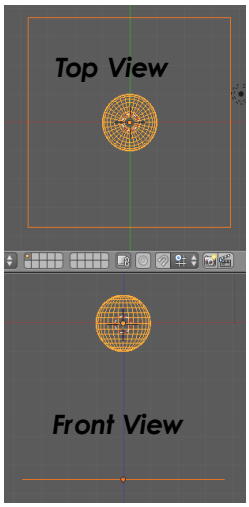
Chapter 21- Game Engine Basics

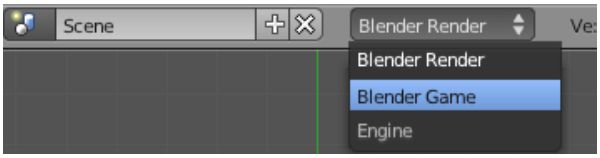
So how can you make 3D games with Blender? How can you use the physics in Blender to actually create animations for you? Have you tried to make dominoes fall realistically in Blender using traditional animation keys? It would be very difficult to do. Using the real-time features in Blender will do a much better job for you and with a lot less work. A blockbuster movie was just produced last year called *2012* that needed 3D animation of falling buildings and debris that looked real so they turned to the *Bullet* physics engine to do the work. Bullet is the same physics engine used for the real-time features in Blender. The Blender game engine uses a programming language called *Python*. Can you make nice games in Blender without knowing Python? The answer is "yes", but if you want to reach a more professional level, knowing Python is a definite. There is a lot of nice documentation on the web for learning Python.



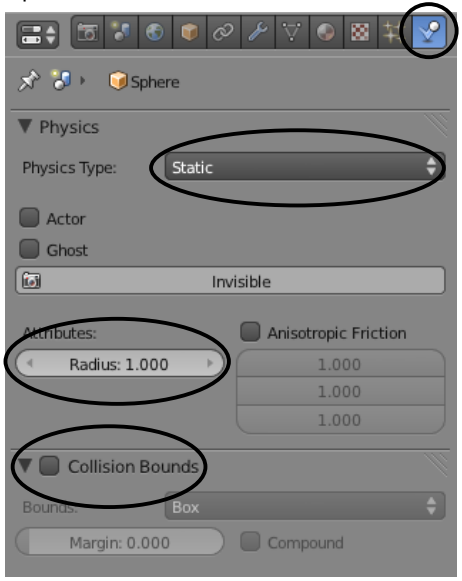
Setting Up The Physics Engine

Let's say you want to use physics to make a ball bounce realistically. The 1st thing you need to do is set up the scene. For my sample scene, I have created a UV Sphere a few Blender units above a plane in a front view. Remember that this scene will be using gravity and reactions. If you make your scene in the top view laying flat, it will work just like real life.

It's now time to set up the real-time animation. When I first looked at Blender 2.5 I couldn't even figure out how to turn something into an actor because the interface changed so much! Here's what you need to do:



To enable the Game Engine physics, go to the top bar and find the box for the *Render Engine*. Change it from "*Blender Render*" to "*Blender Game*". This switches many of your property tool panels to game engine options. We are interested in settings in 3 of these panels:



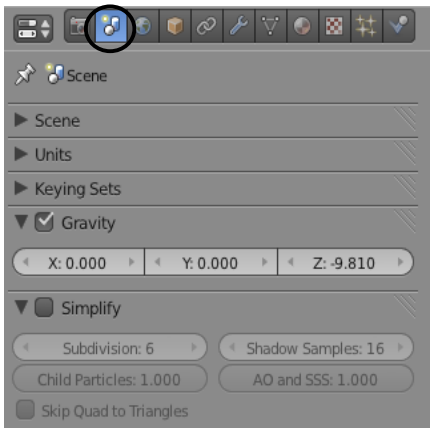
Physics Panel:

In the Physics panel, you control the *Actors* in your real-time animation. By default, everything is "*Static*", meaning that it doesn't react to the physics settings. It can still do things when logic blocks are applied to them, but do nothing otherwise. The other 2 main types we will discuss later are "*Dynamic*" and "*Rigid Body*" actors. You can also make something invisible here.

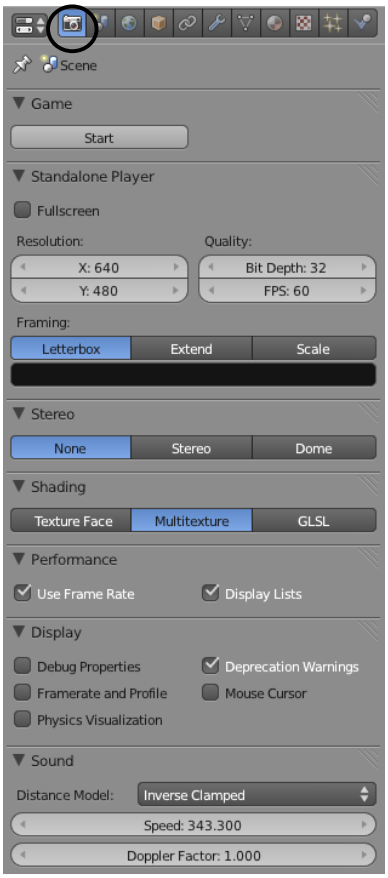
Two other important settings are "*Radius*" which controls the actor size and "*Collision Bounds*" which sets the shape of the actor. All of this will be addressed later.

