

# BlenderPeople 0.8

free.the.people



## BlenderPeople 0.8

© Roland Hess 2006

BlenderPeople is a suite of Python scripts for Blender that, in conjunction with a MySQL database, allow the generation of large scale crowd dynamics, including (but not limited to!) combat scenarios.

### Contents

#### Acknowledgements

1. Requirements ... pg 1
2. System Setup Instructions ... pg 2
3. What's New In This Release ... pg 2
4. Running BlenderPeople ... pg 2
  - a. Creating Objects ... pg 3
  - b. Initializing the Simulation ... pg 6
  - c. Command and Control ... pg 6
  - d. Running the Object Simulation ... pg 8
  - e. Adding Character Animation ... pg 8
5. Working With Actor Stats and Types ... pg 10
6. Effectors ... pg 11
7. Database Structure ... pg 13
8. Building your own character animation library ... pg 16
9. Roadmap ... pg 19
10. How Can You Help? ... pg 19
11. How Can You Get Help? ... pg 19

### Acknowledgements

Thank you to my family for letting me be spaced out and stressed out while the different solutions that eventually made their way into BlenderPeople popcorned around in my brain, and to my wife who always thought that I should be coming to bed *much* earlier than working on something like allows.

Thanks also to the Blender developer community who are always there to offer mockery and ridicule, er, actually, a great deal of help, when faced with a tough matrix math or silly Python n00bery question.

Thanks to graphical.org for hosting both the special Blender binaries you'll need (this is, after all, what graphical does), and for agreeing to host the .iso's of the demo/tutorial DVD.

Finally, a big thank you to Don Kim who gave up some of his time to create the dynamic character animations that come with this release of BlenderPeople. Remember, anyone who uses BlenderPeople out-of-the-box for their productions is building on Don's great work, and you should remember him when you strike it rich!

### 1. Requirements

#### Software:

Blender 2.42a - BlenderPeople Build

MySQL 5\*

MySQLdb 1.0

Python 2.4\*

#### Available at:

OS X:

Windows:

Source Patch:

<http://dev.mysql.com/downloads/mysql/5.0.html>

<http://sourceforge.net/projects/mysql-python>

(MySQLdb installer included for Windows users)

<http://www.python.org/>

\* BlenderPeople 0.8 also works with MySQL4.1+ and Python 2.3. If you already have these installed, there is

no need to upgrade.

## 2. System Setup Instructions

*Previous testers of BlenderPeople 0.6 and earlier, please note: You will have to remove the current BlenderPeople database before using this new version. Use the MySQL browser of your choice to drop the database "dbactors".*

1. You will need to have Blender 2.42a: BlenderPeople version installed. BlenderPeople *will not work with the official Blender release*. If you are an OS X or Windows user, you can find this special version of Blender at the address listed above. I'm guessing that if you run Linux, grabbing the sources and patching them will not be a problem.
2. You should know that you need to be able to find your way around in Blender before attempting to use BlenderPeople. Go to [www.blenderartists.org](http://www.blenderartists.org) for a huge list of links to tutorials. Better yet, go to [www.blender3d.com](http://www.blender3d.com), and buy the 2.3 Blender Guide, then read it cover to cover. See you in three months!
3. Install Python 2.4. If you already have 2.3, you do not need to update to 2.4. Blenderartists.org has extensive instructions on how to set up Blender to work with a full Python installation.
4. Install MySQL. Accepting the default options is the easiest way to do this. I'll assume that if you're running Linux, you know what you're doing and don't need me to hold your hand. The first time you run BlenderPeople 0.8, you are presented with an SQL connection configuration screen, where you can type in your MySQL server's IP address (127.0.0.1, if it's on the same computer), a valid user name and a password. The username and password will have been set up during your installation of MySQL, so be sure to write them down when you're doing that. Unless you're a Golden God when it comes to MySQL configuration, you'll probably be better off using the default "root" username that the installer makes for you.

*MySQL Security Warnings:* Please note that if you are on a broadband connection, or an insecure LAN and there is no NAT (commonly called a DSL or Cable Modem Router) or firewall in place, you should read the MySQL documentation on securing your system first, as MySQL opens up a port and runs a desirable new service on your computer. Also, please be aware that BlenderPeople stores your connection information (IP address, username and password) in a plain text file that is saved within the .blend file. This file is called "MySQLInfo" and can be found through Blender's text window. Don't redistribute this .blend file without removing this information, or you'll be giving the world your database connection information! I am not responsible for you getting haXORd.

5. Install MySQLdb. This is the link between Python and MySQL. The MySQLdb installer for Windows seems to work pretty flawlessly. Mac users, prepare yourselves for some grief. It took me a whole day of fighting with my system to get it installed and recognized. Unfortunately, I just don't have the time to support the nightmare that is Mac/Blender/Python/MySQLdb installation.

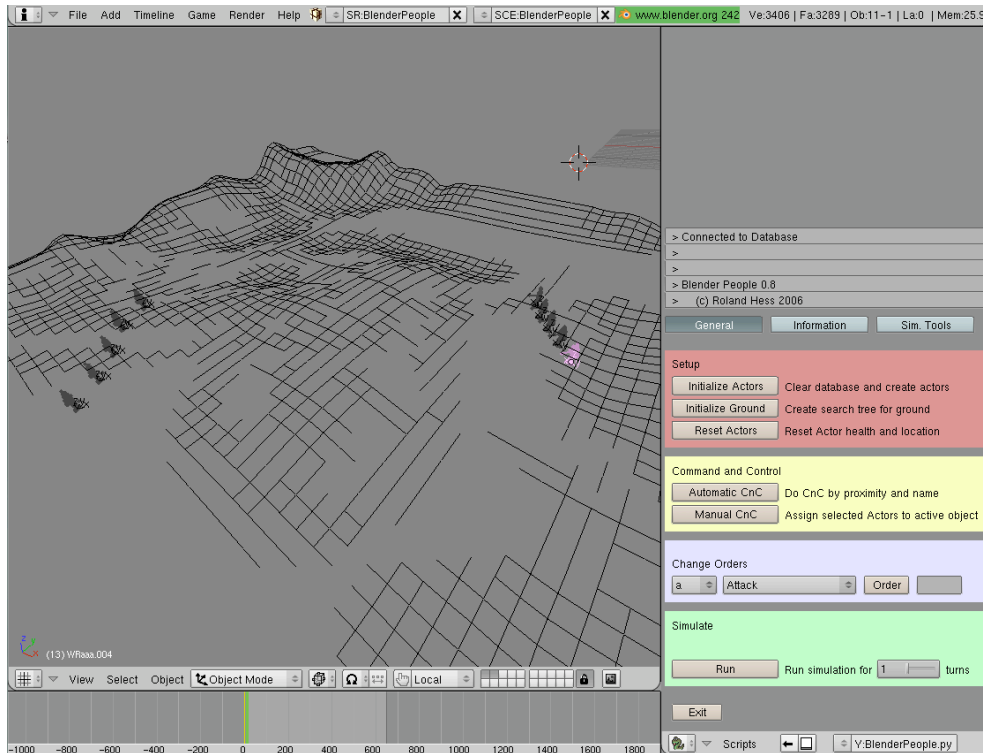
## 3. What's New In This Release

1. Support for character animation.
2. The much-awaited MatchBone NLA Feature.
3. Full in-Blender Action Baking and a new, super cool, NLA blending mode.

## 4. Running BlenderPeople

Run Blender 2.42a:BlenderPeople, hit F1, and open BlenderPeople.0.8.blend.

