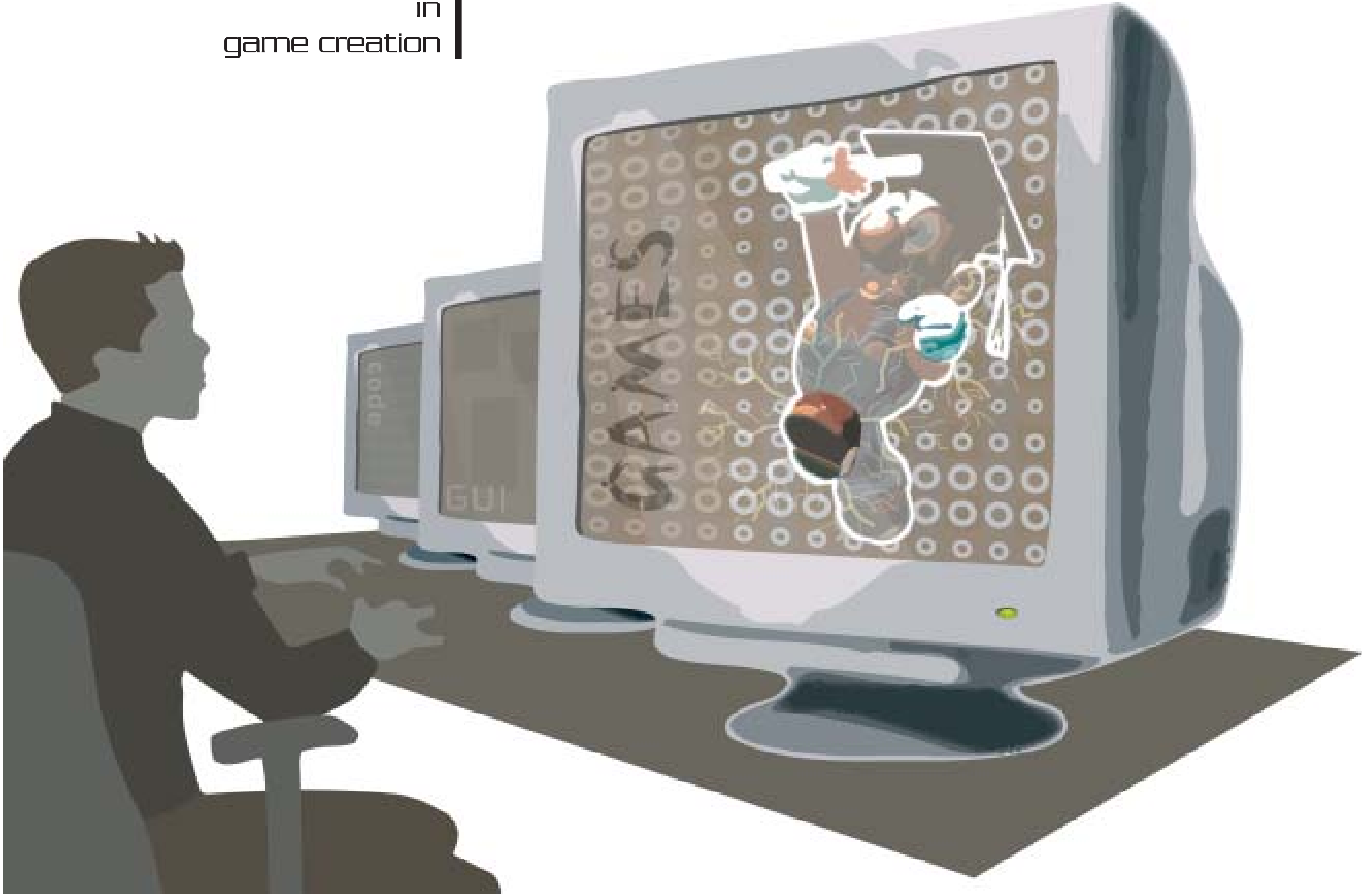
A
Graphical User Interface (GUI)
for
designing interactivity
in
game creation

Jonathan van Wunnik



September 2001 - May 2002

TU team:
dr. P.J. Stappers (UHD)
ir. A. Hoeben
ir. J.C. Verlinden

Company mentor:
dr. ir. M.W. Gribnau

Company:
NaN Holding BV

Summary

The subject of this graduate report is the interactive creation part of the Blender application. Blender, developed by Not a Number BV (NaN), is an application with which users can model, animate and do postproduction of 3D content. NaN has two offices. One in Amsterdam and one in Eindhoven. In Amsterdam, the headquarters are located while the development and content teams are located in Eindhoven. The development team creates the Blender application while the content team creates actual content with Blender for promotional purpose and gives feedback to the development team.

In this project a new Graphical User Interface (GUI) for the interactive creation part of Blender was designed. The interactive creation part allows the user to add, in a visually manner, 'life' (interactivity) to 3D objects and worlds.
The new GUI had to provide means for organizing the networks of logic bricks (the building blocks for creating interactivity graphically), but also means for creating interactivity on a 'higher' level than the current logic bricks concept. Higher level means interactivity such as state engines, behavior and/or artificial intelligence (AI).

To become familiar with the interactive creation part of Blender an interactive 'game' was designed and built within Blender. Different types of 3D interactive content, the design process of the content team of NaN and other game development teams were analyzed. A game development process was extracted and the specifications for a new GUI were determined. A list of criteria was set up to use as guide for the concept creation.

After the analysis phase, the conceptual phase started where (partial) solutions were created for the problems that were found. This resulted in a final concept that was evaluated by two members of the content creation team of NaN. The evaluation showed that the design solves the problems determined and fullfills the demands and wishes.

The results were applied in a final design that offers a GUI for designing interactivity on the level of behavior without (less) coding and a means of organizing all logic used.
The final design can be used as a blueprint for actual implementation in Blender. Although the design has been evaluated with the content team a few times, it is inevitable that modifications will need to take place during and after actual implementation. Besides, valuable insights for future additions to the logic system came up during the process. These include debugging, ordering of states, which kind of logic bricks are (really) needed etc.

Now the question is, will the design be implemented in Blender? This is at the moment of writing uncertain (NaN went bankrupt halfway through the project). Currently NaN is undergoing a re-organization. The prospects are that one or two smaller companies will continue developing and using Blender one way or another. One of those would benefit picking up this project and implementing it.

T.O.C.

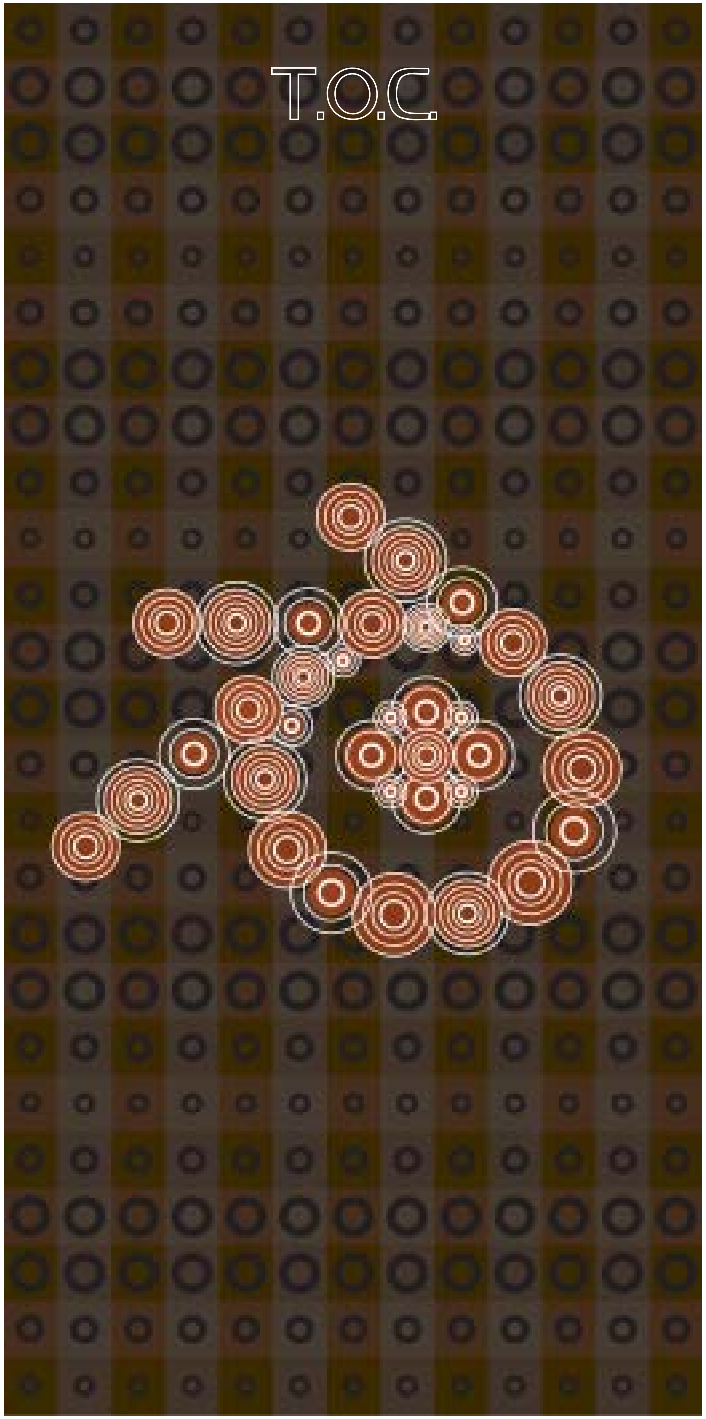


Table of Contents

I Analysis

II Synthesis

1. Introduction

The starting points for this project existed of two main questions. What should a graphical user inteface for creating higher level interactivity (e.g. state engines, behavior and/or artificial intelligence) look like? And in what manner can logic (the building blocks for creating interactivity) be organized?

The process is divided into three main parts: the analysis, synthesis and optimization.

First a company profile is given in section 2. In section 3 the problem definition and assignment are described, before the actual analysis.

The analysis started with building an interactive 3D game within Blender to get familiar with the current interactive creation tools and to get an overview of Blender itself. This is described in sections 4 and 5. Secondly, a survey of different types of 3D interactive content was made, described in section 6. To get an overview of how the content creation team of NaN and other game creation studios do work and which problems they come along, a analysis of these teams was made in section 7 linked together with the development process in sections 8. To define what kind of 3D interactive content should be created within Blender, example games are defined and the 'levels' of interactivity are described in section 9.
In section 10 Blender is compared with other 3D interactive creation applications to obtain information about how these applications make it possible to create interactive content. Also applications that are not directly used for the creation of interactive content, but do have a graphical interface for the workflow of connecting different nodes (resembling connecting logic bricks) are described to explore different design solutions. Section 11 shows the 'look-and-feel' of other graphical user interfaces. What criteria should be of interest for the GUI for creating interactivity? All the information obtained finally resulted in a program of requirements, described in section 12, which the final design has to fulfil.

The synthesis, the second part of the process, existed of bringing all the information, obtained during the analysis, down to a solution for the two starting points mentioned above.
The synthesis started with the creation of ideas, described in section 13. The best parts of these early ideas were mixed and put together concluding into the first concept. In total three concepts were created. The three concepts were not three completely different concepts, as usual within the methodology of the design process. Instead, it was a more iterative process, the second concept continued on the first and the third on the second. Every concept was discussed with some members of the content creation team leading to the next adjusted concept. These concepts are described in section 14. Finally this process of optimizing lead to the 'final concept'. The final concept is described in section 15.

The third part of the process, the optimization, covered the evaluation of the final design. The approach, findings and conclusions regarding the evaluation can be found in section 16. In section 17 the conclusions of the evaluation and the 'look-and-feel' of a GUI (section 11) lead to a 'final design'. This final design describes and shows in short all the aspects of the GUI as designed for creating interactive content.
In section 18 recommendations and future developments are described and finally, in section 19, a process (project) evaluation is given.

2. Company Profile Not a Number [NaN]

NaN Technologies B.V. (Not a Number) is a technology company establishing new standards in real time 3D content creation, playback and delivery of real time 3D for use across all networks and devices. Ton Roosendaal, Creative and Technical Director, founded NaN in June of 1998 to further develop and market Blender Software and its underlying technologies.

At SIGGRAPH 2000 in New Orleans, NaN unveiled Game Blender 2.0, which enables the creation and playback of real-time interactive 3D content such as computer games and product presentations. Blender is designed around a solid-body dynamic simulation, in which forces such as gravity, impacts from weapons, character interactions and collision detection are handled transparently by the software. Traditionally, artists team up with programmers to add interactivity to the product. This is a slow process, especially since reuse of existing software (previous code) is difficult. Therefore, in most productions, interactivity has to be created from scratch. Contrary to this, Blender offers support for design and prototyping interactive behaviour of content in a production. The result is a significant faster development process.

3. Problem definition and assignment

Problem definition

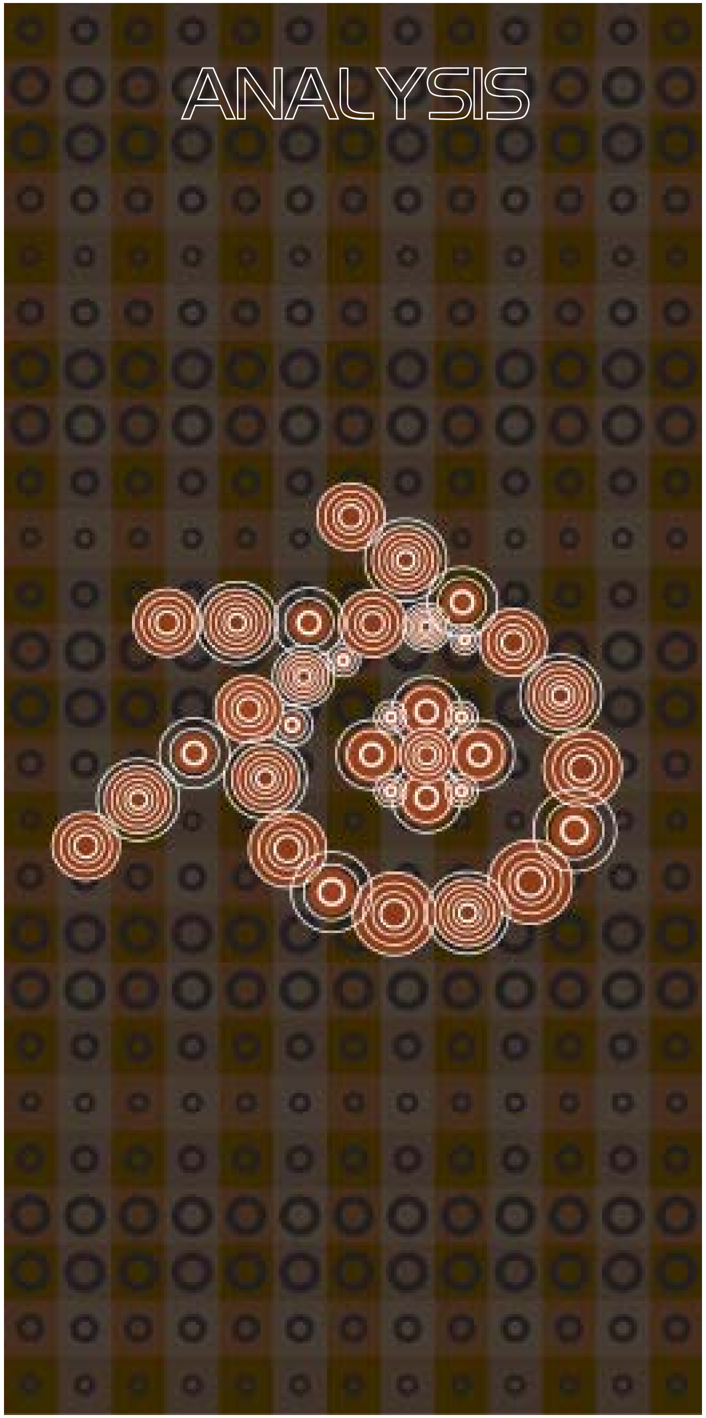
The Graphical User Interface (GUI) for the design of interactivity in Blender is based on a technique (logic bricks), which provides artists a way to create behaviour. To achieve this behaviour, artists have to connect these logic bricks (sensors, controllers and actuators). The current GUI works fine for small projects. When the project grows however, logic brick networks can become complex and confusing. Artists do not have a means to organize the networks now.

There is need to design behaviour on a higher level than the logic bricks technique. The integrated Python scripting language is now the only means to create interactive behaviour with the environment on a higher-level. NaN is investigating ways to ease the design process of creating interactivity. The plan is to create a new GUI for the design and development of higher-level interactive behaviour.

Assignement

Design, prototype and test a new Graphical User Interface (GUI) for creating an interesting interactive production (game) without or with less intervention of programmers (Python scripting). The new GUI must be easy in use, especially for new users to Blender as well as for experienced users. Hereby the new GUI should provide means for organizing networks of logic bricks, but also interactive behaviour on a higher level than the logic bricks concept (e.g. states, behaviour and/or artificial intelligence).

