Workshop – Game prototyping with Blender

CHILDSPLAY

MARTIN STRUBEL

Blender, the well-known freeware modeller, now has some additional talents. Version 2.0, also known as "Game-Blender", includes special functions that allow you to create complete 3D games in next to no time.

Fig.1: Snapshot of a test level created with Blender

Blender 2.0 was released at the Siggraph event in July 2000 by the Dutch firm *Not a Number* (NaN). A significant new release, it includes some quite stunning new features that many a Blender fan had been awaiting feverishly, including a built-in game engine allowing for fast drafting of games models. The initial release of this new version does have one or two problems, a slightly wonky physics engine and missing Linux sound support being the most significant. The package is undergoing very rapid development, though, and a new version appears on the Internet (*http://www.blender.nl*) pretty much on a monthly basis. This is a good thing, but because things are changing so quickly you should expect some compatibility issues to arise. Collision detection and game dynamics are likely to work differently before too long, for example, and a new physics engine is currently being worked on. Also in the development phase is a new Python-API, with which allows even more complex game actions and object types to be defined.

Having said all this, the most current version at the time of writing, version 2.04, works very well indeed. It is extremely simple to create interactive environments without even having to write a single line of code. More complex game concepts can also be created with ease too, but doing so does require a little bit more time and effort. You should be aware, though, that you won't have much fun with Blender 2.x without 3D hardware support. I use a *nVidia TNT2* under XFree86 4.0, but other cards will work too.

How do I produce a game?

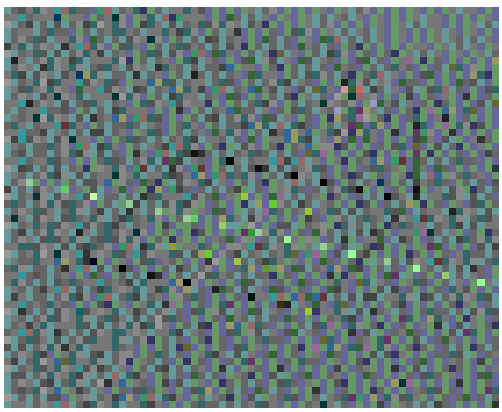
This feature is aimed at those who are already familiar with Blender and virtual reality. We will still go through a very quick overview of the individual steps in the creation of a Blender game environment though, just in case you are a bit rusty.

Create the level: Blender usually organises a game level in a *Scene*. This, in turn, is split up into

sectors, for which transparency tests are performed. In order to be able play a level at a reasonable speed (in other words at a high frame-rate) you should give a lot of thought to the subdivision of a level into sectors using polygon complexities.

- Add moving objects: Generally, animated or moving objects are called *Props* (short for properties – i.e. characteristics). Moving objects, which should be subject to the laws of physics – such as the player, monsters, or bullets, are regarded as *actors*. The engine evaluates collisions between these objects and other objects.
- Interactivity: Certain events trigger actions, which are defined in Blender using what's known as a *SCA* mechanism. *SCA* stands for Sensor-Controller-Actuator. This is when an object (*Prop* or *Actor*) is assigned one or more sensors. These react to certain events such as key actuations and are linked logically with other events by *Controllers*. They can, using *Actuators*, trigger an action such as an animation. The method will be explained later using the example of a Switch-object.
- Process and refine dynamics: The materials menu in version 2.0 has been expanded, a *DYN* (for dynamic materials) option having been added. This makes it possible to define physical properties of a surface (such as friction, elastic reflection, etc.). In addition, global parameters apply to the player object, such as general friction, gravitation and so forth.

An example



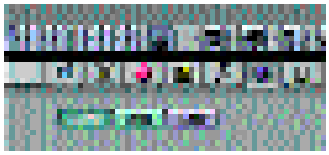
Ok, now that you have the basics under your belt, let's go on to create a simple environment.