Chapter 21- Game Engine Basics

Scene Panel:



The most important setting for the game in this panel is the "*Gravity*". By default, it is set to real gravity, but what if you want to make a game set in space where gravity isn't an issue? You will want to set gravity to zero or something really low. Maybe you want to make a game where objects are pulled to something in the X axis.

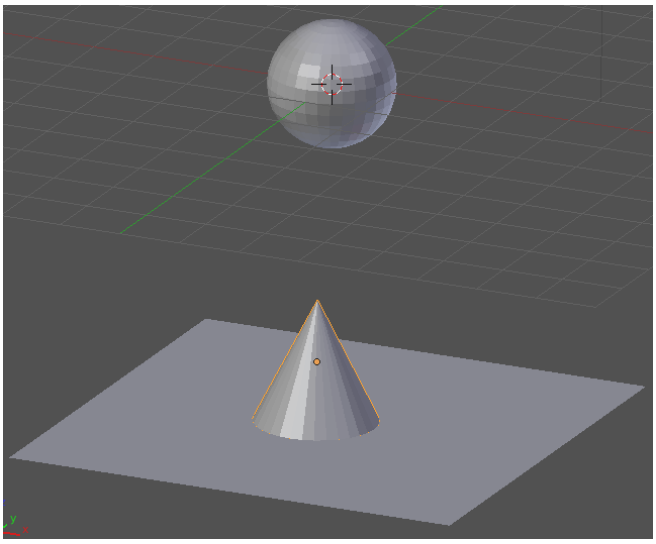


Render Panel:

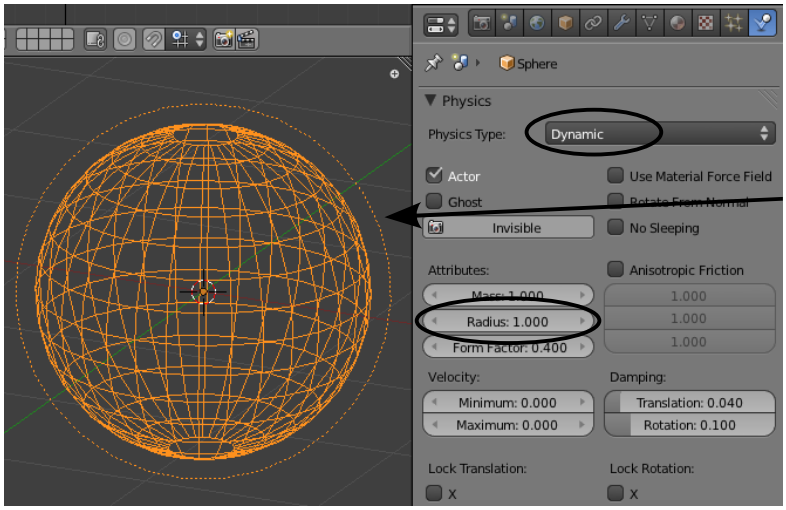
Just like rendering a picture to see your output, this is where you enable the game to play. You can press the "*Start*" button here or just press "**P**" to play in a viewport.

Your end result of making a game is for that game to be played as a standalone (*not in Blender*). This means saving the game as an executable that can launch itself, free of Blender. You can set the size of the game, the color depth, Frames-per-second (FPS), and full screen effects.

Since games rely heavily on sound effects, the game engine has setting features that deal with how the sound is played as well.



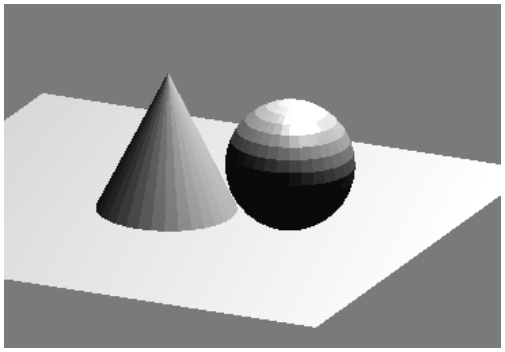
It's now time to apply some physics to the sphere. Add a *Cone* to the scene so the ball has something to deflect off of as it falls. Switch to a shaded view and select the UV Sphere. Rotate your view slightly so you can see what happens when we apply the physics. You want to see the ball drop and how it drops. Now go to the *Physics* panel so we can change some setting.



Change the *Physics Type* to "*Dynamic*". If you are in wireframe mode and scaled the sphere down in size, you will see a dashed circle around it. This circle represents the actual size of the actor. You will need to change the "*Radius*" setting to match the size of the sphere. If this circle is larger than the sphere, when you play the physics, the ball will hover over the plan and never touch it.