ANALYSIS



4. Using Blender for interactive 3D content

4.1 Introducing Blender (history)

Blender is a 3D content creation package, as told in the introduction. It allows modeling, animation, rendering, postproduction, interactive 3D creation and playback.
The interactive 3D creation is not always been a part of Blender. Originally Blender was an in-house tool of the Dutch animation company NeoGeo. When this company stopped existing early 1998, a SGI version of Blender was posted on the web, just for fun. The success and attention it got and also the porting to other operating systems like Linux and FreeBSD, made Ton Roosendaal decide to found NaN halfway 1998.
But to survive and to get attention in the 3D market, it was decided to focus on special key-features to distinguish from other 3D packages. Because of the neglected arena of realtime 3D in other 3D packages, NaN saw an opportunity in extending Blender with a realtime creation part. With the later extension of a webplugin it is now also possible to create and publish interactive 3D content like games and 3D presentations for the web. Therefore, the main point of focus will be more and more on this part of Blender.

4.2 Functionality overview

What will follow now is an overview of the functionality (key features) of Blender. This will be a short description of all the key features offered within Blender. For a official list of features, see appendix 1.

Modeling:
Modeling includes the following types: polygon meshes, curves, NURBS (limited), metaballs and vector fonts.

Deformations:
There are two forms of deformations. Lattices deformations (a lattice consists of a three-dimensional grid of vertices and when moving one of these vertices will cause a deformation of the child object it is assigned to) and bones deformations (used to deform a character during a walk animation, for example).
Animation:
The following kinds of animations are possible within Blender: key frames, motion curves, morphing, inverse/forward kinematics (character animation) and path animation.

Particle systems:
Particles can be used to generate fire, smoke, clouds etc. Every mesh-object can serve as an emitter for particles and every mesh-objects can be used as a particle.

Rendering:
The render engine within Blender supports among other things: solids, transparency, halo/lens-flare effects and radiosity.

Postproduction:
Blender supports also sequence editing of images and postproduction effects with the build in editor.

Real-time 3D:
An engine for playback of real-time 3D is integrated within Blender itself. With this it is possible to playback interactive creations on the fly, without compiling or preprocessing.

Game creation:
The ‘game creation’ part of Blender allows the user to create interactive content within Blender itself without or with less programming. The ‘game engine’ supports the following: collision detection and dynamics simulation, 3D audio, multi-layering of scenes for overlay interfaces and animation features are also supported within the ‘game’ engine.

Embedded scripting language:
Blender has an embedded scripting language: Python. With the addition of Python it is possible to add advanced control of game logic.

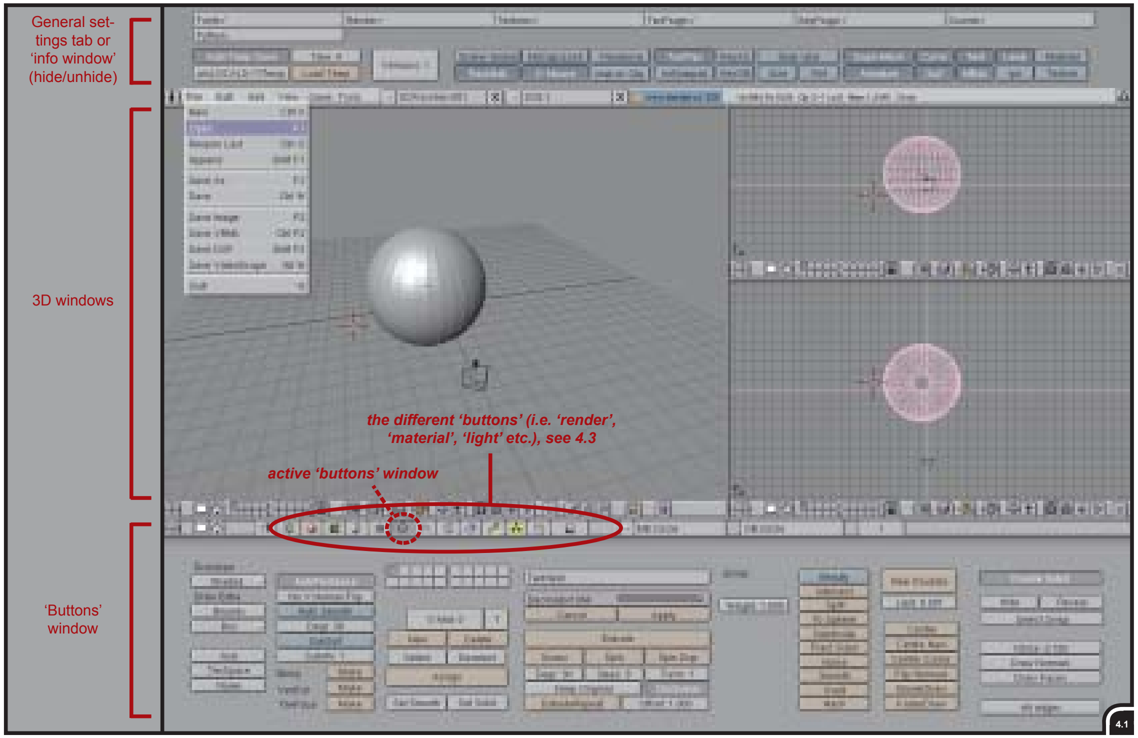

General settings tab or ‘info window’ (hide/unhide)
3D windows
the different ‘buttons’ (i.e. ‘render’, ‘material’, ‘light’ etc.), see 4.3
active ‘buttons’ window
‘Buttons’ window
4.1

fig. 4.1
The full interface of Blender.

4.3 The user interface

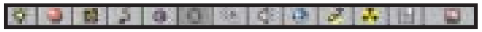
As can be seen in figure 4.1 (previous page), Blender is divided into three parts. From top to bottom: the 'info window', the '3D window(s)' and the 'button window(s)'.

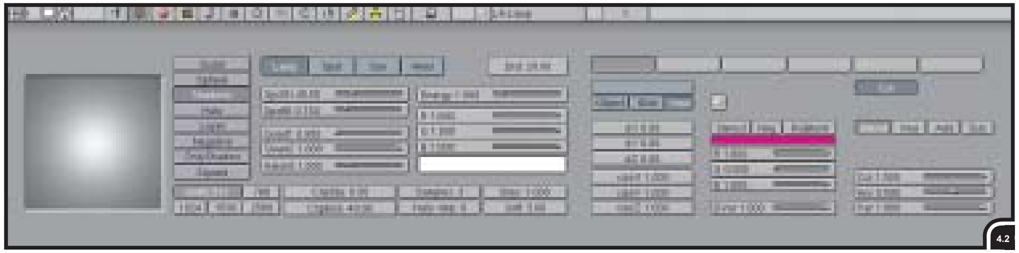
The info window is normally 'hidden' at the top of screen and can be revealed by dragging the menu bar down. The info window shows general settings of Blender. In here you can tell Blender where to store autosave files or activate the tool tips function for instance.

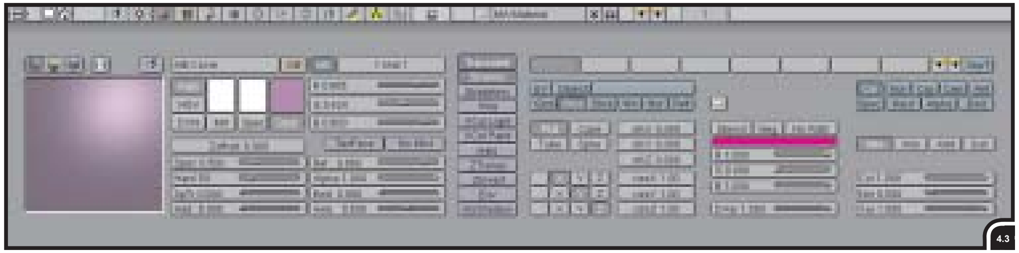
The '3D window' compares to 3D windows in many other 3D packages. In this window objects can be rearranged and edited, animations can be defined and lights and cameras can be added. Many views can be added. These can be perspective or orthogonal (top, front and side) views.
Apart from the '3D views', you can also open the following 'windows' in any 3D window:

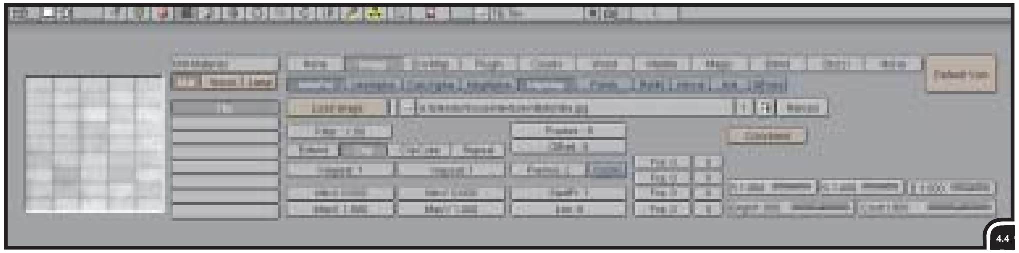
- *The file window:*
  This window provides the interface for loading and saving files.
- *The image (select) window:*
  This window is for selecting and loading images. These images are provided as thumbnails and it is used for loading image textures maps, for instance.
- *The IPO window*
  In this window you can edit the time-lines that are associated to different object properties, like their position and rotation (IPO stands for interpolation).
- *The text edit window:*
  This window is mainly used for writing Python scripts, but it can also be used to keep notes of animation projects for instance. The advantage of using the text edit window is that the notes are stored inside the Blender project file.

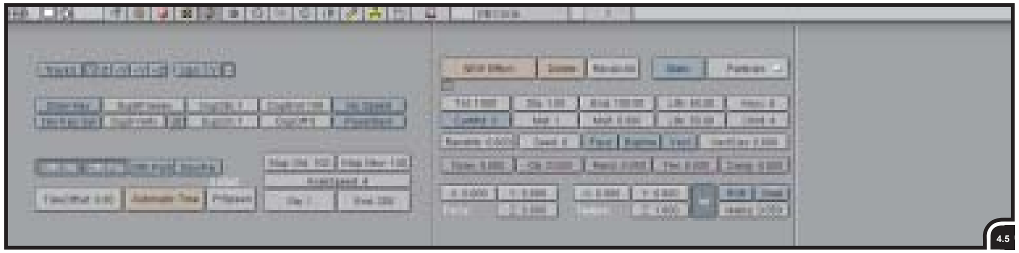
Lastly, at the bottom of the screen is the 'buttons window'. This name can be confusing. The buttons window is strictly speaking a collection of buttons, but basically it is the place in Blender where you can edit, assign and build different aspects of a project. These different aspects are provided in a 'tab' like way. You can switch between them by clicking on the associated button. There are thirteen different 'button windows' (see figure below). What will follow now is a description of every 'buttons window' from the left to the right. The corresponding content of each window is shown in figures 4.2 to 4.14.

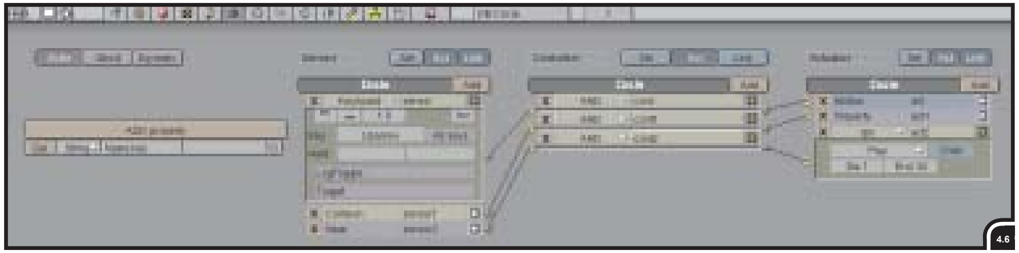



4.2


4.3


4.4


4.5


4.6

- *The lamp buttons (fig. 4.2, previous page):*
  This window allows to change all of the parameters of lamps, like their color, energy, type (regular lamp, spotlight, sun or hemi) and quality of shadows.

- *The material buttons (fig. 4.3, previous page):*
  This window allows the control over properties as object color, shininess, transparency, textures, texture types and texture projection methods.

- *The texture buttons (fig. 4.4, previous page):*
  This window allows the assignment of texture maps to a material. You can choose out of many different texture methods, like a image, clouds, wood, marble, stucci and noise for example.

- *The animation buttons (fig. 4.5, previous page):*
  This window allows to set properties for things like curve following, automatic object duplication and object tracking. It also allows object effects like particle systems and wave effects.

- *The realtime buttons (fig. 4.6, previous page):*
  This window makes it possible to build interactivity into a project. This part of the inteface will be the subject of this project. In section 5.1 this window will be explained in more detail.

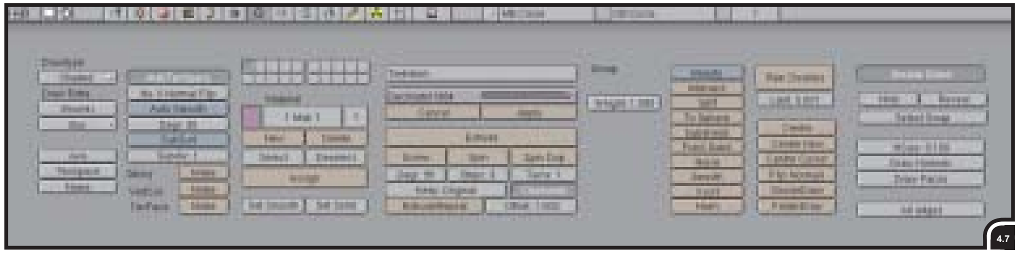
The name 'realtime buttons' is maybe a little bit confusing. It suggests more the 'speed of calculation' than what it really stands for, namely building interactivity. Therefore I will name this part of the interface the ***'interactivity window'***.

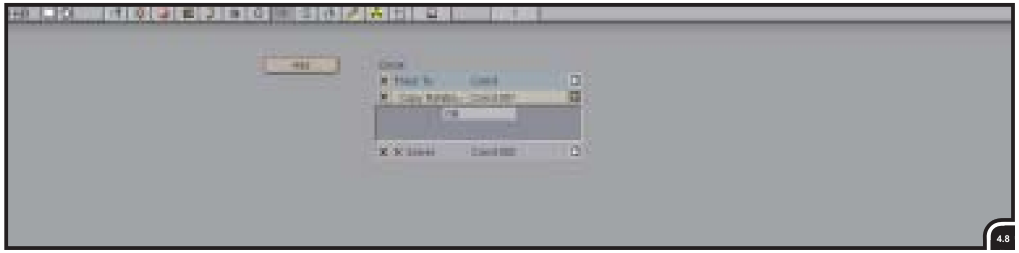
- *The edit buttons (fig. 4.7):*
  This window allows to edit objects in your scene. The buttons that are shown depend on the kind of object that you have currently selected.

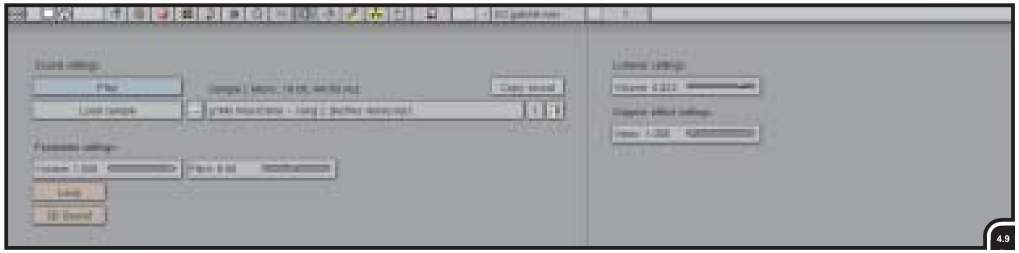
- *The constraint buttons (fig. 4.8):*
  This window allows to add constraints to animations and IK (Inverse Kinematics) skeletons.

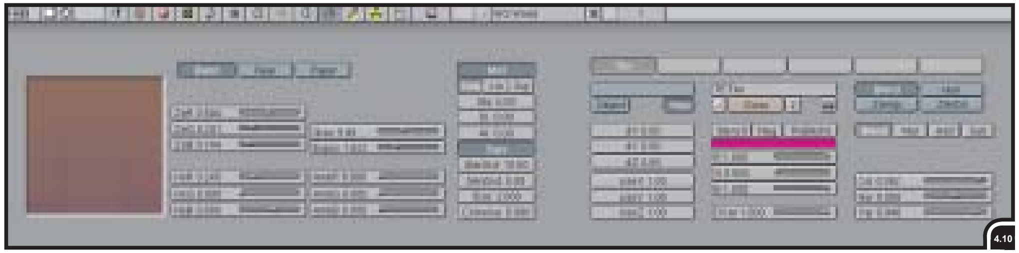
- *The sound buttons (fig. 4.9):*
  This window allows to add sound to a project. The sounds can be assigned to interactive objects. For example, a creaky sound when a door goes open.

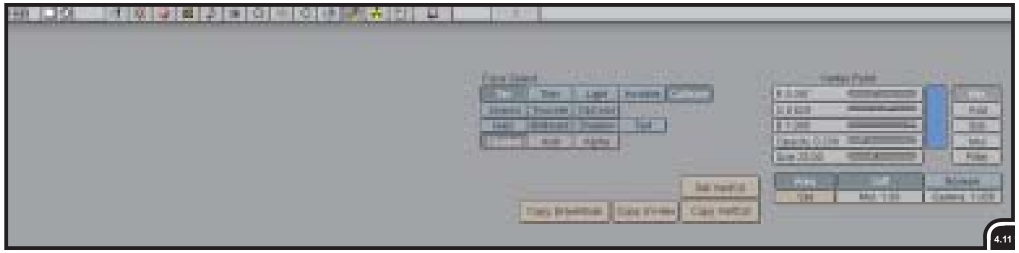
- *The world buttons (fig. 4.10):*
  This window allows to set up horizon colors, ambient lightning, fog effects and starfields.
- *The paint buttons (fig. 4.11, previous page):*


4.7


4.8


4.9


4.10


4.11

This window allows to add or paint a color onto vertices. Besides this, it also allows to take into account if a object will be visible, lighten or will be used for collision in the game engine or not.