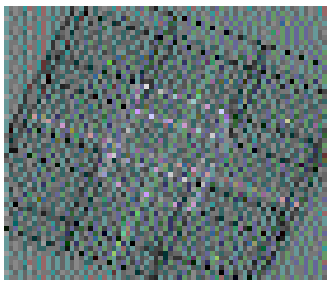
1. First create a *plane*. Now scale it through *EditMode* to the desired size – Blender always sets object scaling to (1.0,1.0,1.0) for sectors as soon as the engine is started. Extrude a point in the Z-direction as can be seen in the picture. Especially important is the direction of the area normals, which are displayed via the Edit button ([F9]) by *Draw Normals* – if applicable increase *NSize*. Inverting the direction of normals is done via W and *Flip Normals*.
2. Leave Edit mode and activate the *Sector* option in the RealTime menu (lilac Pac-Man).
3. The sector thus created is drawn with *Bounding Box*. Copy this sector using Shift+D several times (while holding down Ctrl), so that they join together as in Figure (c).

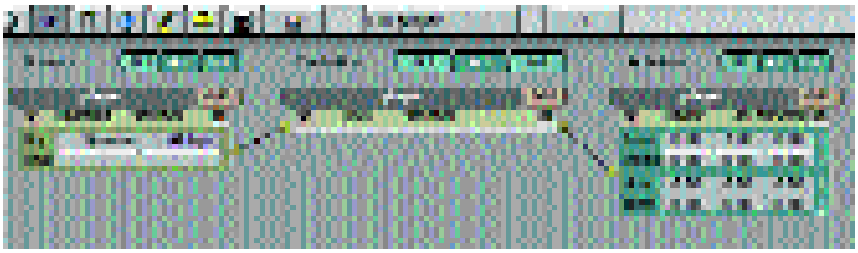
Now add, on top of the ground you've just created, an additional mesh and activate the *Actor* attribute for this, followed by the *Dynamic* and *MainActor* options. You guessed it – this is our player. Dynamic actors possess additional attributes too – see Table 1.

We'll tell you more about attributes later – first we need to define a few sensors for the object, which for the purposes of our example, we've now renamed as *player*.

The link of the signal channels occurs via the yellow blobs by clicking the mouse on the output-blob, holding it down, and drawing the line to the input-ring. The signal channel is







removed by clicking on this. Now create an SCA combination as in Fig. 2 by adding a *Keyboard* type sensor, clicking in the *Key* field and pressing the desired key – for an up-arrow select *Uparrow*, for example. For the actuator, select an *Object* with a *Force* of 1.0 in Y-direction. Now try to add rotation-actions by registering suitable sensors to the left and right arrow keys. For *Torque* use a value of around 0.4 (left rotation) and -0.4 (right rotation) in the third co-ordinate field (Z). You should also activate the axis display of the object ([F9]): *Axis*. Also take note of the L (for *local*) button nearby. When activated, movement occurs with respect to the local co-ordinate system of the actor – if not then the global or superordinate co-ordinate system is used. You can test the control straight away by pressing the hot key to start the engine: P for

Fig. 2:
Creating a sensor for forward movement
Using *Add*, a new sensor, controller or actuator can be added. Via the selection menu, the type of sensor (*Always*, *Keyboard*, etc.), or the logical link of the controller or the action of the actuator is selected. In addition to this any drawing can be entered. With the orange triangle you can pack in the input form respectively. You can find additional details about the display options in Table 2.

Table 1: Actor attributes

Do Fh	Activate Normal force
Rot Fh	Align on the level (e.g. for car on racetrack)
Mass	Mass
Size	Radius of collision sphere
Damp	Damping of movement
RotDamp	Damping of rotation

Table 2: Display options

Sel	Display all selected objects
Act	Display only active object (pale-lilac)
Link	Show links too

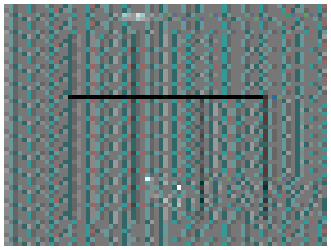


Fig. 3: Visibility of sectors

Play! If you press Esc, play-mode will be ended and the position of the player will be reset. The space bar also ends play mode, but in this case retains the current position of the player.