RoboDude Says: *The game engine likes actors (radius) to be a size of one whenever possible. If you scale it down and also scale the radius circle to match, it may still not work correctly. Pressing "**Ctrl-A**" and applying a reset the Scale and Rotation can usually correct this problem.*

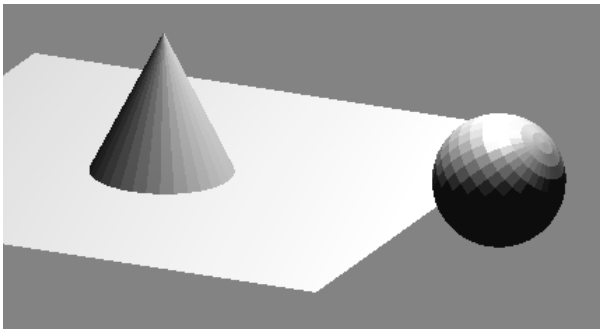
It's now time to test out the system. Switch back *Solid* display mode. With your cursor in the 3D viewport window, press "**P**" to put Blender into game play mode. The ball should fall and hit the cone, but it probably won't act like right. Depending on where you placed the cone, it may even balance on the top of it! If that happens, move the cone slightly to one side and try again. The ball hits the cone, then slides down. It doesn't rotate like a real ball. To exit game play, hit "Esc".

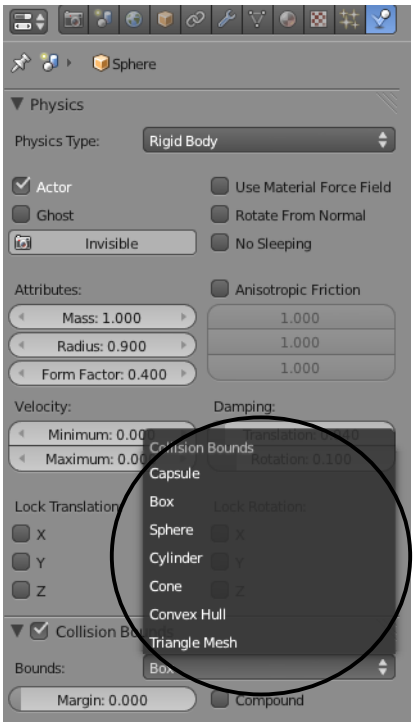


Dynamic and Rigid Body Actors:

A *Dynamic* actor allows you to use physics on it and can fall, bounce and be pushed by forces, but not act like a true solid (*rigid*) body. These actors are great for games where you need to drive or run around in a maze or other scene. A *Rigid Body* actor will like a real solid body. It will spin and deflect when it collides with other objects. Good for some things in the game engine, but better for creating animations like a brick wall collapsing and things bouncing around.

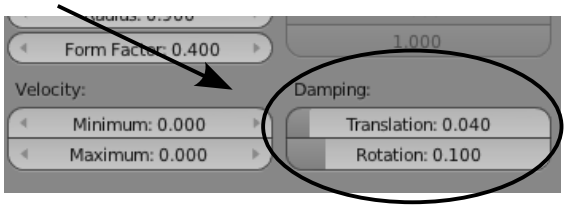
Now change the sphere into a "*Rigid Body*" actor and hit "**P**" to test out the systejm again. The ball should now roll off the plane and fall into nothingness. Press "*Esc*" to exit. Feel free to experiment with some of the other setting like *Mass*. Just like real life, if 2 objects collide with different masses, one will feel the effect more than the other.

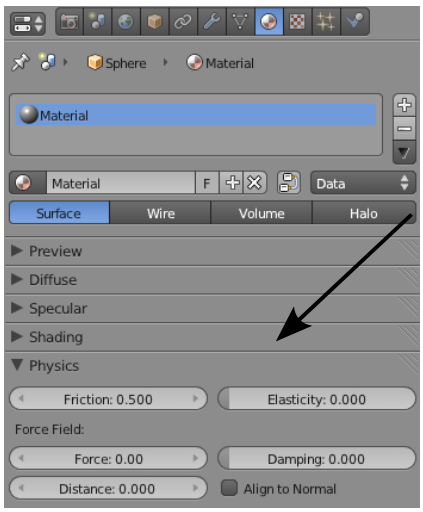




Since you are working with a sphere, you don't notice that even though we are using a rigid body, the actor physics are still calculating to the *Radius* setting in the *Attributes*. If you were to delete the sphere and use a Cube instead, it would roll off the plane like the sphere did. To fix this, you need to turn on "Collision Bounds" and choose a bounds option. "*Box*" would be good for a cube mesh while "*Convex Hull*" or "*Triangle Mesh*" would be better for a more complex shape. You would need to experiment to see which works best for your model.