- *The radiosity buttons (fig. 4.12):*
  This window allows to produce 'realistic' 3D images. Radiosity is based on radiative heat transfer. In real world, colors of objects are not only affected by direct lighting, but it is also affected by lights bounced off from other objects. This window makes it possible to achieve this effect.

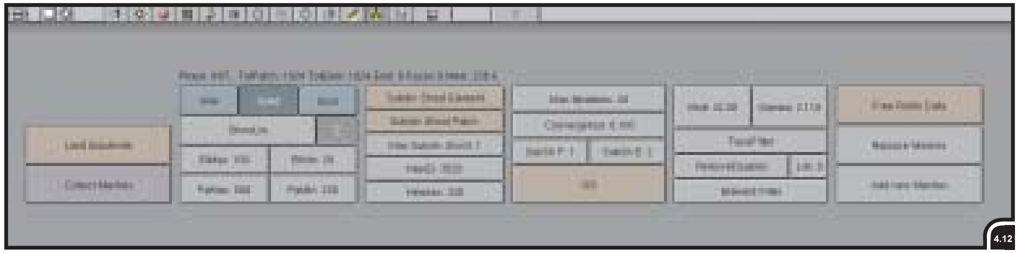
- *The script buttons (fig. 4.13):*
  This window allows to call up scripts. This can be done in three different ways: on 'frame changed', on 'redraw' and on 'onload'.

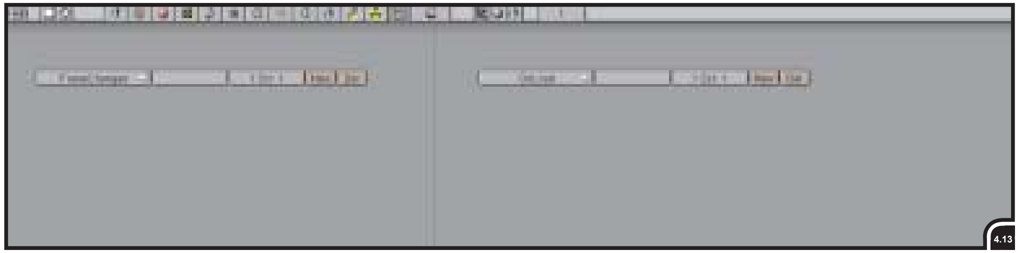
- The display buttons (fig. 4.14):
  This window allows to control the way Blender will render images. In here the size and render quality can be set. With the 'render' button, a still image

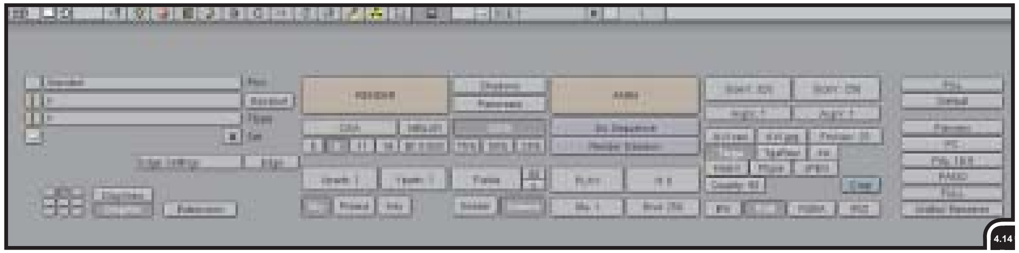
These are all the 'button windows' available within Blender. With all these windows the modeling, texturing, animations, interactivity etc. is done.

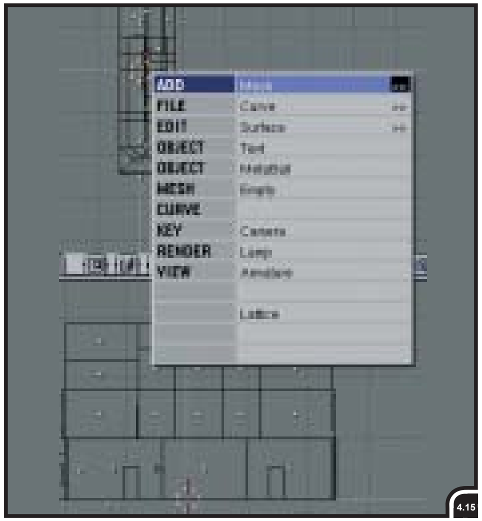
Finally, at the top left of the screen there are the menus. These menus are:
- *File*: standard things like open, save, save as, save image etc.
- *Edit*: things like duplicate, delete, edit mode, rotate, scale etc.
- *Add*: with this menu it is possible to add geometry and objects to a scenes, like add mesh, add curve, add surface, text, camera, lamp etc.
- *View*: this menu let makes it possible to switch the selected view to front, right, top and camera.
- *Game*: this menu is specific for interactive productions within Blender, it can switch on or of the following settings: "enable all frames", "disable sounds", "show framerate and profile" and "show debug properties".
- *Tools*: this menu makes it possible to pack or unpack all external data used (textures, sounds etc.) into one file. Very useful for distribution of files.

The menus file, add and edit (with most of the other function within Blender) duplicated in the toolbox. The toolbox will appear when the spacebar is pressed (see fig. 4.15). A function within this toolbox is selected by clicking it with the mouse. Working with the toolbox can speed up the workflow within Blender.


4.12


4.13


4.14



4.15

5. A small interactive 3D production with Blender

The goal of building a small interactive production was twofold. First of all, it was done to learn the basic features of Blender. How long did it take to learn the different parts of making the interactive 3D production? Second, the steps (sequences) were used in combination with the information obtained from other game development teams (see section 7) to extract a 'user process' of building an interactive 3D production.

5.1 The house

What follows now is a description of building my own interactive 3D production. In section 8.1, a more general game design process will be described.

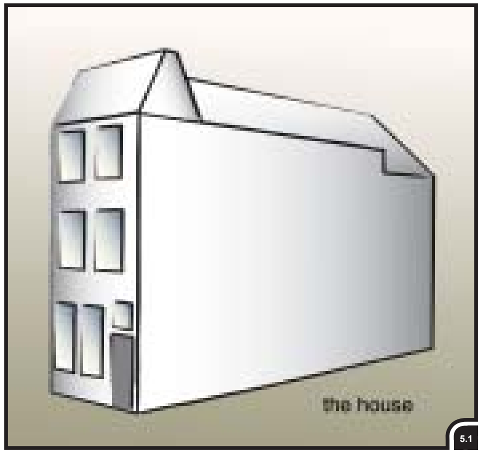
Idea generation (basic concept)

I have chosen to build my own house. It is a big student's house with many rooms. The player can walk around, look around (first person perspective) and open doors after he finds the key for it. It is not a game in the meaning of shooting opponents, but solving a puzzle. There is only the house itself where the character can walk around. The house exists of three floors with seven main rooms and seven smaller rooms. The player is not able to walk outside the house.

Rules, content and behaviour description

The rules are basic. The player has to find all the keys to open the doors. The keys can only be found in one order; the player has to find the key for Anna's room first before he can enter Sandra's room, because the key for Sandra's room is located in Anna's room, for example. Some keys will be hidden in the (already) open hallways and others in the rooms to be opened. After the player has found all the keys and has entered the last room (the cellar) the 'game' is finished. When the player enters this last room (the door to the cellar), he will fall in a deep hole and the message "you pass the test and you are now ready to live in our house" will be shown.

A sketch of the interaction diagram (see fig. 5.5, next page) shows all the related game elements and their connections.



5.1

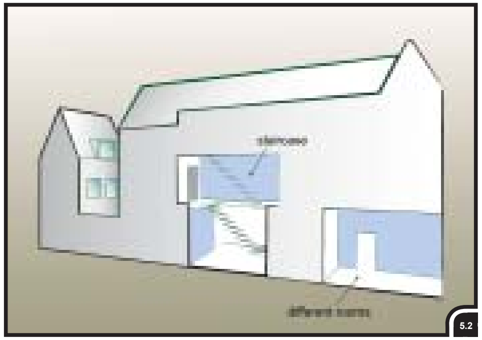
Description of main character and game pieces (assets)

Main character: This is the 'character' the player will move around in the house. The character wants to find a room in Delft, because he is going to study at the Technical University.

Game pieces (assets): These are all the objects that will interact (dynamically) with the player (main character). All the objects that are not static (rooms, environment, buildings etc.). In this case only the keys and the doors are game pieces (assets). The keys will be floating in the air and are placed in different rooms. When the player finds a key, the key will disappear and an overlay picture of the key will appear. The other game pieces, the doors, go open when the player is close enough, provided that the player has the key for the door.

Graphical and sound description (the design 'Bible')

All the rooms need a different look. There will be wooden floors, colored walls and a neutral ceiling (see fig. 5.3). All surfaces will get their 'look' with textures (pictures). Outside the house a sky will be visible. The atmosphere must be a little dark.



5.2

neutral ceiling
neutral coloured walls
wooden floors

5.3

There will be sounds for opening doors (cracking and creaking), picking up a key (bleeping), falling on the ground in the deep hole (crunching) and a continuous sound in the background (sinister).

Building the house

During the process of building the house some difficulties arose. Building the house for the first time, the amount of polygons was too high to get an acceptable frame rate. The house was build again with less polygons.

Lighting was another difficulty. To give each room its own sense of atmosphere, every room had to be on a separate 'layer', because within the game engine light go trough walls and will lit also everything behind it. It was very difficult to set this up properly.

Making it interactive

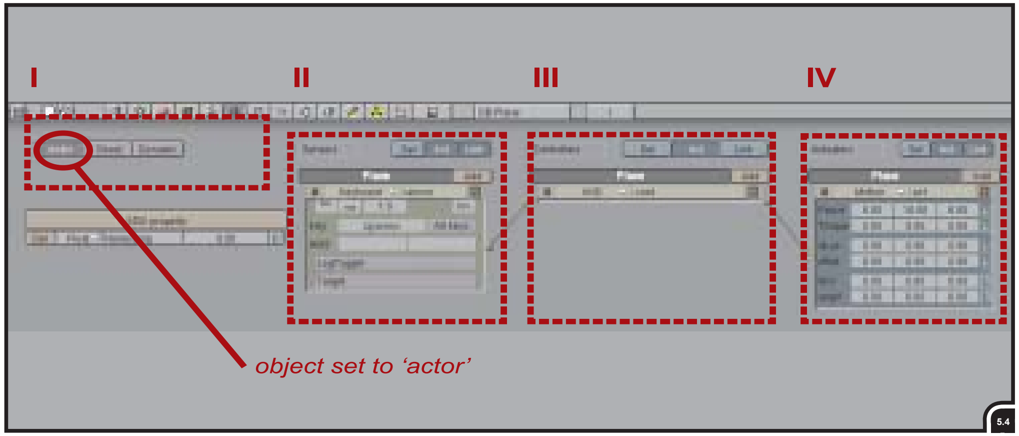
To make the components (objects) interactive, the 'interactive window' was used. A description of the 'interactive window' will be given instead of describing the interaction built particular for the house.

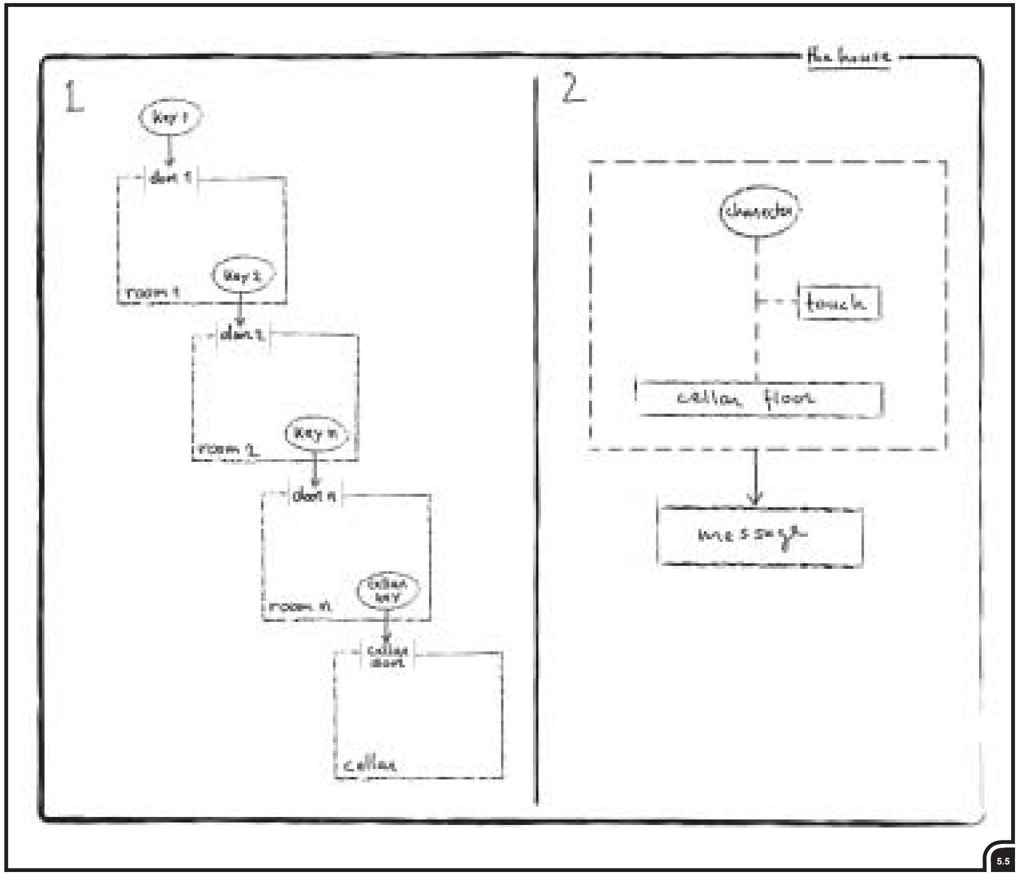
The 'interactive window' is divided in four columns (see fig. 5.4, on next page): the attributes (I), the sensors (II), the controllers (III) and the actuators (IV).

With the attributes it is possible to set an object to be part of the physics engine. To do so the object has to become an 'actor' (see fig. 5.5, next page). Once set to 'actor' the physics engine will evaluate the object and will do collision detection. A ball will bounce when dropped on a floor, for instance. But when the ball also has to obey the laws of physics, the object must not only be set to 'actor' but also to 'dynamic' within Blender.

fig. 5.1, 5.2
Basic concepts for the house.

fig. 5.3
Conceptual representation of colors and textures for the rooms.





With the sensors, controllers and actuators an object can be made interactive. You can think of sensors as the senses of a life form, the controllers are the brain and the actuators are the muscles. The row beneath the sensors, controllers and actuators are called the 'logic bricks'. It is possible to add as many logic bricks as you like.

There is only one way to build an interaction network and that is from left to right. The position of the sensor, controller and actuator bricks are all fixed in one column. Besides, every logic brick (i.e. sensor, controller or actuator that is part of an interaction network) is always assigned to an object, but more than one object can be part of an interaction network.

fig. 5.4
The 'realtime buttons' window.

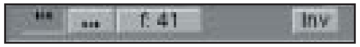
fig. 5.5
Sketch of the 'interaction' diagram for the house.

5.2 List of available logic bricks

The following list is shown here, to give an impression of the possible means to trigger objects within Blender. The means (bricks) to built interactivity into projects.

5.2.1 Pulses (timing)

I will start with pulses (figure below), because pulses are coming from every sensor and trigger both the controllers and the actuators.



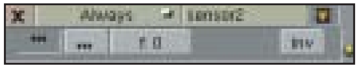
A pulse is like a transistor between the values TRUE or FALSE. Each controller is always evaluated when it receives a pulse, whether the pulse is TRUE or FALSE. The input 'gate' of a controller remembers the last pulse value. This is necessary for controllers being linked by multiple sensors, then it can still do a logical AND or OR operation on all inputs. When a controller is triggered, and after evaluation of all inputs, it can either decide to execute the internal script or to send a pulse to the actuators.
An actuator reacts to a pulse in a different way, with a TRUE pulse it switches itself ON (makes itself active), with a FALSE pulse it turns itself OFF.

Now what is the 'timing' of these pulses? That is sometimes difficult to say within Blender. Anyway, it is not consistent or clear when (precisely) a pulse is send and how an actuator reacts on it. The reason for this is that not all the different actuators evaluate these pulses the same way. Besides, the game engine is, in contradiction to the physics engine, variable in speed. This means that the physics engine 'obeys' its resources to keep on track, while the game engine will slow down when resources drop. This is related to the fact that the physics engine clock runs on time (seconds); every *n*th second the state of objects is updated/evaluated, while the game engine runs on frames; every frame a network of logic bricks is evaluated. Pulses on their turn are 'fired' every *n*th second.

At the moment NaN is busy to make an overview of the current functionality and implementation of the logic system (sensors, controllers, actuators and pulses) in Blender.

