Prof. Monster's

BGE Guide to

Character Setup

by Monster
Version 1.1
Date 19 Jul 2012

This Guide uses:

- Images of ManCandy designed by Bassam Kurdali
- Images and Text by Monster

Images of Blender are from version 2.62

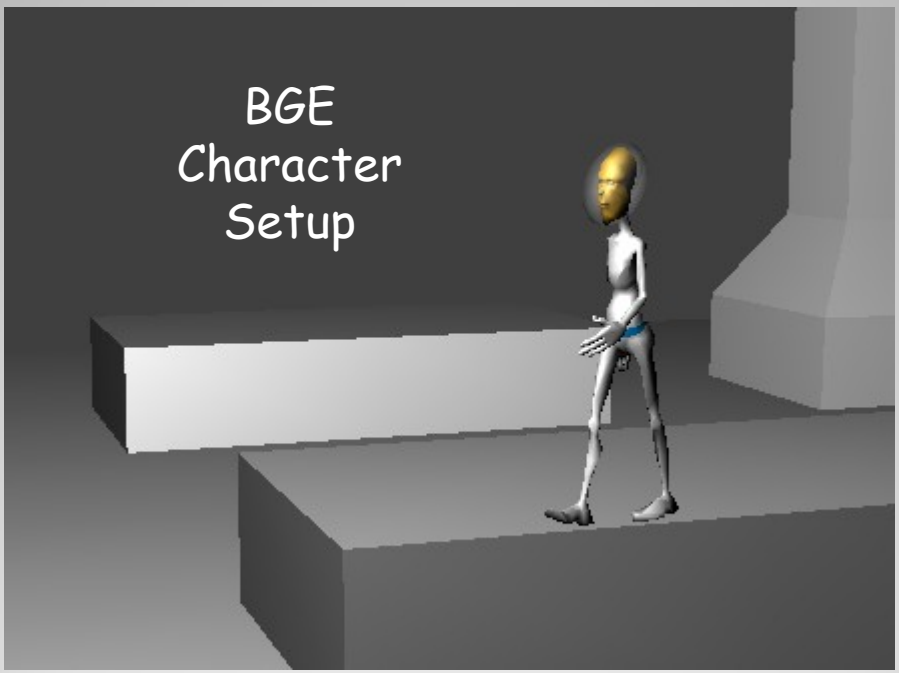
History:

1.1

- Added **Applying Skin**,
- Various small corrections on expressions

References

This guide shows how to can setup your game characters in an easy and flexible way. It allows you to separate physics from animation, presentation and behavior. This makes it very easy to add additional skins/clothes and gadgets.

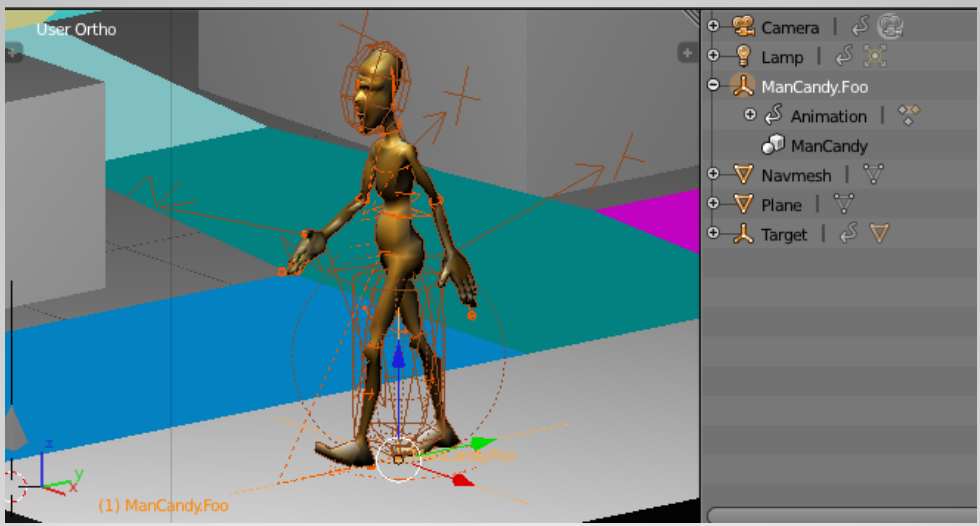


Purpose

A **character** appears as an object with complex animation and behavior. It is often used to act as avatar, person or animal within a game. Each part of a character can become quite complex.

All parts of the character belong together. They should be seen as one complex object – the **Character**.

Hint: All parts of a character can be placed in a group. This prevents the user (level designer) from incidentally destroying the character's structure.



What is a character?

