class

Circle hitBase 内に判定したい中心点と半径をいれ、**CollisionBox** と同じ形で判定をとることができる。円型は回転しても変わらない値のため回転は用意されていない。楕円の判定を行うことはできない。

`class Texture`

画像を扱うための class

`void TextureCreate(string)`で画像を読み込む、引数には画像のファイル名を入力。`void Draw(Box2D,Box2D)`で描画を行う。引数1には頂点情報、2には画像ファイルの座標情報を送る。`Draw`の前に `void Rotate(float)`を行うことで画像を回転させることができる。こちらでも内部的にラジアンに変換してくれるので角度をそのまま送ればよい。`void Finalize()`で画像データの解放を行う。

※入力するファイル名について。このファイルは **OG2D** ライブラリ用に用意されているので最初から `data/image` のファイルパスは入力されている。ここに画像データをいれ、引数には画像のファイル名のみを入力することで正しく動作する。

`namespace DG`

この名前空間には基本的に数学的計算の関数が入っている。

`float ToRadian(float)`はラジアンの値に変換してくれる。

そのほかは行列操作のためのものなので時間がなかったので割愛します。

内容は `_OGSystem.h.cpp` に記載されているので使用する場合はここで確認できる。

キー入力 `class Input` に移動しました。

※`Input` class は `EngineSystem` 内で管理しているので `EngineSystem` の内容で確認できる。

~~`namespace Key`~~

class Object

ゲームオブジェクトを生成する場合に使用する class

`void createObject(Objform,Vec2,Vec2,float)`でオブジェクトを生成する。

Objform には作るオブジェクトが何型なのかを入力、円なら **Ball**、立方体なら **Cube**、それ以外の者には **Non** を入力。(2018/03/20 時の仕様)、1 つ目の **Vec2** には座標情報であるポジションの値を入力、2 つ目の **Vec2** には縦と横のサイズを入力、**float** には初期時の回転の値を入力。

`bool hit(Object)`では判定を取りたい相手の **Object** を引数に渡すことでどちらの型でも判定を行ってくれる。どちらか一方でも **Non** の場合は強制的に **false** を返す。

ここで判定を行う場合は **Collision** には **hit** 関数で自動的に値を送るので **Collision** 側の値をいじる必要はない。

Ball の当たり判定用の半径は **Scale** の **x** が反映されるようになっている。(2018/03/20 現在の仕様)

この **Object** が重力によって落下行動を行うかどうかの設定をするために **bool Gravity** が存在するが現在 2018/03/20 では未使用。

class Sound

サウンドを読み込み再生させる class

コンストラクタ時。もしくは `void createSound(string,bool)`で初期化を行う。

引数 1 は使用するサウンドのファイル名

引数 2 はループの **OnOff** 設定。何も入れない場合 **false** が入っている。

`void play()` :再生

`void stop()` :停止

`void pause()` :一時停止

`void volume(float)` :音量の変更(0~1)

`void pitch(float)` :ピッチの変更(1 が通常状態)

`void looping(bool)` :ループの変更

`bool isplay()` :再生中かどうかを **bool** で返す

`float currenttime()` :再生時間を返す

`float duration()` :再生してるサウンドの全体時間を返す

<p>class EngineSystem</p> <p>この class は基本的に起動から終了まで同じ動作をするものを管理する class。</p> <p>System 内で gameEngine として宣言しており、これを使う形となる。</p> <p>2018/04/01 現在に存在する機能</p>
<p>Cameraclass、camera 視点を変更するための class</p> <p>void gameEngine->camera->MovePos(Vec2)</p> <p>引数の値分現在の位置からカメラを移動させる。</p> <p>void gameEngine->camera->SetPos(Vec2)</p> <p>引数の値の位置にカメラをセットする。</p> <p>Vec2 gameEngine->camera->GetPos()</p> <p>現在のカメラの位置を取得する。</p>
<p>Windowclass、window に関する情報が入った class</p> <p>void gameEngine->SetWindow(int,int,char*,bool)</p> <p>生成するウィンドウの設定を行う。</p> <p>※今までと仕様は同じですが設定する内部の場所が変更。</p>
<p>Inputclass、入力処理を行う class</p> <p>bool gameEngine->in.down(in,int)</p> <p>down,up,on の 3 種類の入力判定。in には Input 内に宣言されている in の中身を使う。詳細は下記記述。int には接続ゲームパッドの番号を入力。0~15。</p> <p>input 内の入力判定ではゲームパッドが存在しない場合は自動的にキーボードのみを取得する形になっている。</p> <p>bool gameEngine->in.key.down(int)</p> <p>down,up,on の 3 種類の入力判定。keyboard のみの判定を取る場合はこれを使う。</p> <p>引数には Input::KeyBoard::A(Input::KeyBoard::Key の中身)もしくは GLFW_KEY_A のどちらかを送ることで判定をとることができる。Input::KeyBoard::Key の詳細は下記記述。</p> <p>bool gameEngine->in.pad[].down(int)</p> <p>float gameEngine->in.pad[].axis(int)</p> <p>down,up,on,axis の 4 種類。axis は-1~1 のスティック倒れている角度に応じて値を返す。ゲームパッドのみの入力の判定を取る場合にこれを使う。</p> <p>bool の引数には Input::GamePad:: BUTTON_A(Input::GamePad::Pad の中身)もしくは GLFW_JOYSTICK_1 を送る</p> <p>float の引数には Input::GamePad::AXIS_LEFT_X(Input::GamePad::AXIS の中身)を送る。</p> <p>Input::GamePad::AXIS と Input::GamePad::Pad の詳細は下記記述。</p> <p>※ゲームパッドのみの判定には例外処理がないので外部で行う必要がある。</p> <p>※bool Input.Pad_Connection に 1 つ以上パッドがあるかの情報があるのでこれを使う。</p>