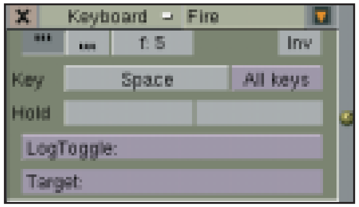
5.2.2 Sensors

Always sensor:



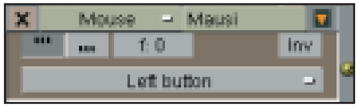
The always sensor gives a pulse 'always'. 'Always' means every frame. The pulses trigger both the controllers and the actuators. The pulses can be set to positive (true) or to negative (false).

Keyboard sensor:



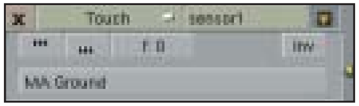
The keyboard sensor provides the interface (interaction) between the game and the user. Every key can be assigned to trigger events.

Mouse sensor:



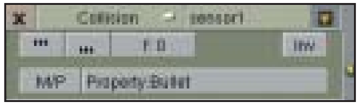
The mouse sensor is able to watch for mouse clicks, mouse movement or a mouse over. But to get the position of the mouse pointer, a python script has to be used.

Touch sensor:



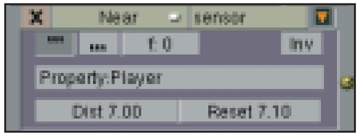
The touch sensor fires a pulse when the object it is assigned to, touches an object with a certain material.

Collision sensor:



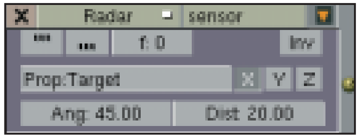
The collision sensor is a general sensor to detect contact between objects.

Near sensor:



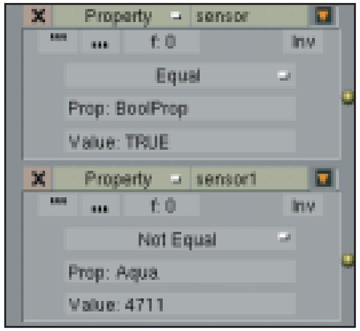
The near sensor can react on actors near the object it's assigned to.

Radar sensor:



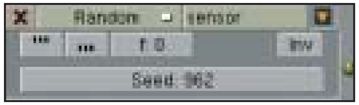
The radar sensor scans the environment for an object along the "x", "y" or "z" axis. The angle and distance to be scanned can be set.

Property sensor:



The property sensor checks an attribute of an object attached to the same object.

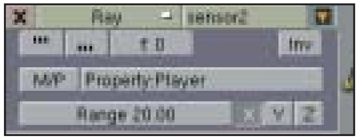
Random sensor:



Random sensor
f: 0 Inv
Seed: 962

The random sensor fires a pulse randomly, every xth frame, according to the pulse settings.

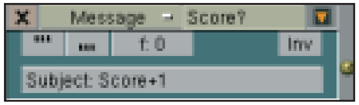
Ray sensor:



Ray sensor2
f: 0 Inv
M/P Property:Player
Range 20.00 X Y Z

The ray sensor casts a ray for the preferable distance to set. If the ray hits an object with the right 'property' or the right 'material' the sensor fires its pulse.

Message sensor:



Message Score?
f: 0 Inv
Subject: Score+1

The message sensor fires its pulse when a 'message' arrives for the object carrying the sensor.

5.2.3 Controllers

AND Controller:

AND cont

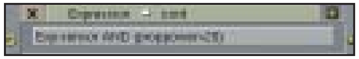
The AND controller combines one or more inputs from the sensors. All inputs must be active to pass the AND controller.

OR controller:

OR cont

The OR controller combines one, two or more inputs from sensors. One or more inputs need to be active to let the OR controller pass the pulse through.

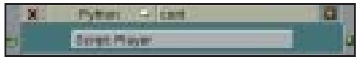
Expression controller:



Expression cont
Exp:sensor AND prop(power<25)

With the Expression controller it is possible to add some 'code' to the game logic. With the Expression controller the output of sensors attached to it can be accessed and through this the properties of the object.

Python controller:

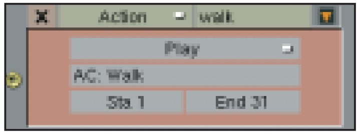


Python cont
Script:Player

The Python controller is the most powerful controller in Blender. A Python script can be attached to it, which allows objects to be controlled ranging from simple movement up to complex gameplay.

5.2.4 Actuators

Action actuator:

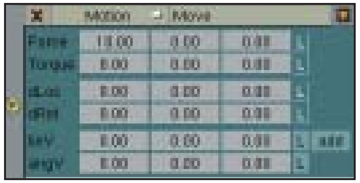


Action walk
Play
AC: Walk
Sta: 1 End: 31

The Action actuator can trigger IPO curves (InterPOlation curves; animation curves). It can play these animations at five different modes:

- *Play:*
  Plays the action from "Sta" (start) to "End" at every positive pulse the Actuator gets. Another pulse while playing is discarded (figure).
- *Flipper:*
  Plays the action from "Sta" to "End" on activation. When the activation ends it plays backwards from the current position. When a new activation reaches the actuator the action will be played from the current position on.
- *Loop Stop:*
  Plays the action in a loop as long as the pulse is positive. It stops at the current position when the pulse turns negative.
- *Loop End:*
  This plays the Action repeatedly as long as there is a positive pulse. When the pulse stops it continues to play the action to the end and then stops.
- *Property:*
  Plays the action for exactly the frame indicated in the property entered in the field "Prop".

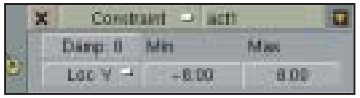
Motion actuator:



Motion Move
Force 10.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

The Motion Actuator is maybe the most important Actuator. It moves, rotates or applies a velocity to objects.

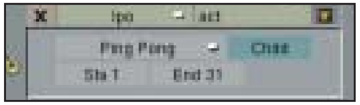
Constraint actuator:



Constraint act1
Damp: 0 Min Max
Loc Y -8.00 8.00

With the Constraint actuator you can limit an objects freedom of movement in x, y and/or z direction to a certain degree.

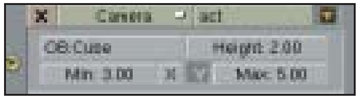
IPO actuator:



Ipo act
Ping Pong Child
Sta 1 End 31

The IPO actuator can play the 'IPO-curves' for the object that owns the Actuator. 'IPO-curves' are the Blender terminology for the animation curves. 'IPO' stands for interpolation.

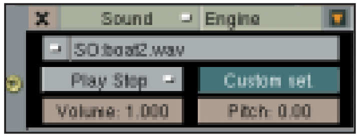
Camera Actuator:



Camera act
OB:Cube Height: 2.00
Min: 3.00 X Y Max: 5.00

The Camera actuator tries to mimic a real cameraman. It keeps the actor in field of view and stays at a certain distance from the object. Also the motion is soft and there is some delay in the reaction on the motion of the object.

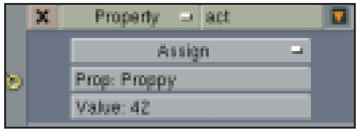
Sound actuator:



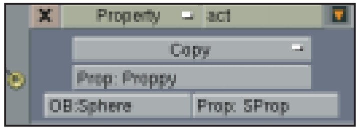
Sound
Engine
SO:boat2.wav
Play Stop
Custom set
Volume: 1.000
Pitch: 0.00

The Sound actuator plays a 'sound object' when it gets a pulse.

Property actuator:



Property
act
Assign
Prop: Proppy
Value: 42

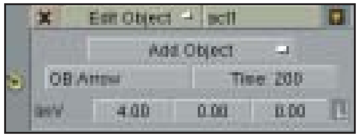


Property
act
Copy
Prop: Proppy
OB:Sphere
Prop: SProp

The Property actuator can be set to three different modes:

- *Assign:*
  Assigns a value or Expression to a Property (figure above).
- *Add:*
  Adds the value or result of an expression to a property.
- *Copy:*
  Copies a Property from the Object with the name given in "OB: Sphere" into the Property "Prop: Proppy". This is a way to pass information between objects (picture above).

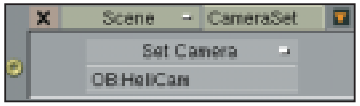
Edit Object actuator:



Edit Object
act1
Add Object
OB:Arrow
Time: 200
linV 4.00 0.00 0.00

This actuator performs actions on objects itself, like adding new objects, deleting objects, etc.

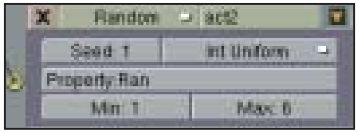
Scene actuator:



Scene
CameraSet
Set Camera
OB:HeliCam

The Scene actuator is meant for switching scenes and cameras in the game engine or adding overlay or background scenes. There are eight different scene actuators: restart, set scene, set camera (figure), add overlay scene, add background scene, remove scene, suspend scene and resume scene.

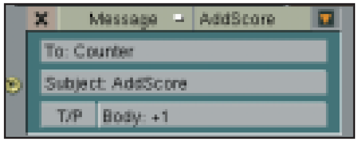
Random actuator:



Random
act2
Seed: 1
Int Uniform
Property: Ran
Min: 1
Max: 6

An often-needed function for games is a random value to get more variation in movements or enemy behavior. There are ten different random actuator types.

Message actuator:



Message
AddScore
To: Counter
Subject: AddScore
T/P
Body: +1

This actuator sends a message out, which can be received and processed by the Message sensor.

5.3 Conclusion

Working with Blender can be rewarding. Learning the basic tools for polygon modeling does not take that much time, assuming that you have modeled before in another 3D package. But modeling for an interactive production is different than modeling for static pictures to be rendered. Polygon count has to be as low as possible to gain an acceptable frame rate. Modeling, texturing and lighting took almost two weeks to be finished. It has to be taken into account that within these two weeks I learned (the basis of) working with Blender and I built the house twice; the second time with fewer polygons to boost the number of frames per second. Compared with simple polygon modeling - for the first time - making the house interactive took less time. The interactive editor is quite straightforward. For a small interactive production as the house it works fine. It took one and a half week to learn and finish assigning the interactivity to the house. For the house, only four different kinds of sensors, one controller and three actuators were used to build the interactivity. Besides, no extra scripting (Python) was used.

Regarding creating interactivity, the following initial observations were made.
For a simple interactive production, there is no need to have any knowledge of scripting. The 'interactive window' interface does not become that cluttered with a small production. So that is a good thing. But as soon as a project become bigger and more complex, the 'interactive window' interface becomes disordered. All the different sensors, controllers and actuators are stacked on top of each other. To gain space these can all be collapsed, but then you do not have an overview of what is inside. A drawback is that there can be a lot of 'wires' connecting the different sensors, controllers and actuators. At this point it is almost impossible to see clearly which one's are connected or not, especially when more than one sensor is connected to a controller.

6. Types of interactive content

After a study of different kinds of interactive content, diverging from simulations to games, the ‘interactive 3D content’ diagram was made (see fig. 6.1, below).
This diagram is constructed in a way I think 3D interactive content can be (sub)divided. This is my perception, their are other possible diagrams and probably it is not complete, but either way its purpose is to give an insight to interactive 3D content.

6.1 Games

Computer game history does not go that far back. The first commercially released games became available in the mid seventies. Despite being a young field there is a distinction possible between games. Games can be divided into two broad categories: skill-and-action games (emphasizing perceptual and motor skills) and strategy games (emphasizing cognitive effort). Each of these two major categories has subcategories. A third category is coming up: hybrids (a combination of two of the ‘standard’ categories).

6.1.1 Skill-and-action games

This is probably the largest and most popular category of the computer games. Most of the computer games are associated with skill-and-action games. Arcade games are skill-and-action games. This class can be characterized by real-time play and heavy emphasis on graphics and sound. The primary skills demanded of the player are hand-eye coordination and fast reaction time.

Shooters (combat games)
Shooters all present a direct, violent confrontation. The player must shoot and destroy the bad guys. The challenge is to position oneself properly to avoid being hit by the enemy while shooting him. Two subcategories can be distinct: First-person Shooters (FPS) and Third-person Shooters (TPS). There are many variations on this theme, most with small variations on the geometry of the situation or the weaponry.

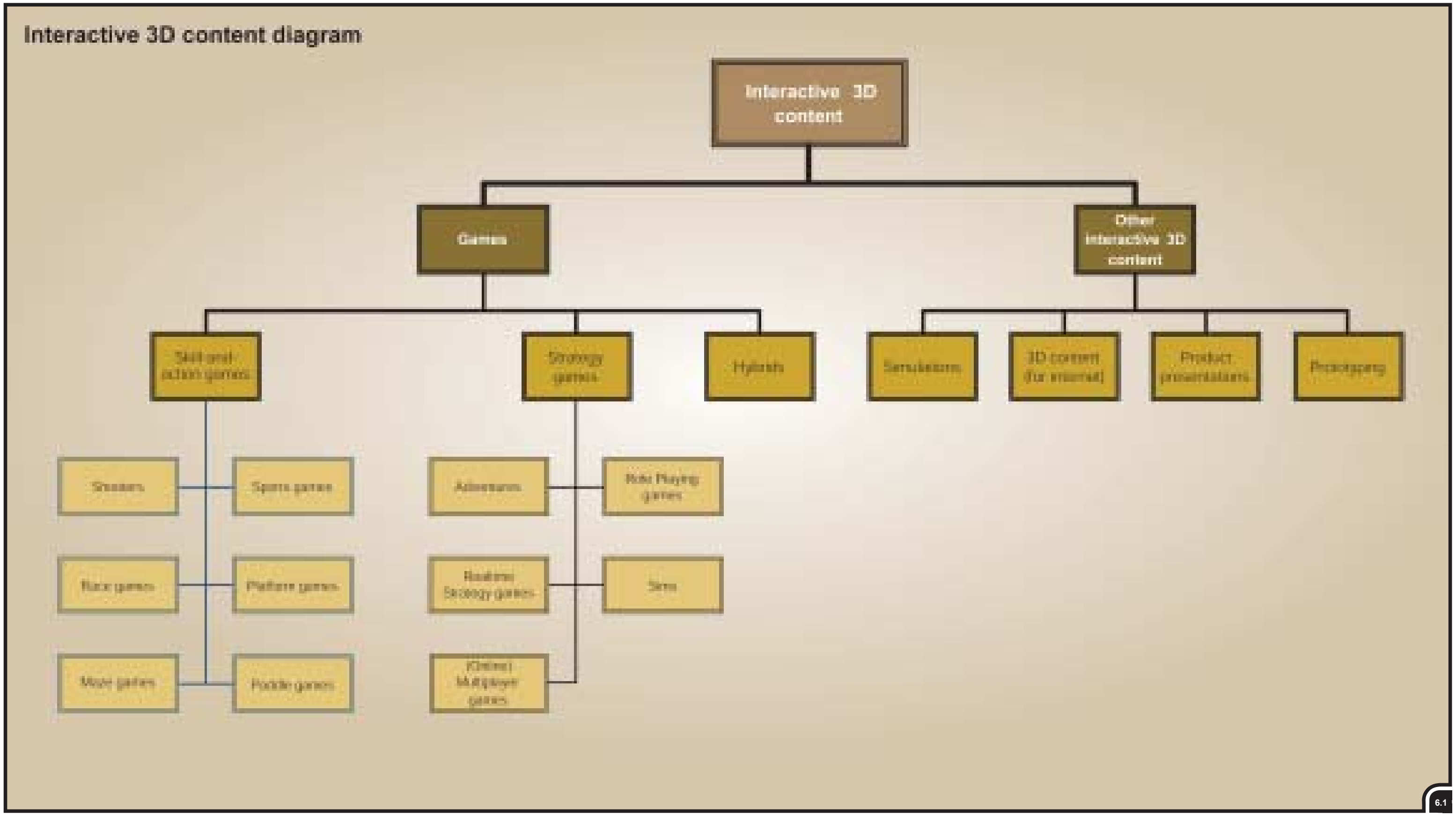


fig. 6.1
Interaction 3D content diagram; (sub)-divided in a way according to me.

Examples:
- Space Invaders *(1978, Taito America Corp.)*
- Wolfestein 3D *(1992, Grey Matters)*
- Doom *(1993, Midway Entertainment)*
- Quake III Arena *(1999, id Software)*
- Unreal *(1998, Epic Games)*

Sports games

These games model popular sports. Sports games offer the player a game he is already familiar with. Thus there are games based on soccer, basketball, baseball, tennis, boxing, skating, golf and many others.

Examples:
- Fifa (soccer) *(Electronic Arts Inc.)*
- Madden NFL *(football) (Electronic Arts Inc.)*
- Tony Hawk's Pro Skater *(2000, Activision)*
- NHL (ice hockey) *(Electronic Arts Inc.)*

Race games

Some race games involve a straightforward race. Others allow the player to move at a constant speed, but extract time penalties for failure to skillfully negotiate an assortment of hazards. A checkpoint has to be reached before the time has elapsed.
The race games can be divided in real life simulation (rally and formula one racing games) and the more 'futuristic' race games. In the futuristic race games the 'car' does not necessarily have wheels. It can hover just above the ground (wipe out).

Examples:
- Outrun *(1985, US Gold)*
- Need for Speed *(2000, Electronic Arts Inc.)*
- Formula One 2000 *(2000, Electronic Arts Inc.)*
- Collin McRae Rally *(1999, Codemasters)*
  Crazy Cars (*1997, Synaptic Soup)*
- Wipe Out *(1999, Psygnosis)*
- Megarace *(2001, Cryo)*

Platform games

With platform games the player has to find his way trough several levels and solve puzzles and 'fight' his way trough impossible courses; he has to climb, jump, dodge, swim etc. It is all about skills. Like the name suggests, the player often has to jump form one 'platform' to the other. Platform games have always a third person of view perspective.

