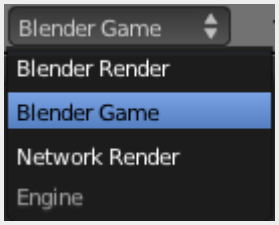
Blender Game Engine 2.5 概要

Blender はインタラクティブな 3D アプリケーションの開発を可能にするゲーム・エンジン（ Game Engine ）を内蔵しています。Blender ゲーム・エンジン（以下 BGE）はパワフルでハイレベルな開発ツールです。ゲーム開発を主たる目的としていますが、インタラクティブな 3D 構造ツアーや教育における物理の研究のような、インタラクティブな 3D ソフトウェアの開発にも利用できます。

標準状態では Blender のレンダー・エンジンは "Blender Render" となっており、ゲームエンジンを利用するためにはレンダー・エンジンを "Blender Game" に切り替える必要が有ります。



もしレンダー・エンジンを "Blender Game" に切り替えていない場合、"Bullet Physics" の設定タブのような、幾つかの機能を利用できません。

Game Engine を利用する

BGE の根幹てきな構造は ロジック・ブリック（ Logic Bricks ）に示されています。ロジック・ブリック（ Logic Bricks ）の目的はいかなるプログラミング言語の知識も使う事無く用意にインタラクティブなアプリケーションの実装を容易に行えるようにする事です。ロジック・ブリック（ Logic Bricks ）には、センサー（ Sensors ）、コントローラー（ Controllers ）、アクチュエーター（ Actuators ）という 3 つの種類が存在します。それぞれに関する詳細は以下に示します:

- センサー（ Sensors ）
- コントローラー（ Controllers ）
- アクチュエーター（ Actuators ）

Python を用いたゲームの記述を好まれる場合でも、ゲーム・エンジンは Blender の他の実装から独立した python api 2 56 0 Python API を持っており、ゲーム操作を行うためのスクリプトを記述する際に利用できます。これは Python Controller の作成と、その Python Controller を Python スクリプトにリンクする事で可能になります。

- Android用ゲーム開発ガイド(eng)
- Android用Blenderのビルド(eng)
- GLESを使うBlenderのビルド(eng)

Standalone Player

スタンドアロンプレイヤーを使えば、Blender のシステムを読み込まずに Blende Game を実行できます。Blender の詳しい知識がない人にもゲームを配布できるようになります（また、無許可の編集もできなくします）。「Save as Game Engine Runtime」は便利なアドオンで、使用前に読み込んでおく必要があります。

次の手順で、動作しているゲームのスタンドアロン版を作れます。

- *File* » *User Preferences* » *Addons* » *Game Engine* » *Save as Game Engine Runtime* » *Install Addon（ボタン）*
  （**Save as Default** ボタンを押して Blender の起動時にアドオンを毎回読み込むようにもできます）。
- *File* » *Export* » *Save as Game Engine Runtime* » *（ディレクトリとファイル名を指定）* » *Save as Game Engine Runtime（ボタン）*

これで、適切な .exe ファイルを実行するとゲームが起動します。必要なライブラリはアドオンが自動的に読み込むことに注意してください。

作成したゲームのライセンスに興味をお持ちなら、関係する議論について Licensing（eng）をお読みください。

Exporting...

ゲームを他のコンピューターに export するときは、ゲームの runtime や付随するライブラリなどのために新たなディレクトリを作ってください。それから、「ディレクトリをまるごと」、目的のコンピューターに転送します。

BGE（Blender Game Engine）におけるカメラ

ゲームエンジンの内部で、カメラはデフォルトの特性（動き方）をさせることもできるし、オブジェクトを追跡することもできます。それにはオブジェクトまたはその頂点を親にするか、Camera アクチュエータを使います。

Note

どのオブジェクトでもカメラにすることができます（後述）。またカメラにオブジェクトを追跡させる場合、どのオブジェクトでも使うことができます（Empty など）

カメラを使うには、カメラの視点（0 NumPad）でゲームエンジンをスタートさせます。またカメラがゆがむのを防ぐため、常にカメラオブジェクトがビューいっぱいに表示されるようにズームしてください。

デフォルトのカメラ

デフォルトではカメラは動きません。

オブジェクトを親にする

カメラにオブジェクトを追跡させます。まずカメラを選択し、つぎにオブジェクトを使い選択し、CtrlP → *Make Parent*。

この場合、オブジェクトが回転するとカメラも回転します。それを避けるには、後述のように頂点を親にしてください。

頂点を親にする

最も簡単な方法として、オブジェクトを選択して⇆ Tab で *Edit* モードに入り、頂点を選択して *Object* モードにもどります。

次に、何もオブジェクトを選択していない状態でカメラを選択し、⇧ Shift を押しながらオブジェクトを選択します。そして⇆ Tab で *Edit* モードに入り、CtrlP を押して *Make vertex parent* を選びます。

これでカメラはオブジェクトを追跡するようになり、またオブジェクトが回転してもカメラは回転しません。

オブジェクトをカメラにする

どんなオブジェクトでもカメラにすることができます。またどのようなプロパティを持っていても良いです。

オブジェクトをカメラにするには、*Object* モードでオブジェクトを選択し、{{Shortcut|ctrl|pad0}}を押します。

元に戻すには、カメラを選択して再度{{Shortcut|ctrl|pad0}}を押します。

レンズシフト

Blender のインターフェースではカメラビューの XY 平面上でビューをずらすオプションがあります。これはビデオプロジェクターのレンズシフト機能に似ています。この機能は通常、Y 軸に沿って画像を動かします（それによって、たとえばテーブルにプロジェクターを置いたときに映像の下半分がテーブルにかかってしまう、ということを防ぎます）。

残念ながら、このパラメータはゲームエンジンで使えません。

投影を操作するには、Python でカメラの投影マトリックスを修正します。

```
import bge
scene = bge.logic.getCurrentScene()
cam = scene.active_camera
```

```
# get projection matrix
camatrix = cam.projection_matrix
#modifying the camera projection matrix by modifying the x and y terms of the 3rd row to obtain a shift of the rendered area
camatrix[2][0] = 2*shiftx
camatrix[2][1] = 2*shitfy
cam.projection_matrix = camatrix
```

shiftx と shifty は視野を単位とします。よってビューを半分上にずらすときは、shifty を 0.5 にセットします。

カメラの投影マトリックスの属性は初期化スクリプトが実行されたあとで設定してください。ゲームが始まった瞬間に実行すると、投影マトリックスがぐちゃぐちゃになってしまいます。

関連項目

- Fisheye Dome camera.

Blender Game Engine のドームモード

この機能は、インタラクティブなプロジェクトを没入型ドーム環境として視覚化できます。拡張性の高いツールとするため、完全なドーム、端を切り落としたドーム(ッ前方および後方)、プラネタリウム、球面ミラーによるドームをサポートしています。

この機能はBlender 2.49 で追加されました。これはPaul Bourkeが開発したマルチパス・テクスチャ・アルゴリズムを使用し、**SAT**(Society for Arts and Technology)の後援のもと、**SATMetalab** immersion research program[1]の一環としてDalai Felintoによって実装されました。簡潔に説明すると、これはシーンを 4 回レンダリングして、平行投影カメラで見たときに魚眼レンズのような投影に見えるようにデザインされたメッシュに貼り付けます。

注意
最大の大きさで投影するために、Blender をフルスクリーンモードにして使ってください。 そのためには、コマンドラインで-W をつけて Blender を起動します。また、トップメニューを消すためにすべてのウィンドウを結合します。そうでなければ、(Ctrl+Up)で最大化しても完全に全画面を使えません（トップバーメニューは約 20 ピクセルあります)。

Dome Camera Settings

Dome Type

メニューで使用するドームカメラの種類を選択します。それぞれのカメラの概要は後述します。また設定項目も後述します。

- Fisheye Dome
- Front-Truncated Dome
- Rear-Truncated Dome
- Cube Map
- Full Spherical Panoramic

使用できる設定項目はカメラの種類によってかわります:

Resolution

バッファの解像度。この数値を下げるとスピードが上がりますが、品質は下がります。

Tesselation

4 がデフォルトです。これはメッシュの分割レベルです（Cube Map では使えません）。

Angle

視野の角度。範囲は 90°から 250°です（Fisheye と Truncated で使えます）。

Tilt

水平軸におけるカメラの回転（Fisheye と Truncated で使えます）。

Warp Data

映像を湾曲させるのにカスタムメッシュを使います。

Fisheye Mode

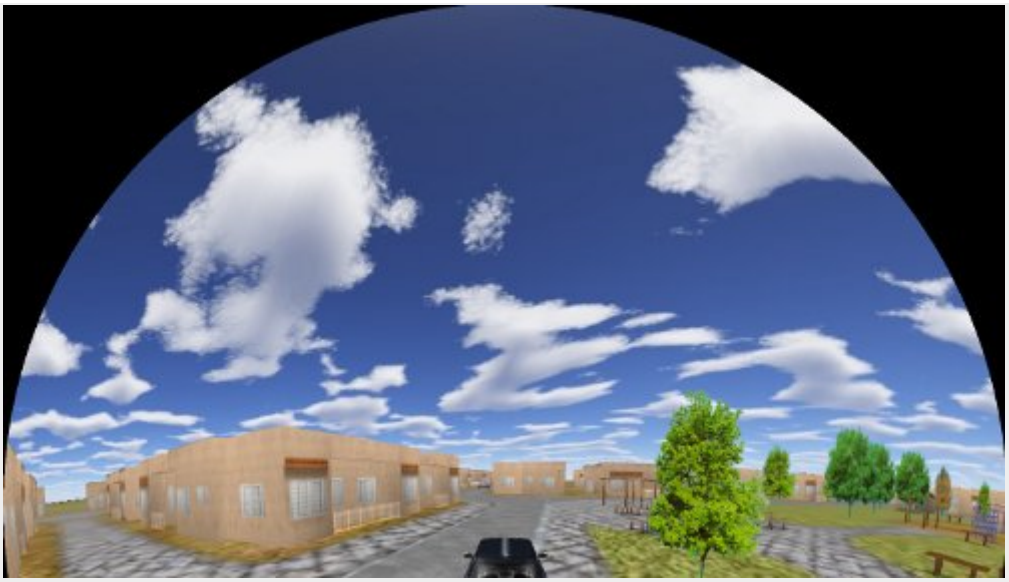
視野 90°から 250°までの正射影魚眼映像。

- 90°から 180°までの場合、4 回レンダリングします。
- 181°から 250°までの場合、5 回レンダリングします。

Front-Truncated Dome Mode

端を切り落としたドーム用にデザインされています。このモードでは魚眼映像がウィンドウの上部および左右に接するように配置されます。

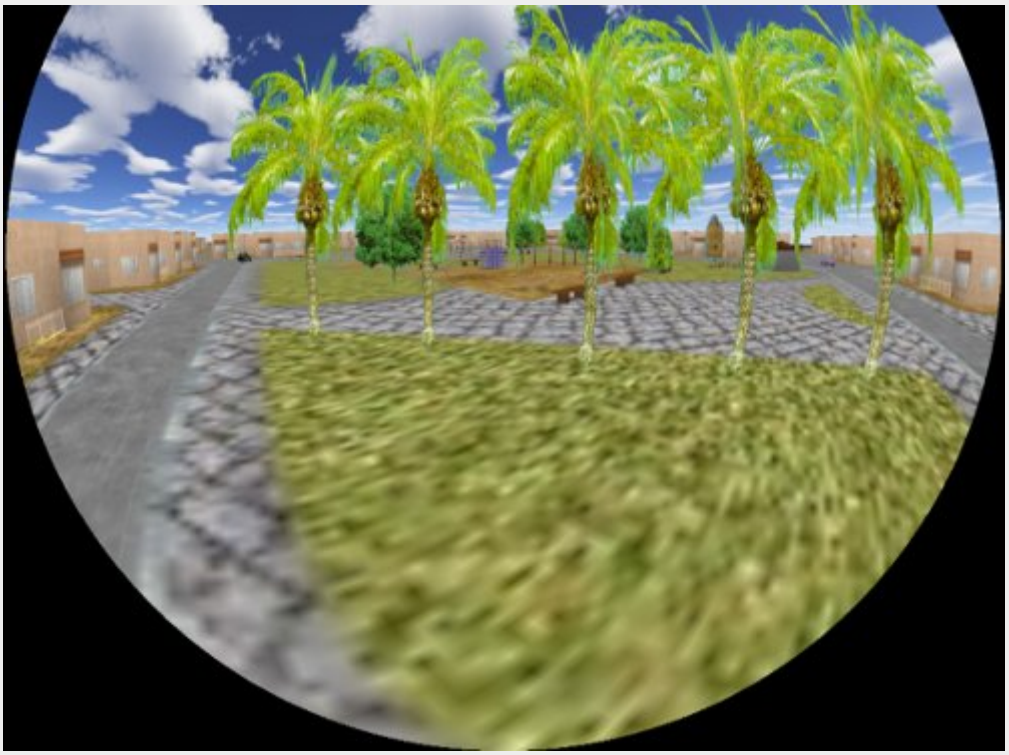
- 90°から 180°までの場合、4 回レンダリングします。
- 181°から 250°までの場合、5 回レンダリングします。



Rear-Truncated Dome Mode

端を切り落としたドーム用にデザインされています。このモードでは魚眼映像がウィンドウの下部および左右に接するように配置されます。

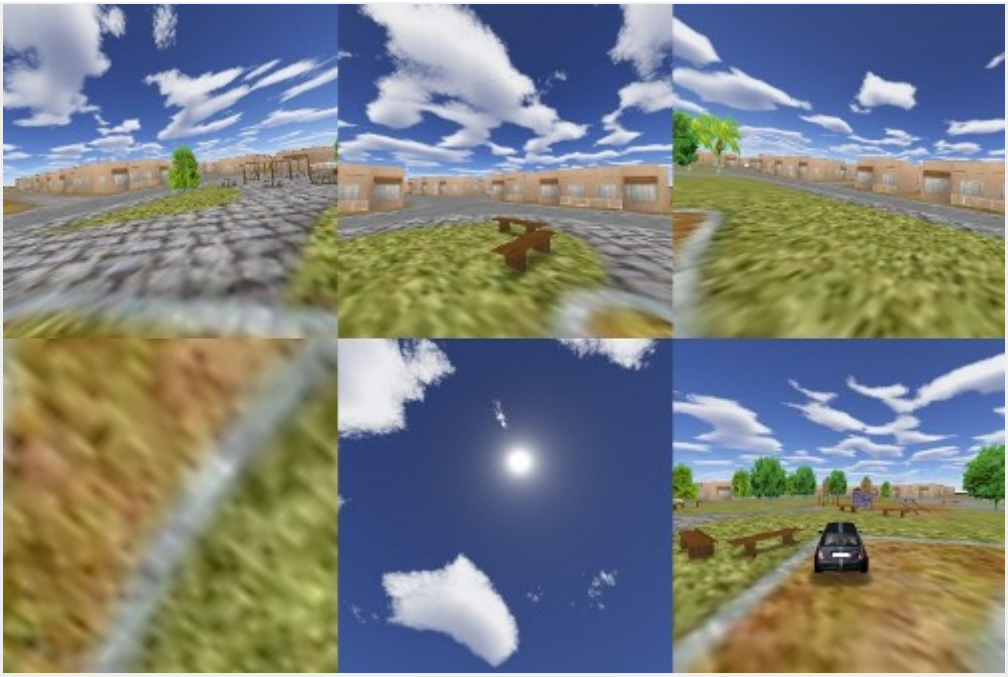
- 90°から 180°までの場合、4 回レンダリングします。
- 181°から 250°までの場合、5 回レンダリングします。



Cube Map Mode

このモードはキューブマッピングのプリレンダーアニメーションに使えます。

- これは 6 回レンダリングします。イメージの順番は Blender の環境マップのフォーマットと同じです。
    - o　１段目：右、後、左
    - o　２段目：下、上、前



Spherical Panoramic

球面パノラマモードです。

- 6 回レンダリングします。
- 上部と下部は **Definition** を 5 以上に設定すると正確になっていきます。

Warp Data Mesh

さまざまな投影環境を使用する場合、フラットなスクリーンとは違ってシンプルな遠近法ではないイメージが必要となることがあります。円筒ディスプレイ、プラネタリウムドームのための新しい方式、バーチャルリアリティ用の直立型ドームなどのために補正されたイメージです。

メッシュのフォーマットについてはPaul Bourkeの記事 を参照。

こういったイメージを生成するために、特別なフォーマットを用いています。

テンプレートは::

```
mode
width height
n0_x n0_y n0_u n0_v n0_i
n1_x n1_y n1_u n1_v n1_i
n2_x n1_y n2_u n2_v n2_i
n3_x n3_y n3_u n3_v n3_i
(...)
```

1 行目はメッシュに適用されるイメージの種類です:**2** = **rectangular**(矩形), **1** = **radial**(放射状) 2 行目はメッシュの寸法をピクセルに換算した数値。残りの行はメッシュの結節点(nodes)です。

それぞれの行は **x y u v i** からなっています。(x,y)は正規化されたスクリーン座標、(u,v)はテクスチャ座標、i は乗算する強度ファクターです。

x の範囲は(-1 x アスペクト比)から(アスペクト比)まで、y は-1 から 1、u と v は 0 から 1、i は 0 から 1 です。負の数が入っているとそのノードは描画されません。

- Warp Mesh データファイルを使うためには、ファイルを作成してテキストエディタに追加する必要があります。
- テキストエディタを開きます（テキストエディタウィンドウ）。
- テキストエディタで、メッシュデータファイル（たとえば myDome.data）を開きます（Text/Open または Alt+O）。
- ゲームフレーミングセットアップに移動します（Window Types/Buttons Window/Scene Page または F10）。
- ドームモードを有効にします。
- Warp Data 欄にファイル名（たとえば myDome.data）を入力します。

Warp Mesh（イメージを湾曲させるのに使うメッシュ）の作成にはmeshmapperというインタラクティブなツールが使えます。これはPaul Bourke's Warpplayerソフトウェアパッケージに含まれています（フルバージョンが必要です）。

サンプルファイル

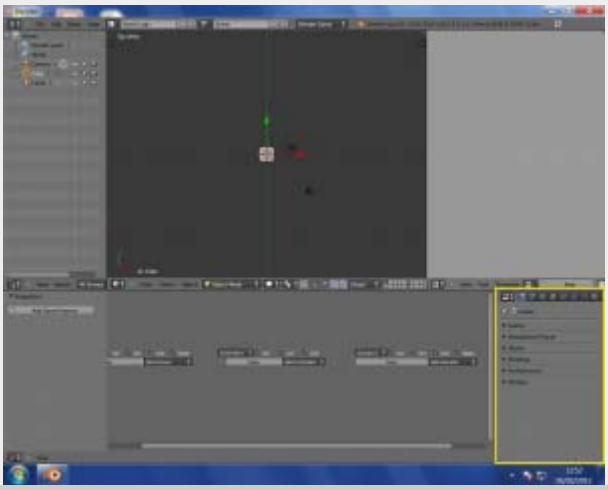
Spherical Mirror Dome 4x3, Truncated Dome 4x3, Sample Fullscreen File 4x3, Sample Fullbuffer File 4x3.