A recommended object structure of a character is this:

- **Physics Object** (also called “Collision Object”)
  - **Armature Object** (also known as “Skeleton”)
    - **Skin Mesh Objects** (incl. Clothes known as “Skin”)
    - **Bone Hooks** (to parent other objects to them)

Other objects might be added to the character as well:
e.g:
Control objects
processing the character’s behavior
LOD manager
switching the details of the character
Sound source
Dealing with the Sound of the character
Cloth manager
Switching cloths
…

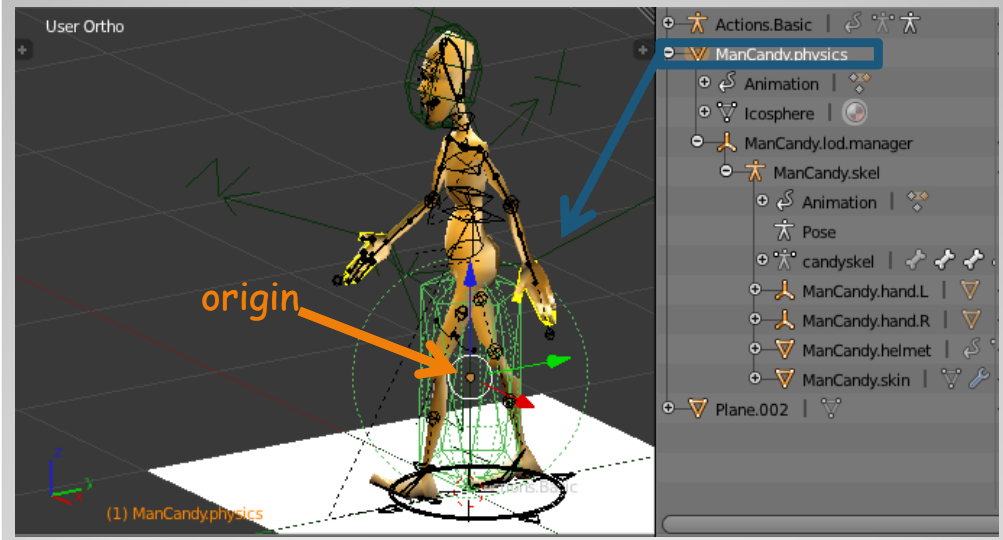
Structure

The **Physics Object** defines the physical presentation of the character. It performs the character's motion e.g. forward motion, turning, falling jumping.

The *Physics Object* interacts with the environment which means it reacts on collisions.

In a lot of game prototypes characters are visualized as cubes or spheres. This is completely valid as the Physics will be exactly that regardless what the final visual representation is.

All other parts of the character need to make sure not to interfere with the *Physics Object (*E.g. **Skin Mesh Objects** should be set to "Ghost" or "No Collision").



Physics

As the **Physics Object** defines the physical presentation of the character it is important to set up the correct physical parameters in the physics panel of this object and the physics parameter of the material.

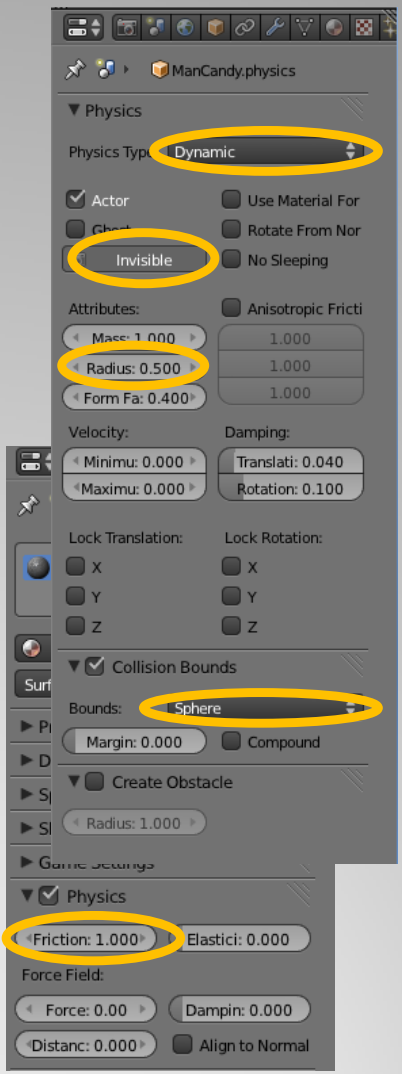
The Physics type determines how this object interacts with the world's physic.

The object itself should be invisible in the game.

The collision bounds do not need to match the visual representation. If a predefined shape is chosen take care of the radius parameter.

Different collision bound shapes result in different physical behavior.

The materials should have a proper friction to avoid unnecessary sliding.