Examples:
- Super Mario Bros. *(1986, Nintendo)*
- Prince of Persia *(1989, Broderbund Software inc.)*
- Tomb Raider *(1996, Core Design Ltd.)*
- Sonic Adventure *(1999, Sega)*

The next two categories are not widely spread anymore, but I think they will become more interesting again with mobile (internet) devices (mobile phones (UMTS), PDA's (Pocket PC's) etc.). Because of the small screen size and lack of extensive input devices, small games like paddle and maze games are perfect to suite these devices. The examples mentioned below are not real 3D games, because they are from before the '3D age', yet some of them have been ported to 3D.

Paddle games

This term covers all the PONG-based games. Paddle games can be seen as the sports games of yesterday. The central element in these games is a paddle-controlled piece. The player has to use the ball as a 'weapon' to batter or he as to use it to catch the (many) ball(s), rather than to deflect it.

Examples:
- Pong *(1958, Willy Higinbotham; he build a pong like game around a computer and a CRT at Brookhaven National Laboratory, [2])*
- Breakout *(1976, Atari)*
- Avalanche *(1978, Atari)*

Maze games

The characteristics of maze games are the maze of paths trough which the player must move. Most of the time the player is pursued by bad 'guys'. The most successful of these is probably PAC-MAN.

Examples:
- Pac-Man *(1980, Namco)*
- Jawbreakers *(1982, Tigervision)*

6.1.2 Strategy games

Strategy games are the second broad class of computer games. These games emphasize cognitation rather than manipulation. This does not mean that skill-and-action games do not have any strategic element at all. The major distinguishing factor between strategy games and skill-and-action games is the emphasis on motor skills. All skill-and-action games require some motor skills: strategy games do not. Another difference between strategy games and skill-and-action games is the fact that strategy games typically require more time to play.

Adventures

These games derive from one of the oldest computer games, called 'adventure'. In these games the adventurer must move through a complex world, accumulating tools and booty adequate for overcoming each obstacle, until finally the adventurer reaches the treasure or goal. The player has to be aware of the different predator animals, unidentifiable creatures, and henchmen lurking around, while finding his path around. Adventures are sometimes closer to puzzles than to games. Adventures present intricate obstacles, which once cracked, provide no longer a challenge to the player.

Examples:
- Larry *(1987, Sierra)*
- Myst *(1993, Cyan Productions)*
- Alone in the dark (1997, I*Motion)

Role Playing Games (RPG's)

In a RPG the player becomes another person. He plays a role. The player is a wizard, a king, a troll or one of the Knights of the Round Table for instance. Most of the times the player can choose between two or more sides he wants to be with and which other character he wants to be. The player must then search through a wide (fantasy) world to find and rescue a princes, but on the way he must fight monsters and thrives. Because most of the time the player is a fantasy character, Role Playing Games are also called Fantasy Role Playing games (FRP).

Examples:
- Diablo *(1996, Blizzard Entertainment)*
- Starwars Galaxies *(2002, Lucas Arts)*

Realtime Strategy games (RTS)/War Games

In RTS games the player is in command of whole army's. The player has to move his men, weaponry, tanks, boats etc. carefully and strategic to conquer the enemy. RTS games can be laid out in WO II, future wars (WO III), the middle ages, in space (fantasy, Star Trek) etc.
In the beginning these strategy games were turn based instead of real time. Now computers and internet connections become faster and the built in 'artificial intelligence' is more capable of handling the decisions to be made in such realtime strategy games, turn based strategy games become out of date.

Examples:
- Myth: The fallen lords *(1997, Bungie Software)*
- Age of Empires *(1997, Ensemble Studios)*
- Civilization II *(1996, MicroProse)*
- Command & Conquer *(1995, Westwood Studios)*

- Commandos 2: Men Of Courage *(2001, Pyro Studios)*
- Gangsters *(1998, Hothouse Productions)*

Sims
Sims are (arcade) simulations of real life situations. The player has to build a city and satisfies its citizens for instance. Or the player has to fly a (passenger) plane. Not arcade like but a 'real' plane with all the possible controls available. The player has to take of, fly to a destination and land the plane in the end. So there are more examples possible, but the main key of sims is to represent 'real' life as real as possible. The player can control near every aspect.

Examples:
- Flight Simulator 2000 *(1999, Microsoft)*
- Sim City 3000 *(1999, Maxis)*
- The Sims *(2000, Maxis)*
- Railroad Tycoon *(1996, PopTop Software)*

(On-line) multiplayer games
Multiplayer games can be all kind of games. The important difference is that the player plays against another human instead of the computer. The interpersonal relationship between players seems to be a very important aspect of playing a computer game. Due to the increasing internet speed and accessibility it becomes more and more popular. The reward of defeating a human opponent can be far greater than defeating the computer.
The two most popular (on-line) multiplayer games are the First Person Shooters (FPS) and the Real Time Strategy games.

Examples:
- Quake III Arena *(2000, id Software)*
- Team Fortress *(2000, Valve Software)*
- Planetside *(2002, Sony On-line)*
- Halo *(2002, Bungie)*

6.1.3 Hybrids

In this category boundaries become blurred. Two or more 'standard' (sub)categories are combined to a new experience in game playing. Hybrids will be more and more common in game design. For example, a game in which the player can fly and shoot in a space craft, land in a space station, steps out of his space craft and starts shooting the 'bad' guys ('Falcone: Into the Mealstrom' by Point Blank). This example is a blend of arcade-space dogfighting and a first person shooter (FPS). Another example is Warcraft. Warcraft is a blend between a Realtime Strategy game (RTS) and a Role Playing Game (RPG).
It must be said that these hybrids are not as new as they sound like. Some games mentioned in sections 6.1.1 and 6.1.2 are already a sort of hybrid, but with the availability of new technologies, faster computers and better graphical cards, it is now really possible to integrate different categories into a new sort of game. Like with the example I mentioned before (Falcone: Into the Mealstrom), it was previously not possible to make a game in where you could fly and shoot around in deep-space and on the other hand could walk around in a small space station (compared to the big space around) and battle man against man. The game engine to support this needs a very fast computer.
As said, hybrids have been around for longer, but were not always that obvious. For instance Deus Ex (2000, Eidos Interactive). This game mixed a variety of genre elements, including action and point-of-view of First Person Shooters, story, character development and exploration of Role Playing games and the economic management and strategic expression of Strategy games.

6.2 Other interactive 3D content

Besides games there is also 'non-game' interactive 3D content, described below.

6.2.1 Simulations

3D simulations can be used to imitate real life situations. A car accident has to be investigated for instance. With the clues from the scene, the simulation can be built and conclusions can be made afterwards. But a set of marbles rolling down a course is also a simulation.
The interactivity to be built in such simulations is not the possibility to navigate trough a scene by ones self, but the interaction of the various elements in the scene itself. The elements have to react on each other.

6.2.2 3D content (for internet)

The creation of 3D (interactive) internet content will become more important and widely spread. I think it will be the next step, but it has to be seen if it will be integrated well. But it will be a welcome addition to the flat interfaces of today's sites, especially when it is used for navigation.
In this area you can also think of 3D (multiplayer) games to be played in the web browser.

6.2.3 Product Presentations

Another possibility is 3D product presentations that make it possible for the client to view the product from all sides before he will buy it or even before it is actually produced. With the addition of interactivity the client is also able to open doors, look inside products or start a virtual CD player for instance. Product presentation is close to '3D content for internet', because most of the time this concept of pre-visualisation is offered on the internet.

6.2.4 Game prototyping

Prototyping is done to make an example of how the end result will look like in a very short period of time. Prototyping can be done for almost everything, but in this perspective you can think of prototyping a concept for a new game. Building the environment, the character, the opponents and the interaction to see if some ideas will work properly, as you would expect. After you or your client is satisfied you can start building the final game.

6.3 Conclusion

This overview of interactive 3D content has been given to describe the terminology of interactive 3D content. This terminology is used throughout the rest of this report. Especially in section 9.3 where example games are listed which are possible to make with Blender at this moment.

7. Game development teams

This section will give a description of existing game development teams. How they manage their projects and deal with problems they come along. The information is obtained from different sources, diverging from the content creation team at NaN itself to the ‘postmortems’ of Gamasutra. The information obtained from the content creation team is also used for the ‘game development process’ described in section 8.1.

7.1 Content creation team NaN

For the interviews a questionnaire has been used (see appendix II) as a guide. Therefore, a brief summary of the different interviews is given with the different members of the content team together.

The content team:
The content team at NaN exists of four people with different backgrounds and nationalities. The different backgrounds are from an architectural and a graphical nature, while the nationalities are Dutch, American and Canadian. All were members of the ‘Blender community’ and were using Blender as a hobby before they start working at NaN.

Environment:
The environment stimulates the work process of all four members. The position of the desks and computers is towards each other. So they can see each other directly, what makes fast communication possible. Besides the positioning of the furniture, the content team owns a lot of movies and games for inspiration and of course relaxation.
To write down initial ideas, most of the members use sketch books. They sketch with pens, pencils and markers. To sketch out the big picture for a project they use a big paper on the wall. This is also used for storyboarding animations. The big advantage of using ‘wall paper’ is that everybody can see it clear how far the project has been progressed and what still has to be done.

Productions ever made with Blender by the content team:
- Graphics (stills)
- Holograms
- Animations
- Fly troughs
- TV commercials
- Games:
  2D shooters (top down view)
  ‘On rail’ shooters (like Virtua Cop and Time Crisis)
  3rd person shooters (like Mechwarrior)
  Race games (from futuristic like Wipeout to more standard race games)
  Platform games
  Puzzle games

All the productions (games etc.) made by the content creation team are (almost) always for promotional purpose on-line.

Design sequence:
- *Assignment briefing:*
  A description of the project will be given by a 3rd party (management and/or PR team). Time schedule is made and the decision is made about how many people will participate on the project

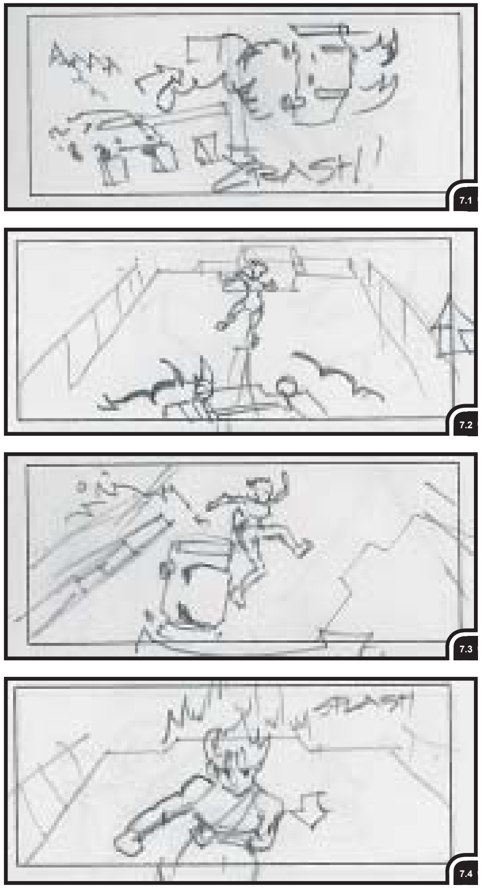


fig. 7.1 - 7.4
An example of a (part of) storyboard for an animation. (Courtesy of and © by Reevan McKay, member of the content creation team of NaN)

- *Brainstorming with the hole team about:*
  Genre, style (cartoons, science fiction, what kind of animations etc.), gameplay and story.

- *Inspiration and reference:*
  Inspiration for the different projects is taken from a wide amount of resources. Including movies, television, commercials, books, comics, magazines, internet and sometimes other computer games.

- *Storyboarding and visualization:*
  Sequences of pictures are drawn for the cut-scenes (animations, no interactivity); for the in-game buildings, characters etc. only a few pictures are drawn to get an impression and to decide which designs will be used.

- *Building the geometry:*
  When all the designs are finished and been approved, the designs will be built in Blender.

- *Game logic (interactivity):*
  When there is enough time the game logic will be laid out on paper. All the objects that participate in the interaction are connected in a way they will interact to each other. But most of the time the game logic is build by one person by trail and error.

- *Sound design:*
  After assigning the game logic is done, one person adds the different sound samples. Strictly speaking is the process of adding sounds a process of building game logic too. For example, the sound of a opening door must only be played when the door actually opens. So not only a IPO actuator, but also a sound actuator has to be added to the door.
  The sounds used in a project are extracted from sound libraries or recorded live. Then these ‘rough’ sounds are edited until they suite the atmosphere they will be used in.

- *Execution:*
  When all the different parts of the game are build, all these parts will be put together and tested until the game is finished for release.

- *Deployment:*
  Finally, pictures for on-line use on the Blender site are rendered.

Task sharing:
The different task of modeling, making textures, animations, music and game logic are divided by the strengths of every individual member and/or what every individual likes to do. The last things to be added are the game logic and music.

Problems/difficulties:
Very little development time. The content creation team gets normally three to six weeks to finish a project, while other game development teams get six to twelve months for a comparable project. So most of the time some initial ideas have to be dropped to finish the project within the given time schedule.

The physics engine. Because of the 'general' nature of the physics engine, it is difficult to make specialized games. During the game development this has to be kept in mind.

The game logic. Making a 3D world interactive causes a lot of frustration. A large production makes the (game) logic interface very complex. There are too many data blocks (i.e. logic bricks) and connection 'wires' that clutter the inteface and makes it very hard to find the way around when adjustments has to be done. There is no means to organize the logic bricks. There is no way to get an overview of all the objects and connected logic related to the project. Another aspect of this, is the fact that all the logic bricks and properties are linked to objects. So when you have to change a property that is spread over more objects, you have to select all the objects one by one and change the appropriate property.

fig. 7.5
Concept Drawing for the Creature [4].

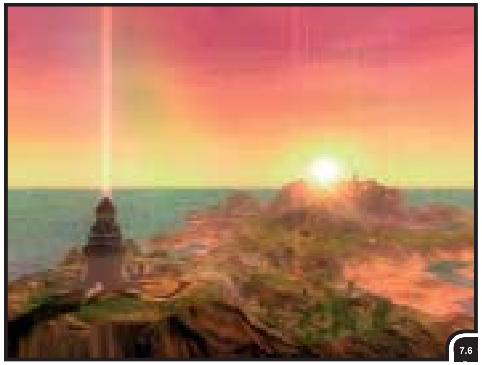


fig. 7.6
The final product. A in-game picture [4].

Consistency. Within Blender sometimes it is hard to predict what's going to happen. The sensors and/or actuators are not predictable and consistent through out a project. When a set of logic bricks work fine in one condition and situation, does not mean that it will work the same on another object and place.

7.2 Postmortems

The 'Postmortems' from Gamasutra [3] describe the process and pitfalls of making a (specific) game. Gamasutra is a part of the Gama Network. The Gama Network provides resources for game development. These resources are available in print (Game Developer magazine), a yearly event (Game Developers conference) and thus on-line (Gamasutra.com).
In this case I will only describe 'what went wrong' according to the 'Postmortems', because that can give extra information on how to improve the game development process. The 'game data' will also be listed to give an overview of how extensive the project really was. The 'game data' are the figures of how many people worked on the project, how long it took (from start to release) and what soft- and hardware was used.

Lionhead Studios' Black & White and Lucas Arts' Star Wars Starfighter are described here, because their game development process involves problems too. It gives insight into the workflow and problems of a major studio during a game development process, compared to the (earlier described, see section 7.1) problems of the (relative small) content creation team of NaN. Besides, I liked to play these games personally.

7.2.1 Lionhead Studios' Black & White [4]

Black and White is a mixture between a role playing game in 'god mode' and an adventure. As a player you can control and influence people in an entire world. The player can choose between good and evil to rule and change the world.

Planning the story. It was very hard to estimate how long it would take to construct and write the story element. The free-form nature of the game required an unfolding tale to give it some structure and lead it to a conclusion. It was expected that the story would take no more than two months, but after a while it was decided to hire a professional games scripter writer. In the end the story was more than 60,000 words, the size of a novel.

Fixing the bugs. Hitting the 'Alpha' (first playable game; all elements are put together), there were more than 3,000 bugs. These bugs had to be reduced to zero within six week. But when one bug was solved, three more were created. By this stage the team was very tired and the only thing that kept the team going was the sense that the end was in sight. The last ten bugs were the hardest to fix and it almost was as if the game did not wanted to be finished.

The project was too big. Black & White got to be so large that it almost felt as if you were lost within it. In the end there were over a million lines of code within the game. Loading up even the most simple of the smallest tools would take many minutes, and compiling the entire game took over a hour, which meant that even a tiny change could take a whole day to implement.

fig. 7.7
Several ship models developed for *Star Wars Starfighter*, and an early screenshot of a pirate bomber [5].

Leaving things out. As in every project some features did not make the finished product. But in this case it was not because of problems caused by software or hardware, but it came down to emotional issues. For example, the original idea of the 'Creatures' was that the player could choose to make any living thing a Creature. The player had to be able to select an ant and grow that, or a human being from a tribe, and raise him or her. One of the artists, spent a long time drawing concepts and sketches depicting what the Creatures could look like at various stages of their development. This of course included humans. But because of the idea that people would have certain expectations from a human. Players would not expect a turtle to learn as quickly as a man, but if the people would be dumped down, they would seem like a proto-hominid race from eons ago, and that was not the intention either.