Put your mouse over the script window and press alt-P. If everything has been installed correctly, you will see the simple BlenderPeople GUI. Two screen views have been created for you: one for running the simulation (BlenderPeople), and one for rendering your animations (Rendering). The GUI has three buttons at the top: General, Information, and Sim Tools. Each of these accesses a different set of controls. The controls for the General button are addressed in this section. The controls for the Information and Sim Tools tabs are addressed in the following sections, "Working with Actor Stats and Types", and "Effectors".



Here is a step-by-step to get you running. Each section contains two parts: one called Do It, which is a quick-start, and one called In Depth, which is, well, duh.

## Creating Blender Objects (People, Ground and Obstacles)

### Do It

You can use the objects included in the sample .blend file, which appear in the 3D window. You have two armies, a red team and a blue team, both set to attack each other. A ground mesh is also included.

### In Depth

*Ground* - here are the rules for creating a ground object:

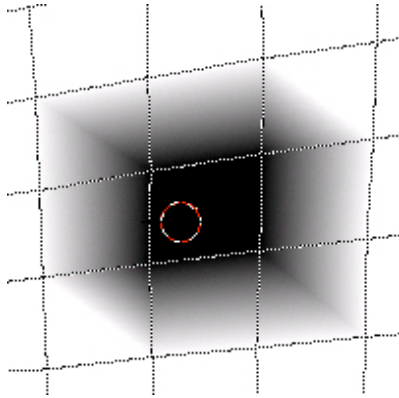
- Mesh object, made of quad faces
- Named "Ground"
- No holes - a pit should be a deep depression, not a missing face.
- The search methods work best above 1000 faces. More faces means slower searches, but I've gone up to 16,000 with tolerable speed results. If you drop below 1000 faces, however, the searches will fail, giving you crazy flying actors.
- Must be unrotated and unscaled. When you are happy with your ground mesh, be sure to hit Ctrl-A to apply any scaling or rotation.

Tips for working with the ground object:

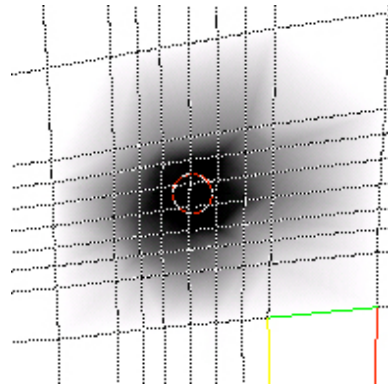
- If Actors will only be moving over a portion of your overall terrain, make a duplicate mesh including only the relevant part. BlenderPeople can generate motion much more quickly on a smaller mesh. In the included file, the object named Ground is on layer 2.
- Meshes that approximate an unrotated rectangle when viewed from above will give the most efficient search results. Other shapes will work fine, but this is the optimized case.

**Vertex painting the Ground object:** BlenderPeople uses the Red channel of the Ground object's vertex paint values to either attract or repel Actors. (Note: although BlenderPeople only reads the Red channel of the vertex paint, I will use the terms "black" and "white" here for the sake of simplicity. Black represents a color value of 0; white, a color value of 255.) A face that is completely black (i.e. all four vertices have a 0 value in the red channel) will be impassable. In this way, you can indicate the location of barriers on the ground. By painting the vertices around barrier objects black, you can make your Actors aware of the barriers. For best results, though, you should gradate your painting away from the barrier, allowing your

Actors to sense that "something" is coming up and they may want to route around it.

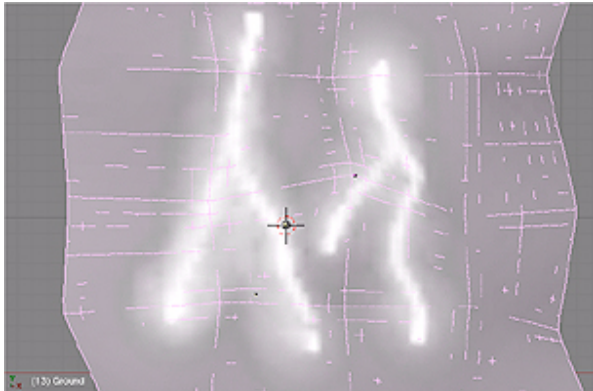


*Too simple - will cause blocky movement*



*Actors will move smoothly around this*

You can also use this to create a preferred path across a field. If your entire battlefield is painted grey (red value around 128), you can paint white paths (red 255) across the field, then graduate the brightness to attract Actors toward it. As long as the paths move in the general direction the Actors are attempting to move, they will tend to congregate toward the brighter paths.



*Actors will gravitate toward the brighter painted pathways*

It is important to remember that Actors set to Terrain in the Actor Stats window do not use full pathfinding algorithms. Though they will try their best to avoid it, they can get stuck if they encounter concave black areas. Don't expect them to be able to navigate a maze of black faces. Actors set to A\* (A Star) pathfinding should be able to successfully navigate the entire terrain. This functionality comes at the cost of time, though. To see the difference, run a step of your simulation, then select an Actor and change it to use A\*, then run the step over. In lieu of using A\* for everyone, I have found that specific March and Target orders work well for directing Actors through more complex terrains.

When you are happy with your vertex painting (and every time you change it) you need to press the **Initialize Ground** button for BlenderPeople to register the values.

Tips for vertex painting your Ground object:

- If your mesh is not detailed enough around barriers (black areas), your Actors will tend to form a box around them, instead of smoothly routing around them. The solution is to increase the mesh detail. If you don't want to subdivide your entire mesh and quadruple your Ground tree search times, then use Blender's loop cut tool (Ctrl-R) to add detail around the area in question while maintaining your quad faces.
- Actors check to see if they will be passing within a "restricted" black area based on their speed. If your Actor's maximum speed is defined as 10 and your black area is less than 10 Blender units in size, it is possible that the Actor may not see it and pass right through it. For this reason, you need to find a balance between Actor speed and the size of your barrier/black areas. A simpler way to put it is: if your Actors are fast enough to move across your black painted area in a single turn, they probably will.

*Actors*

Actors are the BlenderPeople themselves. In the 3D view, Actors are represented by Empties, which are linked with Blender's new DupliGroups feature to a grouped object consisting of a mesh and an armature. If you just want to add Actors to the default scene, you just need to select existing Actors and duplicate them with alt-D. If you want to create your own characters and meshes for use with BlenderPeople (and you're going to want to!), see section 8: "Building Your Own Character Animation Library".

#### *Naming Convention*

It is by their names that BlenderPeople recognizes Actors.

All Actors names must begin with "WR". In fact, any object in the scene whose names begins with "WR" will be treated as an Actor.

The third character of their name designates their team. Actors on the same team will not attack each other, and perform use them when making "ally" decisions. Actors not on the same team will regard each other as opponents. In theory, you can have 62 different teams, (a-z,A-Z,0-9), though I'm not sure why you'd want to. Currently, this demo release of BlenderPeople is set up to work with two teams. You can add more teams simply by creating Actors named to be in a different team. If you wish to do so, you can add lines for Orders for these new teams in the Initialization.py script.

The fourth character in the name shows the Actor type. Types are defined in the MySQL database, and include the following characteristics:

*Speed*: distance an Actor can proceed during one turn

*AttackRadius*: distance within which Actor may do damage to an opponent

*ChargeRadius*: distance within which Actor will charge at an opponent

*CowardRadius*: Actor will not stray further than this from its commander

*BuddyRadius*: Actor will prefer to be within this distance from an allied Actor

*ActorRadius*: Physical size of the actor - used for Barrier collisions

*Health*: 0 and below is dead. Anything it above is alive.