Note

重要: ビューポートはカンバスの横／縦の比を使って計算されます。したがって、画面のサイズが変わると必要なメッシュファイルも変わります。また正しい縦横比を得るために、Blender をフルスクリーンで使用してください。

Camera Editing

（カメラの編集）



Camera Properties

Blender Game で使われるカメラは、ゲームの描画や表示方法に幅広い影響を持ちます。設定のほとんどは、ゲームで使用するカメラのプロパティパネルから操作します。

レンダリングエンジン

これらの操作を行う際には、レンダリングエンジンが Blender Game であることを確かめてください。そうでないと、実際の画面と説明が一致しなくなるでしょう！

カメラのプロパティでは六つのパネルが利用できます。それぞれ、いつもの三角ボタンを使って伸縮できます。各パネルの機能の詳細を以下で説明します。

Game

Game Panel

Start ボタン – Game Engine を開始します。

Standalone Player

Standalone Panel

単体の（Standalone）ゲームプレイヤーの情報を設定するパネルです。単体のゲームプレイヤーは Blender なしで実行できます。詳細は Standalone Player をご覧ください。

Fullscreen –（フルスクリーン）

Off – 新しいウィンドウでゲームを実行します。

On – フルスクリーンでゲームを実行します。

Resolution（解像度）

X

フルスクリーン表示での描画領域の X 方向の大きさです。

Y

フルスクリーン表示での描画領域の Y 方向の大きさです。

Quality（画質）

Bit Depth（ビット深度）

フルスクリーン表示で、各ピクセルの色の表現に使われるビット数です

FPS

フルスクリーン表示での毎秒ごとのフレーム数です

Framing

描画領域にどのように表示を合わせるかを指定します

Letterbox

ウィンドウに描画領域全体を表示し、残った部分を Bar の設定色で埋めます。

Extend

視界を広げることでウィンドウに描画領域全体を合わせます。

Scale

描画領域全体がきっちりウィンドウにおさまるように X か Y 方向に伸縮します

Bar Color

描画領域の周囲の余白を塗りつぶす色です。デフォルトは黒です。

カラー方式を選んでから（RGB, HSV or Hex）、スライダーやホイールを使って余白の色を選びます。

Stereo（立体視）

Stereo パネル

ゲームの立体画像を作るのに使われる、立体視の方式を選びます（もちろん、この立体表示はスタンドアロンプレイヤーで画像をレンダリングするのにも使われます）。

None

立体視効果なしで一枚の画像をレンダリングします。

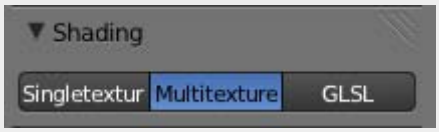
Stereo

専用アルゴリズムにより、立体視用の二重の画像をレンダリングします。利用できるオプションの詳細は 立体視カメラ をご覧ください。

Dome

没入型のドーム状環境の内部からゲームを見る機能です。利用できるオプションの詳細は ドームカメラ をご覧ください。

Shading



Shading Panel

ゲームのレンダリング時に使われる陰影処理の方式を指定します。マテリアル と テクスチャ で使われる陰影処理機能は、Blender Game Engine のものと基本的に同じです。ただ、リアルタイム表示による制約で、一部機能が利用できないことがあります。

Single Texture

一枚のテクスチャ用の機能を使います。

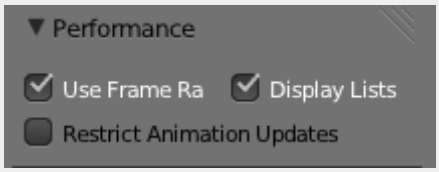
Multitexture

複数枚テクスチャの陰影処理を使います。

GLSL

GLSL の陰影処理を使います。リアルタイムな画像レンダリングの可能性があるなら、常に GLSL を使うとよいでしょう。

Performance



Performance パネル

Use Frame Rate

レンダリングのフレーム数を可能な限り増やす替わりに、フレームレートを重視します。

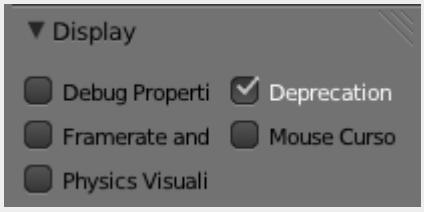
Display Lists

GPU にジオメトリを維持してレンダリングを高速化します。

Restrict Animation Updates

アニメーションの更新数をアニメーションの FPS に制限します（パフォーマンスはよくなりますが、滑らかな再生に問題を起こす可能性があります）。

Display



Display パネル

Game Engine 実行時の表示のオプションがあります。

Debug Properties（デバッグ用のプロパティ）

ゲーム実行中にデバッグ用に印をつけられたプロパティを表示します。デバッグ用プロパティは入力として指定する必要があることに注意してください（例えば Game Properties パネルで i ボタンを使います）。Blender 内でゲームを実行するときだけ利用できます（スタンドアロンプレイヤー版では使えません）。

Framerate and Profile

ゲーム実行中にフレームレートと分析情報を表示します。Blender 内でゲームを実行するときだけ利用できます（スタンドアロンプレイヤー版では使えません）。

Physics Visualization

ゲーム実行中に、物理演算で使われる境界線や相互作用を表示します（Blender 内とスタンドアロン版の両方で使えます）。

Deprecation Warnings

Python API の非推奨機能を使うと警告を表示します。Blender 内でゲームを実行するときだけ利用できます（スタンドアロンプレイヤー版では使えません）。

Mouse Cursor

ゲーム実行中にマウスカーソルを表示します（Blender 内とスタンドアロン版の両方で使えます）。

ステレオカメラ

ステレオカメラは、特殊な眼鏡を通すと 3 次元的に見えるような画像をレンダリングできます。そのために、カメラからわずかな距離だけ離れた二つの画像がレンダリングされます。それによって、われわれの両目から見た映像をシミュレートします。ステレオイメージを見るときには、片方の目は一方の画像だけを見るように制限され、もう片方の目はもう一方の画像だけを見ます。われわれの脳はその二つのイメージを合成して 3 三次元の物体を見ているように感じるのです。

Stereo Settings

Stereo Mode

ステレオカメラの種類をせっていします。これはあとで説明します。

Eye Separation

この値は非常に重要です。これは二つのイメージがどのくらいはなれているか、どのくらい「3D」であるか、ということを決めます。あまり高い値にすると頭痛が起こったり目が疲れたりする可能性があります。

Stereo Modes

Quad Buffer

Above-Below

Interlaced

Anaglyph

Side by Side

Vinterlaced

機能一覧

Blender ゲームエンジンで使用できる機能のリストです。

Editor

- プラットフォーム: GNU/Linux, Windows, Mac
- ロジックブリック・ワークフロー
- Python スクリプティング
- シャドーマップ・ベイキング
- ライトマップ・ベイキング
- 埋め込みプレイヤー (WYSIWYG)
- ランタイムプロファイリング
- 物理効果の視覚化(デバッグ)

Engine

- マルチプルシェーダモード
    - GLSL shading
    - Multitexture
    - Singletexture

- GLSL shaders (2.0, vertex and fragments – Python or built-in).
    - Normal map
    - Parallax map
    - Specular map
    - Color map
    - Detail map
    - Environment reflect (cube, sphere)
    - Vertex color
    - Phong shader
    - Dynamic shadow (Soft shadows and Stencil shadows; harmony branch)
- 物理計算: Bullet
    - Culling system: sphere, box, frustum (through Python)
    - Collision system: capsule, box, sphere, cylinder, cone, convex hull, triangle mesh
- サウンド: WAV, MP3, Ogg
- テクスチャ: tga(+alpha), png(+alpha), jpg, VideoTexture support
- テキスト: Bitmap font and Font objects
- パスファインディング: Navigation mesh with Obstacle Avoidance; Recast and Detour
- アニメーション: Mesh deformation with bones (armatures)
- ミップマッピング
- アンチエイリアシング(MSAA)

不完全および搭載予定の機能

- GL particle system (See xEmitter)
- Terrain engine
- GUI (see BGUI)
- LOD (level of detail – via Python).
- Portals (not yet).
- Network (via Python).
- Multi uv coordinate (via Python).

Licensing of Blender Games

（Blender ゲームのライセンス）

Blender はオープンソース・ソフトウェアであるため、Blender Game Engine と スタンドアロンプレイヤーで配布されるゲームのライセンスは複雑です。この記事ではその問題点と、考えられる解決策を挙げます。

Blender は GNU General Public License（GPL）の下、オープンソース・ソフトウェアとして配布され、所有されます。簡単に言えば、Blender システムそのものが誰にでも利用できる状態である限りは、あなたが Blender を使って作成したもの（スクリプト、テクスチャ、レンダリングした芸術作品等）をすべて所有できます。詳細は http://www.blender.org/education-help/faq/gpl-for-artists/ をご覧ください。

Standalone Player License

（スタンドアロンプレイヤーのライセンス）

残念ながら、このルールは Blender のスタンドアロンプレイヤーで実行されるゲームや芸術作品には当てはまりません。ゲームを配布するには実行形式のファイル（run time）を作る必要があります。実際には Blender の .blend ファイルがスタンドアロンプレイヤーの「内部」に置かれます。このとき、Blender Game Engine に関連する機能だけを含む最小限の Blender が使われます。こうしてできた実行形式ファイルは元のプログラムの「派生品」（すなわちあなたのファイルとスタンドアロンプレイヤーの混成物）に分類されるため、GPL ライセンスである必要があります。

Distributing Games

（ゲームの配布）

あなたの作ったゲームを妥当なライセンス保護をもって配布する方法が、四つ考えられます。

> 1）あなたの作った Blender Game をライセンスで一切保護しません。ほんとうにライセンスが必要ですか？ 古いことわざを思い出してください：「真似されてこそ本物」
> 2）何も問題がないふりをします。BGE ベースのゲームを配布した誰かを、Blender Foundation が告訴するようなことは非常に考えにくい話です。
> 3) Blender Game を実行するための 外部システムを使います。例えば BPPlayer や Gamekit があります（ただ、完全なテストはされていません）
> 4）基本の .blend ゲームファイルを開始させる Game Actuator を使います。まず、別の.blend ファイルを作り、あなたの作ったゲームの内容全体を読み込んで実行するゲームアクチュエータ（Game Actuator）を作ります。次にこのアクチュエータを起動する常時センサ（Always Sensor）を作ります。これで問題を回避できます。あなたの作ったファイルはスタンドアロンプレイヤーの「外部」にあり、GPL で公開する必要はなく、「合法的に保護された」状態です。あなたのゲームはこの仕組みで完全には保護されていませんが、一般に配布されているゲームの大半は似たような保護レベルであると言えるでしょう。.blend ファイルにはアクセス可能ですが、あなたの希望するライセンスの対象外の目的に使われるとは限りません。

（謝辞：このページは Dalai Felinto のブログ記事の内容を基にしています）

Logic, Properties and States

（ロジック、プロパティ、ステート）

ゲームロジックは、ゲームエンジン内のデフォルトのスクリプト記述階層です。ゲーム内の GameObject はそれぞれ、シーン内での動作を制御するためのロジカル要素（ロジックブリック）を保管できます。ロジックブリックは組み合わせて、シミュレーションの進行を決めるようなユーザー定義動作を行わせることができます。

Logic Bricks

（ロジックブリック）

ゲームロジックの主要部分はグラフィカルなインタフェースである Logic Editor を通して組み立てられ、プログラミングの詳細知識を必要としません。ブロック（または “ブリック”）は、 あらかじめプログラムされた機能を表し、ゲーム/アプリケーションを作成するために微調整して組み合わせることができます。 システムは 3 つの部分に分割されます: Sensors、Controllers および Actuators です 。センサーは、衝突、キー押下、マウス操作などの物事の発生を感知します。コントローラーはセンサーの出力に対して論理的な操作を実行し、接続されているアクチュエータを、発動条件が整ったときに起動します。アクチュエータはシミュレーションと直接相互作用します。ゲーム内でこれができるのはアクチュエータだけです（Python コントローラー、および物理演算のような他のシミュレーション要素を除く）。

Properties

（プロパティ）

プロパティ（属性）は他のプログラミング言語における変数のようなものです。ゲーム全体を通じて、もしくは特定のオブジェクト/プレイヤー（名前など）用に使われる、値の保存や取得に使われます。しかしながら、Blender Game Engine ではプロパティはオブジェクトと関連づけられています。プロパティにはさまざまな型があり、Logic Editor の専用の領域で設定できます。

States

（ステート）

もう一つの便利な機能がオブジェクトの ステート（eng）（状態）です。シミュレーション実行中はいつでも、オブジェクトのロジックのうち、現在のステートに属するものがすべて実行されます。ステートは、動作のグループわけに使うことができます。例えばあるオブジェクトのステートが「寝ている」「起きている」「死んでいる」のいずれかであるとすれば、この三つの状態それぞれに、違った動作をするロジックを用意できます。オブジェクトのステートは、そのオブジェクトのコントローラーのロジックブリックで作成し、表示し、編集します。

Category: Game engine

アクチュエータ

アクチュエータは、移動、オブジェクトの生成、サウンドの再生などのアクションを行います。接続しているコントローラから一つ（またはそれ以上）の正パルスを受け取ると、アクチュエータはその関数を初期化します。

アクチュエータ用のロジックブロックは Logic Editor を使って構築し、変更することができます。この手順の詳細はアクチュエータの編集にあります。

以下のアクチュエータが使えます:

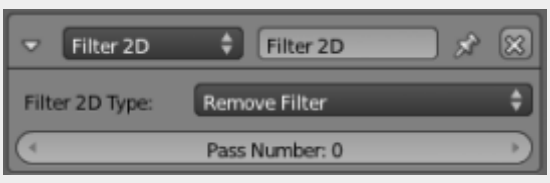
Action	アーマチャのアクションを操作します。アーマチャが選択されているときのみ表示されます。
Camera	滑らかにオブジェクトを追跡するオプションがあります。主にカメラオブジェクトのためのものですが、どんなオブジェクトでも使えます。
Constraint	オブジェクトの位置、距離、回転を制限します。ゲームの中の物理をコントロールするのに便利です。
Edit Object	オブジェクトのメッシュの編集、オブジェクトの追加または削除、またメッシュを変更することもできます（すると衝突メッシュも再生成されます）。
Filter 2D	画面をセピア色にしたり青みがかった感じにしたりなどの特殊効果フィルタです。
Game	ゲーム全体の操作、リスタート、終了、ロード、セーブなどをします。
Message	メッセージを送り、受け取ったオブジェクトを起動したりできます。
Motion	オブジェクトを動かしたり回転させたりします。瞬間移動させるのではなく、物理的に押したり回転させたりするオプションもあります。
Parent	オブジェクトを親にしたり、それを解除したりできます。
Property	オブジェクトのプロパティを操作します。設定、加算、コピーなど。
Random	ランダムな値を生成します。プロパティに格納することもできます。
Scene	.blend の中のシーンをあつかいます。そのシーンをゲームのステージや UI、背景などに使います。
Sound	ゲーム内でサウンドを再生するのに使えます。
State	オブジェクトのステートを切り替えます。
Steering	オブジェクトの経路検索のオプションです。
Visibility	オブジェクトの表示・非表示を変更します。

2D Filter アクチュエータ

2D Filter はイメージフィルタのアクチュエータです。最終的にレンダリングイメージに対して適用されます（FPS ゲームのサンプル画像を提供してくれた人々に感謝）。

2D Filter の種類

次のフィルターが用意されています



2D Filters

Custom Filter

Invert

Sepia

Gray Scale

Prewitt

Sobel

Laplacian

Erosion

Dilation

Sharpen

Blur

Motion Blur

Remove Filter

Disable Filter

Enable Filter

どのフィルターもパラメータはひとつだけです

Pass Number

フィルターを使うパス番号

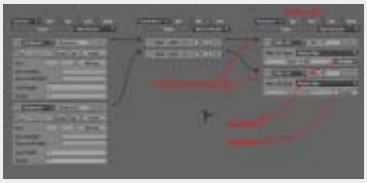
フィルターの詳細については以下の説明をご覧ください。

Motion Blur

（モーションブラー）

Motion Blur を使って動きの表現をするには、以前のフレームの情報が必要です。図"2D Filters: Gmae Logic"は Blender のウィンドウに表示されたロジックブリックです。

2D Filters: *Motion Blur*.



2D Filters: Game Logic.

フィルタを有効にするには:

1. 適切なセンサとコントローラを追加する。
2. *2D Filter* アクチュエータを追加する。
3. ドロップダウンリストから *Motion Blur* を選択する。
4. *Value*(モーションブラーの度合い)

フィルタを無効にするには:

1. 適切なセンサとコントローラを追加する。
2. *2D Filter* アクチュエータを追加する。
3. ドロップダウンリストから *Motion Blur* を選択する。
4. *Enable* ボタンを押して無効モードにする。

Python コントローラを使ってモーションブラーを有効にすることもできます:

```
from bge import render
render.enableMotionBlur(0.85)
```

無効にするには:

```
from bge import render
render.disableMotionBlur()
```

註釈

グラフィックハードウェアと OpneGL がアキュムレーションバッファをサポートしている必要があります (glAccum 関数)。

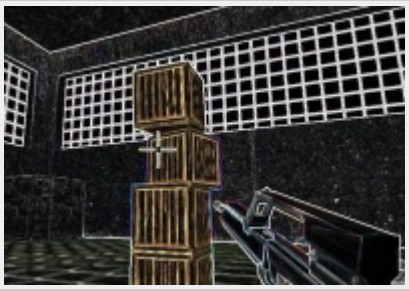
Built-In 2D Filters

(組み込みの 2 次元フィルター)

2D Filter アクチュエータで見ることができるフィルタはすべて同じアーキテクチャを用いています。すべての内蔵フィルタは最終的なイメージを生成するためにフラグメントシェーダを使います。よって、ハードウェアがシェーダをサポートしている必要があります。

2D Filters: *Motion Blur*.

2D Filters: *Sepia*.



2D Filters: *Sobel*.

Blur、*Sharpen*、*Dilation*、*Erosion*、*Laplacian*、*Sobel*、
Prewitt、*Gray Scale*、*Sepia*、*Invert* が内蔵フィルタとして使えます。これらのフィルタはいくつかのパスで使用できます。

フィルタを使うには:

1. 適切なセンサとアクチュエータを作成する。
2. *2D Filter* アクチュエータを作成する。
3. 使用するフィルタ、たとえば *Blur* を選択する。
4. フィルタを適用するパスの番号を指定する。

特定のパスでフィルタを削除するには:

1. 適切なセンサとアクチュエータを作成する。
2. *2D Filter* アクチュエータを作成する。
3. *Remove Filter* を選択する。
4. フィルタを削除するパスの番号を指定する。

特定のパスでフィルタを無効にするには:

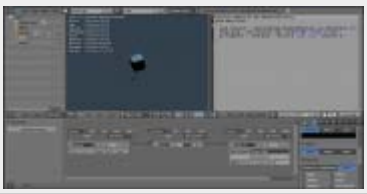
1. 適切なセンサとアクチュエータを作成する。
2. *2D Filter*アクチュエータを作成する。
3. *Disable Filter*を選択する。
4. フィルタを無効にするパスの番号を指定する。

特定のパスでフィルタを有効にするには:

1. 適切なセンサとアクチュエータを作成する。
2. *2D Filter*アクチュエータを作成する。
3. *Enable Filter*を選択する。
4. フィルタを無効にするパスの番号を指定する。

Custom Filters

(カスタムフィルター)



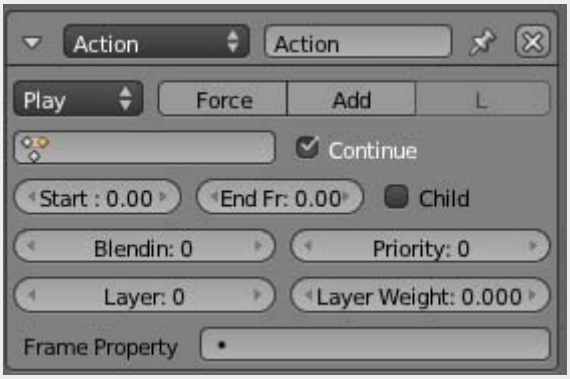
2D Filters: *Custom Filter*.