In summary:
The problems the team of Lionhead Studio's came along were: a lack of planning, lack of prototyping, and they started too late in the project with playtesting.

Game data:
Creator: Lionhead Studios
Publisher: Electronic Arts
Start date: February 1998
Release date: March 2001
Full time developers: 25
Budget: $5.7 million

Hardware used: 800Mhz Pentium III's with 256MB RAM, 30GB hard drives, Nvidia GeForce graphics cards

Software used: Microsoft Dev Studio, 3D Studio Max

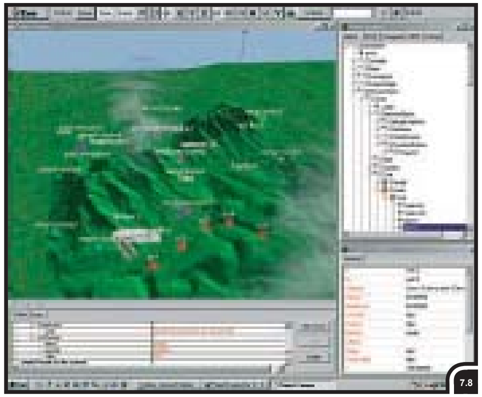
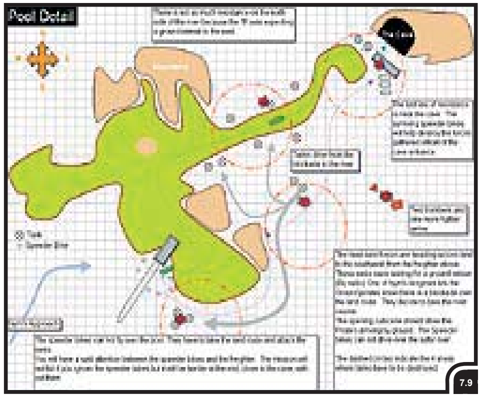

7.8

fig. 7.8
The 'Eve' level design tool used to build Star Wars Star-fighter [5].



7.9

fig. 7.9
Design shematic for one of the Lock missions.
Level designers made elaborate plans, such as this example, for every level [5].

Notable technologies: Bink for video playback, Immersion touch sense for force-feedback mouse

Lines of code: approximately 2 million

7.2.2 Lucas Arts' Star Wars Starfighter [5]

Star Wars Starfighter is a space-combat game in arcade style. As a player you can have the control over three different starfighters (i.e. planes). With the starfighter you have to save innocent citizens from an invasion of pirates. Space dogfights, attack runs and escort missions are the outcome.

Initial lack of detailed design. Initially it was assumed to stay as far as possible from the events in the film Star Wars: Episode I, where the game is based on. Because the game was going to tell one of the first original stories set in the time line of the new film (Episode II, yet to be released). There was no feeling of where the boundaries were with respect to planets, characters, vehicles and the like.
The first game design described a pirate war far divorced from the events of the film. After circulating this design, 'Licensing' contacted the team and explained that the design contained too many pirate elements; the games should contain more elements of the film. The "moving target" nature of this exchange ended up being very disruptive and effectively paralyzed the design effort for weeks at the time that the design team wandered from idea to idea, wondering what fit into continuity with the film and what was straying into areas that should not be entered.

Not enough attention paid to performance. There was a pervasive attitude among many of the team that code problems could safely ignored until they showed up as hotspots on a profiling run. There is some merit to this strategy, since premature optimization efforts can be more wasteful than not fixing the code at all. But since profiling can turn up hidden problems in areas of the code that the team had previously thought complete or issue-free. But in this case it was important that the team had started much earlier than it did to overcome some problems. For example, there were severe performance problems in the collision detection systems that would have been identified immediately if it had been profiled sooner. As it happened, by the time it was realized that the collision detection was working poorly, the best that could be done was apply spot fixes instead of the large-scale reworking that the problem actually demanded.

Space-to-planet. If there was anything about the original Star Wars Starfighter pitch that met with widespread enthusiasm within the team, it was the idea of seamlessly transitioning from planetside environments to the depths of space and back again. Dog fighting close to the planet surface has its own appeal, but the idea about the promise of being able to pull back on the stick and blast off all the way into space that was great. This concept was so exciting to the team that the original game pitch featured this idea predominantly.

First, there were the technical considerations. A planet is big, really big. Even a small planet would require dynamically creating thousands of terrain tiles. Although most of these tiles could be procedurally generated, they would still need to be created and discarded on the fly; depending on the player's location, custom mission-area tiles would have to be streamed in from the hard disk, all while maintaining frame rate. Since the idea was to allow the player to fly absolutely anywhere on the planet, ordering this data on the disk in a streaming-friendly format was problematic. The situation was exacerbated by requiring even the lowest-resolution terrain height maps to be much higher resolution than they really needed to be. This in turn made higher theoretical demands on the streaming and resource systems.

This single feature had introduced a tremendous amount of technical risk to the project, and yet the team had blindly charged ahead anyway because of the idea's inherent coolness factor. The technical issues, however, did not describe the full extent of the problems with this feature. Quite quickly the team also came to realize that there were plenty of game design issues implied by the space-to-planet concept. For example, there was the constant issue of player craft speed. It was decided that the ships should have a top speed of about 450 miles per hour, because dog fighting and bombing ground targets becomes extremely difficult if you move much faster. However, at that speed it would take the player 20 minutes to achieve a low-planet orbit. To circumnavigate a small planet the size of the moon could take as long as 16 hours. Although the team was able to brainstorm several fanciful solutions to this problem, most were time- or cost-prohibitive, and all of the solutions threatened to shatter the illusion that you were in a small fighter craft, engaged in small, intimate battles.

In summary:
The problems the team of Lucas Arts' came along were: initial lack of a detailed game design before starting to build the game, lack (or too late) of testing code on performance issues, lack of technical considerations; the game idea happened to be too complex.

Game data:
Creator: Lucas Arts
Publisher: Lucas Arts
Start date: April 1998
Release date: February 2001
Full time developers: approximately 40

Hardware used: 700Mhz Pentium III's with 256MB RAM, Nvidia GeForce graphics cards, PS2 (Playstation 2) tools

Software used: Windows 2000, Microsoft C++, Metroworks for PS2, 3D Studio Max, Softimage, Photoshop, Bryce, Visual SourceSafe, Perl, After-Effects, Premiere

Notable technologies: Eve level design tool, Miles Sound System, ObjectSpace STL, Macromedia/Secret Level Flash, Planet Blue's Tulip for prerendered cut scene lipsynching

Lines of code: 301,000 (including tools)

7.2.3 More postmortems

To give some more impressions of game projects and there magnitude, there will only list the 'game data'.

Poptop Software's *Tropico* [6]
In Tropico the player becomes a dictator of a small tropical island. As the dictator the player must feed his people and keep them happy, healthy, and safe (both from themselves as well as from outside threats). The dictator must also provide entertainment and places of worship and he has to ensure that the island's economy grows continuously.

Creator: Poptop Software
Publisher: Gathering of Developers
Start date: April 1999
Release date: April 2001
Full time developers: 10 (7 artists, 3 programmers)
Budget: $1.5 million

Hardware used: 550Mhz Pentium III's with 512MB RAM, 40GB hard drives

Software used: Visual C++ 6.0, Visual Source-Safe 6.0, 3D Studio Max 3.1, Character Studio 2.2, Photoshop 5.5, Three Factory plug-in for 3D Studio Max

Notable technologies: Bink Video, Miles Sound System

Lines of code: approximately 150,000 (plus 20,000 for tools)

Muckyfoot's *Startopia* [7]
Startopia is a building and resource management game from the same genre as SimCity and Theme Park. The player takes charge of an abandoned space station and his task is to turn it into a thriving complex where alien space travellers can stay or even live.

Creator: Muckyfoot Productions
Publisher: Eidos Interactive
Start date: March 1999
Release date: June 2001
Full time developers: Core team - 6 programmers, 4 artists, 3 testers, 1 designer, 1 musician (Support resources - 2 programmers, 2 artists)
Budget: $3.0 million

Hardware used: 550Mhz Pentium III's with 128MB RAM, 20GB hard drives, various 3D accelerator cards

Software used: WordPad, MS Visual C++, SourceSafe, Photoshop, Paintshop Pro, 3D Studio Max, paper, pencils and pens

Notable technologies: Direct X, Bink, Peer Engine

Lines of code: 33,5000

Raven Software's *Soldier of Fortune* [8]
Soldier of Fortune is a First Person Shooter (FPS). The player is an ex-special operative agent and he is tasked with infiltrating Soviet-held Prague during the height of the Cold War.

Creator: Raven Software
Publisher: Activision
Start date: April 1998
Release date: March 2000
Full time developers: 20
Budget: multi-million-dollar budget

Hardware used: 550Mhz Pentium's with 128MB RAM, 18GB hard drives, TNT2 graphics cards

Software used: Microsoft Visual C++ 6.0, Microsoft Visual SourceSafe 6.0, 3D Studio Max 2.5, Softimage 3D, Photoshop

Notable technologies: Licensed the Quake 2 engine from id Software (using OpenGL), motion-capture data from House of Moves, force feedback, A3D/EAX 3D sound, World Opponent Network (WON) match making services

Lines of code: 406,044

7.3 Conclusion

Every game development team or game developer has to deal with several difficulties during the game development process, even a small development team as the content team of NaN is. All have to deal with a tight time schedule. But the biggest difference for the content creation team of NaN compared with other development teams is that the content creation team of NaN has only four to six weeks compared with two to four years of development time for a new game. It tends to be that there is less and less time for prototyping. The available time for creating a new game title will be more or less the same, but the complexity and magnitude of a game will expand. Players are ever and again expecting more exciting and new features and game play.
So game development teams need a application with which they can design, build, texture, create interactivity and playtest in a very short period of time.

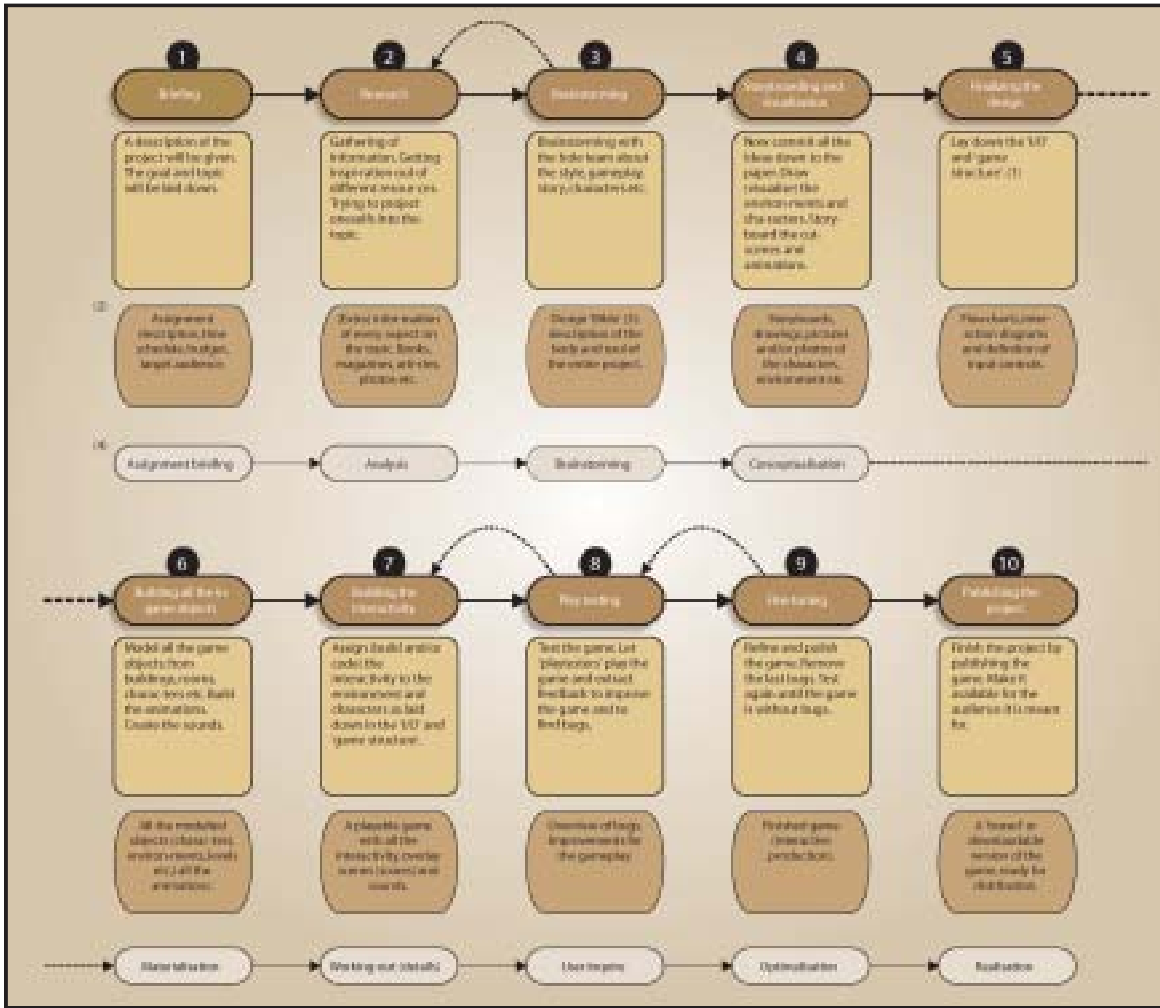
8. Development process

The design process of making the house is described in section 5.1 and an impression of game development teams (especially NaN's content team) and their processes is described in section 7. A more general approach for the game development process will be given now.
Before this is done, it has to be said that game development is complex process. It is the creative mind that (should) guide to develop a great game. It is an artistic process in the first place. Second, it is also a very technical process. A lot of code has to be generated. Probably, the game development can be best compared to the film making and/or product development process.

8.1 Game development process

The game development process, or any other creative process, will never be a real linear process, but a process of different stages which will overlay and will interact with each other. Considering that it is hard to write down a non linear process or a way of thinking, a process in a successive order is given. These are the steps that are, most of the time, taken during game development. This process is shown in figure 8.1, below. To show the game development process can be compared with product development process. The product development process is shown below the game development process.

This diagram (fig. 8.1) results from the process of


1
Briefing
A description of the project will be given. The goal and topic will be laid down.
2
Research
Gathering of information. Getting inspiration out of different resources. Trying to project crosslinks into the topic.
3
Brainstorming
Brainstorming with the whole team about the style, gameplay, story, characters etc.
4
Storyboarding and visualisation
Now commit all the ideas down to the paper. Draw (visualise) the environments and the characters. Story board the cut-scenes and animations.
5
Finalising the design
Lay down the 'I/O' and 'game structure' (1)
(2)
Assignment description, time schedule, budget, target audience.
(Extra) information of every aspect on the topic. Books, magazines, articles, photos etc.
Design Bible (3): description of the body and soul of the entire project.
Storyboards, drawings, pictures and/or photos of the characters, environments etc.
Flowcharts, interaction diagrams and definition of input controls.
(4)
Assignment briefing
Analysis
Brainstorming
Conceptualisation
6
Building all the game objects
Model all the game objects from buildings, rooms, characters etc. Build the animations. Create the sounds.
7
Building the interactivity
Assign (build and/or code) the interactivity to the environment and characters as laid down in the 'I/O' and 'game structure'.
8
Play testing
Test the game. Let 'playtesters' play the game and extract feedback to improve the game and to find bugs.
9
Fine tuning
Refine and polish the game. Remove the last bugs. Test again until the game is without bugs.
10
Publishing the project
Finish the project by publishing the game. Make it available for the audience it is meant for.
All the modelled objects (characters, environments, levels etc.) all the animations.
A playable game with all the interactivity, overlay scenes (cutscenes) and sounds.
Overview of bugs, improvements for the gameplay.
Finished game (interactive production).
A 'boxed' or downloadable version of the game, ready for distribution.
Materialisation
Working out (details)
User testing
Optimalisation
Realisation
8.1

(1) The I/O structure is the system that communicates information between the computer and the player.

The game structure is the internal architecture of causal relationships that define the obstacles the player must overcome in the course of the game.

(2) Deliverables: what is finished after every step.

(3) The design Bible describes the (background) story for the game and it includes sample descriptions of what the play will do such as:
- What form is the interaction taking?
- Are the players making story choices in a branch story composed on scenes, or are they roaming an environment as in a adventure game?
- Do the players pick up items?
- How do the players talk to people they meet?
- What does the interface look like?
- etc.

(4) The product development process for comparison.

fig. 8.1
The game development process.

building the house, the information obtained from the content creation team at NaN and it is loosely based on the product development process.
The following steps are included in the game development process:

- Briefing
- Research
- Brainstorming
- Storyboarding and visualization
- Finalizing the design
- Building the game elements
- Building the interactivity
- Play testing
- Publishing the project

8.2 Conclusion

As said before the game development process will never be a process in a successive order. Within a development team, a lot of steps will and can be done simultaneous. When later on a start is made with creating new concepts for 'building interactivity' (section 13), it can be useful to take this process into account.

Now the 'game development process' has been described in the previous section, I will look whom Blender and especially the interaction creation part is intend for. What is and will be the target group.