In the next step we want to test the sector configuration. Use the space bar to add a camera, position it at the site of the *player*. Make the latter the camera's parent by selecting first the camera, then the *player* while at the same time holding down Shift and then the key combination Ctrl+P for *Make Parent*. Rotate the camera so that it's facing the Y-direction (forwards) of the player. Now make this camera active by selecting it and pressing Ctrl+Num+0 (that's 0 on the numeric keypad). Now switch to *TopView* with Num+7. Press P, control the player, and observe the automatic revealing and masking of the sectors, depending on visibility, as in Fig. 3. The player mesh here is in the form of an arrow for the sake of greater clarity.

When handling sectors there are a few important details to be noted:

- Always process sectors in edit mode; scaling and rotation in object mode leads to undesired effects.
- Sectors can and should overlap somewhat, although interlacing should be avoided.
- Do not apply parenting hierarchies of sectors – this often causes strange behaviour in the visibility test.
- The lilac centre point should lie within the bounding box.
- Visibility is computed using the viewfinder of the camera, taking into account the clipping values. Covering areas are not (yet) evaluated; if you want to suppress the visibility of an adjacent sector, a gap between the sectors larger than 0.5 units has to be created.

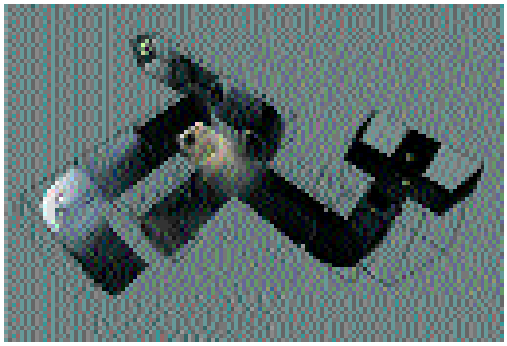


Fig. 4: Simple passage system

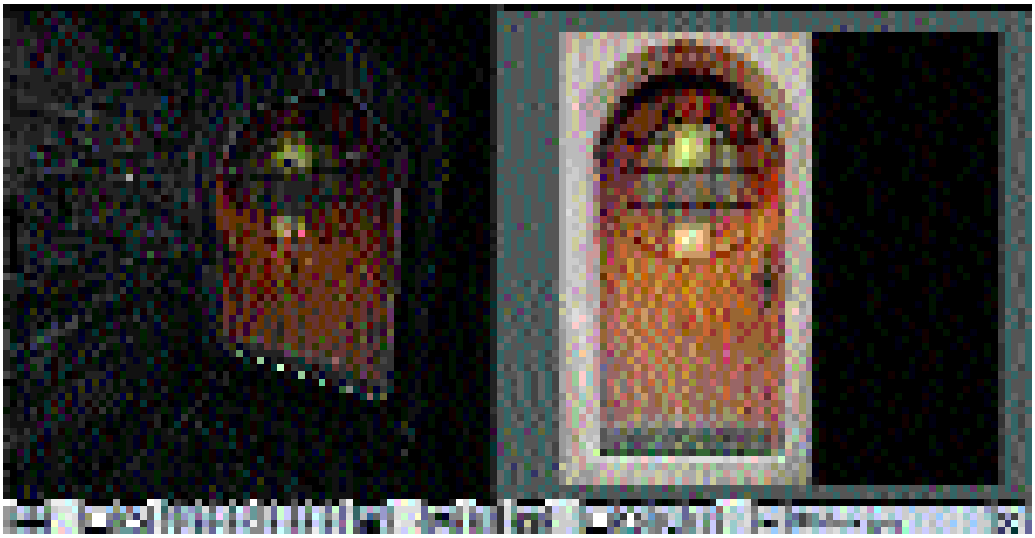
The secret is in the optics – UV mapping