Physics Parameters

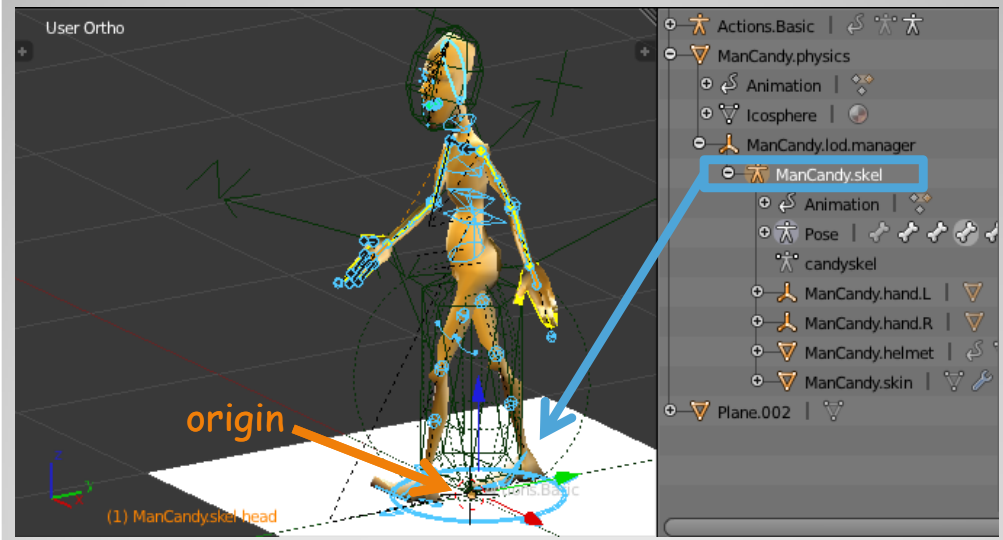
The **Armature Object** is responsible to play the Character's animation. The object itself is invisible. These animations are ALWAYS relative to the armature center. E.g. the walk cycle plays in-place.

Together with a forward motion the character will look like as it is walking forward. The animation should not move the character/armature away from armature center.

The origin of the *Armature Object* does not need to match the origin of the *Physics Object*.

The *Armature Object* needs to know how to play it's animations synchronously with the Physics motion.

In some rare cases the *Armature Object* can act as *Physics Object* as well.



Armature

The **Skin Mesh Objects** are the visible part of the Character. They do not need any logic as they are animated by the *Armature Object* and moved by the *Physics Object*.

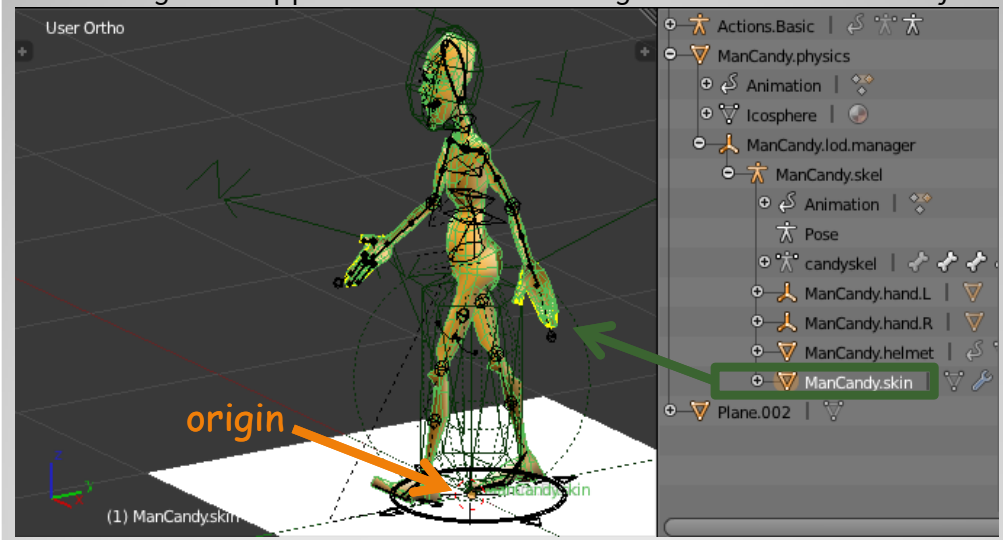
You can have as much *Skin Mesh Objects* as you like. E.g. partial skin, clothes etc.

The physical representation of *Skin Mesh Objects* does not deform with the visual representation. Any physical interaction should be disabled. Otherwise it will result in unexpected and strange behavior.

Attention: It is not possible to apply a new *Skin Mesh Object* to an armature on the fly (in-game).

Hint: The mesh of a *Skin Mesh Object* can be replaced on the fly.

The origin is supposed to match the origin of the *Armature object*.



Skin

A very frequent asked question is:

How to apply the Skin Mesh Objects to the Armature Object?