*Attack*: Actor's relative attack strength - used to determine probability of a hit

*Defense*: Actor's defensive strength - used to determine probability of not being hit

*Intellect*: range: 0-1. Used to determine how well Actor follows orders. Values below .85 lead to some really annoying, stupid behavior.

*FieldofView*: the angle, in degrees, of the Actors peripheral vision, measured from straight ahead. An Actor that can see a full 180 degrees in front of itself has a FieldofVision of 90 (90 left + 90 right = 180).

*MaxTurn*: the angle, in degrees that an Actor can spin around in a single turn. DO NOT make MaxTurn greater than Field of View, or you will end up with Actors that spin like tops when attempting to acquire a target.

*Pathmode*: what pathfinding system the Actor will use. *Terrain* is the default. *None* ignores the vertex painting on the Ground. *Astar* uses the full A\* pathfinding algorithm.

Each of these values (except Pathmode) also has an accompanying variation value, expressed as a decimal, that varies Actor statistics when they are initialized (if you want a 12.5% variance in stats among a given type, you set the variance value to 0.125)

There is only one Actor type included in this release. Actor type are useful if you want to have Actors use different base meshes, different sets and parameters for character animation, and different sets of base statistics.

The fifth character in the name shows the Actor's Command and Control (C&C) level. If you're just running an all-out, no strategy attack/defend simulation, there is no need to use the C&C. If you want different sections of your forces to do different things, though, you must.

Actors decide what to do based on their Orders. Actors can be set to take their orders from any Actor on their team. This allows you to build hierarchical chains of command. The simplest way to set this up is to create your armies as "a" (fifth character) level Actors. You then select your first level of commanders (sergents, etc.) and change their fifth character to "b". If you want another level of commanders (lieutenants, etc.), change theirs to "c". And so on.

Here is the breakdown of a typical name:

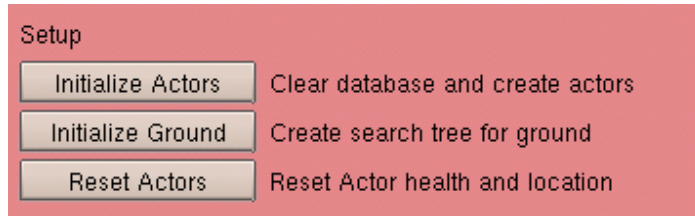
"WRabd.182"

This name indicates an Actor ("WR") on team "a", of type "b", and of command level "d".

## Initializing the Simulation

### **Do It**

Click the Initialize Ground button. When it is finished, a message scrolls onto the info box.  
Click the Initialize Actors button. When initialization is finished, a message appears in the info box.



### **In Depth**

When you start the GUI, BlenderPeople checks for the presence of the appropriate database elements. If they do not exist, it attempts to create them for you.

#### *Initialize Ground*

Clicking Initialize Ground erases the Ground search tree in the database. It then rebuilds the location and vertex paint search tree based on the current Ground object. Depending on the number of faces, this could take some time. As long as you are using the same Ground mesh with the same vertex painting you used the last time you pressed this button, you do not need to do it again. One of the advantages of storing the tree in a database is that it persists from session to session, and can even be easily accessed across a network.

#### *Initialize Actors*

Clicking the Initialize button does the following:

- Deletes all Actor and Action information from the database, including Command & Control links.
- Unlinks all IPO info from Actors
- Populates the database with Actors, as found in the .blend file, and according to their designated types.
- Creates new IPOs for each Actor, with a location and rotation key on the current frame.
- Sets default orders for teams "a" and "b" to Defend.
- Moves actors vertically so that they rest on the surface of the Ground.
- Advances to the first live frame for animation.

#### Hints and Cautions for Initialize

- If you press Initialize Actors and there is no Ground object registered in the database, it will do that first, and may take some time.
- Any animation you have recorded for Actors will be unlinked (essentially erased)
- Command & Control structures will be wiped out.
- Summary information appears in the BlenderPeople info box after the procedure is complete.

#### *Reset Actors*

Clicking the Reset Actors button does the following:

- Unlinks IPOs from Actors.
- Clears the Actions log.
- Creates new IPOs for each Actor
- Reinitializes Actor health and location information in the database.

When you want to rerun a simulation, but don't want to destroy your command and control structure, this is the button to use. If you have added or removed Actors, though, you must do a full Initialize.

## Command and Control

### **Do It**

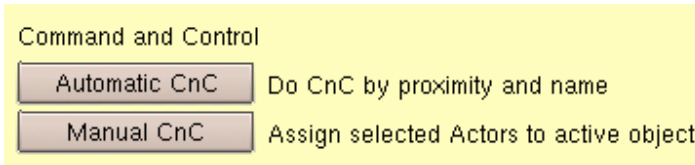
Ignore the Automatic CnC and Manual CnC buttons. Use the dropdown menus under Change Orders to set the Orders for each team. Press the Order button when you want to issue those Orders. See *In Depth* for instructions on using the different order types.



### ***In Depth***

Press the Automatic C&C button, and each Actor will be assigned the nearest Actor on its team that is one level above it in the chain of command.

You can also set up special groups by using the Manual C&C button. This button assigns all selected Actors (dark purple) to the active Actor (light purple) and overwrites any older C&C definitions for these Actors.



The dropdowns and Order button change the fallback orders that all Actors look for when they have no commander.

It is now fairly simple to issue orders to individuals or their commanders. See the "Orders and C&C" topic of the "Working With Actor Stats and Types" section for instructions.

### ***Orders***

These are the orders that can be issued and what they cause Actors to do:

*Defend*: Actors move only to position themselves in relation to their allies. Actors will approach any opponents that come within their ChargeRadius value. Actors will orient themselves to face any opponent that comes within their hearing range.

*StrictDefend*: Actors move only to position themselves in relation to their allies. Actors will attack opponents within their AttackRadius, but will ignore opponents that are merely within the ChargeRadius.

*March180*: Actors proceed along the heading, in degrees. From a top view, 0 is right, 90 is up, 180 is left and 270 is down. Actors will charge and attack opponents that come within the ChargeRadius. Enter your heading in the text box to the right of the March order menu item.

*StrictMarch180*: Actors proceed along the heading, in degrees. Actors will only attack opponents that come within their AttackRadius. Enter your heading in the text box to the right of the March order menu item.

*Mill*: Actors mill around in a random fashion within the confines of the named target object. The best object to use is an Empty. You scale the empty to encompass the area in which you want your Actors to mill about. Actors will attack enemies that come within their ChargeRadius. Enter the name of the target object in the text box to the right of the Mill order menu item.

*StrictMill*: Actors mill around in a random fashion within the confines of the named target object. The best object to use is an Empty. You scale the empty to encompass the area in which you want your Actors to mill about. Actors will attack enemies that come within their AttackRadius. Enter the name of the target object in the text box to the right of the Mill order menu item.

*Target*: This order causes Actors to proceed toward the named target object, attacking opponents that come within their ChargeRadius. Enter the name of the target object in the text box to the right of the Target order menu item.

*Rank90d12*: Actors proceed along the heading, in degrees, and attempt to maintain the distance, after the "d", you have specified from each other. The format is a single string, consisting of "Rank" plus the heading in degrees, plus the letter "d" (for distance), plus the distance for the Actors to maintain

from one another in Blender units. Actors will charge and attack opponents that come within their ChargeRadius. Enter the heading, followed by a "d" followed by the separation distance in the text box to the right of the Rank order menu item.

*RegroupMain*: Actors attempt to come within their CowardRadius to the center of allied forces. They will attack only those opponents who fall within their AttackRadius.

