Chapter 21- Game Engine Basics

We’ve seen that Blender is a powerful 3D rendering and animation program up to this point, but so far, all of the commands that we’ve looked at are in most high-power animation programs. The big difference is in the cost of the program and some features. One thing (besides price) that makes Blender stand out from the others is its integrated **Real-Time animation features** (aka. the Game Engine). The program integrates real-time motion with physics and logic blocks. For example, you can set your gravity in the world buttons, add friction and force settings to your materials, turn objects into actors and move them around, then have them react to other objects in the scene.

You can create games that look as good as professionally produced 3D games and real-time architectural walk throughs where doors can open and close as you approach them. The best part of this is that it can all be done **without computer programming skills**. There are other freeware game creation programs out there, but most require some programming knowledge. Programming skills in Python scripting are helpful in Blender, but not necessary.

This chapter cannot hope to cover everything you need to know about the game engine. We will only look at how to texture your models and describe the interface and logic. We will also look at the game engine’s ability to be written to an animation IPO curve. For a more detailed description, review the Blender downloaded tutorial on the game engine. It is well-written and describes all of the basic command options.

The Blender game engine has seen a great deal of development over the past few releases. The most recent has been a complete reworking of the texturing system. It is now more confusing, but gives much more flexibility. The original game engine physics module is called **Sumo**, which is still available for selection, but all recent work uses the new module, **Bullet**. Bullet is showing a great deal of promise with accuracy and the ability to be used to create animation IPO tracks from the action.

### Setting Up the Physics Engine

The first step to using the game engine is to set it up. To do this, go to the **Shading** and **World** buttons. Under the **Mist/Stars/Physics** tab you will set the engine to **Bullet**. You can also set the **Gravity** at this time. Most likely, these setting will already be selected for you.
Using Logic Blocks

Let’s make a simple scene consisting of a plane and a sphere and set the sphere above the plane. Modify the sphere by pulling one vertex out to form a nose. This will let us know which way is forward when we move it around. Add a material to each one and feel free to set colors for each. We will use a different process for setting textures on the objects than we do for rendering. We are adding materials for physical properties (friction, elasticity).

Here’s what we have so far:

Select the sphere and go to the Game Buttons (little purple pacman button). Here’s what you see in the buttons window:

- **Actor Button** - to turn the object into “live” actor.
- **Add Property** - used when you want something to happen in your scene. Something will happen when it senses this property.
- **Sensors, Controllers and Actuators** - The “brains” of the game engine. Think of it as Input-Process-Output where data is fed in, the computer processes it and something happens. There are a lot of options in these commands.

Let’s turn our sphere into an **Actor**. Click on the **Actor** button and choose **Dynamic**. Look at the important options now available:

- **Damp**: motion dampening - keeps the object from continuing forever when you stop applying force. I like to set this to around 0.4
- **Bounds**: changes the shape of the actor from sphere (from radius) to other shapes.
- **Mass**: how heavy you actor is.
- **RotDamp**: rotational dampening - keeps the actor from spinning forever. I like to set this to about 0.8
- **Actor Size (Radius)** - You will notice a dashed line circle around the sphere when you change this. This is the actor size.

[Image of game engine interface with labels for Damp, Bounds, Mass, RotDamp, and Actor Size]
Chapter 21 - Game Engine Basics

Move your cursor into the 3D window and press “P” for play. If the sphere is above the plane, it will fall to the plane showing you that it is now an actor. Click the “Add” buttons under Sensors, Controllers and Actuators. By holding the LMB down on each block, you can change its type. Change the sensor block from Always to Keyboard.

Next, connect the blocks together. Once you change the sensor to keyboard, you will see a block for Key. Click in that box and type the key you want to use. For our case, we’ll use the “Up Arrow”. We will tie a force to the up arrow so that when we press it, the sphere will move forward.

Now we’ll apply a force to the actor. You will see three columns in the Motion block. They represent X, Y, and Z. The best way to change numbers in these blocks is to hold down “Shift” and click in the box. In the Force block, let’s change the Y number to 10. This is where you need to experiment with numbers. If a block doesn’t move it in the direction you desire, change it back to zero and try a different one. If it moves on the right axis, but the wrong direction, try a negative number. Once you get this motion right, add another row of block under the Sensors, Controllers and Actuators, connect them and adjust your setting to go backwards. To make the object turn left and right, work with the Torque settings and use the left/right arrow keys. There are a lot of options in these buttons. To get a more detailed description of them, refer to the Blender Game Engine documentation available to download from the Blender website.

RoboDude Says: If you want to move something that isn’t an actor, use Loc and Rot numbers. If it’s an actor, use Force and Torque.