9.1 Target group Blender

The interactive creation part within Blender is not only intended for game creators, despite the fact that until now there is only talked about 'game creation'. The possibility to use interactive Blender files within a web browser (Blender 3D Plug-in) and to export Blender files as 'stand alone' productions (like a Macromedia Director™ projector), makes Blender also interesting for other users than game creators alone.
With the 3D plug-in it is, of course, possible to make 3D games integrated in a web site, but also 3D product presentations (on-line shops) and architectural walkthroughs (visualizations of buildings). The possibility to use Blender files within Powerpoint™, Word™ and Director™ means also that 3D multimedia presentations can be built.
The target group can be divided into 'professional' and 'home' users. The professional users can be subdivided again into game- and multimedia developers.

Game artists/creators
Game artists can use Blender to build entire games within Blender. They can build, texture, animate and make it interactive and publish it as a stand-alone game or publish it for use in a web browser with the available plug-in.
Another possibility is to use Blender for concept testing (prototyping). To build - with less programming - game ideas fast and test them.

Multimedia and web designers/developers
Multimedia and web designers can use Blender to build 3D web environments, product presentations, walkthroughs (for architects), 3D site navigation structures, 3D advertisement banners and to enhance presentations in general with (interactive) 3D diagrams, graphs etc.

'Home users'
Because of the wide 'user base' NaN has reached with its free of charge version of Blender (Creator), the home user has to be taken into account too. The home user or hobbyist can use Blender for their own purpose. This can be applied to one or more of the mentioned things above. On the understanding that a home user is willing to learn Blender, because it is not a easy done.

The target group of Blender is widely spread. Assuming that not all the (future) users are able to built their 3D content within Blender (e.g. they import files from other 3D packages), they must be able to make their 3D content interactive within Blender. The different users, with different backgrounds, must be able to use and understand the working and principles of building the interactivity. This process must be easy to adopt and well laid out as well for beginners as for advanced users, whom are using Blender and its current 'interactive building' part.

9.2 What kind of 3D content?

Now I have described the users I will give a more detailed description of what kind of interactive 3D content Blender is (will be) meant for.

Games:

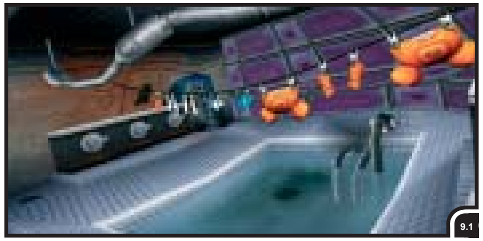
- *Small games which are 'chapter' or 'sequential' based*
  With the rise of internet and the forthcoming second (and third) generation of mobile phones, games that can be developed for these two purposes in a fast and easy way will have a great potential. With Blender these games can be developed very fast within one package. For instance, every week (or month) a new chapter/level can be added to the game.
- *Games which require a short development time*
  This is very related to the 'chapter' based games or games that require to be released in a very short period.
- *Games for mobile devices (PDA's, mobile phones)*
  Small games for small screens, that can be made in an easy way and can be played with the controls available on the mobile device. Also related to 'chapter' based games, with the difference that it not necessarily has to be a chapter based game, but it can also be a game on its own.
- *3D games for internet*
  Games that can be played within a web browser. This can be standalone games or multiplayer games played over the internet.
- *Prototyping games*
  Before committing to the process of making a game, a developer first wants to know if his ideas will work as he intended them to do. Blender can be used for this kind of 'games'. Parts of a new game can be made very quick and tested. In this way a developer can change some ideas and let them aprove by others before building the actual game.

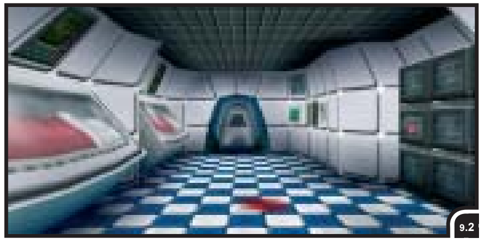
Other 3D content:

- *3D presentations*
  Presentations can be accompanied by (interactive) 3D elements. You can think of 3D diagrams which change over time or when you click on them. This and other 3D enhancements can be included within multimedia/presentation packages like Powerpoint™ and Director™.
- *Architectural walkthroughs*
  As the name suggests, architects can make their buildings in 3D and 'walk' trough it interactively. Clients can see their house from the inside, see how the sun light falls in a room etc. before it is actual build.
- *3D site navigation structures*
  You can also think of navigating trough a website with a 3D menu or you can go even a step further, a virtual room where you can walk around and finding the information you are after.
- *3D advertisement banners*
  Instead of the 'standard' gif animations used on web sites, it is possible to make the banners 3D. This is not a real interactive application, but it is possible to make it interactive with the addition of special placed links within the 3D environment.
- *Simulations*
  Simulations can be made of real life events to show what can happen or what has happened. For instance a car accident can be simulated to show in a clearer way what really happened. In this case the interactivity to be built is how the different cars and their environment will interact with each other.

9.3 Example games

What follows now is an indication of what sort of games are possible to make with Blender and must - at least - be possible to make with the new graphical interface to be designed.
Essentially any kind of game can be made with Blender, but there are some restrictions on the graphical output, the physics engine and the game logic. Graphically not every aspect of most recent games is available in Blender's game engine. There are, for instance, no reflections, particle effects (explosions) and transparency (limited). The particle system mentioned in section 4.2 (4th


9.1


9.2

9.3

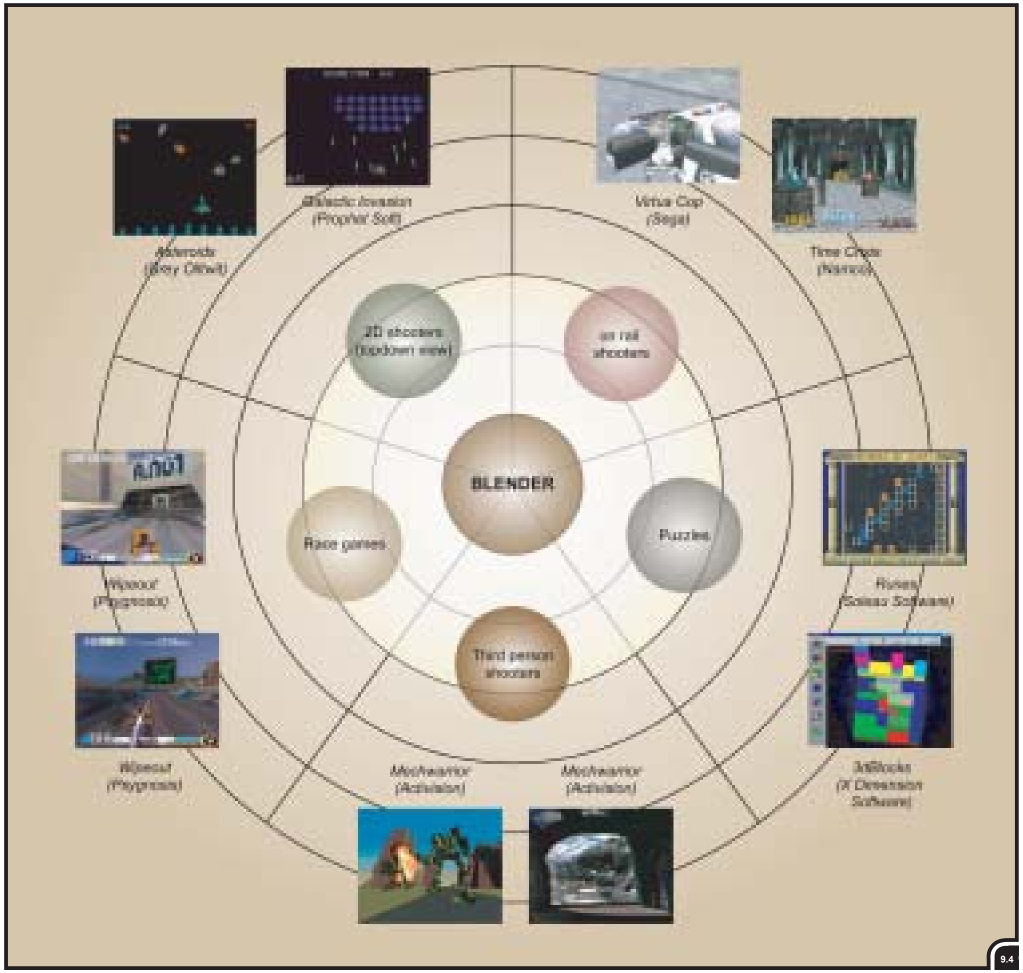
fig. 91 - 9.3
On rail shooter: Hardmashed, made by the content creation team of NaN.

item) does not work wothin the game engine, but is only for rendering animations or stills.
The physics engine has a 'general' nature as mentioned before in section 7.1 (problems/difficulties), which makes it hard to build specialised games. And to make more complicated games (with behaviour etc.), a lot of programming in Python is needed

Despite some lack of features, attractive games can be made with Blender. To give an indication, some examples are shown in figure 9.4. There is made a distinction between the following games, which could have been made with Blender:

- *2D shooters (top down):*
  This are games with a 2D view, but the objects are 3D. The examples given are 'older' games and made with 2D sprites instead of 3D objects.
- *On rail shooters:*
  'On rail' means that the player is not walking on his own, but goes from one predefined location to the next when he finished shooting his opponents. The examples given are 'Virtua Cop' and 'Time Crisis'. The content creation team of NaN made also his own 'on rail' game: Hardmashed (see fig. 9.1, 9.2 and 9.3)
- *Puzzles:*
  With puzzles you can think of 3D 'tetris-like' games, maze games etc. The players has to solve the puzzle before the game continuous to the next level.
- *Third Person Shooters:*
  TPS like mechwarrior. The player follows the main character from a third person standpoint of view, instead of looking trough the eyes with a First Person Shooters.
- *Race games:*
  Any kind of race game; from futuristic (Wipeout) to more regular.

Why are this good examples?
The examples shown in figure 9.4 are all games that can be made within Blender. Some are very old (Asteroids), but they represent what is possible with the game logic within Blender without coding to much in Python. All example games given here are games without real 'intelligent' opponents, what would be the case with a soccer game or a quake alike shooter with a lot of 'intelligent' bots.



9.4

fig 9.4
Example games for Blender.

9.4 Game elements

Below a description of game elements can be found that have to be taken into account building games mentioned.
The game elements are all the individual parts that together make up a game.

- *Story:*
  The story is not an asset like the other elements mentioned here, because it is not something tangible.
  A story can make or break a game. It is (obviously) not enough on its own, but it will help creating a good game. Not every game needs a good story. For instance a car racing game does not need a good story at all, but elements like car handling (game play), sounds and opponent AI will make such a game a great game, if implemented right.
  But the story creates the atmosphere in which the game will play and defines the backgrounds for the characters.

- *Characters:*
  These are all the 'living' creatures that inhabit the game. From the main character (from first person or third person of view) to all the opponents.

- *Environment (levels):*
  This is everything from buildings, rooms, trees, stairs, furniture etc. Basically all the 'dead' or static elements within a game.
  A game is (normally) divided into different levels. Each level has its own environment and specific objects that define the level.

- *Sounds:*
  A game is not finished without sound. Sound is as important as nice graphics. With sound a game comes alive. Think of sounds for footsteps, shooting, explosions, water (fountains), environment (suspense) etc. Although sounds are not visible, they can have a location within a game environment. These are '3D sounds'. For instance, when the character is walking towards a fountain the sound of the running water has to become louder and louder when the character comes closer to the fountain.

- *Animations:*
  The environment and of course the characters need to be animated to come alive. For instance doors have to go open and elevators have to go up and down. Concerning the characters, these have to walk, walk upstairs, shoot, crawl, swim, die etc. All these different 'states' have to be pre animated.

- *Cut scenes:*
  These are small (pre-rendered) animations (movies) between different levels to tell the story and to connect the different levels. Cut scenes can deepen the atmosphere of the game when implemented right, but are often assumed as interruptions of the game play, because the player can only watch and cannot interact with what he sees.

9.5 Levels of interactivity

Interactivity is no game element in the strict sense, it (only) makes it possible to play the game. It defines the scope a player can interact with the game elements. Links are made between elements that defines the 'game play'.

Interactivity can be divided into different levels of 'complexity'. These levels are described below. Separations between these levels are not always that obvious and can be interpreted in different ways. The conception of how these levels of interactivity have to be subdivided are derived from NaN itself. These levels from - 'simple' to 'complex' - are :

Logic
↓
States
↓
State engines
↓
Behaviour
↓
Artificial Intelligence (AI)

- *Logic:*
  'Logic' is the lowest level of interactivity. In this level the basic interactivity is assigned to the different game elements. For instance, assigning a key to a object to let it move or assigning a sensor to a door to let it be opened when something comes near.
- *States:*
  'States' are the different conditions a character can be in. A character can have walk, run, bump into a wall, fall, jump, shoot, die, wounded etc. states. All these states are used to play (trigger) different animations.
- *State engines* (see note 1)*:*
  'State engines' are advanced or automatic 'states'. These 'state engines' describe the transitions between different states at specific moments or situations. For instance, when a character bumps into a wall, the character has to stop walking, even as the player still pushes the forward (walk) button, so the character must transit from a 'walking' state into a 'stand still' state.
- *Behaviour:*
  'Behaviour' is harder to describe. It is hard to say when a 'state engine' becomes 'behaviour'. Behaviour assumes some kind of intelligence or at least that game opponents (re)act in a way like they are intelligent. This means that game opponents will behave different when they are in a different 'state'.
  An example of 'behaviour' is when a guard will start shooting on a player when this player enters the guard's sight of view. Or when an alien only hunts down all the yellow gnomes and leaves the red ones for what they are.
- *Artificial Intelligence:*
  AI is the highest level of interactivity. AI is the name for decision-making techniques like: decision trees, fuzzy logic, neural networks, genetic algorithms and others. These techniques are used to develop group (team) 'behaviour'. This means that different game elements will react to the 'behaviour' of other game elements. This all is used in RTS games, squad maneuver for FPS (pathfinding), team play for a sports game or for deciding what are tactical good locations for building a base (terrain reasoning) etc. [9].

9.6 Conclusion

In this project, the graphical interface for creating interactivity facilitate at least building the example games mentioned in section 9.3, but in a more fluid and neatly workflow than is possible with the current interface.
Concerning the levels of interactivity, the highest level that must be supported is the level of 'behaviour'. The level of AI is too specialized, too much coding would be involved creating AI. This means for the graphical interface that AI will not be included, but left to scripting. But maybe it is possible to include 'autonomous' packages of logic like pathfinding, flocking etc. that can be assigned to game elements within a game to provide a means of AI within the interaction creation part of Blender.

(1) Strictly speaking, a 'state engine' should be called a 'state network'. A state network executed by an engine. However, to the user a 'state network' can be explained as a representation of the 'state engine'. Throughout the rest of this report the term 'state engine' will be used.

10. Blender compared to other applications

How does Blender compare to other 3D (interactive) content creation applications? First, to give an impression, Blender is plotted with other applications against the 'development process' described earlier in section 8.
Second a more detailed description of other 3D interaction creation applications is given; their functionality, advantages and disadvantages. But also programs that are not intended for building interactivity but use a graphical interface to provide an overview or way of organizing objects.
The reason why these other interactive creation applications are described here, is to obtain information and inspiration about means of creating interactive content.

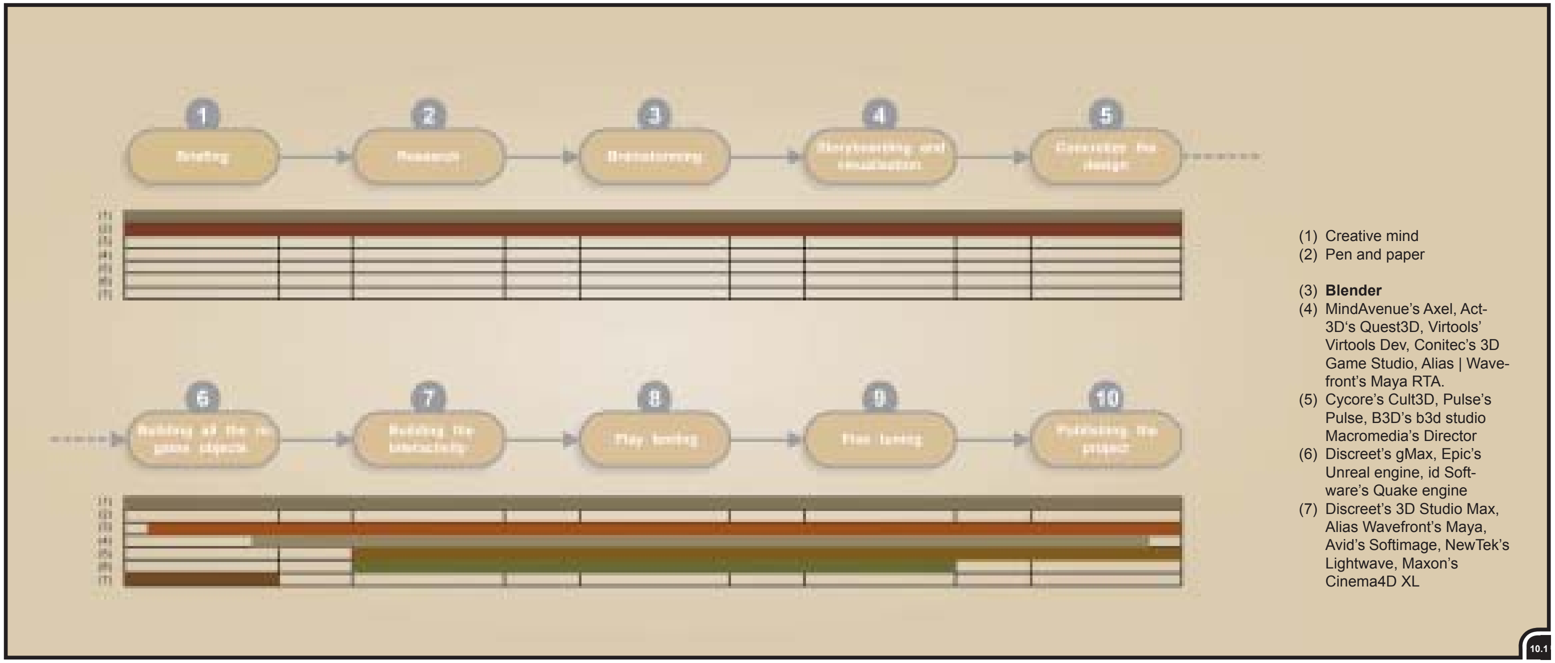
10.1 Overview

In figure 10.1 Blender and other 3D (interaction) applications are plotted against the 'development process'. This is shown to give an indication of the steps taken during the (game) development process Blender and the other applications can be used. The following five categories can be distinct. The numbers indicated after every category can be found in figure 10.1.