*RegroupCommander*: Actors attempt to come within their CowardRadius to their designated Commander. They will attack only those opponents who fall within their AttackRadius.

*Retreat*: Actors attempt to move away from their nearest opponents and from the center of the opposing forces. They do not attack opponents, regardless of their proximity.

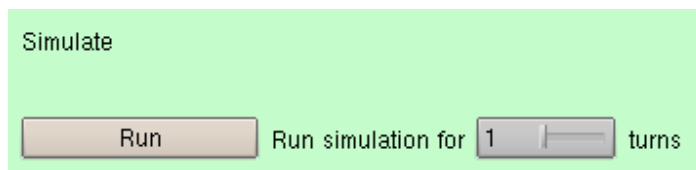
*Attack*: Actors seek out the nearest opponent within their FieldofVision. If none exists, they listen for an opponent, then turn to face it. If no opponent is within their FieldofVision or within hearing range, the Actor begins looking around to identify an opponent.

*Directed*: Actors will only attack opponents who are within the bounds of the specified object. The best object to use for this purpose is an empty, as the attack radius around the object is determined by its uniform scale. You can use this to set up long-range attackers such as archers, etc. You alter (or create) an actor Type to have an AttackRadius that covers the distance you want the projectile to be able to shoot. Actors of this type can then be directed to attack anything that comes within the bounds of the designated object. They will remain stationary, unless directly attacked themselves, or ordered to do otherwise. Enter the name of the target object in the text box to the right of the March order menu.

## Running the Simulation

### **Do It**

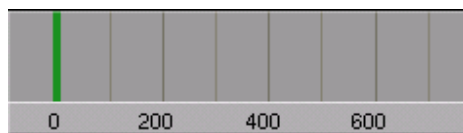
Set the slider beside the Run button to 1, and press Run. This will run the simulation for 1 turn, generating 12 frames of animation. Adjust the slider to run it more times without intervention.



### **In Depth**

If you've run several hundred frames and want to preview your animation, go ahead. The simulation stores the motion in IPOs, so you can move freely backward and forward without having to do any recalculation. If you want to continue simulating, check the Console for the last generated frame number, then set Blender to that frame number + 12, and resume with the Run button.

To scan through your animation, use the provided frame position window:



## Adding Character Animation

Once the object-level motion has been generated, and you are happy with it, I recommend that you save your .blend file.

*IMPORTANT! Before you begin with the character animation tools...*

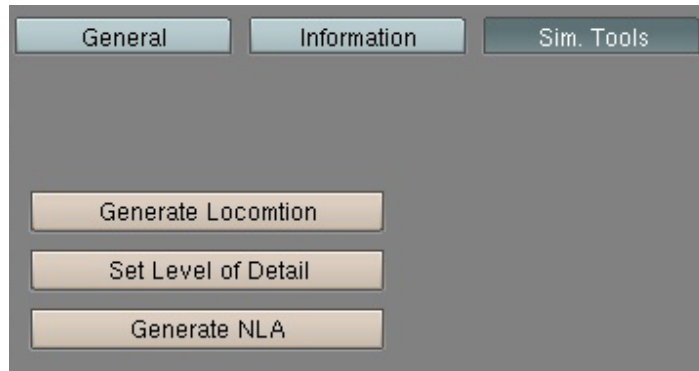
At this point, you need to disable Blender's global undo system by pulling down the user preferences window, clicking the Edit Methods button, and unclicking the Global Undo button. The baking procedure

used in character animation is seen by the Undo system, and when creating walking animation for dozens (or hundreds or thousands) of Actors, the Undo system can balloon Blender's memory use quite quickly past 2GB of RAM, most likely crashing your system!

### **Do It**

Click the "Sim Tools" button. Select your Actors in the 3D view. Hit the "Generate Locomotion" button to create the walking portion of the animation.

Select your Actors in the 3D view. Hit the "Generate NLA" button to generate the auxiliary animation. That's it. Really. Press alt-A and watch the Actors run around and fight.



### **In Depth**

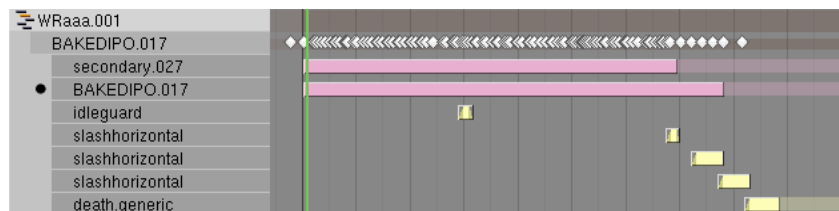
There's not a lot to say at this point. Once there is a library of character animation that is properly registered with the database (which is done automatically when you first run BlenderPeople), adding character animation really is as simple as hitting those two buttons. You can read about the process of creating your own library of armatures, meshes and actions in section 8: Building Your Own Character Animation Library

These buttons will do the procedures on only the selected objects. It may be a good idea to run the Locomotion procedure on a few actors at a time, just to make sure you're getting the results you expect.

*The Generate Locomotion button is destructive!* Once you use it on an Actor, you cannot un-use it. The object-level lpo motion is destroyed and replaced with equivalent character level motion. Future versions of BlenderPeople will provide hooks so that this process is reversible, but for now, make sure you're happy with the object level motion before you start generating walking animation.

**Generate Locomotion** is also the single biggest time gobbler in BlenderPeople. It's a complex procedure that uses a lot of system resources. Do a test on ten Actors or so, to gauge how long it will take your system before you go generating walking animation for 1,000 Actors.

Unlike Generate Locomotion, **Generate NLA** is a reversible procedure, and the results are quite editable if you want to tweak them. The NLA entries for a finished Actor will look like this:



The strips shown in yellow here are the ones generated by the Generate NLA procedure. Strip names that begin with "secondary" and "BAKEDIPO" are generated by the auto-walking. Don't mess with those two. If you don't like the way the NLA worked out (there is some randomness involved), you can delete the strips, select the Actor in the 3D view, and hit the Generate NLA button again. When building NLA, BlenderPeople chooses from all of the available actions for a certain activity (like Attacking), based on how well they fit in the allotted time and how heavily those actions are weighted in the database, meaning that you may get a different result each time you run the procedure. Alternately, you can simply relink the strips you don't like to different actions, or change its size or location along the timeline.

The **Set Level of Detail** button also works only on the objects selected in the 3D view. The default setup of

BlenderPeople has the Actor empties linked to a group of objects with a very low resolution mesh object, suitable only for quick tests and populating backgrounds. The master objects for this group are found on layer 5. On layers 6-10, however, are progressively higher resolution groups for objects that will appear closer to the camera. While there is not at this time a system for changing the Actors' level of detail on-the-fly, it can still be controlled by the user with relative ease.

Advance the timeline to a frame of your choosing, select the some Actors and the camera that the scene will be rendered from. Hit the Set Level of Detail button. BlenderPeople will swap in the correct dupliGroup object for each of the selected Actors, based on their distance from the camera and ranges that have been pre-entered into the database for you. Judicious use of this feature can drastically improve your rendering times, while indiscriminate use could send your computer spouting flames and smoke down an elevator shaft!

## 5. Working with Actor Stats and Types

The Information button at the top of the GUI allows access to a set of controls for view and changing information about your Actors and Actor Types.

### Actor Information

Select an Actor and press Get Stats.