カスタムフィルタによって、GLSL を使って好きな 2D フィルタを定義することができます。使い方は内蔵フィルタと同じですが、*2D Filter*アクチュエータで *Custom Filter*を選択する必要があります。そしてテキストエディタでシェーダプログラムを書き、アクチュエータにスクリプト名を設定します。

青セピアの例:

```
uniform sampler2D bgl_RenderedTexture;
void main(void)
{
  vec4 texcolor = texture2D(bgl_RenderedTexture, gl_TexCoord[0].st);
  float gray = dot(texcolor.rgb, vec3(0.299, 0.587, 0.114));
  gl_FragColor = vec4(gray * vec3(0.8, 1.0, 1.2), texcolor.a);
}
```

Action アクチュエータ



Action Actuator

Action アクチュエータはアーマチュアが選択されているときのみ表示されます。アクションはアーマチュアに格納されているからです。

アクチュエータ共通のオプションについては こちら をご覧ください。

専用オプション:

Action Playback Type

Play –
TRUE パルスを受けた時 IPO の開始フレームから終了フレームまで一度だけ再生します

Ping Pong

Flipper
TRUE パルスを受けた時 IPO の開始フレームから終了フレームまで一度だけ再生します(FALSE パルスを受けた時は逆方向に再生します)。

Loop End
TRUE パルスを受けた時 IPO の終了フレームから開始フレームまでを繰り返し再生します

Loop Start
TRUE パルスを受けた時 IPO の開始フレームから終了フレームまで繰り返し再生します

Property

Action

使用するアクション

Continue

オン・オフを切り替える際、前回のフレームを記憶します。無効にすると毎回最初から再生されます。

Start Frame

再生を始めるアクションのフレーム

End Frame

再生を終えるアクションのフレーム

Child Button

Blendin

モーションブレンディングに使うフレーム数

Priority

実行の優先順位。低い数値にすると、高い数値のアクションをオーバーライドします。2 つまたはそれ以上のアクションが同時に起こるときは、優先順位の高いチャンネルはスタックの中で低い位置になければいけません。

Frame Property

アクションのカレントフレーム番号をこのプロパティに登録します。

Property

ここに指定したプロパティに基づいてアクションの位置を決めます。再生モードが Property の場合のみです。

Layer

Layer Weight

Camera Actuator

カメラにオブジェクトを追従させます。

オプション

アクチュエータ共通のオプションについては こちらをご覧ください。

専用オプション:

Camera Actuator

Camera Object

カメラが追従する Game Object の名前

Height

Game Object の中心からの高さ。カメラがこの位置に留まり続けようとします

Axis

カメラの追従する軸(X か Y)

Min

Game Object を追従するカメラの最小距離

Max

Game Object を追従するカメラの最大距離

Damping

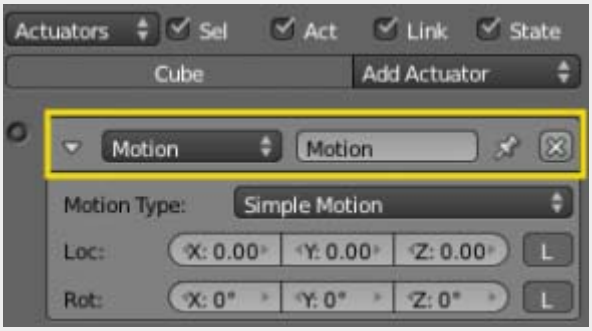
カメラを対象の後ろに動かす強制力。0 から 10 の範囲で指定します。値が高いほどカメラが制約範囲内（Min/Max/Height によるもの）に位置調整するのが速くなります。

関連項目

- Camera
- Fisheye Dome camera

アクチュエーター共通オプション

（アクチューエーターの共通オプション）



アクテイエーターの共通オプション

すべてのアクチュータに共通する、次のようなボタン、入力欄、メニューがあります：

三角形のボタン

アクセス情報を 1 行に折ったみます。拡大し展開します

クチュアエータ種類メニコニコュー

アクセルエイターの種類を選びます

アクセルエターター

アクセッションの名前です。ユーザーが定義できます.Python でアクセスにアクセスする際に使います。選択されたオブジェクトが重ならない名前にする必要があります。

x **ボタンン**

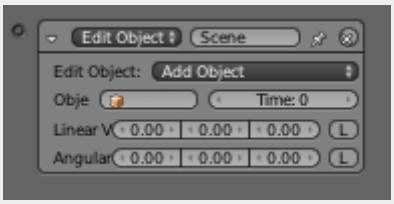
アクチューターを削除します。

オブジェクト編集（オブジェクト編集）アクチューター

編集オブジェクト（オブジェクト編集）アクチューターで、ゲーム中のオブジェクトの設定を編集できます。

オスープの共通シヨンはアクチュエーター共通のオスープションを参照してるくださんい。

特別なオプション:



オブジェクトの編集

オブジェクトの編集(オブジェクト編集)

オブジェクトの編集(オブジェクト編集)アクチューターのオプションのメニュー

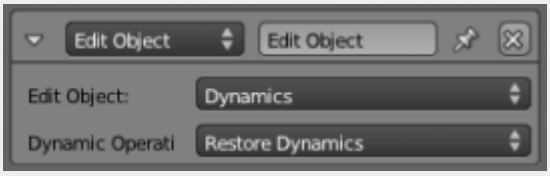
ダイナミクス(力学処理)

トラック先(追尾)

メッシュ交換(メッシュ切替え)

エンドオブジェクト(エンドオブジェクト)

オブジェクトを追加する(オブジェクト追加)



編集オブジェクト(オブジェクト編集)アクチューター – ダイナミクス(力学処理)

ダイナミクス(力学処理)

オブジェクトの力学処理のオプションを設定する*ダイナミック操作(力学処理の操作)*メニューを提供します。

質量を設定する(質量を設定)

現在のオブジェクトの物理演算用の質量を設定できます(範囲:0 – 10,000)。

剛体を無効にする(剛体力学を無効化)

剛体状態を無効にします。衝突が無効になります。

剛体力学を有効にする(剛体力学を有効にする)

危険状態を有効にします。衝突が有効になります。

力学を一時停止する(力学処理を一時停止する)

オブジェクトの力学(速度)を中断します。

復元力学(力学処理を再開)

オブジェクトの力学(速度)を再開します。

編集オブジェクト(オブジェクト編集)アクチュウター – Track to(追跡)

トラック(追跡)

そのオブジェクトは他のオブジェクトを 2D または 3D で「注視」するようにします。

オブジェクト(オブジェクト)

追求するオブジェクト。

時間(時間)

ターゲットオブジェクトの方を向くことになるフレーム数(範囲:0–2000)。

3D(トグル)

2D(X、Y)または 3D(X、Y、Z)での追跡を切替えます。

*アップ軸(上方向の軸)*メニュー

上を指して(X、Y、Z)軸(デフルトは Z 軸です)。

*トラック軸(トラック軸)*メニュー

ターゲットオブジェクトを指し示す(X、Y、Z、–X、–Y、–Z)軸(デフルトは Y 軸)。

*上軸(上方向の軸)メニュー*と*トラック軸(トラック軸)*メニューで同軸を指定した場合、機能しなくなります。

編集オブジェクト(オブジェクト編集)アクチューター – 置換メッシュ(メッシュ変更)

メッシュを置き換える(メッシュ変更)

メッシュを他の物で置き換えます。表示と物理演算の双方向を一緒に、別々に変更できます。

メッシュ

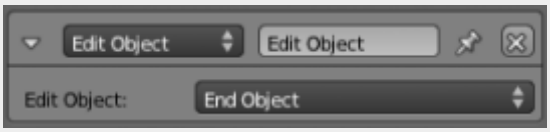
現在のメッセージを置き換えるメッシュ名。

Gfx ボタン

表示するメッセージを置き換えます。

物理ボタン

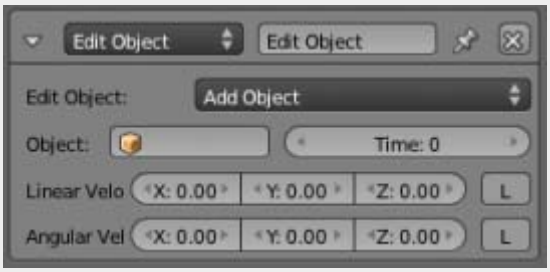
物理演算に使用するメッシュを置き換えます(「組合」は除きます)



オブジェクト編集(オブジェクト編集)アクチューター – End Object(エントロブジェクト)

エンドオブジェクト(エンドオブジェクト)

現在のオブジェクトを消失させます。



オブジェクトを編集する(オブジェクト編集)アクチューター – オブジェクトを追加する(オブジェクト追加)

オブジェクトを追加する(オブジェクト追加)

オブジェクトを現在のオブジェクトの中心に追加します。追加されるオブジェクトは、別の非表示レイヤーに配置されて必要があります。

オブジェクト(オブジェクト)

追加するオブジェクト名。

時間(時間)

そのオブジェクトは消えるまで生存し続ける時間(フレム数).0 で無限。

線速度(線速度)

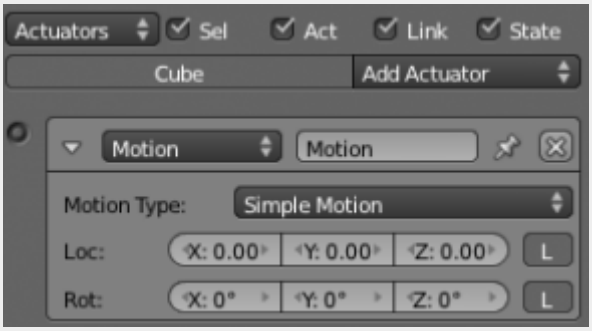
作成されたオブジェクトの線形速度。オブジェクト指向時の、初期速度の設定に便利です。

角速度(角速度)

作成されたオブジェの角速度。

Actuator Editing

(アクチュエータの編集)



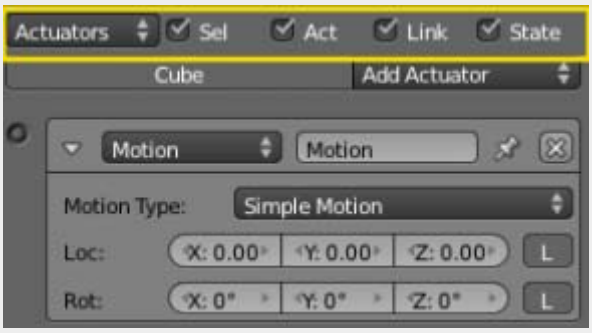
典型的なアクチュエータのあるアクチュエータ列

アクチュエータは Logic パネルの右の列で作り、編集できます。このページでは通常の列コントロールと、個々のアクチュエータで一般的に使われるパラメータについて説明します。

図は、アクチュエータを一つ持つ、典型的なアクチュエータ列を示しています。列の最上部にある列見出しには、現在の Game Logic にあるすべてのアクチュエータを操作するためのメニューやボタンが表示されています。

Column Heading

（列見出し）



アクチュエータ列の見出し

列見出しのアクチュエータ列には、アクチュエータの選択やその表示詳細度を設定するためのコントロールがあります。必要のないアクチュエータを隠して、必要なものを見つけやすくするのに役立ちます。それぞれ別々に操作できます。

Actuators

Show Objects	全オブジェクトを展開します。
Hide Objects	名前を表示するバーだけになるように全オブジェクトを折りたたみます。

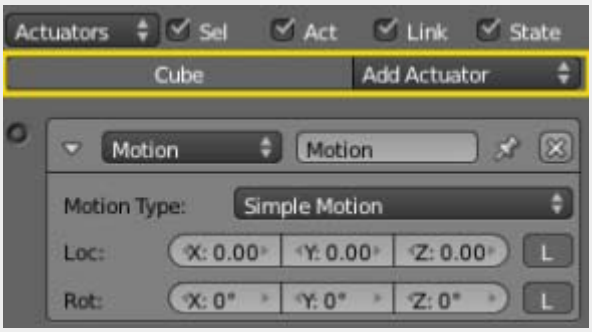
Show Actuators	全アクチュエータを展開します。
Hide Actuators	名前を表示するバーだけになるように全アクチュエータを展開します。

ヘッダーにある 4 つのボタンを使って、表示されるアクチュエータを抽出できます:

Sel	選択中のオブジェクトの全アクチュエータを表示
Act	アクティブオブジェクトに属するアクチュエータのみ表示
Link	コントローラーに繋がっているアクチュエータのみ表示
State	アクティブなステートを持ったコントローラーに繋がっている、アクチュエータのみを表示

Object Heading

（オブジェクト見出し）



オブジェクト見出し

アクチュエータ列には、アクチュエータがオブジェクトごとにグループ分けされて並んでいます。デフォルトでは、選択中のオブジェクトのアクチュエータがすべて並びますが、列の見出し部分にある抽出機能を使って変更可能です。

オブジェクトのアクチュエーター覧の見出し部分には、二つのボタンがあります:

Name

オブジェクト名

Add

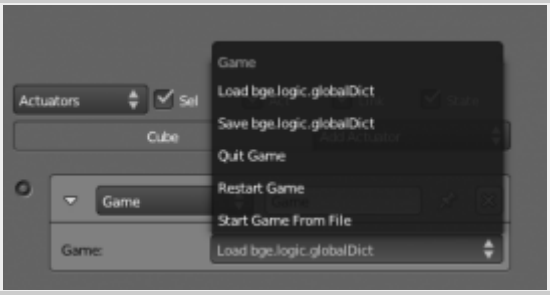
クリックすると利用可能なアクチュエータの選択メニューが現れます。選択して、オブジェクトに新たにアクチュエータを追加します。利用可能なアクチュエータの一覧は、Actuators をご覧ください。

Category: Game engine

Game アクチュエータ

Game アクチュエータはゲームに関連した機能、たとえばリスタートや終了、ロードなどを行います。

Game actuator



Game

種類

Load bge.logic.globalDict

.bgeconf から *bge.logic.globalDict* をロードします。

Save bge.logic.globalDict

.bgeconf に *bge.logic.globalDict* をセーブします。

Quit Game

このアクチュエータが起動すると Blender player が終了するようにします。

Restart Game

このアクチュエータが起動すると Blender player がリスタートするようにします（ファイルをリロードします）。

Start Game From File

このアクチュエータが起動すると Blender player は指定されたパスの.blend ファイルをスタートします。

File

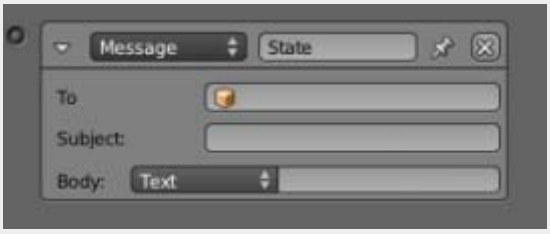
ロードする.blend ファイルのパス。

使用する際の注意

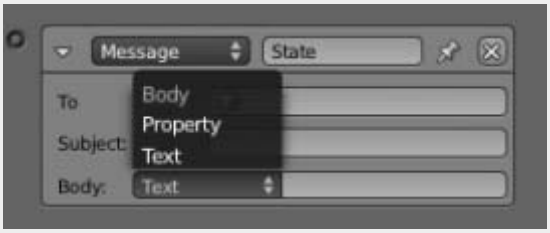
keyboard センサで Esc を検出するようにしている場合、Python ファイルでのエラー等が起きて *Quit Game* アクチュエータが失敗すると、ゲームが終了できなくなります。データは quit.blend（*File* » *Recover Last Session*）から復旧できるかもしれません。

Message アクチュエータ

Message アクチュエータは、シーン内で、またはシーン同士の間でデータを送ることができます。



Message actuator



Message actuator Options

To

メッセージを送る相手のオブジェクト。すべてのオブジェクト（または他のシーン）にメッセージを送る場合はこの欄は空にします。

Subject

メッセージの件名。たとえば"end-game"というメッセージを送る場合、message センサで"end game"を検出し、AND コントローラを介して Quit Game アクチュエータを起動する、という使い方ができます。

Body

メッセージの本文（これを読めるのは Python のみです）。

Text

ユーザの指定したテキスト。

Property

指定したプロパティ。たとえばハイスコアなど。

使用する際の注意

Message アクチュエータは、たとえばオブジェクトにスコアを送ったりするのに使えますが、シーンをまたいでデータを送るのにも使えます（もう一つの方法として は bge.logic.globalDict を使います）。

Motion Actuator

このアクチュエータはオブジェクトを移動/回転させます。移動、回転、または動的な移動を行う、Simple および Servo の二つの操作方法が用意されています。Simple モードの処理はオブジェクトの物理設定に左右されます。

アクチュエータ共通のオプションについては こちら をご覧ください。

専用オプション:

Motion Type
動きの種類を決めます

Simple Motion
: さまざまな動きを直接適用します

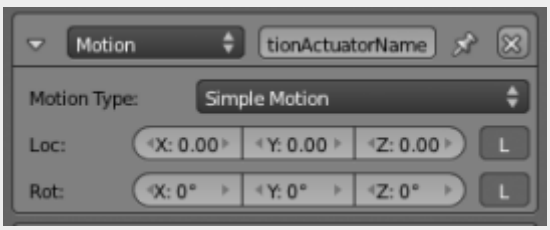
Servo Control
: 目標速度と、その速度に到達するまでの早さを指定します

オブジェクトの衝突

Simple 動作モードではオブジェクトは始点と終点以外の座標を一切通らないため、他のオブジェクトを通り抜ける可能性があります。オブジェクトの物理設定が Dynamic/Rigid Body/Soft Body に設定されると実行される Servo モードでは、この問題を回避できます。

Simple Motion

（単純運動）



Motion actuator for Simple Motion

Loc
: オブジェクトはパルスを受信するたびに、各軸方向に指定値だけジャンプします。

Rot

オブジェクトはパルスを受信するたびに、各軸を中心にして指定量だけ回転します。1 回転は値 7.2 で表されます（0.02 で 1 度）。

L

指定した座標がグローバル（灰色）とローカル（白色）のどちらなのか指定します。

Servo Control

Servo Control を動作させるには、Physics ウィンドウで Dynamic をオンにし、オブジェクトを Actor にする必要があります。

Servo Control

（サーボ制御）

Motion actuator for Servo Control

Servo control は物質界のものの動きを模倣する強力な手段です。Servo controller は、オブジェクトに与える力を調整し、指定した速度になるようにします。運動の比例 – 積分 – 微分（PID）の数式を使います。

Reference Object

アクチュエータの所有者が運動の基準として使うオブジェクトを指定します。例えば動く足場などです。空欄にすると World を基準にします。

Linear Velocity

三軸それぞれにおける線速度で、オブジェクトの速度の目標値になります。

L

指定した座標がグローバル（灰色）なのかローカル（白色）なのかを指定します

X, Y, Z

オブジェクトに適用される力の最大/最小値を設定します。無効にすると(X、Y、Z ボタンが灰色になります)力の上限はなくなります。

Proportional Coefficient

比例係数を調整します。実際の速度と目標線速度との差への反作用を制御します。

Integral Coefficient

積分系数を設定します。この動きのこれまでの誤差の合計への反作用を制御します。

Derivative Coefficient

微分係数を設定します。反作用を制御します。

Parent Actuator

現在のオブジェクトの親を変えることができます。

アクチュエータ共通のオプションについては こちら をご覧ください。

専用オプション:

Parent Actuator

Scene

必要な操作を選びます

Set Parent

このオブジェクトを現在のオブジェクトの親にします

Parent Object

親オブジェクトの名前

Compound'

このオブジェクトの形状を親の形状に足します(親の形状がすでに compound されているときのみ)

Ghost'

親である間このオブジェクトを Ghost にします

Remove Parent

現在のオブジェクトの全ての親を取り除きます

Parent Object

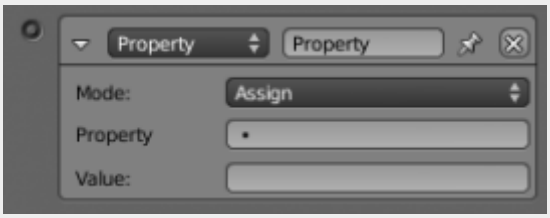
親オブジェクトの名前

Property(プロパティ)アクチュエーター

Property(プロパティ)アクチュエーターは、アクティブになると指定したプロパティの値を変更します。

共通のオプションについてはアクチュエーター共通のオプションを参照してください。

特別なオプション:



Property(プロパティ)アクチュエーター

Mode(モード)

Assign(適用)

*Property(プロパティ)*に指定したターゲットプロパティを、*Value(値)*に指定した値にします。

Add(追加)

*Value(値)*を *Property(プロパティ)*の値に加算(負の値を入れると減算)します。*Bool(ブーリアン)*では、0(または負の値)以外が True として加算されます。

Copy(コピー)

他のオブジェクトのプロパティからこのアクチュエーターのオーナーのプロパティにコピーします。

Toggle(切り替え)

0 と 1、そして 0 以外の数字の場合は 0 に切り替えます。ON/OFF スイッチに便利です。

Level(レベル)

アクチュエーターがアクティブになると 1、非アクティブになると 0 になります。

Property(プロパティ)

このアクチュエーターが変更するターゲットプロパティ。

Value(値)

プロパティを変更するのに使用する値。

例

たとえばゲームのキャラクターが "HP"(ヒットポイント)というプロパティを持っているとします。あとどのくらいのダメージを受けたら死亡するかということを決める数値です。HP は整数で、100 から始まるとします。

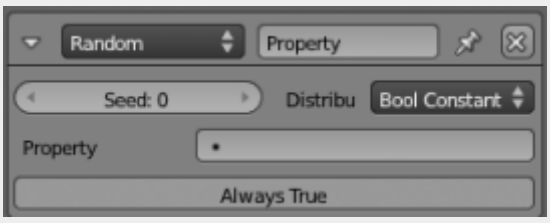
二つの *Collision(コリジョン)*センサーを追加します。一つは敵の弾、もう一つは回復薬の回収用です。 最初のセンサーは、(*AND* コントローラを介して) *Add Property(追加プロパティ)*アクチュエーターにつなぎ、変更するプロパティ名 "HP" を指定して、*Value(値)*は-10 に設定します。プレーヤーは敵の弾に当たるたびに HP を 10 失います。 二つ目のセンサーは、(*AND* コントローラを介して) *Add Property(追加プロパティ)*アクチュエーターにつなぎ、変更するプロパティ名 "HP" を指定して、*Value(値)*は 50 に設定します。プレーヤーが回復薬にぶつかるたびに HP が 50 増えます。 次に、上限を設定するために *Property(プロパティ)*センサーを設定し、interval(区間)モードで 100 以上のときの反応するようにします。そして(*AND* コントローラを介して) *Assign Property(適用プロパティ)*アクチュエータにつなぎ、プロパティを100に変えるように設定します。これでHPが100以上になってしまう場合は 100 に設定されます。

Random Actuator

オブジェクトのプロパティにランダムな値をセットします

アクチュエータ共通のオプションについては こちら をご覧ください。

専用のオプション:



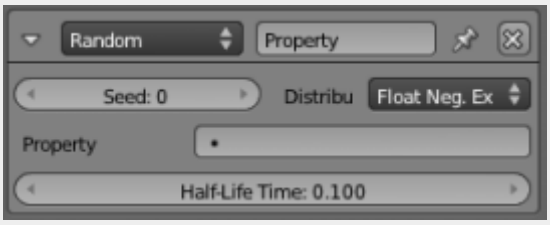
Random Actuator

Seed(シード)

乱数生成器の開始シード値です(1 – 1000 の範囲)

Distribution(分布)

乱数が抽出元となる分布のメニューです。デフォルトの Boolean Constant は True または False 値を返し、テスト目的に役立ちます。



Float Neg. Exp.