1. Make sure there is no modifier on the **Skin Mesh Object**
2. Ensure the *Skin Mesh Object* has the right alignment to the **Armature Object**. E.g. Facing the same directions.
3. Apply any transformation on the *Skin Mesh Object* (pos/rot/scale). You can do that with <ctrl-A>
4. Select the *Skin Mesh Object*
5. <Shift> select the *Armature Object*
6. Armature-Parent the *Skin Mesh Object* to the *Armature Object* <ctrl+P>. The *Skin Mesh Object* automatically gets an armature modifier added.

- Only bones matching the name of an animation channel will animate with an action (for channels see dopesheet editor).
- Only vertices assigned to vertex groups with the same name as bone name will deform with an action.
- Meshes can be replaced (ReplaceMeshActuator) on the fly. The new mesh should have according vertex groups to be animated. This allows changeable skins/clothes.

Hint: You can use envelopes too, but they do not always fit that well.

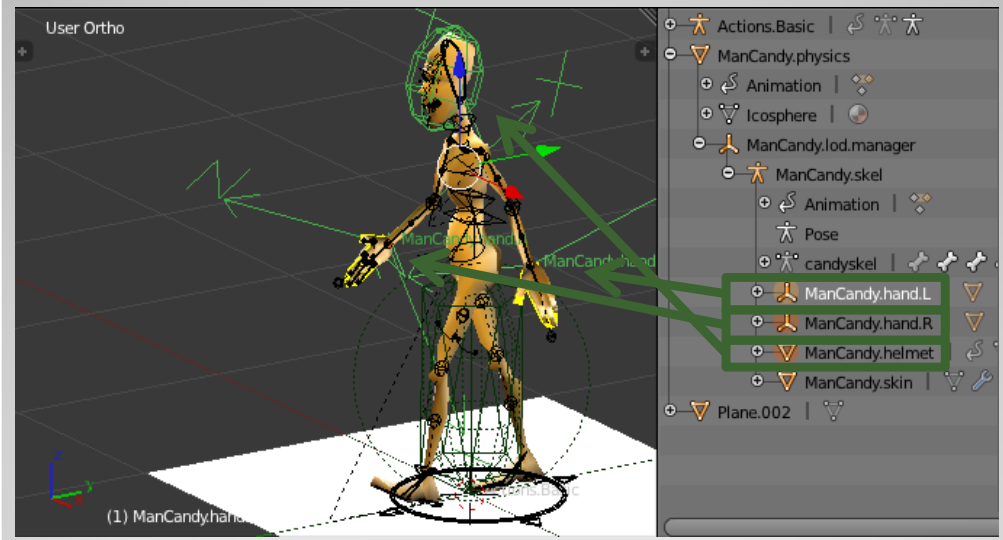
Applying Skin

The **Bone Hooks** are objects (usually empties) that are directly parented to bones. This enables to user to parent other objects to the hooks. The hooks and their children will follow the bones' animation. E.g. hold things with the right hand.

The hooks are recommended as they are easy to recognize and because it is not possible to parent objects to bones on the fly.

If objects are not supposed to be removed/added on the fly they can be parented at design time. (e.g. gadgets, helmets, caps, collision detectors).

To sense collisions with "bones" **Collision Detectors** can be parented to a bone. They can be switched to Physics type "sensor" to enable collision detection.



Bone Hooks

The **Behavior Object** performs the behavior of the character.

That means it collects the input on all involved objects, calculates the appropriate reaction and triggers the involved actuators (including other objects).

The applied logic can be very simple or very complex. In most cases it is quite complex.

Hint: There is no restriction to pure Logic bricks or pure Python. The most efficient solutions are a mix of both.

The behavior object does not need to be part of the character's object tree as it does not matter where it is. It just needs to exist.

Any object of the character can act as behavior object. The most obvious candidates are the *Physics Object* and the *Armature Object*.

Behavior

Other

There can be **other objects** (character parts) be involved. They should be placed in the character's structure where they fit most.

Examples:

- **LOD** (Level-Of-Detail) **Control Object** [as seen in the above images] – manages to switch the level of detail
- **Cloth Manger Object** – manages to switch cloths
- **Sound Manager Object** – manages to play sounds/voices
- **Special Effects manager** – Manages special effects (particle systems)
- **Collision Detector Objects** – Detect collisions with other objects
- **Save Manager** – saves/loads the character's state to/from discs
- **etc.**

You learned a method how you can setup a character in the BGE. This should provide you with an idea how to create complex characters for your game.

I hope it helps you to create a fancy game without discovering to much obstacles on your way.

The road to success is long and I hope this little information can point you into the right direction.

You can find me under www.blenderartists.org as Monster

Good luck
Monster

Thank you