※2018/04/09

GameEngine(input)の変更点

class 名の変更

input	in
keyBorad	key
gamePad	pad

key,pad,in すべてに共通する on,up,down の引数を名前空間 In にまとめました。

現状の形としては

```
gameEngine->in.down(In::B1)
```

```
gameEngine->in.key.down(In::Z)
```

```
gameEngine->in.pad[0].down(In::BUTTON_A)
```

となります。

gameEngine(camera)の変更点

```
void Move(Vec2) → void MovePos(Vec2)
```

に変更になりました。

Camera の追加関数

```
void MoveSize(Vec2) :カメラのサイズの移動
```

```
void SetSize(Vec2) :カメラのサイズ設定
```

```
Vec2 GetSize() :カメラのサイズを返す
```

in	キーボード番地	ゲームパッド番地
B1	Z	A
B2	X	B
B3	C	X
B4	V	Y
CU	↑	↑
CR	→	→
CD	↓	↓
CL	←	←
L1	Q	LB
R1	E	RB
D1	ENTER	BACK
D2	SPACE	START
SR	N	右スティック押し込み
SL	B	左スティック押し込み

ゲームパッド番地名は **Logicool** コントローラの配置名を使って記載しております。ほかのコントローラと名前が一致しない場合がございます。

Key	キーボード番地
A~Z	A~Z
ENTER	Enter
SPACE	Space
ESCAPE	ESC

Pad	ゲームパッド番地
BUTTON_A	A
BUTTON_B	B
BUTTON_X	X
BUTTON_Y	Y
BUTTON_L1	LB
BUTTON_R1	RB
BUTTON_BACK	BACK
BUTTON_START	START
BUTTON_L3	左スティック押し込み
BUTTON_R3	右スティック押し込み
BUTTON_U	↑
BUTTON_R	→
BUTTON_D	↓
BUTTON_L	←
AXIS_LEFT_X	左スティック X 座標値
AXIS_LEFT_Y	左スティック Y 座標値
AXIS_RIGHT_X	右スティック X 座標値
AXIS_RIGHT_Y	右スティック Y 座標値

※タスクの追加について

タスクの処理や追加、初期のタスクの設定は WinMain.h と OGTask で行っている。

追加や削除を行う場合は、

- 1、WinMain.h に存在する名前空間にタスクを登録、削除をする。
- 2、OGTask.h の include ファイルに制作したタスクのヘッダーファイルの記入、削除。
- 3、class _OGTK の public の中に新規タスクの class を生成、削除。
- 4、OGTask.cpp の _myGameUpdate、_myGameRender、_myGameFinalize 内の switch 内に生成した class の処理の記述、削除。

※break 等を忘れないこと。

- 5、_myGameInitialize の nextTask に最初に起動したいタスクの記入。

これでタスクの追加を行うことができる。タスクからタスクへの移動は Update 内の引数にて行う。

SampleTask, SampleTask2 についてはタスクの記述方法、処理方法。

各種読み込みと使用の例を記述している。

不要になったら削除してください。

履歴	2018/03/21	金子 翔	記入
履歴	2018/03/23	金子 翔	追記
履歴	2018/04/01	金子 翔	追記
履歴	2018/04/09	金子 翔	追記