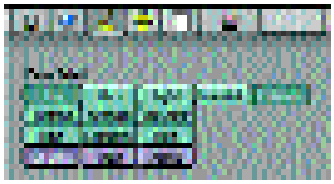
If you switch the current 3D window into camera view using Num+0, our test level looks rather bland – it lacks texture. Nor does our terrain have any sensible boundary. Like the old idea of a flat Earth, we fall off the edge of the world as soon as we leave a marginal sector. For this reason it is best to create a few level elements in order to add a little more spice. The simplest option is labyrinthine systems of passages and underground spaces, which require little planning from the point of view of sector organisation. We'll start with simple passage elements as shown in Figure 4. To do this, download the demofile from *http://www.section5.de/game/demos/*. This contains what you might call a mini-adventure in which a goal has to be reached. You should be careful though, as you could get yourself killed.

Select one of the passage elements and press the / key on the numeric keypad. This switches you to LocalView, which means that only selected objects are displayed. Now press F for Face Select

Table 3: Mapping options	
Cube	cubic mapping
Cylinder	cylindrical mapping
Sphere	spherical mapping
Bounds to 64/128	Use current view for projection, adjust boundaries to 64x64 or 128x128 respectively
Bounds to 128	As above, boundaries 128x128
Standard 64/128/256	Quadratic mapping 64x64/128x128/256x256
From Window	Use current view for projection

Table 4: Draw modes	
Tex	Textured areas
Tiles	Tile image for animated or combined textures
Light	Area uses dynamic lighting
Invisible	Cannot be seen
Collision	Collision detection
Shared	Share vertex colours
Twoside	Double-sided area
ObColor	Use object colour (material)
Halo	Halos, always turned towards the camera
Opaque	Covering texture
Add	Adding texture (halos)
Alpha	Alpha texture (water, pane of glass, etc.)





[top]
Fig. 5: UV mapping (assigning)

[above]
Fig. 6: The paint buttons: Draw modes

[left]
Fig. 7: Image window: tile mode

mode, in which UV texturing is performed. To do this, open a second window with the Image Window (Shift+[F10]). If you select the front door of the level (*entrance*) for instance, the UV mapping looks like Fig. 5.

In Face Select mode, areas are selected with the right mouse button (cross-wire selection via B also works). The associated UV co-ordinates of the areas selected are displayed on the right in the Image Window and can be moved, scaled and rotated with the normal Blender commands. But this mapping is a tedious task, especially with multi-surface objects. Luckily, Blender offers the option of automatic mapping. Position the mouse cursor over the 3D Window (Fig. 6 left) and press U. When you do this, the options described in Table 3 will be offered. For the passage system, cube mapping with a size of 0.60 has been used throughout.

Take note of the red and green edge marking of the active areas. These help in orientation. For the active areas, Draw modes (view attributes) are displayed under the paint buttons (see Fig. 6)

You will normally use options selected in the picture. The ceiling light fitted at the entrance (*lantern*) also uses the *Tiles*, *Twoside* and *Alpha* options, as well as *Halo* and *Add* for the middle areas. If you want to change the Draw Modes of several areas at once, select the desired faces, select Draw Mode and then use *Copy Draw Mode*. This will copy the attributes of the active areas onto the selection. In the same way, *Copy UV+tex* copies the texture plus mapping, and *Copy VertCol* copies the vertex colours.

Areas without any texture assigned are displayed in the texture view in a hideous pink. In this view the direction of the area normals is also relevant, which means the faces are visible only from the outer side (to which the normal is pointing). The direction of the area normals of a fairly uncomplicated or closed object can be oriented automatically by selecting the desired vertices or areas via Ctrl+N (or Shift+Ctrl+N). If this fails, the object might possess overlapping or superfluous areas internally (as can sometimes arise with extrusion).