Float Neg. Exp.

指定した半減期を持ち指数関数的に減衰する値

Property

値を受け取る Float 型のプロパティ

Half-Life Time（半減期）

半減期（範囲：0.00 -10000.00）

Float Normal

Float normal

正規分布から得られる乱数

Property

値を受け取る Float 型のプロパティ

Mean（平均）

正規分布の平均値（範囲：-10000.00 から +10000.00）

SD（標準偏差）

正規分布の標準偏差（範囲：0.00 から +10000.00）

Float Uniform

Float uniform

最大/最小値間で均等に選ばれたランダムな値

Property

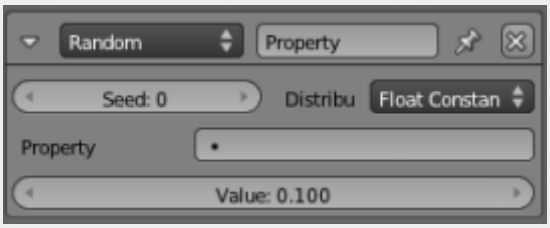
値を受け取る Float 型のプロパティ

Min

最小値（範囲：-10000.00 から +10000.00）

Max

最大値（範囲：-10000.00 から +10000.00）



Float Constant

Float constant

定値を返します

Property

値を受け取る Float 型のプロパティ

Value

値（範囲：0.00 から +1.00）

Random Integer Poisson

Int Poisson

ポアソン分布から得られる乱数

Property

値を受け取る Integer 型のプロパティ

Mean（平均）

ポアソン分布の平均値（範囲：0.01 から +100.00）

Random Integer Uniform

Int uniform

最大/最小値間で均等に選ばれたランダムな値

Property

値を受け取る Integer 型のプロパティ

Min

最小値（範囲：-1000 から +1000）

Max

最大値（範囲：-1000 から +1000）

Random Integer Constant

Int constant

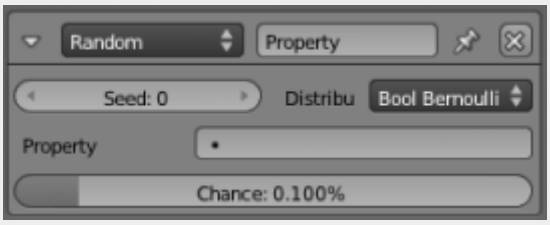
定値を返します

Property

値を受け取る Integer 型のプロパティ

Value

値（範囲：0.00 から +1.00）



Random Bool Bernoulli

Bool Bernoulli

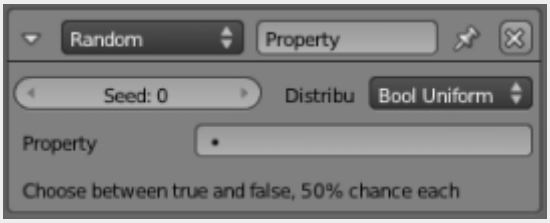
指定した比率で TRUE パルスを含む、ランダムな分布を返します

Property

値を受け取る Boolean 型のプロパティ

Chance（確率）

TRUE を受け取る比率



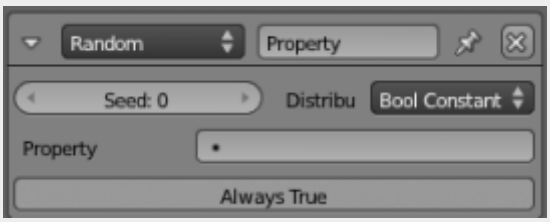
Random Bool Uniform

Bool uniform

50 対 50 の確率で True か False を受け取ります

Property

値を受け取る Boolean 型のプロパティ



Random Bool Constant

Bool constant

定値を返します

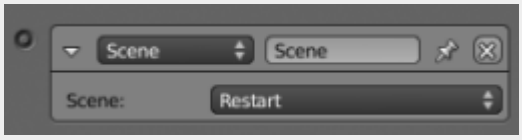
Property

値を受け取る Boolean 型のプロパティ

Value

値（True または False）

Scene アクチュエータ



Scene actuator

Scene アクチュエータは.blend ファイルの中のシーンを扱います。たとえばゲームのステージや UI、背景などをコントロールします。

このアクチュエータには 8 つのモードがあります。

Scene actuator options

Restart

現在のシーンをリスタートします。シーン内のすべてのものはリセットされます。

Set Scene

指定したシーンに変更します。

Set Camera

使用するカメラを変更します。

Add OverlayScene

別のシーンを追加し、現在のシーンの上に描画します。これはたとえばゲーム画面に体力や弾薬、スピードメーターなどの表示をオーバーレイして常に見えるようにする場合に便利です。

Add BackgroundScene

OverlayScene と反対に、現在のシーンの下に別のシーンを描画します。

Remove Scene

シーンを削除します。

Suspend Scene

シーンを中断します。

Resume Scene

中断したシーンを再開します。

Sound Actuator

音声ファイルを一覧から選ぶか、外部の音声ファイルを新たにリンクします。

Sound Actuator

アクチュエータの共通のオプションについては こちら をご覧ください。

専用のオプション:

音声ファイル名

一覧から音声ファイルを選ぶか、Open を押して外部の音声ファイルをリンクします。

Play Mode:（再生方法）

Loop Bidirectional Stop: 正パルスの受信中のみ再生＋逆再生します

Loop Bidirectional: 再生後に逆再生します

Loop End: ループ再生します

Loop Stop: 正パルスの受信中のみループ再生を続けます

Play End : 再生します

Play Stop: 正パルスの受信中のみ再生を続けます

Volume（音量）

再生音の音量を調整します

Pitch（音高）

再生音の高さを調整します

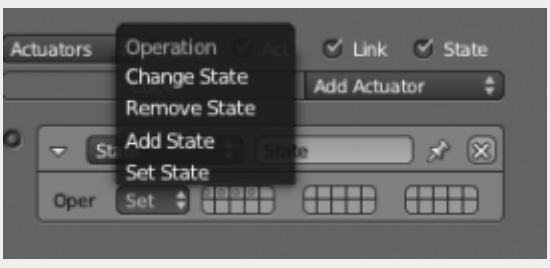
3D Sound

有効にすると、さまざまなパラメータを使って音の空間を表現できます

State アクチュエータ

State アクチュエータは簡単な操作で複雑なロジックを作成することができます。複数の違ったステート（状態）を持ち、それに対して処理をします。

State actuator



State actuator options

処理

Change State

現在のステートから指定したステートへ変更します。

Remove State

アクティブなステートのうち、特定のステートを削除します（停止します）。

Add State

アクティブなステートに、指定したステートを追加します（起動します）。

Set State

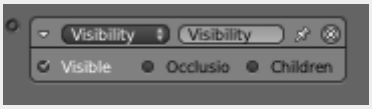
現在のステートから指定したステートに移動します。他のステートは停止されます。

Usage Notes

State アクチュエータを使うと、何百ものプロパティを扱わずにロジックの階層を作ることができます。うまくつかえば非常に有益ですが、python で問題を回避できる場合もあります。

Visibility アクチュエータ

Visibility アクチュエータはオブジェクトの表示・非表示をコントロールします。



Visibility actuator

Visible

表示・非表示を切り替えます。

Occlusion

オクルージョンの有効・無効を切り替えます。Physics タブで初期設定する必要があります。

Children

再起設定の有効・無効を切り替えます。有効にすると、visibility および occlusion の設定が再帰的に（子オブジェクトやそのさらに子オブジェクトに）適用されます。

使用する際の注意

Visibilityアクチュエータを使うと描画を節約できますが、物理計算に関してはそうではありません。よってLOD（レベル・オブ・ディテール）という意味では限界があります。LODのためにはEdit Objectアクチュエータのreplace meshモードを見てください。ただし、ロジックがLODを反転してしまう可能性に気をつけてください。

コントローラ

コントローラはセンサからのデータを集めるブリックです。 センサが起動すると正パルスが発信され、停止すると負パルスが発信されます。コントローラの役割は、それらの情報をつなぎ合わせて適切な反応を生み出すことです。

コントローラー用のロジックブロックは Logic Editor を使って構築し、編集できます。この操作の詳細は コントローラーの編集 ページをご覧ください。

コントローラの種類

入力を処理する方法は 8 通りあります。

- AND
- OR
- XOR
- NAND
- NOR
- XNOR
- Expression
- Python

以下の表はコントローラの種類と反応をまとめたものです。1 列目は出力です。接続されているセンサから送られてくる正パルスの数を表しています。それに続く列は、それぞれの種類のコントローラの反応です。True は、コントローラにおいて条件が満たされ、それに接続されたアクチュエータが起動するということです。False は条件が満たされず、何も起こらないということです。それぞれのコントローラについてはあとのセクションでさらに解説します。

註釈
コントローラには 2 つ以上のセンサが接続されているとみなされます。1 つしか接続されていない場合は表中の"全て"の段のふるまいになります。

正のセンサー	コントローラー					
	AND	OR	XOR	NAND	NOR	XNOR
なし	False	False	False	True	True	True
ひとつ	False	True	True	True	False	False
複数あるが全てではない	False	True	False	True	False	True
全て	True	True	False	False	False	True

AND Controller

AND コントローラは新規のコントローラを作成したときのデフォルトのタイプです。その理由は、これがもっともよく使われるということと、センサからのパルスを単純にアクチュエータに渡したい場合に上手く動作するからです（コントローラを介さずに渡すことはできません）。

AND コントローラは接続されているすべてのセンサの状態が正の場合のみアクチュエータを起動します。センサが一つだけ接続されている場合は、センサから発信がそのままアクチュエータの起動につながります。複数のセンサが接続されている場合は、すべてのセンサが同時に起動していなければいけません。これが名前の由来です。センサ 1 とセンサ 2 がコントローラに接続されていた場合、センサ 1 および（AND）センサ 2 が同時に起動している必要があります。

例

たとえばメニューボタンを作りたいとします。プレイヤーがそのボタンをクリックすると、次のシーンに移動します。

メニューボタンに *Mouse Over* と {{Literal|Left Button} }の二つの Mouse センサ を作成します。二つのセンサを同じANDコントローラにつなぎ、そのコントローラは *Set Scene* アクチュエータにつなぎます。このようにすると、マウスがメニュー上にあり、なおかつクリックされた場合だけ、次のシーンに進むということになります。

オプション

AND Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x ボタン

センサーを削除します

Controller Editing

（コントローラーの編集）

コントローラー列

コントローラーは Logic Panel の中央の列で作成/編集できます。このページではコントローラー全般で使われるコントロールやパラメータと、オブジェクトの状態（State）の作り方、変更の仕方について説明します。

図はコントローラー列にひとつだけコントローラーがある様子を示しています。列の最上部にある列見出しに、現在の Game Logic にあるコントローラーのうちどれを表示するのか決めるメニューやボタンがあります。

Column Heading

（列見出し）

コントローラー列の見出し

列見出しには表示するコントローラーの選択や表示の詳細度を決めるボタンなどがあります。必要のないコントローラーを隠し、必要なものを目立たせるのに便利です。

Controllers

Show Objects	すべてのオブジェクトを展開する。
Hide Objects	すべてのオブジェクトを名前の表示されるバーのみに縮小する。
Show Sensors	すべてのコントローラーを展開する。
Hide Sensors	すべてのコントローラーを名前の表示されるバーのみに縮小する。

コントローラーとオブジェクトの表示は別々にコントロールできます。

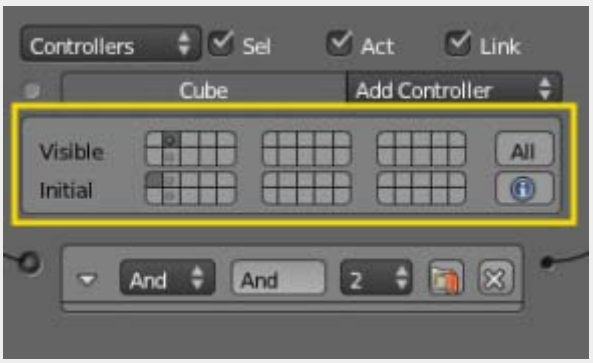
また表示するコントローラーの種類を指定できます。

Sel	選択されているすべてのオブジェクトのコントローラー。
Act	アクティブなオブジェクトのコントローラー。
Link	コントローラにリンクされているコントローラー。

Object Heading

（オブジェクト見出し）

コントローラー列のオブジェクト見出し、使用中ステートボタンがオフの状態



コントローラー列のオブジェクト見出し、使用中ステートボタンがオンの状態

列内には、コントローラーがオブジェクトごとに並びます。デフォルトでは選択中のオブジェクトのすべてのコントローラーが表示されますが、列見出しの抽出機能を使って変更できます。

オブジェクトのコントローラー一覧の見出しには三つの項目があります:

(使用中ステートボタン)

オブジェクトで使われているステートを表示します。パネルの詳しい使い方は States をご覧ください。

Name

オブジェクト名

Add Controller

クリックすると利用できるコントローラーの種類を選ぶメニューが現れます。コントローラーを選ぶと新たなコントローラーがオブジェクトに追加されます。利用できるコントローラーについては Controllers をご覧ください。

Expressions Controller

（式コントローラ）

コントローラは式を評価して正または負のパルスをアクチュエータに送ることができます。

- 式の評価が **True**（真）の場合、正のパルスを送ります。
- 式の評価が **False**（偽）の場合、負のパルスを送ります。

Expression Controller

Expression（式）

枠内に記述する式には変数（variable）、定数（constant）、および 演算（operator）を組み合わせることができます。

以下のルールに従う必要があります。

変数

使えるのは:

- **sensors names**: センサの名前
- **properties**: プロパティをオブジェクトに登録して、式の中で使えます

これらの名前にはスペースを含んではいけません。

演算

算術演算

演算子: *, /, +, -

返り値: 数値

例: 3 + 2, 35 / 5

論理演算

- 比較演算子: <, >, >=, <=, ==, !=
- ブーリアン演算子: AND, OR, NOT

返り値: 真または偽

例: 3 > 2 (True), 1 AND 0 (False)

条件文(if)

以下の文を使います。

```
if( expression, pulse_if_expression_is_true, pulse_if_expression_is_false )
```

式(expression)の評価が真の場合、

- `pulse_if_expression_is_true` が真ならアクチュエータに正パルスを送る。
- `pulse_if_expression_is_true` が偽ならアクチュエータに負パルスを送る。

式の評価が偽の場合、

- `pulse_if_expression_is_false` が真ならアクチュエータに正パルスを送る。
- `pulse_if_expression_is_false` が偽ならアクチュエータに負パルスを送る。

例

たとえば、オブジェクトに `coins` というプロパティをつくり、値は 30 にしておきます。

```
coins > 20
```

これは真を返します（アクチュエータに正パルスを送ります）。

たとえば、オブジェクトが

- `Key_Inserted` というセンサを持ち、値は真
- `Fuel` というプロパティを持ち、値は偽

という場合、

```
Key_Inserted AND Fuel
```

これは偽を返します（アクチュエータに負パルスを送ります）。

これは以下と同じです。

```
if (Key_Inserted AND Fuel, True, False)
```

また、

```
if (Key_Inserted AND Fuel, False, True)
```

こう書くと、`Key_Inserted AND Fuel` が偽の場合に正パルスが送られます。

さらに、

```
if ((Key_Inserted AND Fuel) OR (coins > 20), True, False)
```

これは真を返します。この場合はアクチュエータに正パルスが送られます。

Game Expressions

コントローラは式を評価して正または負のパルスをアクチュエータに送ることができます。

- 式の評価が `True`（真）の場合、正のパルスを送ります。
- 式の評価が `False`（偽）の場合、負のパルスを送ります。

変数

使えるのは:

- **sensors names**: センサの名前
- **properties**: プロパティをオブジェクトに登録して、式の中で使えます

これらの名前にはスペースを含んではいけません。

演算

数学的演算

演算子: *, /, +, -

返り値: 数値

例: 3 + 2, 35 / 5

論理演算

- 比較演算子: <, >, >=, <=, ==, !=
- ブーリアン演算子: AND, OR, NOT

返り値: 真または偽

例: 3 > 2 (True), 1 AND 0 (False)

条件文(if)

以下の文を使います。

```
if( expression, pulse_if_expression_is_true, pulse_if_expression_is_false )
```

式(expression)の評価が真の場合、

- **pulse_if_expression_is_true** が真ならアクチュエータに正パルスを送る。
- **pulse_if_expression_is_true** が偽ならアクチュエータに負パルスを送る。

式の評価が偽の場合、

- **pulse_if_expression_is_true** が真ならアクチュエータに正パルスを送る。
- **pulse_if_expression_is_true** が偽ならアクチュエータに負パルスを送る。