As you watch your physics in action, you may notice some other reactions that seem a bit off. For example, the ball may slide a bit, or not enough. It may not bounce much or it may spin too much, or not enough. We have 2 places where we can control some of these factors. The first place is in the Physics panel. You will find a block for *Dampening*. The "*Translation*" slider controls the amount of sliding in a direction (like being on ice) while the "*Rotation*" slider controls resistance to spinning. These 2 features will be discussed more when we talk about making a game.

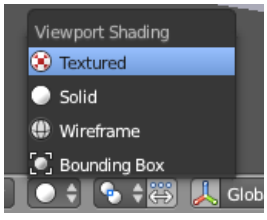




The second place to make changes to reactions is in the *Materials* panel. Add a material to the sphere. And find the *Physics* settings. If you want something to bounce, adjust the "*Elasticity*" slider, "*Friction*" controls slippage. You can also provide *forces* and other *dampening* here as well. *For these to work properly, you usually need materials set on both interacting objects (ex. Elasticity on both the sphere and the plane).*

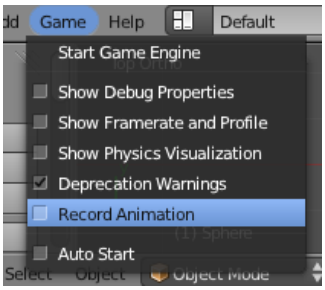
Materials in the Game Engine:

Some things that work in rendering do not work in the game engine while other features do. For example, a standard image texture may display in the game engine, but many adjustments to that texture may not work. There has been a lot of development in texture work for the game engine and we will examine some of that in the UV mapping chapter. For now, just work with straight Diffuse material color. To see what things will look like in a game, change your view type from "Solid" shading to "Textured" shading. Press "**P**" and your view will reflect what will be seen in a saved game. Since the next section deals with applying game physics to an actual saved animation, texture can be handled exactly as we have in previous chapters.

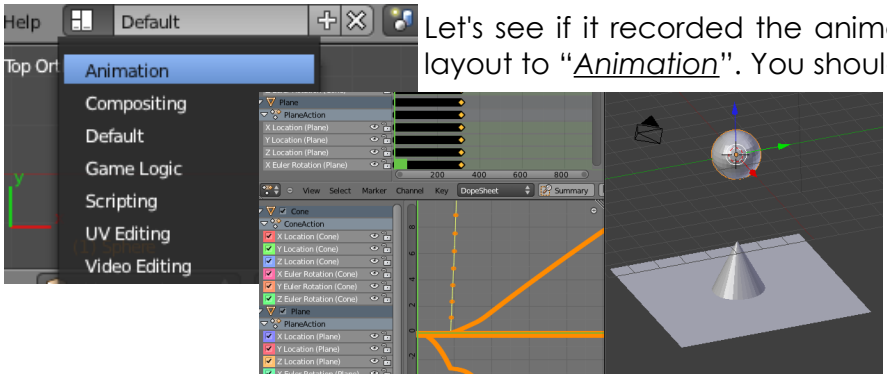


Using Game Physics in Animation

So far, you have a ball dropping on a cone and rolling off the plane. It works when you press "**P**" to enable the game engine, but what if you want to use this reaction in a movie? If you press "**Alt-A**" to play an animation, nothing happens. That is because the reaction has not been written into an animation curve... yet.



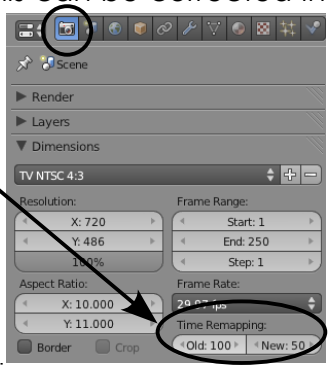
Writing the game physics to an animation curve is a simple process. In order to write to a curve, go to the "*Game*" pull down menu and select the "*Record Animation*" option. This will enable the game record feature. Now, press "**P**" to run the game engine. Let the physics run through, then "*Esc*" the game engine. Go back to the "*Game*" menu and turn "*Record Animation*" off. You should also change the *Engine* back to "*Blender Render*".



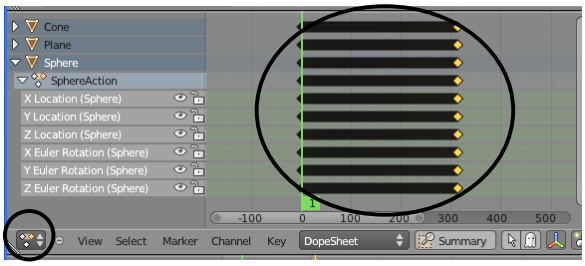
Let's see if it recorded the animation. Switch your screen layout to "*Animation*". You should see animation curves in the *Curve Editor* window. Press "*Alt-A*" to confirm the animation. You can now work with your scene exactly as you would for any other animation work including materials and textures.

RoboDude Says: *Remember to TURN OFF the "Record Animation" feature after you have recorded your motion. If you leave it on and accidentally press "P" again, it will try to over write your saved animation curves!*

The only problem you may encounter when saving a movie file will involve the *speed* of the animation. The physics may be run slow in the final movie. This can be corrected in several ways.