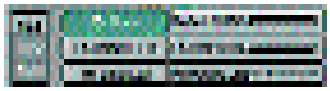
Textures

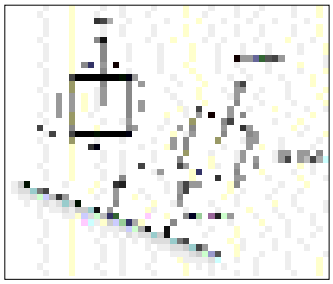
For each area, an image can be selected or loaded into the image window using *Load*. When you do this, note that the dimensions of the texture must correspond to a power of two, for example 128x128 or 512x64. It won't be shown in the texture view otherwise.

In the image window header further options are shown which activate tile mode for animation (see Fig. 7). Subdivision is done with the left number buttons. For an animated texture, which can also be put together rather elegantly from a sequence using the *montage* ImageMagick tool, activate *Anim* and set start and end frames with the number buttons on the right. The *Cycle* option has no effect for the moment. The partial image in the activated tile mode is selected by holding down Shift then clicking the left mouse button on the image. For *tiled textures*, such as the torch texture in our mini-adventure, the rule is that the dimension of the subdivision in tile mode (see Draw Modes) must correspond to a power of two, but not the total dimension. For a 5x5 subdivision, then, with a tile size of 32x32 this therefore gives the total quadratic dimension of the texture image of 5 x 32 = 160.

A bit of dynamics – the physics engine

The current physics engine evaluates collisions of actor-objects with areas over a sphere with a radius *Size* (see actor attributes in Table 1). In the case of actors such as deformed Kraken monsters, this naturally leads to a problem when it comes to collision detection. The solution to this will have to wait until Blender 2.1 (with its improved engine) appears. If we limit ourselves to a first-person shoot-'em-up there is no need for us to worry too much