例

たとえば、オブジェクトに **coins** というプロパティをつくり、値は 30 にしておきます。

```
coins > 20
```

これは真を返します(アクチュエータに正パルスを送ります)。

たとえば、オブジェクトが

- **Key_Inserted** というセンサを持ち、値は真
- **Fuel** というプロパティを持ち、値は偽

という場合、

```
Key_Inserted AND Fuel
```

これは偽を返します（アクチュエータに負パルスを送ります）。

これは以下と同じです。

```
if (Key_Inserted AND Fuel, True, False)
```

また、

```
if (Key_Inserted AND Fuel, False, True)
```

こう書くと、**Key_Inserted AND Fuel** が偽の場合に正パルスが送られます。

さらに、

```
if ((Key_Inserted AND Fuel) OR (coins > 20), True, False)
```

これは真を返します。この場合はアクチュエータに正パルスが送られます。

NAND Controller

NAND は Not And の略です。論理学では、Not は結果を反転します。つまり Not And は *AND* コントローラの正反対の働きをします。*AND* コントローラはすべてのセンサが発信しているときだけアクチュエータを起動します。それと反対に、*NAND* コントローラはすべてのセンサが起動してはいない場合のみアクチュエータを起動します。センサが一つだけ接続されている場合は、センサが発信していないときだけアクチュエータを起動します（単純な NOT と同じように働きます）。

例

---製作中---

オプション

NAND Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x **ボタン**

センサーを削除します

NOR Controller

NOR コントローラは *OR* コントローラの反対です。センサが一つも発信していない場合だけアクチュエータを起動します。

例

---製作中---

オプション

NOR Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x **ボタン**

センサーを削除します

OR Controller

OR コントローラは、接続されたセンサのうちすくなくとも一つが起動していると、アクチュエータを起動します。センサが一つだけ接続されている場合は、センサから発信がそのままアクチュエータの起動につながります。複数のセンサが接続されている場合は、どれか一つのセンサは起動していなければいけません。*OR* は排他的ではありません。つまり、*AND* コントローラが起動するような状況（すべてのセンサが起動している）でも同様に起動します。排他的論理和には *XOR* コントローラを使ってください。

例

EscかQを押せばゲームが終了するようにしたい場合、それぞれのキーに対して Keyboard センサを設定します。それを同じ*OR*コントローラにつなぎ、そのコントローラを Quit this game アクチュエータ につなぎます。これで、どちらかのキーが押されればゲームは終了します。

オプション

OR Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x ボタン

センサーを削除します

Python Controller

Python コントローラはユーザがプログラムしたスクリプトに従って入力を処理するコントローラです。Python スクリプト、または Python コードの含まれるファイルを使います。*Python* コントローラには二つのモードがあります。*Script* と *Module* です。どちらもテキストエディタで書いて.blend ファイルの内部に保存することもできるし、外部ファイルにすることもできます。

BGEにおけるPythonについてはここを参照。

BGEのPython APIについてはここを参照。

Script Mode

Script モードでは、コントローラはスクリプトにリンクされ、ロジックが進む前にスクリプト全体が実行されます。スクリプトに記述されていることはすべて同じフレームにおいて起こります。そのため、処理の中で同じ値または属性が何度も変わった場合、最終的

な結果だけがゲームとして表示されます。たとえばオブジェクトがまず(100.0,100.0,100.0)に移動し、それから(0.0,0.0,0.0)に移動した場合、表示されるのは(0.0,0.0,0.0)への移動だけです。

Module Mode

Blender 2.49 からは Python モジュールコントローラが加わりました。*Python* コントローラのドロップダウンメニューで *Script* のかわりに *Module* を選んでください。そして、そのモジュールのなかで関数を定義し、その関数をコントローラで呼び出します。エディットボックスには"myScript.py"のかわりに"myModule.myFunc"という風に指定します。その関数はコントローラが呼び出されるたびに処理されます。しかし、myFunc のスコープの外にある関数や変数は一度だけしか処理されません。これは最初に一度だけ変数を初期化したい場合に便利です。

Python モジュールコントローラが扱える属性の数に制限はありません。つまりパッケージが自動的にサポートされるということです。"myModule.myFunc"と同様に"myPackage.myModule.myFunc"とネストされたパッケージも動作します。"myPackage.myModule.myInstance.myMethod"のようなメソッドも動作します。Python コントローラは、一つの引数をとる関数に対す引数として関数に渡されます。

このためライブにスクリプトを編集できます。Python モジュールコントローラについてさらに学ぶには、

*Thread http://blenderartists.org/forum/showthread.php?t=156672
*Video http://download.blender.org/apricot/live_bge_edit.ogv

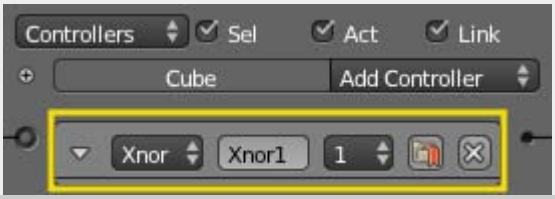
XNOR Controller

XNOR コントローラは、混乱するような名前ですが、単純に *XOR* コントローラの反対です。一つより少ないまたは多い数のセンサが発信しているときだけアクチュエータを起動します。

例

---製作中---

オプション



XNOR Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x ボタン

センサーを削除します

XOR Controller

XOR コントローラは、排他的論理和のコントローラです。
このコントローラーは次の条件を満たすとき正の（TRUE/真の）信号を出力します:

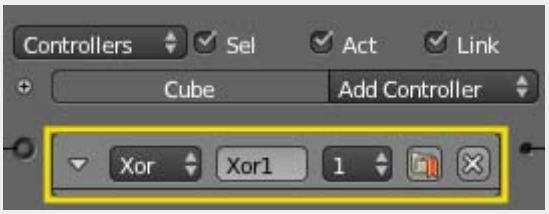
- 入力の一つ（一つだけ）が TRUE、かつ オブジェクトが指定状態（Designated State）であるとき

これ以外では負の（FALSE/偽の）信号を出力します。

例

---製作中---

オプション



XOR Controller

Controller Type メニュー

コントローラーの種類を指定します

Controller Name

コントローラーの名前で、ユーザーが指定できます。オブジェクト内で重複しない名前にする必要があります。Python でコントローラーにアクセスする際に使われます。

State Index

このコントローラーが実行されるステートを設定します。

Preference ボタン

オンにすると、オフにしている他のすべてのコントローラーより先に実行されます（開始時スクリプトに便利です）

x ボタン

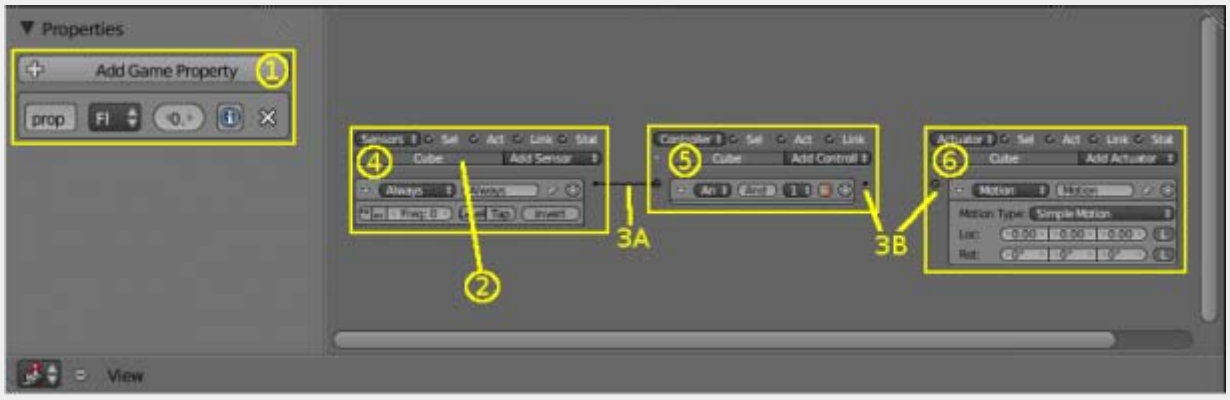
センサーを削除します

Logic Editor

ゲームを構成するさまざまな俳優たち（＝オブジェクト）用の、ゲームロジックの作成、編集作業の中心となるのが Logic Editor です。

関連する 3D パネルで選択中のオブジェクトのロジックが、logic brick として表示されます。センサー、コントローラー、アクチュエータの三つの列を持つ表として表示されます。ロジックブリックを繋いでいるリンクは、センサー–コントローラー間、コントローラー–アクチュエータ間に信号を伝えます。

ロジックパネルをより良く理解してもらえるように、メニューの一部を拡大して番号をつけたものが、下の画像です。 下の画像に付けられた番号を追いながら、個別に各セクションを見ていきます。



Logic Panel の構成部品

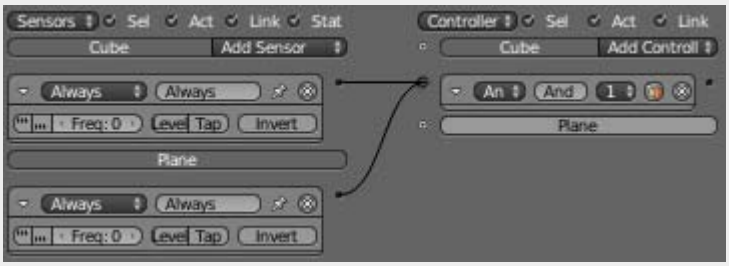
1 Properties

プロパティは、他のプログラミング言語における変数に似ています。これらは、オブジェクトに関連付けられたデータを保存したり、アクセスしたりするために使用されます。以下の種類があります:

- *Timer* – 定義された番号から始まり、カウントアップします。
- *String* – テキストを保持します。
- *Float* – –10000.000 から 10000.000 の小数を保持します。
- *Int*eger – –10000～10000 の整数を保持します。
- *Bool* – True または False のいずれかです。

より詳細な説明については、Propertiesを参照してください 。

2 Associated Object(s)



Logic for several objects.

Blender game engine のロジックは、オブジェクトに結び付けられます。 オブジェクトは、その名前順に一覧表示され、それらが選択されている場合、ロジックウィンドウに表示されます。 あなたは上の図のように単一のオブジェクトを選択するか、右の図のように複数のオブジェクトを選択することができます。

オブジェクトのロジックは、そのロジックに関連付けられているオブジェクト(s)が選択されているときにのみ表示されます

3 Links

リンク(3A)は、オブジェクト間の論理的な流れの方向です。リンク線は LMB  で一方のリンクのノード(3B)から他方へドラッグすることで描かれます。 リンクは、センサからコントローラへ、もしくはコントローラからアクチュエータへにだけ描画することができます。 送信ノード(センサとコントローラの右側にある黒い丸)は、複数の受信ノード(コントローラとアクチュエータの左側にある)に送信することができます。 受信ノードは、同様に複数のリンクを受け取ることができます。

あなたが直接アクチュエータにセンサをリンクすることはできません。 アクチュエータは、センサに戻るほうにリンクすることはできません。 もしアクチュエータが完了した後にセンサーを起動させたい場合は、アクチュエータのセンサーを使用してください。

4 Sensors

センサーは、すべてのロジックのアクションを開始します。センサーは、オブジェクトの接近や、キーボードのキーの押下や、時限イベント等々を “感知” します。 センサーがトリガされると、パルスがすべてリンクされているコントローラに送信されます。

より詳細な説明については、Sensorsを参照してください 。

5 Controllers

コントローラは、ロジック、センサーからのパルスの評価、およびそれに応答してアクチュエータへのパルスの送信を扱います。コントローラにはさまざまな種類があります:

- *AND* – すべての接続されたセンサーは正のパルスを送信するため、正の値でなければなりません。
- *OR* – 1 つ以上の接続されたセンサーが正である必要があります。
- *XOR* – 排他的論理和:どちらか片方だけ、接続されたセンサーが正である必要があります。

- *NAND* – 否定論理積、AND のビット反転のコントローラ。
- *NOR* – 否定論理和、OR のビット反転のコントローラ。
- *XNOR* – 排他的 NOR コントローラ。
- *Expression* – あなた自身の表現をお書きください。
- *Python* – Python スクリプトやモジュールのセンサを制御します。

より詳細な説明については、Controllersを参照してください 。

6 Actuators

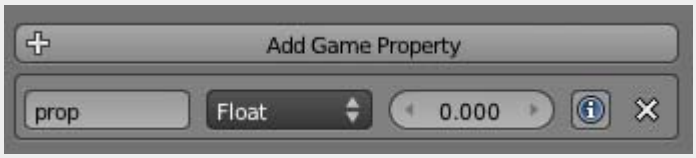
アクチュエータは、オブジェクトまたはゲームに何らかの方法で影響を与えます。アクチュエータは動き、音、プロパティ、オブジェクト、などを変更します。これらの変更は、他のオブジェクト、物理学、プロパティなどにも及び、またそれらは他のロジックブリックへのトリガイベントを発生させることもできます。

より詳細な説明については、Actuatorsを参照してください 。

プロパティ

プロパティは変数に相当するものです。これはオブジェクトに格納されており、たとえば弾薬、体力、名前などを表すことができます。

プロパティパネル



The properties panel.

View » *Properties* または N でプロパティを開きます。

プロパティパネルは 6 つの要素から成っています。

Add Property button

新しいプロパティを追加します。デフォルトでは *Float* プロパティで、名前は prop、すでに同じ名前がある場合は後ろに番号が付きます。

Name field

プロパティの名前。この名前を使って Python プログラムまたは式でプロパティにアクセスします。その場合ディクショナリ形式でアクセスします(GameObject["*propname*"])。名前は大文字と小文字が区別されます。

Property type menu

プロパティの種類(後述)。

Value field

プロパティの初期値。

Debug button

デバッグを有効にします。*Game* メニューの *Show Debug Properties* も有効にする必要があるので注意してください。*Show Debug Properties* を有効にすると、デバッグを有効にしてあるすべてのプロパティが、オブジェクト名、プロパティ名、数値、としてゲームの実行中に表示されます。プロパティが問題を起こしている疑いがある場合に便利です。

Delete button x

それぞれのプロパティの横には削除ボタンがあります。

プロパティの種類

プロパティには5種類あります。

Timer	初期値から始めて、オブジェクトが存在する限りカウントアップしていきます。たとえばゲームのステージをクリアするのにかかった時間などを調べるのにつかいます。
Float	-10000.000 から 10000.000 までの少数。精密な数値が必要な場合に遣います。
Integer	-10000 から 10000 までの整数。たとえば弾丸の数など、少数が必要ない場合に使います。
String	テキストを値として持ちます。128 文字まで使えます。
Boolean	真か偽の二値を持ちます。明かりのスイッチなど、二つしか状態がないようなものに便利です。

プロパティを使う

ロジックエディタではプロパティセンサとプロパティアクチュエータが使えます。

Property Sensor

Property センサには 4 つのモードがあります

Equal

プロパティ値とセンサの値が等しい場合に正パルスを発信します。

Not Equal

プロパティ値とセンサの値が異なる場合に正パルスを発信します。

Interval

プロパティ値がセンサの *Min* から *Max* の価の範囲にある場合に正パルスを発信します。
最低値のみを指定したい場合は、自身のプロパティ名を *Max* 欄に、パルスを送る最低値を *Min* 欄に入力します（訳注：最高値は自分自身の値なので、常に範囲内にある）。
最高値のみを指定したい場合は、自身のプロパティ名を *Min* 欄に、パルスを送る最高値を *Max* 欄に入力します。
値を比べるために他のプロパティ名を入力することもできます。

Changed

プロパティの値が変わったときに正パルスを発信します。

センサの基本的な使い方については センサ を、プロパティセンサについては プロパティセンサ を参照してください。

Property Actuator

Property アクチュエータには 4 つのモードがあります。

Assign

Value 欄の値を *Prop* 欄に与えます。同じオブジェクトのプロパティのみに使えます。

Add

プロパティの値を増加させます。マイナスの数値をしてすれば減少させます。Bool に関しては 0 以外の値（マイナスでも）は真とみなされます。

Copy

他のプロジェクトのプロパティをアクチュエータのオーナーのプロパティにコピーします。

Toggle

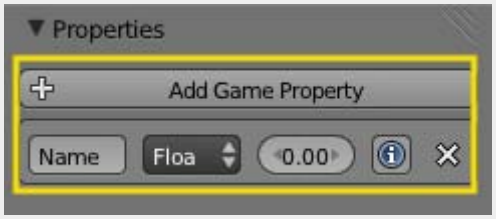
0 は 1 に変え、0 以外の値は 0 に変えます。オン／オフのようなスイッチに便利です。*String* プロパティには使えません。

アクチュエータの基本的な使い方については アクチュエータ を、プロパティアクチュエータについては プロパティアクチュエータ を参照してください。

Property Editing

（プロパティの編集）

Logic Editor パネル左側にあるパネルを使って、ロジックのプロパティを作成したり編集したりできます。最上部のメニューでは利用できるプロパティの型の一覧が見られます。



Property Panel

Add Property ボタン

一覧に新たなプロパティを加えます。デフォルトは“prop”という名前の *Float* 型のプロパティで、同名のプロパティがすでにあれば、後ろに数字がつきます。

Name 欄

プロパティの名前を入力します。Python や Expression（式）からアクセスする際に使われます。Python では辞書型の参照方式を使います（(GameObject[“*propname*”]）。名前の大文字小文字は区別されます。

Type メニュー

プロパティの型を決めます（下記参照）。

Value 欄

プロパティの初期値を決めます。

i 情報ボタン

デバッグ時にプロパティを表示します。デバッグをオンにすると、ゲーム実行中画面の左上隅にプロパティの値が表示されます。デバッグをオンにするには、*Game* メニューにある *Show Debug Properties* にチェックをつけてください。デバッグ対象となったプロパティはすべて、ゲームプレイ中オブジェクト名、プロパティ名、プロパティ値で表示されます。プロパティが何か問題を起こしていると思える場合に役立ちます。

x

プロパティを削除します。

センサ

センサはロジックが何かをする要因となるものです。これはオブジェクトに近づいたり、キーが押されたり、時間によってイベントが起動したり、などのトリガイベントです。センサが起動すると、リンクしているすべてのコントローラに正のパルスが送られます。

どのタイプのセンサー用ロジックブロックも Logic Editor を使って構築し、変更します。この作業の詳細は センサーの編集 ページをご覧ください。

現在、以下のセンサーが利用できます:

Actuator	特定アクチュエータが実行信号を受け取ったことを検知
Always	一定間隔で信号を出力し続けます
Collision	オブジェクト間またはマテリアル間の衝突を検知
Delay	指定した論理ティック値だけ出力を遅延させます
Joystick	ジョイスティックの特定操作を検知
Keyboard	キーボード入力を検知
Message	テキストメッセージかプロパティ値を検知
Mouse	マウスイベントを検知
Near	一定距離内に進入したオブジェクトを検知
Property	所有オブジェクトのプロパティ変化を検知
Radar	一定距離内、軸から一定角度内に進入したオブジェクトを検知
Random	ランダムにパルスを生成
Ray	軸角方向に光を放射して衝突を検知
Touch	オブジェクトと他オブジェクトとの接触を検知

Actuator sensor

（アクチュエータセンサー）

Actuator sensor

Actuator 欄に指定したアクチュエータが起動すると正のパルスを発信します。

指定したアクチュエータが非アクティブ化すると、センサーは偽のパルスを送ります。

センサー共通のオプション もご覧ください。

専用のオプション:

Actuator

アクチュエータ名。同じオブジェクトに属している必要があります。

Always sensor

（常時センサー）

Always sensor

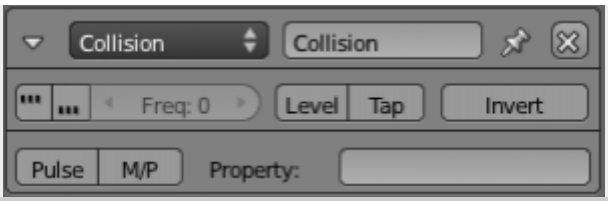
一定の論理 tick ごとに発信する必要がある場合、またはスタートアップ時に発信させたい場合（*Tap* を有効にする）に使います。

センサー共通のオプション もご覧ください。

このセンサには共通オプション以外の設定項目はありません。

Collision sensor

（衝突センサー）



Collision sensor

Touch センサに似ていますが、プロパティやマテリアルで抽出できます。指定したプロパティ/マテリアルを持つオブジェクトが衝突した場合のみ正パルスが送られます。どのオブジェクトに対しても動作させるには欄を空のままにします。