Method #1: Remap the timing in the Render panel.
Find the "*Old*" and "*New*" mapping settings. If you need the movie to run twice as fast, set "*New*" map to 50 (50%) and adjust your *end frame* to half. If you need it to run slower, like ½ speed, try a new map of 200 and double your end frame.



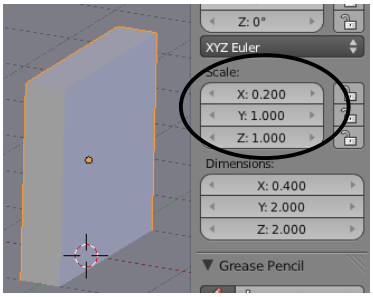
Method #2: Scale Keys in the Dope Sheet.
Another method is to select *All* keys in the *Dope Sheet* window and *Scale* them in the "*X*" axis (*"S" to scale and "X"- drag the mouse*).

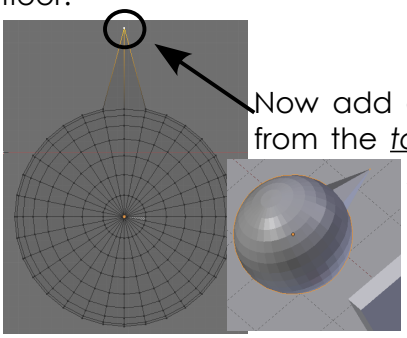
Using Logic Blocks

We have talked about using the physics for animation, but now it's time to look at using Blender for *Real-Time* animation like an architectural walk-through and yes, *games*.

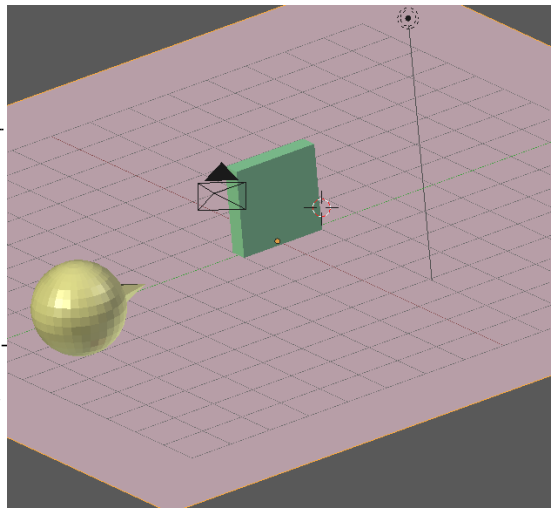
Scene Set Up:

Start a new scene and make a Cube resting on a plane. Using the "**N**" key to open the *Transform* bar, change the scale X of the *Cube* to 0.200. We'll use this as a wall block. For the *Plane*, scale the X and Y to 10.000. This will be our floor.





Now add a *UV Sphere*, enter *Edit Mode* and select a single vertex from the *top view* as shown. Use the "G" key to pull it out from the sphere. This will indicate the forward direction when we turn this into an actor and move it around with the arrow keys. Make sure that it is above and not touching the plane. This could cause it not to work when we turn it into an actor.



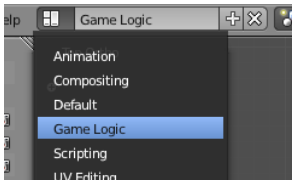
We now have a basic scene to work with. Add a *Material* to each object and change the *Diffuse* color for each so they stand out. You should have something like this scene.

Setting the Actor:

It's now time to turn the Sphere into a *Dynamic Actor*. Start by setting the Engine from *Blender Renderer* to *Blender Game* (*page 21-1*). Go to the Physics panel and select "Dynamic" for they type. To keep the actor from sliding or spinning too much in the game, we'll set Translation Dampening up to 0.400 and Rotational Dampening up to 0.900. You may need to experiment with these later, but these settings should be good. If these settings are too low, you will notice that your actor "coasts" a lot after you take your finger off the key. This is also controllable in the materials settings with friction.

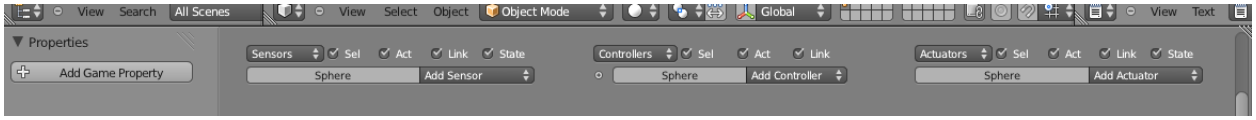
We shouldn't need to change the radius size since we didn't scale the sphere, but if you did, adjust the radius size to match, then hit "*Ctrl-A*" to reset scale and rotation settings.

It's now time to switch to the "*Game Logic*" screen layout so we can add some controllers.