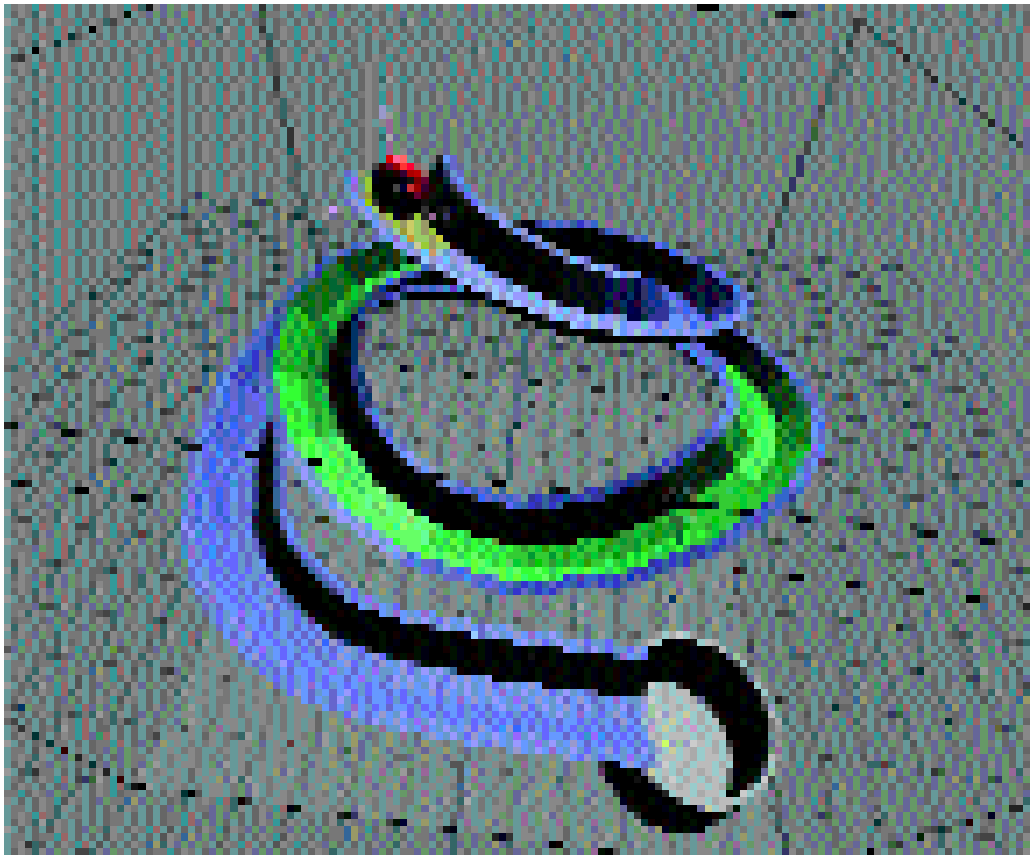




[top]
Fig. 8: Dynamic materials

[above]
Fig. 9: Normal forces to actors

[right]
Fig. 10: Bob run with various materials



about our own appearance. A spherical shape will suffice for simple prototyping.
What is interesting though, is the ground and the movement of the player. In our first test for the sectors you will certainly have been bemoaning the all-too smooth movement. The reason for this lies in the fact that the value for *Damp* or *RotDamp* may be too low – which applies globally for the associated actor (see Table 1). If you increase the damp values, forward and turning movements are braked more quickly regardless of the ground. Another globally-applicable value can be found under the World Buttons: Gravitational acceleration, with the standard value of 9.81 m/s≤ is the default.

But by experimenting with various masses you will find a fault in the engine: Heavy bodies fall faster than light ones. This is due to the fact that Newtonian physics ($f = m * a$) has not been correctly implemented.

Ground material

Each surface can be assigned a so-called dynamic material: If you use the Material button ([F5]), as well as the *RGB / HSV* colour choices, you'll also find a *DYN* option (see Fig. 8). Selecting this will reveal sliders for various parameters, whose functions we'll explain in a moment. But first here's a useful snippet of information on physics on an oblique

Table 5: Overview of the parameters of dynamic materials

Fh Norm	Fh in normals direction of the ground area
Reflect	Reflection/elasticity of the ground
Fh Dist	Distance of the "soft" or elastic surface from the actual area
Fh Damp	Damping of the soft surface (elasticity)
Fh Frict	Friction components parallel to the surface
Fh Force	Resistance force of the elastic surface

Table 6: Actuator features

object	Apply force etc to object
Constraint	Restrict place/orientation to one area
Ipo	Play Ipo animation
Camera	External camera flying alongside
Sound	Play sound unfortunately does not yet function under Linux
Property	Change user-defined attribute of an object
Edit Object	Change, add, remove or track objects (tracking)
Scene	Change scene, restart or change camera

plane: if a body is on a slope, it will obviously slide downwards (provided static friction is overcome). In Blender, unfortunately, static friction does not exist yet, so it is only possible to work with sliding friction, which is usually dependent on the rate of slide. We are not going to explain in detail how downward movement comes about, as we hope you remember some physics lessons. However, the key here is normal force (the force which exerts resistance to the gravitational force of a body on a surface). In Blender this resistance force is referred to as *Fh*. To make an elastic surface possible, this force is effective from a certain distance, namely *Fh Dist*, from the surface. If this distance between actor and plane reduces, a force of elasticity also acts (*Fh Force*). Finally, with *Fh Damp* the damping of the elastic resilient movement is controlled. Regardless of this, using *Reflect*, a hard elasticity of the surface at *Fh Dist* = 0.0 can be set.