センサー共通のオプション もご覧ください。

専用オプション:

Pulse ボタン

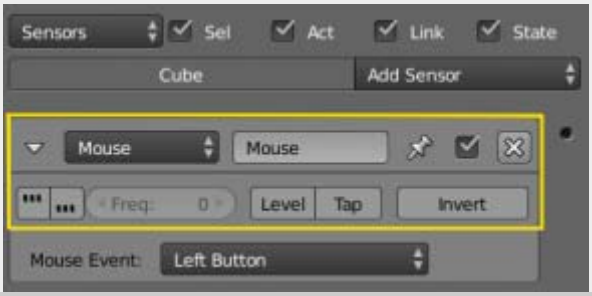
直前のパルスの要因となったオブジェクトとまだ触れているときにも、他のオブジェクトとの衝突を検出します。

M/P ボタン

フィルタリングする要素をマテリアルとプロパティで切り替えます。

Sensor Common Options

（センサー共通のオプション）



Common Sensor Options

すべてのセンサに共通のボタン、入力欄、メニューがあります:

三角形のボタン

センサの詳細を表示／非表示します。

センサーの種類 メニュー

センサの種類を参照。

センサー名

センサの名前。Python でセンサにアクセスする際に使います。そのオブジェクトのなかで唯一の名前である必要があります。

X ボタン

センサを削除します。

起動信号(Trigger)について

コントローラーは、接続されているセンサー(状態は任意)によって起動されない限り作動することはありません。
センサーの状態が変化すると、接続されているコントローラーが起動します。センサの状態が負から正、または正から負へ変わると、センサーは接続されているコントローラーを起動します。センサーが非アクティブ状態からアクティブ状態になるときも、接続されているコントローラーを起動します。

以下のパラメータは、センサーに接続されたコントローラーの起動方法を指定します:

 (True level triggering)

これをセットすると、センサの状態が正である限り、真のパルスを出し続けます。パルスの間隔は周波数で設定します。

 (False level triggering)

これをセットすると、センサの状態が負である限り、偽のパルスを出し続けます。パルスの間隔は周波数で設定します。

Freq(周波数)

周波数という名前に反して、このパラメータはパルスを送る時間間隔を指定します。単位はフレームです(論理 tick と呼ぶこともあります)。デフォルトの値 0 の場合は、まったく間隔をあけません。Level 起動のパラメータが少なくとも一つ有効な場合にのみ使われます。
パルスがそれほど頻繁でなくてもいい場合は、この数値を上げておくとパフォーマンスの節約になります。

例: (デフォルトのフレームレートを 60 Hz(60 フレーム/秒)と仮定します)。

freq	意味	起動信号を持つフレーム	起動信号を持たないフレーム	フレーム周期	周波数(フレーム/秒)
0	センサーは次のフレームで起動信号を送ります。	1	0	1	60
1	センサーは第 1 フレームで起動信号を送り、再度起動信号を送るまでに 1 フレーム待ちます。結果として半分の速度になります。	1	1	2	30

29	センサーは第 1 フレームで起動信号を送り、再度起動信号を送るまでに 29 フレーム待ちます。	1	29	30	2
59	センサーは第 1 フレームで起動信号を送り、再度起動信号を送るまでに 59 フレーム待ちます。	1	59	30	1

Level ボタン

組み込みのステートマシンの状態（State）が変化するときにパルスを送ります。状態についてはStatesをご覧ください。

以下のパラメーターはセンサーの状態算出の方法を決めます:

Tap ボタン

センサーの状態を正に変え、（センサーの算出値が正であったとしても）1 フレーム後に負に変えます。状態が変わるため、接続されたコントローラーは（Level と）同じように起動します。Tap と Level はどちらか一方だけを選択できます。

TRUE level triggering を設定している場合、センサーの算出値が偽になるまで真／偽のパルスが交互に送られ続けます。

Tap が設定されていると *FALSE level triggering* は無視されます。

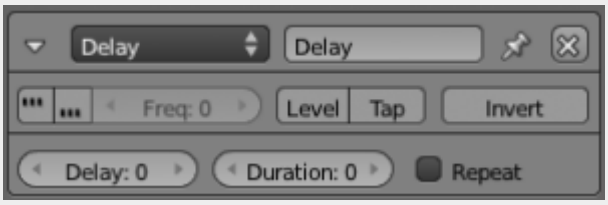
Invert ボタン

センサの出力を反転します。

真のパルスを送るべきときに偽のパルスを、偽のパルスを送るべきときに真のパルスを送ります。ただし、*Tap* が設定されている場合は、センサーは反転したセンサーの状態に基いて、コントローラーに起動信号を送ります。

Delay sensor

（遅延センサー）



Delay sensor

指定した ticks だけ反応を遅らせます。これは他のアクションが先に起こるべきである場合や、時間によって起こるイベントの場合に便利です。

センサー共通のオプションもご覧ください。

専用オプション:

Delay

正のパルスを送る前に待つ論理 ticks 数

Duration

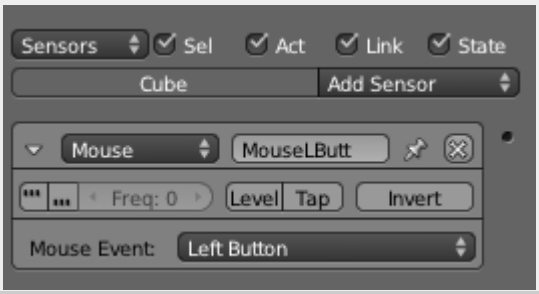
負のパルスを送る前に待つ論理 ticks 数

Repeat ボタン

遅延時間が終わった後、再始動します

Sensor Editing

（センサーの編集）

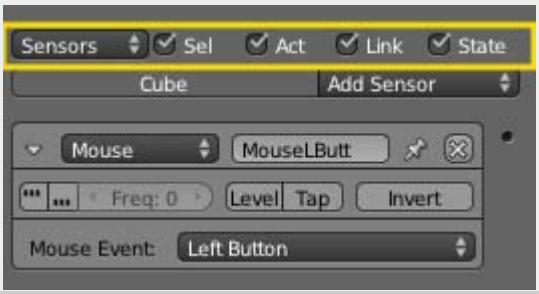


Sensor Column with Typical Sensor

センサはロジックパネルの左の列で編集します。

Column Heading

（列見出し）



Sensor Column Heading

Sensors サブパネルの一番上には *Sel*、*Act*、*Link*、*State* の4つのボタンがあります。また Sensors サブパネルのタイトル *Sensors* をクリックするとメニューが現れます。これはセンサの表示・非表示を切り替えるもので、必要なセンサだけを見るのに便利です。

Sensors メニューはラベルのようですがメニューです。4つのオプションがあります。

Show Objects	すべてのオブジェクトを展開する。
Hide Objects	すべてのオブジェクトを名前の表示されるバーのみに縮小する。
Show Sensors	すべてのセンサを展開する。
Hide Sensors	すべてのセンサを名前の表示されるバーのみに縮小する。

センサとオブジェクトの表示は別々にコントロールできます。

また表示するセンサの種類を指定できます。

Sel	選択されているすべてのオブジェクトのセンサ。
Act	アクティブなオブジェクトのセンサ。
Link	コントローラにリンクされているセンサ。
State	アクティブなステートを持つコントローラーに接続されているセンサ。

Object Heading

（オブジェクト見出し）

Sensor Object Heading

センサは選択されたオブジェクトごとに表示されています。デフォルトでは、選択中のオブジェクトのすべてのセンサーが列内に並びますが、列見出しにある抽出機能を使って変更可能です。

センサーー覧の見出し部分には二つの項目があります:

Name

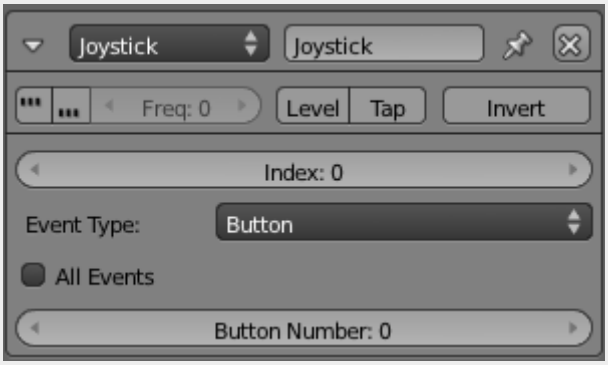
オブジェクト名

Add Sensor

クリックすると利用できるセンサーの種類を選ぶメニューが現れます。センサーを選ぶと新たなセンサーがオブジェクトに追加されます。利用できるセンターについては Sensors をご覧ください。

Joystick sensor

（ジョイスティックセンサー）



Joystick sensor

ジョイスティックの動きやジョイスティック付随するイベント（ハット、ボタンなど）を検出します。ジョイスティックは複数使用できます（「Index」をご覧ください）。ジョイスティックの操作ボタン等の正確な配置は使用するジョイスティックの種類や形式に左右されます。

共通オプションは センサー共通のオプションをご覧ください。

専用オプション:

Index（番号）

使用するジョイスティックを指定します。

All Events（全イベント）

指定したタイプのイベントはすべてトリガに使われます。

Joystick Events

Event Type（イベント種類）

イベントの種類を指定します。

Joystick Single Axis

Single Axis（単軸）

一つの軸を使います。

Axis Number（軸数）

1 = 横軸（左右）

2 = 縦軸（前後）

3 = パドル（上下）

4 = ジョイスティック（ねじり左右）

Axis Threshold（軸のしきい値）

精密さを指定（範囲は 0 – 32768）

Joystick Hat

Hat（ハット）

指定したハット番号を使います

Hat Direction: 使う方向を指定（up, down, left, right, up/right, up/left, down/right, down/left）

Joystick Axis

Axis(軸)

Axis Number: 軸を指定

Axis Threshold: 精密さを指定(範囲は 0 – 32768)

Axis Direction: 使う方向を指定(Left, Right, Up, Down)

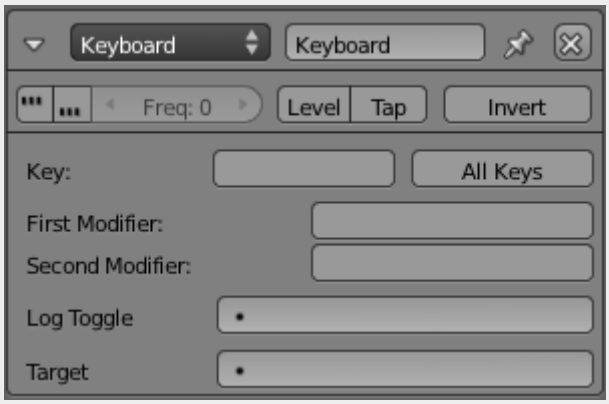
Joystick Button

Button(ボタン)

使うボタンの番号を指定します。

Keyboard sensor

(キーボードセンサー)



Keyboard sensor

キーボード入力を検出します。入力をString プロパティに保存することもできます。

共通オプションについては センサー共通のオプション をご覧ください。

専用オプション:

Key

ひとつのキーの押下を検出します。何も書かれていない欄をクリックして、検出したいキーを押します。これがアクティブキーになり、正パルスを発信します。この欄をクリックして続けて別の場所をクリックすると、キーの登録は消去されます。キーが離された時、偽パルスを発信します。

All keys ボタン

押すと他のボタンは隠され、すべてのキー押下に対して正パルスが発信されます。これは*Python*コントローラでカスタムキーマップを使う場合に便利です。

Modifier ボタン

Key と似ていますが、*Key* で指定したボタンが押されたときにいっしょに押下されていた場合だけ検出されます。キーは 2 つ指定できます。ここにキーが登録されると、両方またはすべてのキーが押された場合だけ正パルスが発信されます。これは CtrlR や⇧ ShiftAltEsc などの操作で特定のアクションをさせるときに便利です。

LogToggle

キーストロークが *String* プロパティに保存されるかどうかを *Bool* 値で指定します。

Target

キーストロークを保存する *String* プロパティを指定します。*Property* センサとともに使うと、たとえばパスワードの入力操作などに利用できます。

Message sensor

（メッセージセンサー）

Message sensor

テキストまたはプロパティ値を送信するのに使います。ゲームエンジンのどこかでメッセージが送られると、*Message* センサは正パルスを発信します。*Subject* で指定した件名の場合だけパルスを発信するようにもできます。

共通オプションについては See センサー共通のオプション をご覧ください。

専用オプション:

Subject

ここで指定したメッセージの受信時にセンサーを起動させます（空欄のままでも構いません）。

メッセージの送り方については Message Actuator をご覧ください。

Mouse sensor

（マウスセンサー）

Mouse sensor

マウス入力を検出します。使えるのは以下のイベントのみです。

- *Left button*.
- *Middle button*.
- *Right button*
- *Wheel Up*, スクロールアップ
- *Wheel Down*, スクロールダウン
- *Movement*, マウスによるあらゆる動き
- *Mouse over*, オブジェクトにマウスが重なったことを検出
- *Mouse over any*, 何らかのオブジェクトにマウスが重なったことを検出

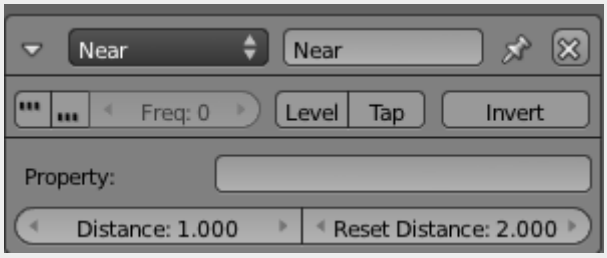
上記の状態が終わると、偽のパルスが送信されます。

具体的なマウスの動きや反応に対するロジックブリックはありません（たとえばプレイヤー視点でカメラを動かしたりなど）。それは Python で記述する必要があります。

センサー共通のオプション もご覧ください。

Near sensor

（近接センサー）



Near sensor

指定した距離内に近づいているオブジェクトを検出します。*Collision* と同様に、プロパティによってオブジェクトをフィルタリングすることもできます。

センサー共通のオプションについては こちら をご覧ください。

専用オプション

Property

指定したプロパティを持つオブジェクトのみに検出を制限できます。

Distance

オブジェクト同士の接近を検出する距離です（Blender 単位）。

Reset

センサがリセットされる（負パルスを発信する）距離です。

註釈

1）近接センサーの検出は他のオブジェクト（壁など）を「突き抜けて」行われます

2）オブジェクトの物理設定で「Actor」を有効にしていないと検出されません

Property（プロパティ）センサー

Property sensor

Property（プロパティ） センサーは、自身を所有するオブジェクトのプロパティの変更を探知するセンサーです。

共通するオプションは 共通する Sensor オプションを参照してください。

特別なオプション

プロパティの評価

Evaluation Type（評価タイプ）

*Property（プロパティ）*が *Value（値）*に対してどう評価されるかを指定します。

Greater Than（大きい）

プロパティの値がセンサー内の *Value（値）*より大きい場合（値は含まない）、TRUE パルスを送ります。

Less Than（小さい）

プロパティの値がセンサー内の *Value（値）*未満の場合、TRUE パルスを送ります。

Changed（変更された）

プロパティの値が変更されると、すぐさま TRUE パルスを送ります。

Interval（区間）

プロパティの値がセンサーの *Min（最小）* and *Max（最大）*の間※の場合、TRUE パルスを送ります（※最小と最大は含まない）。

Not Equal（違う）

プロパティの値がセンサー内の *Value（値）*と違う場合、TRUE パルスを送ります。

Equal（同じ）

プロパティの値がセンサー内の *Value（値）*と適合する場合、TRUE パルスを送ります。

他のプロパティ名を入力し、プロパティ同士を比較することもできます。

Radar sensor

（レーダーセンサー）

Radar sensor

Near センサに似ていますが、軸から特定の角度の範囲（オブジェクトの中心を指す見えない円錐形）への進入を検出し、また接近は軸上での距離で測られます。

センサー共通のオプションについては こちら をご覧ください。

専用オプション

Property: 指定されたプロパティを持つオブジェクトのみに検出を限定します。

Axis: レーダーの方向を決めます。+ および − の記号は軸上での正負の向きです。

Angle: 円錐の広がりを指定します。

Distance : 円錐の長さを指定します。

このセンサーはたとえば自動的に行動するキャラクターが正面方向しか見えないようにするのに便利です。ただし、他のオブジェクトの向こう側にあるオブジェクトも検出します。

Random sensor

（ランダムセンサー）

Random sensor

Seed 値をもとにして、ランダムなパルスを発信します。*Seed* が 0 だとランダムでないパルスになり、これはテストなどに使います。値の範囲は 0 – 1000 です。

Seed 値を変えずに再実行すると、パルスの間隔は同じ並びになります。

センサー共通のオプションについては こちら をご覧ください。

Ray sensor

（レイセンサー）

Ray sensor

指定した軸方向にレイを発射して、何かに当たると正パルスを発信します。指定したプロパティまたはマテリアルを持つオブジェクトのみ検出することもできます。

センサー共通のオプションについては こちら をご覧ください。

専用オプション

ボタンなどは *Radar* と多くが共通しています。

Property

指定されたプロパティを持つオブジェクトのみに検出を限定します。

註釈

1) Property 欄が設定されていない限り、このセンサーは他のオブジェクト（壁など）を通過してオブジェクトを検知します

2) 「Actor」が有効でないオブジェクトは検知されません

Axis

光線の方向を決めます。+ および - の記号は軸上での正負の向きです。

Range

光線の長さを決めます（Blender 単位です）

X-Ray Mode ボタン

X 線モードを有効にします。これは指定されたプロパティまたはマテリアルを持たないオブジェクトを通り抜けるようになります。

Touch sensor

（タッチセンサー）

Touch sensor

他のオブジェクトとぶつかったときに正パルスを発信します。 *Material* 欄でマテリアルによるフィルタリングができます。この欄に指定したオブジェクトとぶつかった場合だけパルスが発信されるようになります。すべてのオブジェクトを検出する場合は空にします。衝突が起こったときに正パルスが発信され、離れたときに負パルスが発信されます。"True Pulse triggering"を有効にすると、オブジェクトに触れている間はパルスが発信され続けます。

センサー共通のオプションについては こちら をご覧ください。

センサの種類

Actuator sensor

Actuator sensor

Actuator 欄に指定したアクチュエータが起動すると発信します。

GameLogic python API.

Always sensor

Always sensor

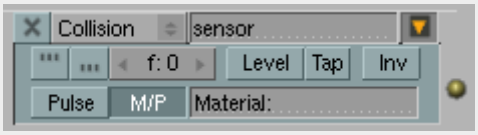
決まった ticks ごとに発信する必要がある場合、またはスタートアップ時に発信させたい場合（*Tap* を有効にする）に使います。

このセンサには共通オプション以外の設定項目はありません。

GameLogic python API.

Collision sensor

Collision sensor



Collision sensor

Touch センサに似ていますが、プロパティによってフィルタリングできます。指定したプロパティを持つオブジェクトが衝突した場合のみ正パルスが送られます。どのオブジェクトに対しても動作させるには欄を空のままにします。

Pulse ボタンは、直前のパルスの要因となったオブジェクトとまだ触れているときにも、他のオブジェクトとの衝突を検出します。

M/P ボタンはフィルタリングする要素をマテリアルとプロパティで切り替えます。

Delay sensor

Delay sensor

指定した ticks だけ反応を遅らせます。これは他のアクションが先に起こるべきである場合や、時間によって起こるイベントの場合に便利です。

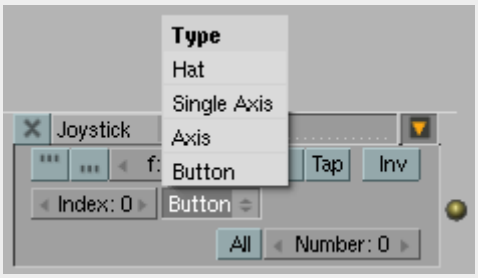
3 つのオプションがあります。

- *Delay* 正のパルスを送る前に待つ ticks 数
- *Duration* 負のパルスを送る前に待つ ticks 数。
- *Repeat* 遅延時間が終わった後、再始動します。

GameLogic python API.