ID#	Name	Type	Team	Self
< 8 >	WRbaa.005	a	b	Self
xSpeed	Orders	Params	Commander ID	
< 0.000 >	Target	Empty.001	< 0 >	
Speed	Attack	Defense	Health	Intellect
< 7.05 >	< 2.03 >	< 2.81 >	< 4.37 >	< 0.950 >
R-Attack	R-Charge	R-Coward	R-Buddy	R-Actor
< 7.30 >	< 24.74 >	< 29.36 >	< 7.10 >	< 1.800 >
View	MaxTurn	Loyalty	Pathfinding	
< 2.737 >	< 1.682 >	< 0.000 >	terrain	

Get Stats    Select Cmndr    Select Subs    Set Stats

To view the information for an individual Actor, select that Actor in a 3D window, then click the "Get Stats". BlenderPeople queries the database for the Actor's statistics, which are then displayed in the GUI. These statistics correspond to the categories described in the *Creating Blender Objects* section of this document. A few notes are in order, though:

- ID*: Although you can alter it here, changes to this ID number will not be written back into the database. Each Actor has a unique, unchangeable ID. It is how the database keeps track of them.
- xSpeed*: This is the speed multiplier for the Actor's orders. For example, a speed multiplier of 0.5 with a Retreat order would have the Actor retreating at half it's maximum speed. Likewise, an Actor with an *xSpeed* of 2 and an order to Attack would execute Attack behavior at twice its normal speed.
- CommanderID*: This is not the name of the Actor's commander. This value corresponds to the *ID* number of the Actor's commander.
- R-Attack, R-Charge, R-Coward, R-Buddy, R-Actor*: All define radius values in Blender units.
- View, MaxTurn*: Values are in radians.
- Pathfinding*: Pathfinding algorithm that this Actor uses.

If you wish to alter the settings, you may do so. When you are satisfied, press the Set Stats button to write the information back to the database. Note that you can do this in the middle of a simulation with no ill effects. You can reset the health of a dead Actor, change an individual Actor's orders (and if that Actor is a commander, all Actors below it in the chain receive those orders, too!), reduce an Actor's speed, intellect, attack/defense values - whatever is appropriate to the situation.

### Orders and C&C

You can use the Actor information panel to issue orders and fine-tune your Command and Control structure. An example will best illustrate its uses. Out of your entire army of 1,000 Actors, you have a group of 100, all set to the same commander, that are not doing a good job of locating the enemy. You want to change their orders, but who in that whole bunch is the commander? Select one of the Actors, then click the "Select Cmndr" button. This button selects the current Actor's commander in the 3D window. Once their commander is selected, click the Get Stats button. The commander's stats, including orders, come up in the stats window. First, click the "Self" button. This breaks the Actor off from receiving orders from up the chain of command. He will now act on whatever orders he is issued. (Clicking the "Self" button immediately updates the database, so if you only grabbed this Actor's stats to set it to be its own commander, you do NOT need to press "Set Stats".) Change the Actor's orders from "Attack" to "Target" and enter the name of one of the enemy Actor's objects into the parameter widow beside the orders menu. Now, hit "Set Stats". What you just did was break off a commander from the C&C heirarchy and issue it orders to proceed directly toward an enemy object.

If you want to see all Actors that are under a particular commander, select that commander in the 3D window, then hit "Select Subs". All subordinate Actors are selected, and the commander remains the active (light pink) object.

## Types

Speed	< 0.00 >	< 0.000 >	Field of View	< 0.00 >
Attack Rad.	< 0.00 >	< 0.000 >		< 0.000 >
Charge Rad.	< 0.00 >	< 0.000 >	MaxTurn	< 0.00 >
Coward Rad.	< 0.00 >	< 0.000 >		< 0.000 >
Buddy Rad.	< 0.00 >	< 0.000 >	Name	
Health	< 0.00 >	< 0.000 >	Name	
Attack	< 0.00 >	< 0.000 >	Type Identifier	
Defense	< 0.00 >	< 0.000 >		a
Intellect	< 0.000 >	< 0.000 >		

a, FastWeak    Get Type    Set Type    New

You can access the current list of Actor Types via the pop-up menu at the bottom of the GUI. In the image above, it reads "a, FastWeak". The entries in the menu show the type identifier (used when naming your Actors) followed by the Type's name. You can view and edit a Type's values by choosing it from the menu and clicking the "Get Type" button.

In the GUI, the values in the on left represent the targeted value for the statistic. The ones on the right show the variability of the value when Actors are initialized. To give an Actor Type a speed value of 9, with a variability of 10%, you would enter a 9 in the left column and a 0.1 in the right column. When Actors of this Type were initialized, their Speed values would be 9 +/- 10%, or in the range of 8.1 to 9.9. The following are some notes on a couple of the statistics:

*Intellect:* Be careful when choosing Intellect values. Intellects below 0.8 produce silly behavior in Actors, as they will tend to do very foolish, seemingly random things during the simulation. Try it and see.

*FieldofView:* This is value is in degrees. If FieldofView is less than MaxTurn, Actors can spin continuously when attempting to acquire a target.

*MaxTurn:* The maximum amount in degrees that an Actor can rotate in one turn. If MaxTurn is greater than FieldofView, Actors can spin continuously when attempting to acquire a target.

When you have things set the way you want, press the "Set Type" button to write your chages to the database. Types will persist in the database across multiple simulations, regardless of Initialization.

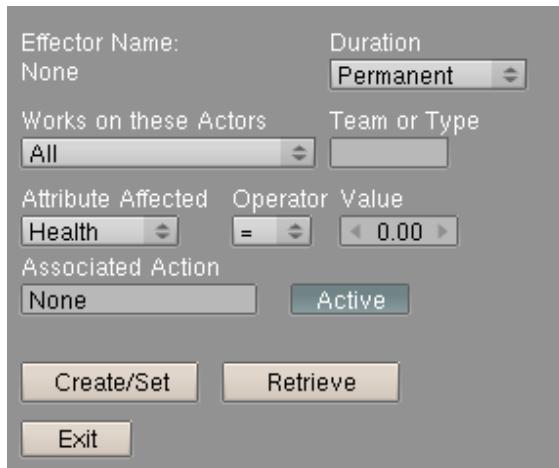
Pressing the "New" button creates a Type with the next available Type Identifier and loads it's information (mostly zeros) into the GUI. Define the new Type as you see fit, give it a name, then click "Set Type".

## 6. Effectors

**Effectors** are Blender objects that can affect the statistics (and therefore the behavior) of your Actors. To register an object from your 3D scene as an Effector, you make it the active selection (light pink) in a 3D window, set your options in the Effector window, then press the "Create/Set Effector" button.

In addition to the parameters seen and explained below, Effectors have an area of influence. This area is the average of the X,Y and Z scales of the object, as seen in the object's properties pallet in the 3D window. For this reason, it is highly recommended that you use Empties for your Effectors. If you simply scale the empty in the 3D window to cover the area you want to have influenced, it will work correctly. If you want something like a rock to be an Effector, try creating an empty, parenting it to the rock, then setting the Empty as the Effector. You could use the rock object itself, but then you have to play around with getting the scale to match the visual size of the object, which can be tricky and screw up your texturing.

### The Effector Panel in Sim. Tools



*Effector Name:* Displays the name of the Effector object. This is set for you, based on the name of the Blender object.

*Duration:* Permanent or Temporary. A Permanent Effector does its work directly on the Actor's information in the database, so any changes it makes are unreversible. Using an Effector to kill Actors (setting their Health to 0) would be a good use for this. A Temporary Effector modifies Actor information on the fly, and does not touch the actual database information. This means that after an Actor passes outside of the Effector's area of influence, its effects disappear. An example would be a temporary Effector on a "muddy" area, that cuts Actor speed in half - once they are away from it, their speed returns to normal levels.