Often sliding on a slope is not desirable, especially in an adventure game with a walking player. For this reason the resistance force *Fh* can be laid in the Z direction by switching off the *Fh Norm* option, which cuts out the downwards movement – see Fig. 9, left. In the case on the right, the *Fh Rot* option for the actor has been activated, whereby it orients itself to the plane.

You'll find another demofile, *bob.blend*, at *http://www.section5.de/game/demos/*. This shows a bobsled run. It has been assigned several different materials, each with different friction forces *Fh Frict* – see the shaded image (by Z) in Fig. 10: yellow means *Fh Norm* is switched off, high *Fh Frict*. Green means Fh Norm on, low *Fh Frict* and blue means average *Fh Frict*. Play around a bit with the dynamics. Also try switching into camera view with Num+0 and show textures by using Alt+Z.

Interactivity – GamePlay

Let's get back to our adventure. Perhaps you have already discovered the switch in one of the back passages, or even solved the puzzle already. As well as the simple, pre-applied SCA actions such as moving objects by forces, animations and other things can be played back using an IPO curve. You will find additional features in the short overview in Table 6.

Let's examine the example of a switch in the adventure. When the player stands close enough to the switch and presses the "operate" key, an animation of the switch should be played and the action should be triggered (in this case the light switching on). To do this, you must be able to assign status to an object. This is achieved by means of self-defined property attributes. With the real time buttons ([F8]) in Fig. 11, a self-defined attribute can be added using *ADD property*; enter type, name and initial value in the corresponding fields. Property-attributes are also used to define certain object classes, for example *bad*, *good*, etc.

Let's look at the switch in more detail: Select the

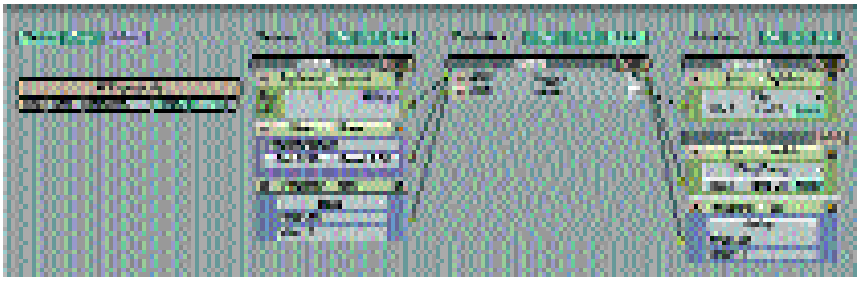


Fig. 11: Switch concatenation

switch object – this gives an SCA combination as in Fig. 11. Note the property attribute called *on*. The D ("debug") means that in wireframe mode the value is displayed in the 3D window.

Let's follow the signal paths: When E is pressed and the *Near* condition is met, an Ipo animation is played with the option *PingPong*, which means that if tripped again the switch will reset itself. The property attribute *on* is also set to the value 1. This activates an animation for the lamp *Lamp.004*.

Switching on lamps unfortunately does not yet occur via Energy-IpoCurve. The near check functions as follows: If an object with the property attribute *player* comes nearer than a distance of *Dist* (2.20) this condition is met. If it moves away further than the *Reset* distance (2.30) the condition is reset. Check to see if the *player* really does have the property *player*, and test the near check in the wire mesh view. When the near event is triggered, the switch lights up in blue. The switch object can be more easily achieved with *Collision-Sensor*.

When playing our mini-adventure, you will already have discovered which objects have interactivity, and how the portcullis can be opened. Take a look at the associated SCAs at your leisure.