Joystick sensor

Joystick sensor



Joystick sensor events menu

ジョイスティックイベントを検出します。

Index で、使用するジョイスティックを指定します。

Event Type メニューでイベントの種類を指定します。

Button

使うボタンの番号を指定します。

All Events

指定したタイプのイベントはすべてトリガに使われます。

Axis

Axis Number 軸を指定

Axis Threshold 精密さを指定

Axis Direction 使う方向を指定: Left, Right, Up, Down

Hat

指定した *Hat number* を使います

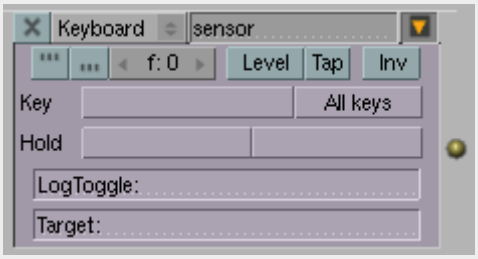
Hat Direction 使う方向を指定: up, down, left, right, up/right, up/left, down/right, down/left.

Single Axis

一つの軸を使います。

GameLogic python API.

Keyboard sensor



Keyboard sensor

キーボード入力を検出します。入力をStringプロパティに保存することもできます。

Key 欄はひとつのキーの押下を検出します。何も書かれていない欄をクリックして、検出したいキーを押します。これがアクティブキーになり、正パルスを発信します。この欄をクリックして続けて別の場所をクリックすると、キーの登録は消去されます。

All keysボタンを押すと、他のボタンは隠され、すべてのキー押下に対して正パルスが発信されます。これは*Python*コントローラでカスタムキーマップを使う場合に便利です。

Modifier ボタンは *Key* と似ていますが、*Key* で指定したボタンが押されたときにいっしょに押下されていた場合だけ検出されます。キーは 2 つ指定できます。ここにキーが登録されると、両方またはすべてのキーが押された場合だけ正パルスが発信されます。これは CtrlR や⇧ ShiftAltEsc などの操作で特定のアクションをさせるときに便利です。

LogToggle 欄にはキーストロークが *String* プロパティに保存されるかどうかを *Bool* 値で指定します。

Target 欄にはキーストロークを保存する *String* プロパティを指定します。*Property* センサとともに使うと、たとえばパスワードの入力操作などに利用できます。

GameLogic python API.

Message sensor

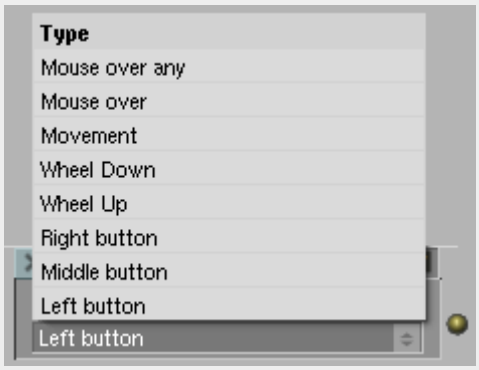
Message sensor

テキストまたはプロパティ値を送信するのに使います。ゲームエンジンのどこかでメッセージが送られると、*Message* センサは正パルスを発信します。*Subject* で指定した件名の場合だけパルスを発信するようにもできます。

GameLogic python API.

Mouse sensor

Mouse sensor



Mouse sensor input type

マウス入力を検出します。使えるのは以下のイベントのみです。

- *Left button*.
- *Middle button*.
- *Right button*
- *Wheel Up*, スクロールアップ
- *Wheel Down*, スクロールダウン
- *Movement*, マウスによるあらゆる動き
- *Mouse over*, オブジェクトにマウスが重なったことを検出
- *Mouse over any*, 何らかのオブジェクトにマウスが重なったことを検出

具体的なマウスの動きや反応に対するロジックブリックはありません（たとえばプレイヤー視点でカメラを動かしたりなど）。それは Python で記述する必要があります。

GameLogic python API.

Near sensor

Near sensor

指定した距離内に近づいているオブジェクトを検出します。*Collision* と同様に、プロパティによってオブジェクトをフィルタリングすることもできます。

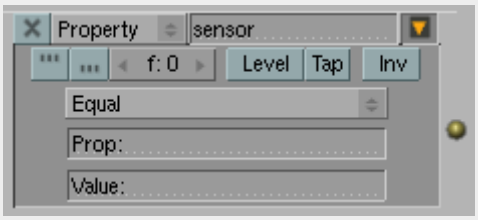
Property 欄で指定したプロパティを持つオブジェクトのみに検出を制限できます。

Distance はオブジェクト同士の接近を検出する距離です（Blender 単位）。

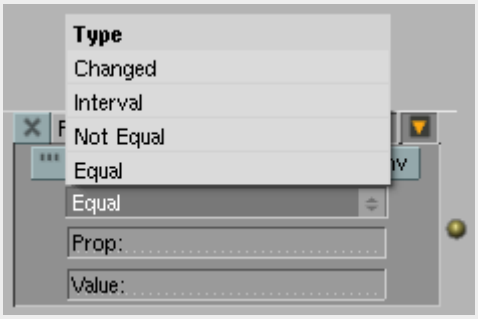
Reset はセンサがリセットされる（負パルスを発信する）距離です。

GameLogic python API.

Property sensor



Property sensor



Property sensor operation type

プロパティの変更を検出します。*Property* センサには 4 つのモードがあります。

Equal はプロパティ値とセンサの値が等しい場合に正パルスを発信します。*Prop* はプロパティ名、*Value* に正パルスを発信する条件となる値を指定します。

Not Equal はプロパティ値とセンサの値が異なる場合に正パルスを発信します。入力欄は *Equal* と同じですが、*Value* にはプロパティ値と一致しないことが条件となる値を指定します。

Interval は、プロパティ値がセンサの *Min* から *Max* の価の範囲にある場合に正パルスを発信します。最低値のみを指定したい場合は、自身のプロパティ名を *Max* 欄に、パルスを送る最低値を *Min* 欄に入力します（訳注：最高値は自分自身の値なので、常に範囲内にある）。最高値のみを指定したい場合は、自身のプロパティ名を *Min* 欄に、パルスを送る最高値を *Max* 欄に入力します。値を比べるために他のプロパティ名を入力することもできます。
Changed はプロパティの値が変わったときに正パルスを発信します。

GameLogic python API.

Radar sensor

Radar sensor

Radar sensor axis menu

Near センサに似ていますが、軸から特定の角度の範囲（オブジェクトの中心を指す見えない円錐形）への進入を検出し、また接近は軸上での距離で測られます。

Property 欄は、指定されたプロパティを持つオブジェクトのみに検出を限定します。

Axis メニューは、レーダーの方向を決めます。+および-の記号は軸上での正負の向きです。

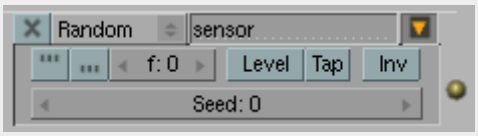
Angle で円錐の広がりを指定します。

Distance で円錐の長さを指定します。

これはたとえば自動的に行動するキャラクターが正面方向しか見えないようにするのに便利です。ただし、他のオブジェクトの向こう側にあるオブジェクトも検出します。

GameLogic python API.

Random sensor

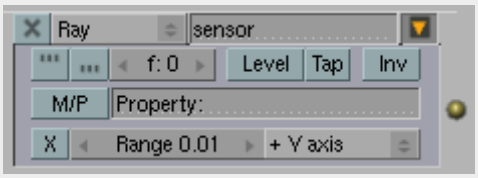


Random sensor

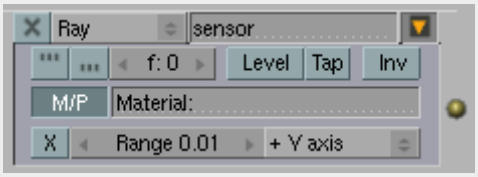
*Seed*値をもとにして、ランダムなパルスを発信します。*Seed*が０だとランダムでないパルスになり、これはテストなどに使います。

GameLogic python API.

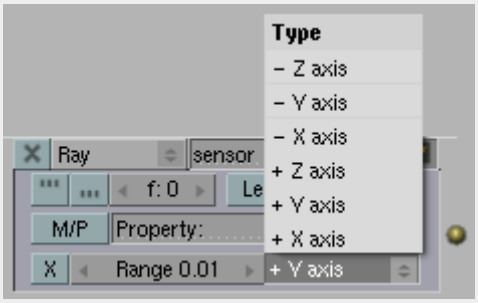
Ray sensor



Ray sensor (toggle collision on Property)



Ray sensor (toggle collision on Material)



Ray sensor axis menu

指定した軸方向にレイを発射して、何かに当たると正パルスを発信します。指定したプロパティまたはマテリアルを持つオブジェクトのみ検出することもできます。

ボタンなどは *Radar* と多くが共通しています。

X-Ray Mode は X 線モードを有効にします。これは指定されたプロパティまたはマテリアルを持たないオブジェクトを通り抜けるようになります。

GameLogic python API.

Touch sensor

Touch sensor

他のオブジェクトとぶつかったときに正パルスを発信します。 *MA* 欄でマテリアルによるフィルタリングができます。この欄に指定したオブジェクトとぶつかった場合だけパルスが発信されるようになります。すべてのオブジェクトを検出する場合は空にします。衝突が起こったときに正パルスが発信され、離れたときに負パルスが発信されます。"True Pulse triggering"を有効にすると、オブジェクトに触れている間はパルスが発信され続けます。

GameLogic python API.

States

(ステート/状態)

BGE では、オブジェクトは複数の「ステート」を持つことができます。ゲームのプレイ中はいつでも、オブジェクトの現在のステートによって行動が決まります。例えば、ゲーム内のキャラクターが覚醒、睡眠、死亡の各ステートを持っているものとします。大きな音がしたときの彼の行動はいつも、現在の状態によって決まります。うずくまる(覚醒時)、目を覚ます(睡眠時)、何もしない(死亡時)といった具合です。

How States Operate

(ステートの操作方法)

ステートはコントローラーを通じて作成し、使用します。ステートのシステムに直接操作されるのはアクチュエータやセンサーではなく、コントローラーだけであることに注意してください。各オブジェクトは複数のステートを持つことができますが(上限は 30 で、デフォルトは 1 です)、ある時点でのオブジェクトのステートは必ずひとつに決まります。コントローラーは常に、操作対象となるステートを定義しなければなりません。a) ロジックの条件を満たし、かつ b) オブジェクトが現在、指定したステートにある場合にだけ、出力パルスを送ります。ステートはオブジェクトのコントローラー設定で作成し、編集します(詳細は下記をご覧ください)。

単純なゲームでは、ステート設定は自動で行われます。デフォルトで、各オブジェクトはひとつのステートを持ち、すべてのコントローラーがステート 1 を使うよう設定されます。したがってゲームに複数のステートが必要ないのなら、ステートをあえて設定しなくともすべて動作するでしょう - ステートについて頭を悩ます必要はまったくありませ

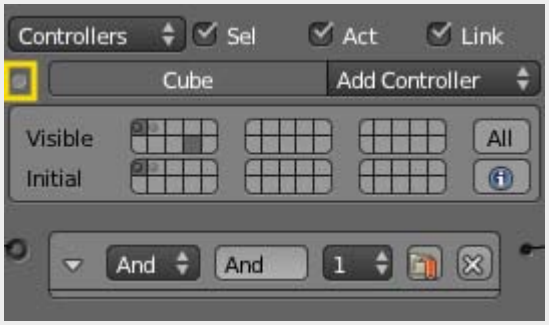
ん。

{{{2}}}

アクチュエータの一つである State アクチュエータは、オブジェクトのステートビット値を設定したり解除したりでき、センサーの信号に対するオブジェクトの反応を現在のステートによって決めることができます。上の例では、キャラクターには「大きな音」センサーにつながった「覚醒時」「睡眠時」「死亡時」ステート用のコントローラーがあります。これでキャラクターの現在のステートによって異なった動作を行わせることができ、この中にはある条件下でキャラクターのステートを切り替えるものがあるかもしれません。

Editing States

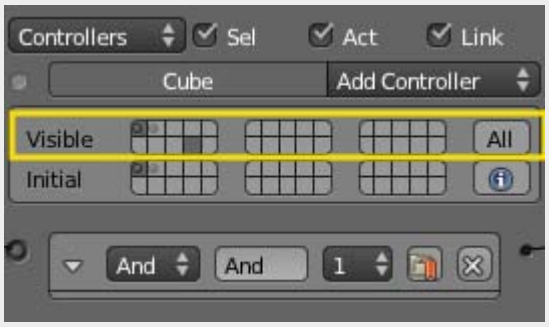
（ステートの編集）



State Panel Button

ステートは、Game Logic パネルのコントローラー（中央）列を使って作成し、編集します。ステートパネルを表示するには、ステートパネルボタンをクリックします。パネルには利用できるステートが 30 ずつ、二つにわけて表示されます。Visible ステートと Initial ステートです（下記参照）。ゲームのステートシステムを設定するには、オブジェクトのロジック内で、各コントローラーに妥当なステートを選びます。

オブジェクトのステートやその関連項目の表示はそのオブジェクトのステートパネルを使って行います。ボタンを使ってオンとオフを切り替えます。パネルは Visible および Initial のふたつの部分にわかれています。

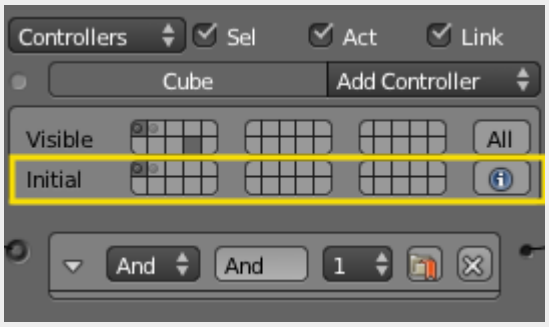


State Panel Visible

Visible ステート

Visible 領域では、利用できる 30 のステートがそれぞれ、薄い灰色の四角形で表現されています。このパネルは、オブジェクトのロジックブリックで見えるロジックを示しています。右側にある All ボタンをクリックすると、オブジェクトの論理ブリックのすべてが表示され（もう一度クリックすると非表示）、ステートパネルの四角はすべて薄い灰色になります。もしくは、個々のステートをクリックしてそのロジックを目に見えるようにすることができます（複数の四角をクリックできる点に注意してください）。もう一度四角をクリックすればステートの選択を解除します。

使用中のステート（＝そのステートで実行されるコントローラーがオブジェクトにある場合）にはドットがつき、Game Logic にコントローラーを表示すると、四角は濃い灰色になります。列見出し部分にあるステートボタンを使って、連結されたセンサーやアクチュエータの表示を切り替えられます。



State Panel Initial

Initial ステート

Initial 領域でも、利用できる 30 のステートが薄い灰色で表現されています。ゲームが実行されたとき、オブジェクトの開始時のステートとして、このうち一つをクリックできます。

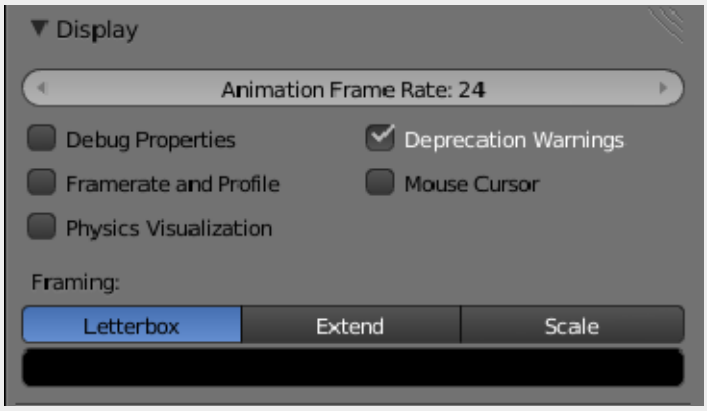
右側には I (Information) ボタンがあります。クリックして Game メニューの「Show Debug Properties」メニューをクリックすると、ゲームの実行中、そのオブジェクトの現在のステートが画面左上隅に表示されます。

Display

（表示）

プロパティウィンドウの *Render* にある *Display* では、ゲーム実行中のアニメーションの最大フレームレート、フレームレートや分析情報の表示、デバッグ用のプロパティ、物理演算用のジオメトリの視覚化、警告、ゲーム実行中のマウスカーソルの表示、指定解像度を持つウィンドウに合わせてゲームの画面枠の決め方を変えるオプションを指定できます。

Options



Render にある Display

Animation Frame Rate

ゲーム実行中の最大フレームレート。最小は 1、最大は 120 です。

Debug Properties

有効にすると、ゲーム実行中にデバッグする設定にしていたプロパティの値が フレームレートや分析情報 といっしょに表示されます。

Framerate and Profile

有効にすると、ゲーム実行中に行われる計算値と、デバッグする設定のプロパティをすべて表示します。各値については フレームレートや分析情報 で説明されています。

Physics visualization

物理演算に使われる（殻や衝突形状のような）範囲や相互作用を視覚的に表示します。

Deprecation Warnings

非推奨の機能が使われると警告を表示します。非推奨の機能とは古すぎるか利用不能な OpenGL グラフィックカードの機能などが該当します。

Mouse Cursor

ゲーム実行中にマウスカーソルを表示するか非表示にするかを選びます。

Framing

Blender Game Engine の画面枠の決め方を *Letterbox*、*Extend* および *Scale* から選べます

Letterbox

必要なら縦横に余白を置いて、表示ウィンドウにゲームの描画領域全体を表示します。

Extend

縦または横方向に視界を広げることで、表示ウィンドウにゲームの描画領域全体を表示します。

Scale

表示ウィンドウに合わせて、ゲームの描画領域を伸縮します。

Color Bar

画面枠を **Letterbox** 形式にしたときの余白の色を指定できます。

Performance

（性能）

ゲームを開発するときにゲーム開発者やソフトウェア/ハードウェア開発者は、ユーザーがそのゲームで最高の体験ができる基本利用環境を決め、いくつかのツールを使ってゲームを特定のプラットフォームや OS にあわせて最適化します。

こうしたツールのほとんどは、ゲームが開発実行される特定のゲームエンジンに向けたソフトウェアです。

Blender Game Engine にも開発中のゲームを最適化するための視覚的なツールがあり、ゲーム開発者は最適な利用環境と、ゲーム実行に最低限必要なソフトウェアおよびハードウェアをテストすることができます。

こうしたツールを、Blender では *Properties Window* の *Render* コンテキストにある *System* および *Display* から利用できます。 特定の性能調整と計測用オプション、ゲーム実行中のフレームレートや Blender ウィンドウ（ゲームビューポート）へのレンダリング方法の調整オプションの他、グラフィックカードのメモリに割り当てられたジオメトリの管理用コントロールがあります。

Blender Game Engine レンダリングシステムのコントロール

System/システム – ゲーム実行中の、シーンのレンダリング用コントロール

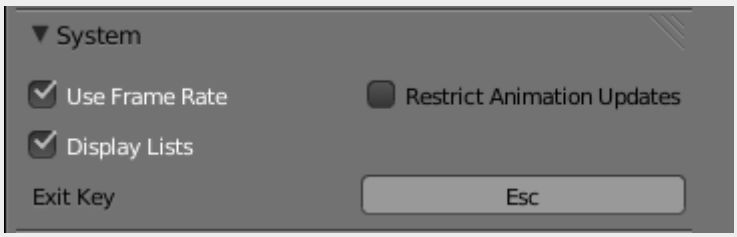
Blender Game Engine 性能の計測

Display/表示 – ゲーム実行中の、性能に関する特定データの表示用コントロール

System

プロパティウィンドウの Render タブにある *System* では、ゲーム開発者はフレーム廃棄やフレーム描画の制限に関わるシステム性能のオプション、Blender Game Engine を止めるキー、ジオメトリの管理をグラフィックカードの内蔵メモリで行う、といったオプションを指定できます。