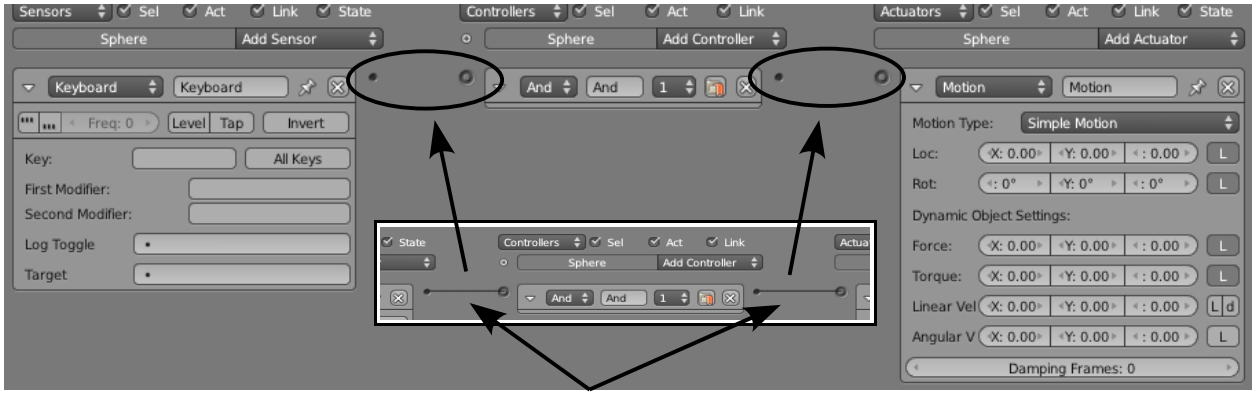


Logic Block Construction:

Now that you've switched to the *Game Logic* screen layout, you will see the logic block window at the bottom. Think of this as an "Input-Process-Output" model, but called "Sensor-Controller-Actuator". You will also see a place to add a *Property*.



There are a lot of different types of sensors, controllers and actuators that you can use, more than we will discuss here. After you get a feel for working with this chapter, there are many discussions and examples on the internet addressing practical examples of all these. To get started, lets add a "***Keyboard***" sensor, a "***Add***" controller, and a "***Motion***" actuator.



First thing, connect the blocks by dragging a line. To disconnect the, drag backwards.

The first thing we want to do is make the sphere move forward when we hit the Up arrow key. Click in the box by the word <u>Key</u>. It will say "*Press a key*". Hit the <u>Up</u> arrow key to assign it. There are other options, but we do not need them for this exercise.

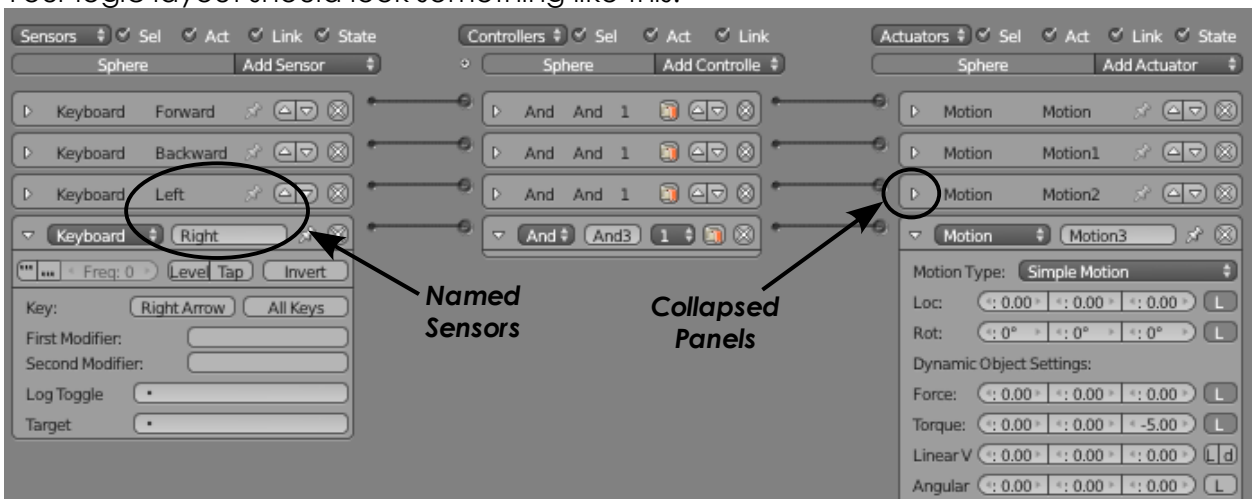
Think of the *Controller* as the computer processor. By default, we hit "*And*", meaning that if we tie more than 1 sensor to it, all sensors must be in a true state in order for an actuator to function. There are other expression available in the controller.