*Works on these Actors:* Effectors can work on All (every Actor), a specific Team, or a specific Type of Actor. If you choose Team or Type, enter the appropriate designation in the *Team or Type* box to the right. When choosing All, the *Team or Type* box is ignored.

*Attribute Affected:* This menu shows what Actor statistic the Effector works on.

*Operator:* What to do to the Attribute chosen in *Attribute Affected*.

*Value:* The numeric value to use with Operator and Attribute.

Basically, you're building an equation to work on Actor attributes. If you choose "Speed" from *Attribute Affected*, "+" from *Operator*, and set *Value* to 4, you'll be creating an Effector that adds 4 to the speed attribute of any Actor coming within it's influence.

*Associated Action:* If you want there to be a special Action associated with this Effector, enter its name here. A large boulder set as a kill effector (Health=0) might include a "Death" or "Flying Death" action, so that future versions of BlenderPeople will know to animate a character dying or flying through the air upon encountering this Effector.

*Active:* Whether or not the Effector is evaluated. It's just an on/off switch.

*Create/Set:* If the Object you have selected is already a registered Effector, it will modify its database entry to match what you see in this panel. If it is a new Effector, it will create a new entry in the database.

*Retrieve:* Selecting an Effector in the 3D window and pressing *Retrieve* will display that Effector's information in the window.

## 7. Database Structure - What to touch and what not to touch

### Database and Browsers

BlenderPeople uses an external database to store all Actor and ground information, to track actions across the length of the simulation, and to create and maintain the system of links between different types of data (actions, behaviors, groups). While there are a few tools included in the BlenderPeople GUI for dealing with individual Actor stats and Actor types, I have held off on creating tools for accessing the rest of the database. There is a source project in the works that will make coding graphic interfaces for Python scripts significantly nicer, both for me and for you, and I have chosen to wait until it's available before doing any further work in this area.

If you plan to build your own character animation library, or to do anything more with the database than what the provided tools can handle, you'll need a map of the database and a database browser.

For directly browsing and editing the database, there are a number of tools available:

*MySQL Administrator and Query Browser:* Free tools offered by MySQL on their website. If you just want to edit the database directly, all you need is Query Browser. The interface is a bit weird, but it's free and it works.

*Microsoft Access and ODBC:* Many people will poo-poo this, but for browsing a simple database like BlenderPeople's, MS Access with an ODBC link will be just fine. I'm not going to give you a step-by-step on how to set up ODBC with MySQL, or how to link Access tables afterward, but if you've ever used the program at all, it's pretty simple. Not free.

*phpMyAdmin:* More savvy users who already have Apache and PHP installed on their workstations can use phpMyAdmin, a PHP suite for managing MySQL databases. I've used it, and it's not that bad. Free, and available at [phpmyadmin.net](http://phpmyadmin.net).

### Database Structure

Some of this information is duplicated in other sections of this documentation, and is included here to make this section a complete reference.

A database in MySQL is called a "schema". The name of BlenderPeople's schema is "dbactors" ("db" signifies at a glance that this is a database). Schema are divided into tables of data. The tables you will find in dbactors are (in alphabetical order, the "tb" prefix shows at a glance that these are tables):

**tblactionlinks:** contains information linking actor behaviors, actor types and Blender actions

**tblactions:** the running list of behaviors for each Actor in a simulation

**tblactors:** the master list of all registered actors and their statistics for this simulation

**tblastar:** structure used for calculating A\* pathfinding. Done in memory, now deprecated.

**tblcharacters:** contains the link structure between actor types, groups and Level of Detail

**tbleffectors:** the master list of registered effectors

**tblgroundtree:** contains the current ground search and height information

**tbl locomotion:** the links between Actor types and walk and run cycles

**tblorders:** default orders on a team-by-team basis

**tblruncycles:** information about runcycle Actions for locomotion

**tbltypes:** the different types of Actors available to be registered

**tblwalkcycles:** information about walkcycle Actions for locomotion

Here is each table explained. A designation of DNM means "Do Not Mess with this value -- you could screw up your simulation and make your mother hate you."

## **tblactionlinks**

Actions are linked to Actor types in this table. Each action that you want to associate with an Actor type needs the following information:

- TypeID*: the Type of Actor that will use this action ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.
- Orders*: ignored for now. Future versions of BlenderPeople will allow you to differentiate behavioral actions based on varying orders. For example, an Actor exhibiting an Attack behavior while under Retreat orders would use a different set of Actions than an Actor exhibiting Attack behavior while under Defend orders.
- Action*: the behavior with which to use this Action. Enter something from the previous list (Defend, Attack, Search, March, Charge, Retreat, Mill, Regroup)
- Likelihood*: A weighting value from 0 to 1 for how likely this Action will be to be selected from all available Actions. Since a behavior can have several Actions associated with it, this value will help to determine how often this particular Action is chosen.
- OptimumFrames*: the length in frames of the Action. As this is how you designed the Action, BlenderPeople will try to use it at this size in the NLA.
- MinFrames/MaxFrames*: the minimum and maximum size in frames that you are willing to let this action be compressed or expanded. Try scaling your Action as an NLA strip and playing back the results. You are trying to find the maximum range over which the Action still looks good.
- ActionName*: the actual name of the animation Action in BlenderPeople

## **tblactions**

This information in this table is generated as you run the initial simulation. It contains a list of all behaviors and orders that Actors do, with an associated frame number.

- ActorID*: the ID number of the Actor, linked to the ID field in tblactors.
- Frame*: the frame on which this behavior was begun
- Orders*: ignored for now.
- Action*: the behavior that the Actor began on this frame -- see section 8 for allowable values

## **tblactors**

The master list of Actors and their current statistics.

- ID*: a unique identifier within the database. All tables that link to actor information do so using this ID value
- Type*: the Actor's type, this determines which sets of character animations and dupligroups are used, as well as the ranges for initial statistic generation
- Team*: which team the Actor is on, a single character. You can have up to 26 teams.
- ObjectName*: the name of the Actor's object in the Blender interface
- OrderSpeed*: This is the speed multiplier for the Actor's orders. For example, an order speed of 0.5 with a Retreat order would have the Actor retreating at half its maximum speed.
- Orders*: the orders that this Actor will follow. See the Command and Control heading of section 4 for the list of allowable orders
- CommanderID*: the database ID number of this Actor's commander
- Speed*: the distance in Blender units the Actor can travel over a level surface in one turn (12 frames)
- Attack*: Attack strength of the Actor. Determines both hit probability and damage
- Defense*: Helps to determine hit probability when being attacked
- Intellect*: Percentage of the time the Actor will correctly follow orders. Values below .9 can lead to some really stupid animation
- Health*: Amount of damage the Actor can receive before being killed
- FieldofView*: amount of arc around which the Actor can see, in radians. This is the amount away from straight ahead to one side that the Actor can see, so the final field of view is actually twice this
- MaxTurn*: the maximum amount in radians that an Actor can turn in one... er, turn. If this value is greater than FieldofView, Actors could potentially begin spinning and never stop.
- Weapon*: Not currently in use. Future versions will allow this field to enact mesh swapping and alter combat effectiveness
- AttackRadius*: maximum distance in Blender units that the Actor can be from another Actor and still do attack damage

*ChargeRadius*: if an Actor is closer than this value to an opponent, the Actor will charge at that opponent  
*CowardRadius*: Actors who are further than this value from their commander will cheat toward it  
*BuddyRadius*: Actors who are further than this value from their nearest ally will attempt to cheat toward it  
*Loyalty*: Not currently in use. Future versions will use this dynamic value to trigger automatic retreats or other actions  
*x,y*: DNM; the location of the Actors in the current simulation  
*ActorRadius*: the physical radius of the Actor, in Blender units. This value is used to prevent Actors from walking through each other.  
*Heading*: DNM; the Actor's orientation in radians  
*Turn*: turns run from -6 to 6, and determine the order in which Actors are evaluated  
*CurrentState,OldState*: Not in use.  
*Pathmode*: Tells the Actor whether to ignore terrain, or to use simple or a\* pathfinding  
*PassOrder*: Not currently in use. Future versions will use this for a better orders system

### **tblastar**

Not in use. This functionality has been duplicated much more efficiently directly within Python.