Options



System tab at the Render Context

Use Frame Rate（フレームレートの使用）

フレームレートの制限なしに実行できるかどうかを決めます。フレームレートはプロパティウィンドウの *Render* タブにある *Display* で指定します。フレームレートに関するさらなる情報は Display をご覧ください。

Display Lists（表示一覧）

有効にすると、Blender は GPU メモリに割り当てられているメッシュジオメトリの一覧を管理します。ジオメトリとテクスチャを割り当て可能な GPU メモリがあれば、ゲーム実行中のビューポートレンダリングを高速化できます。

Restrict Animation Updates（アニメーション更新の制限）

有効にすると、Display タブで指定された値よりも GPU のフレームレートが大きくなると、ゲームエンジンがフレームを廃棄します（再描画の途中であっても廃棄し、画像が「引き裂かれたように」ひずむことがあります）。

Exit Key

実行中のゲームエンジンを終了するキーを設定できます。このボタンをクリックして、終了に使いたいキーを入力します。デフォルトは Esc キーです。

Game Physics

ゲーム用の物理処理の設定はWorldパネルにあります。その設定によってゲームエンジン内のシーンにおける物理のルールおよび重力が決まります。選択された物理エンジンに基づいてBlenderは自動的にアクターを下方に動かします。 望みの動きになるようにアクターを調整したあと、その動きを固定したIpoカーブとしてベイクすることもできます（詳細はロジックのアクターを参照）。

オプション

Physics Engine

使用する物理エンジンの種類を選択します。

Bullet

デフォルトの物理エンジンです。活発に開発中です。これは移動と衝突検知をあつかいます。衝突によって運動量が移動します。

None

物理計算をしません。物体は重力の影響を受けることはなく、移動しているオブジェクトはその動きを続けます。

Gravity

重力加速度です（$m.\ s^{-2}$）。ここのオブジェクトは質量と大きさのスライダを持っています（オブジェクトの物理を参照）。フレームレートを考慮する際に（Renderを参照）、Blenderはこの情報を使ってオブジェクトがどのくらいの速さで下方に加速されるかを計算します。

Physics Steps

Max

描画のためにゲームスピードが遅くなる場合の、1 ゲームフレームあたりの物理計算ステップの最大値。高い数値にすると物理計算がリアルタイムに追いつきやすくなります。

Substeps

物理計算ステップあたりのシミュレーションサブステップの数。高い数値にするほど物理的に正確になります。

FPS

ゲームフレームレートの設定値。物理計算の固定タイムステップの長さは実際のフレームレートとは関係なくこの設定値の逆数になります。

Logic Steps Max

描画のためにゲームスピードが遅くなる場合の、1 ゲームフレームあたりのロジックフレーム数の最大値。高い数値にするほど物理計算と同期します。

Occlusion Culling

オクルージョン・カリング（隠れたポリゴンを省略する技術）と視野錐台に最適化された Bullet DBVT を使います。

Resolution

オクルージョンバッファのピクセルサイズ。高い数値にすると正確になります（しかし遅くなります）。

オブジェクトの種類

Objects type menu

ゲームエンジンにおいて、オブジェクトに対する物理計算にはいくつかの種類があります。*Physics Type*メニューでオブジェクトに適用する物理計算を選び、プロパティを設定します。するとオブジェクトは物理エンジンによって評価されるようになり、通常のオブジェクトとは異なる扱われ方になります。

設定

オブジェクトに対する物理計算の種類はプロパティウィンドウの Physics パネルで設定します。

以下のオブジェクトが選べます。設定項目についてはそれぞれの頁で解説されています。

- Occluder
- No Collision
- Sensor
- Static
- Dynamic
- Rigid Body
- Soft Body

衝突境界（Collision Bounds）

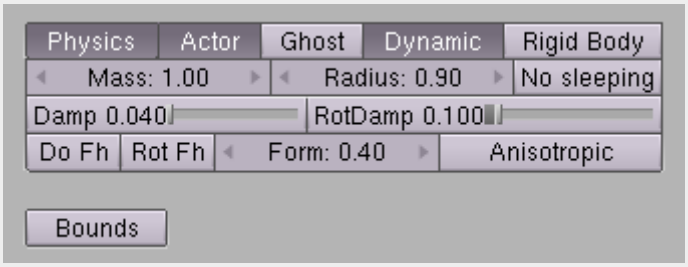
それぞれの物理タイプには衝突境界の設定があります。Blender ゲームエンジンにはオブジェクトの形状を扱う方法が何種類もあります。ひとつの面ごとに衝突を高精度で計算することをゲームエンジンが避ける理由は、それには高い処理能力を必要とするからです。箱型や球体に近い形のオブジェクトについては、箱型や球体として扱うと計算が速くなります。

境界はオブジェクトの中心から計算されます。中心はふつうピンクの点で表示されています。より便利な位置に中心点を動かすこともよく行われます。それには*Editing*（F9）で 3 つのボタンのうちのどれかを使います（オブジェクトの中心を参照）。

Convex Hull および *Triangle Mesh* はワールドオブジェクトに使うと良いでしょう。床、壁、木など。

Blender にはくぼんだ形のオブジェクトを扱う一般的な方法はありません。たとえば、コップの中でサイコロを転がす動きなどは計算するのが難しいです。

アクティブでないワールドにおける物理計算



Objects physics with no active world

シェーディングコンテキスト（ボタンウィンドウ、F5）においてアクティブなWorldがない場合は、*Object type*のかわりに"Physics"というボタンが見えるでしょう。それを有効にすると、図"*Objects physics with no active world*"のようなボタンが現れます。この場合に私用できるオプションについてはここを参照.

基本的にはワールドは作成しなければいけません。ワールドを作成すると物理計算の環境もセットアップされるからです。

Dynamic

有効にすると、オブジェクトはリアルな物理的性質を持ちます。重力に引かれたり、他のオブジェクトにぶつかってはね返ったり、質量をもったり、摩擦によってスピードダウンしたりします。

設定

Actor

Staticを参照。

Ghost

Staticを参照。

Invisible

Staticを参照。

Use Material Force Field

有効にすると、オブジェクトはDynamic Materialsと相互作用します。現在これはデフォルトのbullet物理エンジンではサポートされておらず、Sumoエンジンでも完全にはサポートされていません。Blender2.24 では完全にサポートされています。

Rotate From Normal

オブジェクトを回転させるために法線を使います。サーフィスの方向とは違う方向を指します。

No Sleeping

デフォルトでは、回転の物理計算の適用されているオブジェクトの動きが止まると、回転物理は無効にされます。そのオブジェクトの動きを計算する必要はなくなったと物理エンジンが判断するためです。これはある場合には正しく、またコンピュータの処理能力を節約することになります。しかし別の場合、たとえばサッカーゲームなどでは、誰かがボールをまた蹴って動かすまで物理計算を維持する必要があります。

Attributes

Mass

質量は、オブジェクトを動かすのに必要な力の大きさに関わります。質量が大きいほど、オブジェクトを動かすのにより大きな力が必要になります。質量は物体が落ちる速さには関係しません。速度を落とすにはdampeningを使ってください。物体が落ちる速度を上げるにはgravityを上げてください（通常は 9.81 です）。

Radius

球形の境界（bound）を設定した場合の球の半径、および Fh/FhRot を決めます。オブジェクトの見た目と実際の大きさの関係を変えるためにこの数値を変えます。境界は *Bounds* で有効になり、そのデフォルトモードは *Box* です。
境界の半径を変える際、画面上に円が表示され効果を確認できます。この瞬間は 3D ウィンドウにおいてすべての境界形状が円として反映されます。それはゲーム中の反応とは違う場合があります。

Form Factor

剛体（rigid body）オブジェクトがコントロールされる度合いです。数値を高くすると、オブジェクトが転がりにくくなります。とくに平らな表面上において効果ができます。

Anisotropic Friction

Staticを参照。

Velocity（速度）

Minimum

速度の最小値を設定します。

Maximum

速度の最大値を設定します。

Damping（減衰）

Translation

減衰はオブジェクトの動きの自由度に影響します。宇宙空間では減衰はほぼまったくないと考えられます。一方、水中の場合は減衰の効果を非常に高くするべきです。

Rotation

通常の減衰と似ていますが、これはオブジェクトの回転に影響します。数値を高くしても、他の種類の運動量には影響しません。

Lock Translation and Rotation

オブジェクトの回転および移動を、任意の X、Y、Z 軸の組み合わせのみに制限できます。

Collision Bounds

Object Typeを参照。

No Collision

オブジェクトの衝突を無効にします。他のオブジェクトから決して触れられないようなオブジェクトにこれを適用すると、パフォーマンスの節約になります。

Invisible

Staticを参照。

Occluder

選択的オクルージョン（occlusion＝ふさぐ）とは、オブジェクト（オクルーダー）が他のオブジェクトを隠す（GPU に送らないようにする）能力のことです。実装は tree selection と同じです。

オクルージョンはデフォルトでは有効になっていません。これを有益に使うためにはよく理解する必要があるからです。適切に使用されない場合、ゲームがスローダウンすることがあります（ただしそれほど多大にではありませんが。処理に使う時間は自動的に調整されます）。

動作の仕組み

オクルージョンはすくなくとも一つのオクルーダーが存在する場合に起動します。オクルーダーの設定は physics パネルにあります。

オクルーダーはネットワークベースのオブジェクトのみに使われなくてはいけません。これは物理計算に関する限り、衝突計算をしないということと同じです。オクルーダーモードが他の物理モードと相互排他的である理由は、オクルーダーはそれ専用にデザインされるべきで、すべてのメッシュをオクルーダーにするべきではない、ということを強調するためです。しかしながら、Python とロジックブリックを使って、他の物理オブジェクトにオクルーダーの機能を与えることも可能です。これについては後で触れます。

オクルーダーオブジェクトが視野錐台に入ってくると、ゲームエンジンはそのオブジェクトの面から Z デプスバッファを作成します。面が片面表示か両面表示かは重要です。表側が見えている面および両面表示になっている面だけが Z デプスバッファの作成に使われます。複数のオクルーダーが視野錐台内にあるときは、ゲームエンジンはそれらを結合して最も手前にある面を残します。

Z デプスバッファの解像度は World 設定の"Occlu Res"ボタンでコントロールできます。

デフォルトでは、解像度はビューの最も長い辺に対して 128 ピクセル、他の辺に対しては長さに応じて設定されます。128 というのは非常に低い解像度ですが、カリング（描画から除外すること）のためには十分です。解像度は 1024 まで上げられますが、CPU の使用も増大します。

ゲームエンジンは DBVT ツリーを走査して、個々のノードが完全にオクルーダーに隠されるかをチェックします。隠される場合は、そのノード（およびそれが内包するすべてのオブジェクト）を消去します。

更なる最適化のため、ゲームエンジンはすくなくとも一つのオクルーダーが視野錐台にあるときのみ、Z デプスバッファを作成、使用します。それ以外の場合はオクルージョンを使わない場合と比べてパフォーマンスの低下はありません。

使い方

オクルージョンカリングを使うのが有益でない場合もあります：

- オクルーダーが小さく、多くのオブジェクトを隠さない場合。
  この場合、オクルージョンカリングは CPU を遅くするだけです。
- オクルーダーが大きいが、隠されるオブジェクトがシンプルな場合。
  この場合、オブジェクトを GPU に送ってしまったほうが良いでしょう。
- オクルーダーが大きく、多数の複雑なオブジェクトを隠すが、それが非常に予想しやすい状況の場合。
  たとえば、複雑なオブジェクトがたくさん詰まった家などです。ここでオクルージョンカリングを使ってもうまく働きますが、特定のロジックを作って表示・非表示を切り替えたほうがもっとパフォーマンスはあがります。たとえば、カメラが屋内にあるときだけ表示されるようにオブジェクトを設定するなど。

オクルージョンカリングが最も意義を持つのは、オクルーダーが大きく（ビルや山など）、多数の複雑なオブジェクトを隠し、それが結果を予想しにくい状況である場合です。しかしながら、パフォーマンスをそれほど気にし過ぎなくても良いです。適切でない使い方をしたとしても、パフォーマンスの低下はある程度に抑えられるアルゴリズムになっています。

オクルーダーは視覚的に描画されるオブジェクトであってもかまいませんが、面が多すぎると Z デプスバッファの作成に時間がかかるので注意してください。たとえば、地表はオクルーダーの候補としてはあまり良くありません。面や重なっている部分が多すぎます。オクルーダーをシンプルかつ透明（invisible）にして、複雑なオブジェクトの内部に配置することもできます（たとえば複雑な建物の壁の内側に沿って）。オクルーダーに穴をもたせて、そこからオブジェクトが見えるようにすることもできます。

複雑な地形からオクルーダーを作るときにありうる手順を以下に示します。

- 地形を複製して、低ポリゴンに作り直す。
- 水平になっているパーツをすべて削除して、急勾配な部分だけを残す。
- すべての面を透明にして、オクルーダーモードをセットする。

さらに最適化するため、オクルーダーをいくつかのオブジェクトに分割して（個々の山や壁からそれぞれオクルーダーを作る）、地表と同じ作業をします。このようにすると、オクルーダーに隠された地形がレンダリングされなくなります。

ワイヤーフレームモードでゲームを実行すると、オクルージョンが動作していることを確認できます。

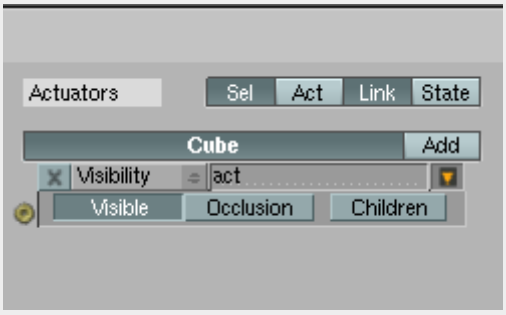
ゲーム内でのコントロール

Python を使って、オブジェクトのオクルージョン能力を有効／無効にできます。

```
obj.occlusion = True
obj.setOcclusion(True, False)    #param1=occlusion, param2=recurse in children
```

オブジェクトは必ずしも GUI 上でオクルーダーとして設定されていなくてもかまいません。どのようなメッシュでも、static オブジェクトでも dynamic オブジェクトでも、オクルージョン能力を有効にできます。

Visibility アクチュエータを使うこともできます:



アクチュエータのレイアウトが変わったので注意してください。排他選択の Visible ボタンと Invisible ボタンは、別々の機能に関する二つのオン・オフボタンに置き換えられました（Visible/Invisible と Occluding/Non-occluding）。

アクチュエータを使う場合は、Visibility と Occlusion 両方の設定（有効／無効）を指示したことになってしまいます。個別に設定する場合は Python を使ってください。

既知の制限

- オクルージョンカリングを使う場合、ゲームが始まった時点ですくなくとも一つのオクルーダーが存在していなくてはいけません。オクルーダーはアクティブなレイヤーにある必要はありません。その後は、どのメッシュオブジェクトに対してもオクルージョンの有効・無効をゲーム内で設定できます。

Rigid Bodies

設定項目はDynamic Objectsと同じです。

これは回転の物理計算を有効にして、オブジェクトに生き生きした感じを与えます。たとえばボールが床に落ちた場合、床が水平だとしてもその場で止まるのは不自然でしょう。回転の計算を有効にすると、たとえ完璧に水平な面にオブジェクトが落ちたとしても自然な感じに転がるようになります。

Sensor

Sensor は static および dynamic オブジェクトを検知しますが、他の衝突、センサ、オブジェクトは検知しません。これは Near センサや Radar センサを設定した物理オブジェクトに似ています。Near オブジェクトや Radar オブジェクトと同様に、

- static かつ ghost です。
- デフォルトで invisible です。
- 正しく衝突を検出するため、常にアクティブです。
- static オブジェクトおよび dynamic オブジェクトを検出します。
- その親オブジェクトとの衝突は無視します。
- broadphase フィルタリングします。そのベースは、
    - Actor option: 衝突が検出されるためには Actor を有効にしている必要があります。
    - property/material: 適用されている衝突センサにおいて指定されているもの。

Broadphase フィルタリングはパフォーマンスのために重要です。衝突点は broahphase フィルタを通ったオブジェクトの場合だけ計算されます。

- 衝突センサがアクティブでない場合は自動的にシミュレーションから除去される。

Near オブジェクトや Radar オブジェクトとは違う点は、

- どのような形状をとることもできます。triangle mesh でも良いです。
- デバッグのために視覚化できます（Visible アクチュエータを使ってください）
- 複数の衝突センサがこれを使えます。

その他の点として、sensor オブジェクトはふつうのオブジェクトです。字通に動かしたり、親子の関連付けをしたりできます。dynamic オブジェクトを親にすると、そのオブジェクトに対して高度な衝突コントロールを提供できます。

衝突検知の種類は形状によって異なります。

- box、sphere、cylinder、cone、convex hull ではボリュームの衝突検知をします。
- triangle mesh ではサーフィスの衝突検知になりますが、マージンを設定すればサーフィスにボリュームを与えることもできます。マージンはサーフィスの両面に設定されます。

パフォーマンスのヒント:

- Sensor オブジェクトは Near や Radar よりも上手く動作します。シーングラフ最適化によって synchr

onizations が少なく、また複数の衝突センサを持つことができます（たとえばそれぞれ違ったプロパティによってフィルタリングしたりなどできます）

- 可能な場合はシンプルな形状（box や sphere）を使ってください。

- 常に broadphase フィルタリングを使用してください（フィルタリングするプロパティやマテリアルを設定せずにセンサを使わないでください）。
- 衝突センサは必要なときだけ使用してください。センサオブジェクト上で衝突センサが無効になっているときは、シミュレーション内から削除され、CPU を消費しません。

わかっている制限:

- Blender をデバッグモードで実行している場合、このような警告が出ます。
  "warning btCollisionDispatcher::needsCollision: static-static collision!"
  リリースモードではこのメッセージは出ません。
- sphere、cone、cylinder では衝突マージンは効果がありません。

設定

Invisible

Staticを参照。

Collision Bounds

Object Typeを参照。

Soft Body

衝突したときに変形するオブジェクトです。

Settings

Actor

Staticを参照。

Ghost

Staticを参照。

Invisible

Staticを参照。

Attributes

Mass

Dynamicを参照。

Welding

結合のしきい値。これよりも近い距離にある頂点は同じものであるとみなされます。0.0 に設定すると、チェックが無効になり、シーンのローディングが速くなります（重複した頂点がなければ無効にして大丈夫です）。

Position Iterations

位置の計算を反復する回数。数値を大きくすると正確になります。

Linear Stiffness

柔体のリンクの直線的な硬さ。

Friction

動摩擦です。

Margin

衝突計算の余裕。小さい値にするとアルゴリズムが不安定になります。

Bending Constraints

コンストレイントの制約も曲げます。

Shape Match

Threshold にもとづいてシェイプマッチングします。

Cluster Collision

Rigid to Soft Body

柔体と剛体の間で cluster collision を有効にします。

Soft to Soft Body

柔体同士の間で cluster collision を有効にします。

Iterations

クラスタの反復数。

Collision Bounds

Object Typeを参照。

Static（静止）オブジェクト

物理計算には関わりますが、重力の影響は受けないオブジェクトです。

Actor

有効にすると、物理エンジンの中でオブジェクトは世界（単純な壁や床）とは違った扱いになります。このオブジェクトはほかのアクティブなオブジェクトから見えるようになります。

この時点では、オブジェクトは通常の物理効果（たとえば重力）の影響は受けないがただの背景とも違った見られ方をする、という状態です。これは物理的な性質（重力など）をもつ必要がないオブジェクトの場合に便利です。たとえば壁についているボタンなど。

Ghost

有効にすると、他のオブジェクトをすり抜けるようになります。これは反発や摩擦などの物理計算の時間を節約できます。他のオブジェクトとの衝突は依然として検出され、ゲームロジックに報告されます。

Invisible

有効にするとオブジェクトが「透明（Invisible）」になり、レンダリングされなくなります。

Radius

境界とする球形および material physics の半径。

Anisotropic（異方性）

X、Y、Z それぞれの軸における摩擦力をコントロールします。異方性は特定の方向における摩擦に関係するものです。たとえば、スケートボードのホイールは前方には転がりやすいですが、横方向には滑りにくいです。