Let there be light

The correct lighting of a scene is really the key to a good atmosphere. This is difficult to achieve within the game engine, since as yet not all light sources react exactly as they do in the rendering part. Also, you have to activate the *Light* option for each area, which might have the effect of slowing down the view. So, if dynamic lighting is not absolutely necessary, lighting conditions can be simulated by vertex colours. To do this, position the light sources as usual, select the desired mesh object and, under the edit buttons, use *Vertcol Make*. This will transfer the lighting conditions as vertex colours onto the object, which can be checked in the vertex paint mode (V). This means that additional shade effects or colourings can be added by reprocessing with the paintbrush (see also paint buttons). Reprocessing in vertex paint mode is usually necessary to make the colour shadings between the areas slighlty more subtle. Blender pros might also like to make use of the radiosity method to generate a realistic lighting model. But if you do, bear in mind that existing UV-texturing gets lost during the radiosity process in the current

version (2.04) of Blender. Also, if the areas have too high a level of subdivision in the radiosity resolution, a level may well look optically perfect, but from the point of view of speed it can easily become unplayable.

For dynamic lighting you must, as explained above, *activate* the light option for the corresponding areas. The rule here is that only lamps that are in the same layer as the mesh object contribute to the lighting. But previously only lamps of the type *Lamp*, *Spot* and *Sun* functioned as desired. In the case of the *Lamp* type, the sphere option has no effect yet, but better light attenuation can be achieved by increasing the value for *Quad1*, which controls the, physically more correct, quadratic attenuation of the intensity of a point light source.

Tips and tricks

Whew! If you've followed us this far you must be eager to create your own level by now. Before you do though, keep the following tips in mind:

- **Number of polygons**: Always keep the number of polygons as low as possible. For characters in particular you should not need more than 500 faces. The number of vertices or faces of an object or a scene can be read from the status line (normally top right in the window), e.g. Ve: 287, Fa: 458. For an object, switch into LocalView using Num-[/]. If you want to risk a more complex game, subdivide the sector groups into scenes, which should not contain more than 5000-8000 polygons in total (depending on the graphics card used).
- **Modelling**: You can safely model and texture a level as a whole within an object, and later split it into individual objects in edit mode with P – the texturing will not get lost. The same applies for merging objects using Ctrl+J. What counts most when modelling is the direction of the area normals for collision and visibility, otherwise your player might fall through the floor.
- **Texture**: Keep the textures in as small a format as possible. Texture-Mipmapping (automatic scaling/filtering) will be used only in later versions.
- **Clipping/Popping**: If your game is played in an outdoor environment (countryside, motor racing, etc.), an unwanted side-effect called popping (a sector appearing suddenly) might occur depending on the camera clipping used. Eliminate this by creating some *Mist* under the world buttons. To do this, set the initial mist value *Sta* a bit lower than the clipping value set using *ClipSta* for the camera (the latter can be found among the edit buttons). *Dist* may also be fairly small.

In future Blender versions, you can expect a few new features such as automatic sector optimisation, better visibility tests and a bit more automation, especially for simulation of lighting conditions. The latest news can of course be found on the Blender Web site. Of course, you can bet we'll report on any significant developments here. ■

The author

Martin Strubel has been an enthusiastic Blender fan for two years and is currently developing games environments for NaN.

Important key combinations	
RealTime	
P	Play mode, start game
Esc	Stop Play mode, reset positions
Spacebar	(within Play mode) As above, but retain positions
View	
Num+0	Camera-view
Ctrl+Num+0	Activate selected camera
Z	Toggle wire mesh/ area view
Shift+Z	As above, with shaded area view
Alt+Z	Textured view
Modelling (Edit mode)	
B, B twice	Cross-wires-/circle selector
E	Extrude
P	Split selection as object (separate)
Ctrl	Snap vertices onto grid or gradually scale/rotate
Ctrl+N	Reorient normals outwards (Shift = inwards)
General object processing	
G / S / R	Displace (grab), scale, rotate
Shift+D	copy object or vertex
Alt+D	Linked copy: copy object linked
Ctrl+J	Merge objects (join)
Ctrl+P	Make Parent
Texturing, colours	
F	FaceSelect mode
U	Automatic UV-Mapping
V	VertexPaint
Shift+K	colour whole object with current vertex colour