Applying Materials

Before the release of version 2.48, game texturing was a pretty easy process. Materials and textures used in rendering and animation were basically different from UV texturing used in the game engine. With the introduction of new features and more crossing over between rendering and game texturing, the developers have pulled many of the features together which, in my opinion, have caused some confusion. The reason for treating game texturing differently stems from the concept that things in a game need to be kept simple in order for the game to run smoothly and many feature in normal render texturing can slow game play down considerably. UV textures can basically be “mapped” onto the mesh faces which minimize texture calculations while in game play. While traditional textures can be used in the game engine to some degree, I will focus on the more traditional UV mapping process here.
You already know that you can apply a material to an object and it will display in the game engine. First, start by making sure you are in Textured Draw type (Alt-Z will also get you there). For my scene, I will be using a basic cube to demonstrate UV texturing. Place a material on the cube.

In textured mode and the cube selected, set one of your viewports to the UV/Image Editor window. At the bottom of that window you will see some options - View, Image and some up/down arrows. The arrows are used to browse the list of already opened images in the scene. Use the Image menu and Open a texture that you would like to use. I will select a brick texture to use for the cube. Now go into edit mode and switch the selection type from vertices to faces. Hit “A” once or twice to select ALL faces. It’s time to apply the UV mapping by pressing “U” in the 3D window for the UV Mapping option menu. For this example, I will select Cube Projection. Since we had selected the texture beforehand, it will automatically display on all faces (or you can hit the arrows and choose from a list). In the UV window, you will notice faces and vertices show up on the texture sample. These can be moved, scaled, etc. to change the way the texture maps. Instead of selecting all faces, we could have just selected one and applied the texture to just that face (a difference between rendering materials and UV texturing).

Now exit edit mode and press “P” to see how your cube looks in the game engine while running. When done, press Esc to exit play.

### RoboDude Says:
To keep your game as streamlined as possible, always keep your meshes as simple as possible with the fewest number of vertices and faces. Use textures to show detail when possible instead of modeling that detail into the mesh. (i.e. windows in a building, signs on a wall)
Chapter 21- Game Engine Basics

The cube looks good, but we have some more options. Go to the edit buttons and find the **UV Calculations** and **Texture Face** panels. The texture Face panel will only be visible after hitting the “U” key to apply UV Mapping. **Both panels are only visible in edit mode.**

**Copy**- At times, press these for effects to be applied.  
**Light**- Selected faces will reflect light for better scenes.  
**Invisible**- Selected faces will not be seen in the game.  
**Collision**- Actors will not be able to pass through these faces.  
**Two-Sided**- Faces will display textures on both sides of the face (if only one-sided, face is invisible from opposite side.  
**Add**- Textures with dark/black areas will display those areas as transparent. Great for text on a black background.  
**Alpha**- textures with a transparency channel will display this.

UV Calculations are used to display the texture on a face in various ways. For example, if you change “Cube Size”, place your cursor back in the 3D window and press “U” again, select “Cube Projection” and the texture will reflect the changes.

There are a lot of other things that can be done beyond this discussion. Look to blender.org and the forums for more help. **Don’t be afraid to experiment!**

Using Game Physics in Animation

It is possible to use the game engine physics to record an animation (IPO) curve. Now, instead of trying to insert location and rotation keys for a falling (moving) object in the IPO and 3D window, you can set up a **Bullet** model and tell Blender to record the action to an IPO curve for animation. No more frustration trying to make something look like it’s moving correctly in a fall/fly/bounce. Blender will do it for you! To start out, you need to set up your game logic to do what you want. Set up your actor to be a **Rigid Body** actor instead of **Dynamic**. Dynamic actors will not just slide around, but roll or angle when needed automatically. You may want to set **Bounds** also.

When you have the action correct, go to the “Game” pull-down menu and select the “**Record Game Physics to IPO**” option.  
Hit the “P” button to run the action. When finished, hit the “Esc” key. The action will be written to an IPO curve for the object and will now work with “Alt-A” and animate! Apply materials and textures as normal and make your movie.

**RoboDude Says:** Remember to turn off the Record Game to IPO button when finished. It will make a new curve everytime you hit Play!
Create a scene similar to the one discussed in this chapter. It should include one actor and a plane, both textured for game mode. Apply physics to the actor (**Dynamic**) so that he can move forward and back, turn left and right. Create a brick (cube) and make it a **Rigid Body** and apply a *Box* under *Bounds* setting to it. Duplicate it several times (Shift-D) and stack them up. Drive your actor through them so they fall. When finished, create an IPO curve of your action so a movie can be rendered from the scene.

**Call the instructor when finished**