The Motion actuator works for dynamic and static objects. When moving a *Static* object, you will want to use the <u>Loc</u> and <u>Rot</u> motion outputs. You are setting a step movement or rotation. ***You probably do not want to use these for Dynamic actors!*** If you do, an actor might walk right through a wall. Think of this as real life. To move a *Dynamic* object, it needs a push (<u>Force</u>) or turning force (<u>Torque</u>). You will see columns for X,Y, and Z. **Let's set the Y Force to 5.00.** Hit "**P**" to test out your scene. Adjust the force if more or less is needed. If it goes the wrong direction, try a negative number or try the X column. Adjust actor *Dampening* to improve stopping.

Now that you have the sphere moving forward, add more sensors, controllers and actuators to make it move backwards. In my case, all I would need to do is give it a Y force of -5.00 (*or any speed you wish*). To make it turn, you will need to apply a Torque in the Z column. **A Torque of 1.00 may be enough If not, try higher**. You should now have 4 directional keys for the sphere. *It's also a good idea to name your sensors. You may have a lot of them. You can also collapse them by clicking the small triangle.*

***RoboDude Asks:* Having trouble with the sphere rolling strangely when moving forward?** *Try going to the Materials panel and reducing the Friction of the sphere or floor (page 21-4). If your actor spins when it his the wall, also lower friction for the wall.*

Your logic layout should look something like this:



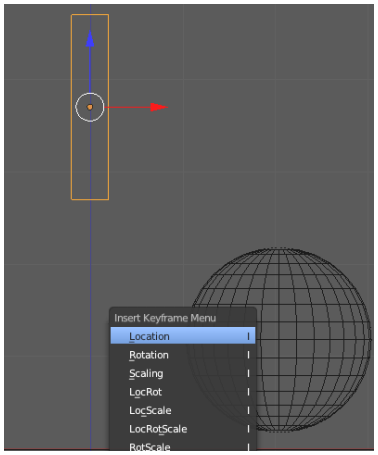
Let's add a ***Jump*** command using the *Space Bar*. Since you want him to jump and not fly, we will need to connect 2 *Sensors* to a *Controller* to make this work. One *Keyboard* sensor for the space bar and one *Collision* sensor with a named *Property*.

Select the *Floor* plane and add a *Game Property* (found to the left of the logic blocks). Give it a name called "*floor*". This is case sensitive.

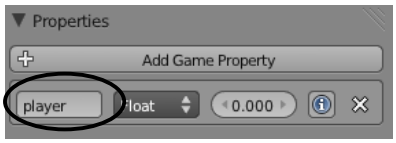
Now go back and select the *Sphere* and add a *Sensor-Controller-Actuator*. Make the sensor a *Keyboard* and assign the *Space Bar*. Use an *And* controller and a *Motion* actuator. Give it a *Force* in the Z-direction of **100**. *Since the force will only be applied momentarily when in contact with the floor, it will need to be high in order to have a good jump.* Now, we need to add another *Sensor* and make it *Collision*. In the *Property* block, type "*floor*". Tie this sensor to the same controller as the keyboard for jump.

Because it is an *And* controller, both sensor states must be true in order for the actor to jump. Adjust the force.

Using Animation in a Game:



Now that we have basic motion down, let's try an animation in the game. We will make the Cube act like a rising door when the actor gets close to it. We first need to add some animation keys to the cube. With the *Cube* selected and at *Frame 1*, hit "**I**" to insert a *Location* key. Move up to *Frame 60*, raise the cube high enough for the actor to pass under it and hit "**I**" again to insert another *Location* key. If it helps, change back to the *Animation* or *Default* screen layout during this step, then return to the *Game* screen.



Back in the *Game* window layout, select the *Sphere* and give it a *Property*. Name it something like "*player*"

Select the *Cube* once more and add a *Sensor-Controller-Actuator* to it. This time, you will add a *Near* sensor, *And* controller, and an *F-Curve* actuator. Set is up as shown:

Distance-Reset:
Adjust for actor distance when trigger is activated. The reset distance (usually higher than distance) resets the trigger.

Start-End Frames:
Set these numbers to match the range of frames you wish to play during the action.

When the actor with the property name "player" gets within the sensor's trigger distance, the actuator occurs. There are several different playing options in the *F-Curve* actuator- *Play* plays the frames and stops; *Ping-Ping* plays frame forwards and backwards; *Flipper* plays forward, stops, then plays backwards during the trigger reset; and *Loop* occurs the entire time when activated.

These are just the basics of the Game Engine. With practice, experimentation, and a little research, you will be able to build some amazing games. Games are played through the *camera's view* so you will want to set the camera's location or child-parent it to the *Actor*. When you're ready to test the game outside of Blender, you need to enable exporting through the *User Preferences* in the *File* menu. Go to *Add-Ons* and select "*Game Engine:Save As Run time*". Now go to File-Export and save as a .exe file.