### **tblcharacters**

This table holds the links between the different Actor types and the dupliGroups they use within Blender.

*TypeID*: the Type of Actor that will use this group ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.  
*GroupName*: the name of the Group you just created  
*MaxDist*: the maximum distance from the camera that this particular group should be used. If you're not bothering with Level of Detail, set this to something very high, like 10000.  
*MinDist*: the minimum distance from the camera that this particular group should be used. If it is the highest resolution group, or you are not using Level of Detail, set it to 0.  
*ArmatureObject*: the name of the armature object within the group. Blender needs this information to properly target NLA strips within groups.

### **tableffectors**

This table contains the list of all registered Effectors for this simulation.

*EffectorName*: Displays the name of the Effector object as found in the Blender interface  
*Affects*: which set of Actors the effector will work on. See the section on Effectors for all of your options  
*AffectsValue*: the actual value to use in conjunction with the Affects field  
*Attribute*: the Actor statistic the Effector works on  
*Operator*: + - / \* =  
*Action*: If you want there to be a special Action associated with this Effector, enter its name here.  
*Value*: the numeric value to use with operator and attribute.  
*Active*: Whether or not the Effector is evaluated  
*Duration*: whether the Effector works directly on Actor stats in the db, or effects them on-the-fly. See the section on Effectors for all of your options  
*x,y*: DNM; the current location of the Effector in the simulation  
*Size*: the spherical radius in Blender Units of the Effectors area of effect

### **tblgroundtree**

This entire table is DNM. It's the persistent storage of a quadtree of the Ground object, caching height and color information.

### **tbllocomotion**

This table holds the links between Actor types and the different sorts of locomotion available to them.

*TypeID*: the Type of Actor that will use this action ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.  
*MoveType*: "walk" or "run". Only "walk" is currently supported.

*ActionName*: the name of the walkcycle Action in Blender  
*likelihood*: ignored in this version of BlenderPeople. Future versions will allow you to attach multiple walk and run cycles for greater animation variety

### **tblorders**

The default orders that Actors follow if they do not have a commander.

*Team*: the team that will use these default order  
*Orders*: the default orders  
*SpeedMultiplier*: the speed applied to the default orders

### **tblruncycles**

Not used in this version.

### **tbltypes**

This is the list of different types of characters available to the simulation.

Fields in this table duplicate the fields found in tblactors, with one exception. Each field, eg. AttackRadius, has a partner field, eg. AttackRadiusV. This partner field is a percentage factor (0.1=10%) around which generated statistics for Actors will fall. (A Type with an AttackRadius of 20 and AttackRadiusV of .25 would generate Actors with AttackRadius values from 15-25 (25% on either side of 20).

The Armature field is not used.

### **tblwalkcycles**

This table contains the information for the different walkcycle Actions in the Blender interface

*WalkName*: the name of the walkcycle Action in Blender  
*leftstart/leftend*: the start and end frames during which the left foot is touching the ground in your walkcycle  
*rightstart/rightend*: the start and end frames during which the right foot is touching the ground in your walkcycle  
*length*: the distance covered by a single step in your walkcycle. One way to determine this is to attach your walkcycle to your armature in the Action window, then keyframe your entire armature object forward until foot slipping is at a minimum. Use empties, or some other method, to measure the distance covered by the forward motion. This value does not have to be exact for BlenderPeople to generate slipless animation, but the closer you get it, the better the final animation will look.  
*height*: the distance from the ground to the foot controller at its highest point in the stride  
*leftname/rightname*: the names of the left and right foot controller bones  
*likelihood*: ignored in this version of BlenderPeople. Future versions will allow you to attach multiple walkcycles to a single Actor type, creating variations along the length of the final walk.

## **8. Building Your Own Character Animation Library**

Building a high quality character animation library for BlenderPeople is not an easy task. For that reason alone, I encourage you to share any successes you have with it with the worldwide Blender community. The more good BlenderPeople set ups there are floating around, the more people will be inclined to use it.

### **A. Models**

#### **A Strong Basis: The Armature**

The success or failure of your efforts will most likely hinge on your armature. Here are the rules:

1. Only bipeds are currently supported (so no horses yet -- sorry)
2. Legs must be IK chains (so no BVH motion capture yet -- sorry again)
3. A single controller must be responsible for both foot placement and foot/leg rotation
4. No Hinge bones -- Action Baking doesn't like Hinges yet

5. All controllers should be contained within the armature, i.e., no IK constraints to Empties

And the final rule, which is so important that it's outside the list:

*Armatures, when not in edit mode, must face to the right of the screen from a top view.* To be sure, pop up the n-key properties panel for your Armature object in the 3D view and make sure that it has no rotation or scaling (rotations all 0, scaling values all 1). If it has either rotations or scaling, hit alt-r and alt-s to remove those values, then re-enter edit mode and perform any needed scaling or rotation there. If you don't have your armature set up properly in this way, nothing will work correctly.

At the current time, walking velocity thresholds for building locomotion are hard-coded, so you are encouraged to maintain the same scale as the included armature for best results. This restriction will be removed in the future.

## Meshes

You can create as many different versions of your mesh object for Level of Detail as you care to. If you are going to create multiple meshes for different levels of detail, you must link each one to a different armature object (the different armature objects can all be instances of a single armature, though. They do not have to be separate copies).

## Groups

Once you have both an armature and a mesh object, you will need to link them together, and make sure that the armature properly deforms the mesh.

*Currently, MatchBone only seems to work when using the Parent Mesh Deform method.* Adding the armature to the Mesh as a modifier will not work correctly.

When you are happy with the way that your armature deforms your mesh, select them both and hit Ctrl-G to make them into a new Group.

## Register Groups in the Database

Using your database browser (see Section 7), you will need to make an entry for each Group and each Actor type. For each Actor type that will use this group, create an entry in the "tblcharacters" table of the database.

*TypeID:* the Type of Actor that will use this group ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.

*GroupName:* the name of the Group you just created

*MaxDist:* the maximum distance from the camera that this particular group should be used. If you're not bothering with Level of Detail, set this to something very high, like 10000.

*MinDist:* the minimum distance from the camera that this particular group should be used. If it is the highest resolution group, or you are not using Level of Detail, set it to 0.

*ArmatureObject:* the name of the armature object within the group. Blender needs this information to properly target NLA strips within groups.

## B. Actions

### Create Actions

Now that you're happy with your armature and meshes, you can start creating actions. You *could* start animating before you have a good mesh, but you'll find that your visualization of the action with stick figures or b-bones is significantly more forgiving than it will be with the final mesh.

You can create as many actions as you care to for any of the coded behaviors in BlenderPeople. The more actions you create per behavior, the more varied and realistic your final animation will be. Please note that you don't have to create actions for each of the behaviors. Behaviors that do not have a linked action will simply be ignored when NLA is built. The coded behaviors are:

*Defend:* Actors are mostly stationary in defend mode. They are waiting in place for something to happen.

*Attack:* Actors are actively attacking other actors

*Search*: Actors are in attack mode, but cannot see or hear enemies from their current location. Actors will turn in place and look around.

*March, Charge, Retreat, Mill, Regroup*: at the present, these behaviors should not be associated with actions. As these behaviors only take place while an Actor is walking (or running), they should be upper-body only. Future versions of BlenderPeople will give you the ability to incorporate different upper body actions with the existing walking animation, but for now, linking actions to these behaviors will give unpredictable results.

### Register Actions in the Database

Actions are linked to Actor types in the "tblactionlinks" table. Each action that you want to associate with an Actor type needs the following information:

*TypeID*: the Type of Actor that will use this action ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.

*Orders*: ignored for now. Future versions of BlenderPeople will allow you to differentiate behavioral actions based on varying orders. For example, an Actor exhibiting an Attack behavior while under Retreat orders would use a different set of Actions than an Actor exhibiting Attack behavior while under Defend orders.

*Action*: the behavior with which to use this Action. Enter something from the previous list (Defend, Attack, Search, March, Charge, Retreat, Mill, Regroup)

*Likelihood*: A weighting value from 0 to 1 for how likely this Action will be to be selected from all available Actions. Since a behavior can have several Actions associated with it, this value will help to determine how often this particular Action is chosen.

*OptimumFrames*: the length in frames of the Action. As this is how you designed the Action, BlenderPeople will try to use it at this size in the NLA.

*MinFrames/MaxFrames*: the minimum and maximum size in frames that you are willing to let this action be compressed or expanded. Try scaling your Action as an NLA strip and playing back the results. You are trying to find the maximum range over which the Action still looks good.

*ActionName*: the actual name of the animation Action in BlenderPeople

## C. Walkcycles

### Create Your Walkcycle Action

Currently, each Actor Type in BlenderPeople can be associated with a single Walk action. Support for running and multiple walkcycles is already coded, but has been disabled in this release because it is not yet up to my standards.

Walkcycles should follow the standard rules of such things, and many tutorials are available on the Internet for making good walkcycles. Keep in mind that BlenderPeople will be taking over control of the actual foot controller bones for automated walking, so don't do anything radically fancy with that part of things. Feel free to include whatever secondary motion (head dipping/arm swing/etc.) you like, though. BlenderPeople will make good use of it, synchronized with the footsteps.

### Register Walkcycles in the Database

A walkcycle will require two database entries, one in "tblwalkcycles" and one in "tbllocomotion".

In "tblwalkcycles", you will need to enter the following information:

*WalkName*: the name of the walkcycle Action in Blender

*leftstart/leftend*: the start and end frames during which the left foot is touching the ground in your walkcycle

*rightstart/rightend*: the start and end frames during which the right foot is touching the ground in your walkcycle

*length*: the distance covered by a single step in your walkcycle. One way to determine this is to attach your walkcycle to your armature in the Action window, then keyframe your entire armature object forward until foot slipping is at a minimum. Use empties, or some other method, to measure the distance covered by the forward motion. This value does not have to be exact for BlenderPeople to generate slipless animation, but the closer you get it, the better the final animation will look.

*height*: the distance from the ground to the foot controller at its highest point in the stride

*leftname/rightname*: the names of the left and right foot controller bones  
*likelihood*: ignored in this version of BlenderPeople. Future versions will allow you to attach multiple walkcycles to a single Actor type, creating variations along the length of the final walk.

In "tblocomotion", you will need to enter the following information:

*TypeID*: the Type of Actor that will use this action ("a" is the one provided with BlenderPeople). This value will link to one of the Actor types from the tbltypes portion of the database.  
*MoveType*: "walk" or "run". Only "walk" is currently supported.  
*ActionName*: the name of the walkcycle Action in Blender  
*likelihood*: ignored in this version of BlenderPeople. Future versions will allow you to attach multiple walk and runcycles to a single Actor type, creating variations along the length of the final walk.

## 9. Roadmap

### **BlenderPeople 0.9-1.0: Refinement, Speed and Tools**

Up until 1.0, I'll be working on bug fixes and speed optimizations. I'm currently working on (well, thinking about) ways to approach Mesh, material and armature variation to give randomization throughout population groups. I would also like to include a non-combat set of Actors and actions, more of a "city streets" sort of thing. If anyone is interested in helping to create photorealistic meshes of business people or armatures actions appropriate to that sort of thing, give me a shout.

## 10. How Can You Help?

### **There are a lot of ways that you can help this project!**

One way that you can help is to show your support. Pop into the BlenderPeople development blog at <http://www.harkyman.com/bpblog> and leave me some love in the comments. It really does keep me going to know that people are interested in the project. You can also put the banner art or the buttons from <http://www.harkyman.com/bp.html> on your website. If your really crazy, you can go to CafePress and get a BlenderPeople shirt or hat. I made a couple for myself, and if you're really into it and buy one, I'll get a couple of bucks.

I *will not* be asking the general Blender community to make donations (hit my tip jar! stuff). However, if you are a wealthy individual and you think this is cool, you could buy me a used BMW Z3, or pay for me to go to SIGGRAPH next year (\$ prevented me from attending the 2006 show), or buy me a nice new dual core system to replace my older-every-day Athlon XP 1800 development machine.

Another way that you can help is in bug reporting. See something that's going completely bonkers? Let me know in the blog comments. And I don't mean stuff like "I can't get it to run!" because that's almost certainly an installation/system configuration problem. If anyone wants to proofread this very document, that would be helpful too.

Above all, the best way to help is to actually use BlenderPeople! If you use it to make some cool animation, or create a new library of meshes and actions, by all means I want to hear about it. And if you actually use it in some kind of production capacity, be it commercial, freebie or whatever, please oh please let me know.

## 11. How Can You Get Help?

### **Help is available...**

If you're having system configuration problems with **MySQL** or **MySQLdb**, come on over the forums on [blenderartists.org](http://blenderartists.org). Look me up under "harkyman", and look for a "BlenderPeople support" thread. The odds are that most of your questions will already have been addressed somewhere in the thread.

If you're having problems with **Python**, and with getting Blender to recognize your full Python install, there is tons of help available in the [blenderartists.org](http://blenderartists.org) Python forum. Seriously, getting a full install of Python running with Blender is one of the prerequisites of BlenderPeople that is addressed elsewhere, so don't bring that one to me.

If you would like to see a video demo of BlenderPeople in action, you can head over to [GraphicAll](http://GraphicAll.com), the website that hosts Blender test builds and has generously offered to host an ISO of a video demo DVD. The

video will be available on YouTube in the typical crap version that they have, but you are encouraged to watch the 1/2 hour video in DVD quality. For those who don't know, you download the ISO file, crack open your DVD burning program and most of them will have an option for burning a DVD from an .iso file. Don't feel like doing a several hundred MB download? Contact me via email at [me@harkyman.com](mailto:me@harkyman.com), and I'll be more than happy to make and send you a DVD. Is \$15 fair for my time in making the DVD, taking it to the post office and mailing it? I think so. So, \$15 via paypal will get your own copy of the DVD. Personally, I don't care if one person takes the time to download the ISO, then burns it and you all pass it around for the cost of postage. Or seed a torrent. I'm just trying to provide options, people.

Although I try to provide support through the BlenderArtist forums, I'm only doing this in my spare time, and I probably won't be able to give each and every person who has some esoteric problem the hours of attention and therapy sessions they would like. People who are using BlenderPeople on larger productions can email me for more direct help.

And finally, if all else fails, try reading these docs. I know it sounds crazy, but it just might work.