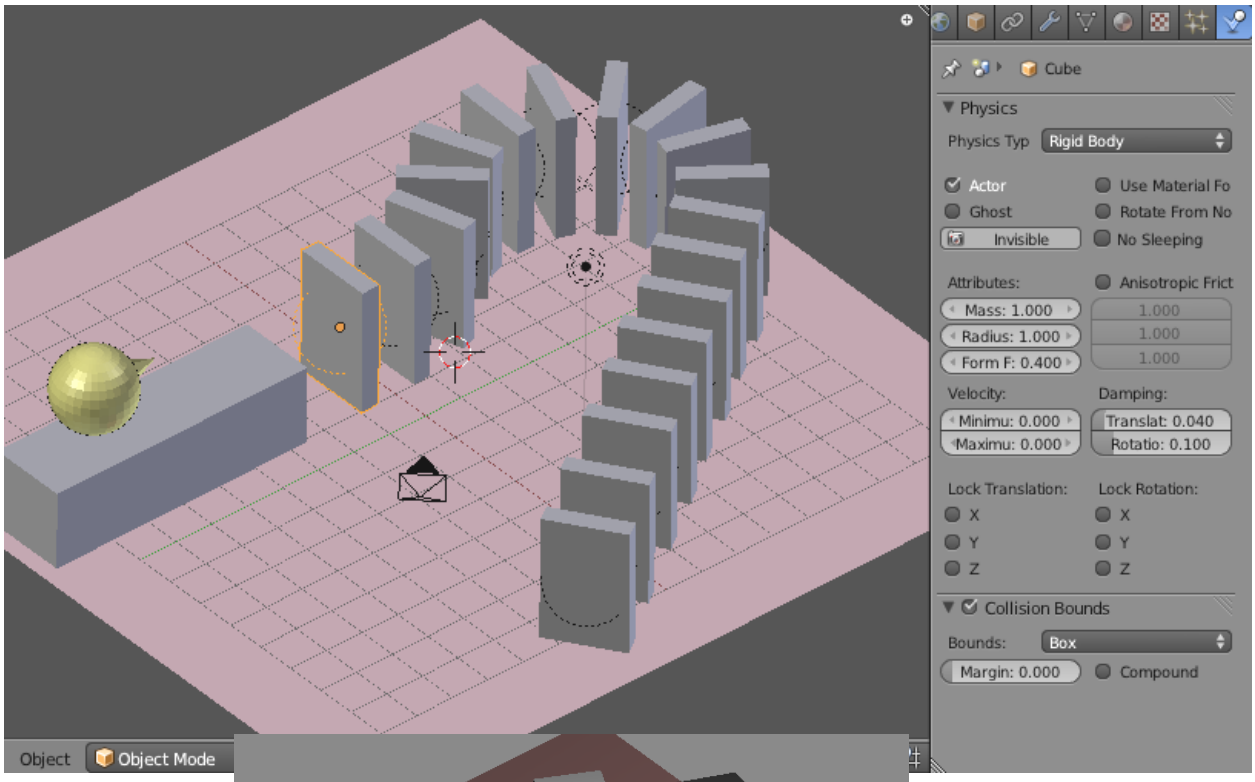
RoboDude Says: *When making a game, try to keep face counts on meshes as low as possible. The game must actively count and deal with the faces in a game. Detailed meshes will slow things down considerably. The best way to simulate detail is through detailed textures, which will be discussed in the next chapter.*

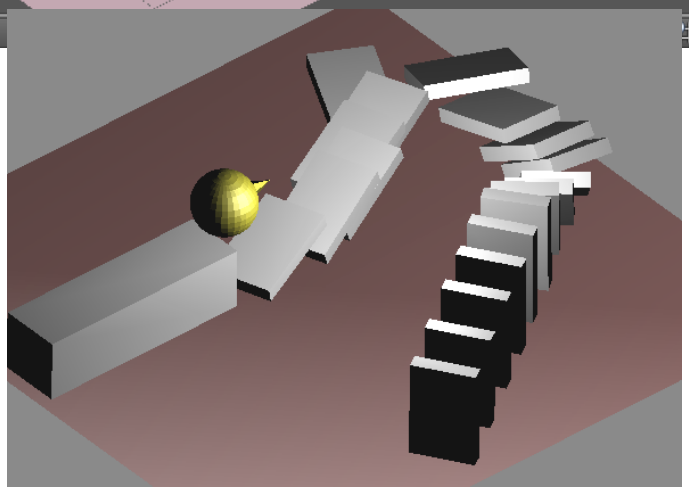
Real Time Practice Exercise

PRACTICE EXERCISE

For this activity, your job is to design a maze full of motion. Create an actor that can be moved around with the arrow keys as discussed in the previous pages and make him start by knocking down some dominoes. To make a domino, start with a cube, scaled into the shape of a domino. After shaping, hit "*Ctrl-A*" to apply scale and rotation (*reset settings*), the turn it into a "*Rigid Body*" actor and use "*Box Collision Bounds*". Duplicate it a few times and test it out to see if you can knock the first one over and that, in turn, knocks the others over. Add as much other detail to your scene as possible and more motion.

If time permits, save the motion to an animation curve and make a movie.





***** Call the instructor when finished*****