- Blender as reference (3).
- Other 3D interactive content creation applications like Blender. The building and animation features of these applications are (most of the time) less extensive compared with Blender. But these applications have the opportunity to import 3D data from more specialized 3D applications.
  Alias|Wavefront's Maya RTA (Real Time Author) can not completely be compared with applications like Blender, because it is a sort of extension (plug-in) for Maya. It allows to built and edit interactivity within Maya and when finished to export the data to Macromedia's Director, which on his turn makes it possible to publish the project (4).
- Applications mainly or alone for the creation of (interactive) 3D *web* content. All of these applications need another (specialized) package to build the geometry (5).
- Game level editors. A lot of game engines provide a editor to make your own or adjust existing levels within a existing game. Discreet's gMax is maybe a bit different from the rest, because it provides the functionality of Discreet's 3DStudio Max (interface, polygon editor, texturing, animation editor etc. are the same), so it is possible to make custom levels, characters, vehicles etc. for existing game titles, but only for games which support gMax. Games that already (or will) support gMax are: Quake III, Flight Simulator 2002, Dungeon Siege and Microids (6).
- Specialized 3D creation, modeling and animation applications. These applications can not build interactivity. They often provide an option to export the data to applications that are capable of building interactivity (like the export option to the Shockwave 3D format for Macromedia's Director) (7).

10.2 Other 3D interactive creation applications

First, other 3D interactive creation applications that can be compared with Blender in functionality are described. A description of their functionality is given and a list of advantages and disadvantages.
Second, other (parts of) applications that make use of a graphical manner of creating and organizing content will be described in section 10.3.
Finally some other 3D web creation tools with less



(1) Creative mind
(2) Pen and paper

(3) **Blender**
(4) MindAvenue's Axel, Act-3D's Quest3D, Virtools' Virtools Dev, Conitec's 3D Game Studio, Alias | Wavefront's Maya RTA.
(5) Cycore's Cult3D, Pulse's Pulse, B3D's b3d studio Macromedia's Director
(6) Discreet's gMax, Epic's Unreal engine, id Software's Quake engine
(7) Discreet's 3D Studio Max, Alias Wavefront's Maya, Avid's Softimage, NewTek's Lightwave, Maxon's Cinema4D XL

fig 10.1
Blender compared to other packages, plotted against the 'development process'.

extensive possibilities for creating interactivity (compared to Blender) will be described in section 10.4.

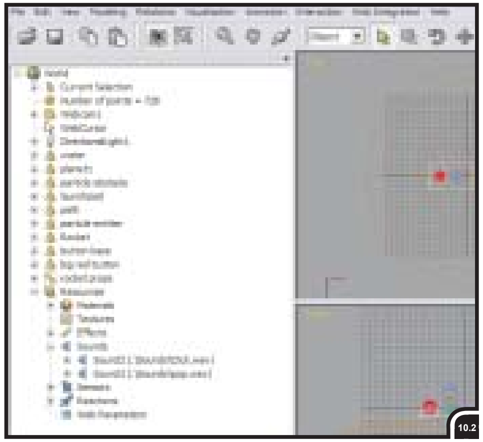
10.2.1 MindAvenue's Axel

Axel is a program that is dedicated to creating and publishing 3D interactive *web* content. It provides a set of (basic) modelling, animation and interactivity tools. To overcome the basic modelling tools, Axel provides (only) VRML import. To view on-line content created with Axel, a special Axel plug-in is needed.

Now the question is: how does the 'interactivity editor' work, what is its functionality?

Axel has three kind of interaction 'types':
- Sensors
- Reactions
- Handles

With sensors it is possible to trigger reactions. The following different sensors are available:


10.2

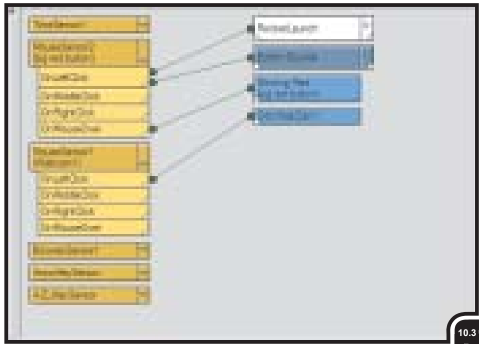

10.3

fig. 10.2
The 'world explorer' within mindAvenue's Axel.

fig. 10.3
MindAvenue Axel's 'interaction editor'. Within the left row the 'sensors' and in the right row the 'reactions' and 'handles'.

- Mouse
- Time
- Proximity
- Position
- Orientation
- Parameter Change
- Parameter Range
- Time Pulse
- Download
- Refresh rate
- Keyboard (two different kinds: A-Z key sensor and Arrow key sensor)

Reactions are actions that occur when they are triggered by a sensor. Reactions can be animations, sounds, color changes, and so on. They can be triggered by mouse overs, time marks, key strokes, and so on. The following different reactions are available:
- Play animation
- Play sound
- Set parameter
- Toggle parameter
- Set WebCam ('WebCam' is the name Axel uses for the camera)
- Set Pose
- Set material
- Set geometry
- Tooltip text
- Hyperlink
- Download Axel stream
- Go to Time
- Control Time
- Call JavaScript

With the addition of handles to objects it is possible for users to manipulate these objects. Handles can also be added to the webcam (camera) so that users can navigate in and around the 3D world. When a handle is added to an object or to the webcam (camera), a reaction and a sensor are automatically created and linked. The following different handles are available:
- Translate
- Rotate
- Scale
- Push
- Bend
- Orbit WebCam
- Rotate WebCam
- Translate WebCam
- Zoom WebCam

Advantages:
- Axel has a 'world explorer' (see fig. 10.2). In the world explorer all the objects and their settings, materials sensors etc. are shown. From within this window you can select everything you want.
- The 'parameter editor' is a content selective window. It shows all the 'parameters' of a selected object or - most useful - all the common parameters if more than one object is selected so these can be edited all at once. It also shows the properties of selected sensors, reactions and handles.
- To add more functionality or advance interaction it is possible to use the 'Call JavaScript' reaction. This can also be seen as an disadvantage, because you have to know Java, but for a better integration within a web page it can be good to have.

Disadvantages:
- The interaction editor is divided into two fixed columns (see fig. 10.3). In the left column all the sensors are located and in the right column all the reactions and handles. The position of the sensors is alphabetical and fixed.
- The interaction editor shows always all the sensors, reactions and handles used in a production. There is an option to only show the selected sensors, controllers and/or handles. But this is of no use at all, because you do not have an overview anymore and all the properties editing is done outside the interaction editor. So it is not necessarily to take apart a (or group of) 'logic brick(s)'.
- The 'parameter editor is an advantage, but at the same time a disadvantage: it is the only place where properties can be edited. So it is not possible to edit settings of the 'sensors', 'reactions' and 'handles' directly in the 'interaction editor'.
- To create a new sensor, reaction or handle, a object, parameter, material, animation etc. - depending on what has to be triggered - has to be selected first or non of these can be created.

10.2.2 Act3D's Quest3D

With Quest3D it is possible to create interactive, realtime 3D applications. It can produce standalone animations, visualizations, screensavers, web content and games. Quest3D has no facility for creating object meshes or textures itself. For this you need a separate application. To view the content on-line, a special Quest3D plug-in is needed.

How does the 'interactivity editor' work?

Everything in Quest3D is represented as a 'channel'. A channel can store data or perform an action or both. For instance, a sound sample would be one channel, a texture would be another and a command to rotate an object might be a third.

Quest3D comes with many predefined drag-and-drop channels. These predefined 'channels' are essentially C++ functions and often only wrappers for DirectX functions. The interface is divided into three different sections: the channel section, the object section and the animation section. The channel section is the place in where the interactivity is built. To give an overview of what is possible with Quest3D, a list of some channels is given. A predefined channel is often a set or group of other channels. Some channels need another channel before they can perform their function. For instance, a channel that can clear the screen (redraw) needs a color channel before it can clear the screen. This color channel exists of three values: R, G and B. In Quest3D these color values are children of the clear screen channel. In effect, channels are simply functions and procedures and the connections represent function calls and parameter-passing between them. To connect two channels, a line is dragged from the bottom of one channel to the top of another, but the connection will be made only if the channels are of the right type to supply valid data to one another.

Here are some of the available channels:
- DX8 3D Object:
  This channel joins a set of other channels to form a channel that can be displayed by rendering it with a DX8 3D Render channel.
- Text:
  This channel can store any text (strings). This channel is the base channel for many text operating channels.
- MessageBox:
  This channel will show a Microsoft Windows style default Message Box.
- DX8 Collision Response:
  The DX8 Collision response is a channel that can replace the position vector of an object or a Camera. The collision response makes it possible to let a camera or object react as expected when a collision occurs.
- DX8 Light:
  This channel will add a light to a scene.
- This channel holds sound data and it plays sounds when a command is send to it. Commands to this channel are send with the DX8 Sound command channel.

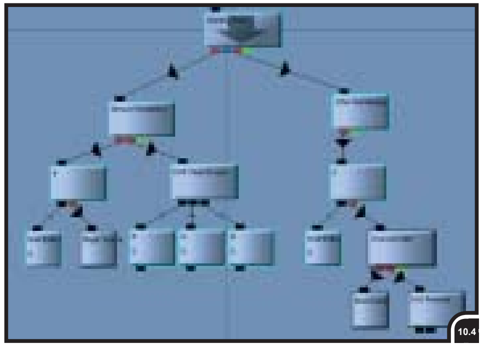

10.4

fig. 10.4
Channels connected within the channel-graph view of Quest3D.

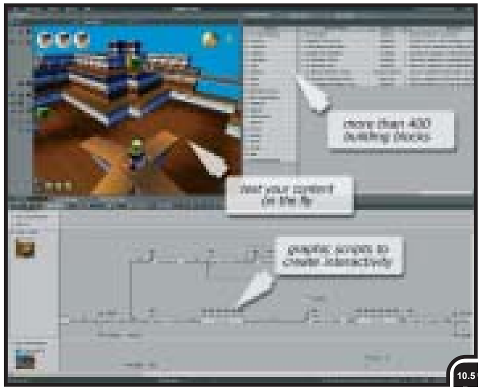
There are some 95 available predefined channels.

Advantages:
- Channels can be combined into groups to collapse the view. To view which channels are in a group, simply double click on this group and a new view with all the channels in it will be shown.
- Quest3D allows only connections between two channels if they are both of the right type.

Disadvantages:
- Essentially, building interactivity with the channels is still programming and it requires a good understanding of the DirectX API to use well.
- For a relative small project a huge number of channels is required, because everything is a channel. So this turns the channel-graph view (editor) into a spidery mess.
- The channels are grouped by type and within these groups there are sub-groups. The layout is far from clear. It is very hard to find the channel you are looking for and to select a channel, a lot of groups have to be clicked open.

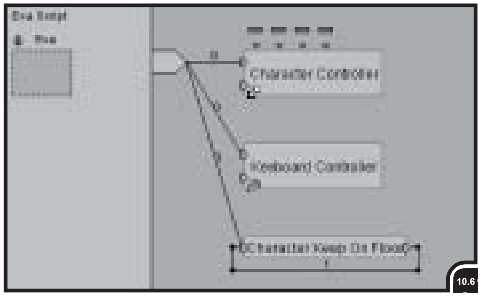
10.2.3 Virtools' Virtools Dev

Note:
Virtools Dev and Maya RTA (described in sections 10.2.4) were not actual evaluated in this project. The information to give an description is obtained from different resources like internet and magazines (3D World).
The reason they are described here, is the fact that they seem to be very interesting applications to build interactivity with.



10.5

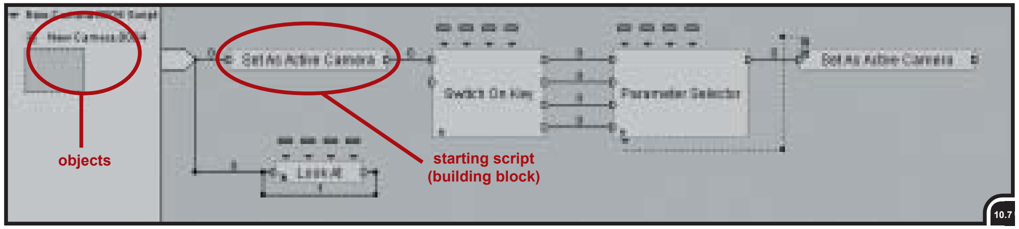
fig. 10.5
Full interface of Virtools.



10.6

fig. 10.6
'Building blocks' within the shematic view.

With Virtools it is possible to create interactive, realtime 3D applications just like Axel and Quest3D. The difference is that Virtools uses "behaviors" as a sort of 'building blocks' (or 'logic bricks'). To create interactivity within Virtools, a starting 'building block' has to be selected from a list of almost 400 predefined scripts and must be assigned to an object. This 'building block' has to be dragged over an object in the 3D view to assign it or has to be dragged directly into the schematic view (interaction editor). When released over an object, the starting script for this object is added to the 'schematic' view (see fig. 10.7). In the schematic view all the objects with a 'building block' assigned are listed on top of each other on the left of the screen. This list of objects can be



10.7

fig. 10.7
Another example of the shematic view with 'building block'.

collapsed. When a object in this list is unfold, the building blocks are shown for this object from left to the right (see also fig. 10.7).
Just like Quest3D, Virtools has no facility for creating object meshes or textures itself, another application has to be used.

Some of the predefined 'building blocks' (scripts) that are available within Virtools are:

- Set as active camera
  Sets the selected camera as active camera.
- Look at
  This scripts makes it possible to let the active camera always look at an object it is attached to.
- Character controller
  When this 'building block' is dragged over an object (character) a parameter window appears in where animations (created earlier) can be assigned to different states of the object. These (standard) states are: stand animation, walk animation, backward walk animation and run animation (see fig. 10.6, previous page).
- Keyboard controller:
  This 'building block' makes it possible to use the arrow keys to navigate a character. Assigned to the character, it translates the arrow keys that are pressed into messages. These messages are received by the character controller and executes the animation corresponding to that message.
- Character keep on floor:
  This 'building block' will cause a character to remain on the floor. To let this script work the 'behavioral engine' must know that an element is to be treated as a floor. So only an object has to be marked as floor.
- Object slider:
  The object slider 'building block' causes a character to collide with objects. To let this work, the object slider requires, just as with the keep on floor building block, that elements to be treated as obstacles are selected and assigned to this 'building block'.

These are just a few examples of the available 'building blocks'. As mentioned before, there are some 400 predefined 'building blocks', but it is also possible to program your own "behaviors" with Virtools Development Kit (SDK). But this will (definitely) not be an option for the non programmers, because this has to be done in C++.

Advantages:

- It is possible within Virtools to directly drag a 'behaviour' (building block) onto an object in the 3D view.
- It is possible to create new and reusable behaviors by graphically combining existing ones and save them as a new 'building block'.
- Visually drag-and-drop 'building blocks' onto objects directly in the 3D view.

10.2.4 Alias|Wavefront's Maya RTA

Maya RTA is a plug-in for Maya. It adds he RTA (Real Time Author) interaction editor to the interface of Maya. The 'interaction editor' allows the creation and connection of sensors, actions, and viewers. The 'interaction editor' workflow is similar to Maya's HyperShade editor where shader networks are created (see section 10.3.1).
With the 'interaction editor', the networks that are built control scene behaviors and ways for the user to interact with the 3D content.

Sensors can trigger actions, which are events that will (only) occur when played back within the final Shockwave application (Macromedia's Director). This means that when an environment is built and has to be tested, it first has to be exported to Director.

The following sensors are available:

- Touch:
  Detects the user touching an object within the 3D scene. The user selects the object from the list of geometry meshes within the scene. Touching can be defined in a variety of ways including clicking on with cursor and mouse, or simply rolling the cursor over or off of the specified object(s).
- Keyboard:
  Detects the user pressing or releasing a specified keyboard key or keyboard key plus modifier (Shift, Alt, Ctrl).
- Mouse:
  Detects the user pressing or releasing a specified mouse button.
- Proximity:
  Detects the current camera moving into or out of a volume type selected by the user and placed within the 3D scene. Proximity sensor volume types are cube, cylinder, and sphere. They can be transformed (translated, rotated, scaled), parented, and animated as necessary.
- Actions as sensors:
  The completion of most RTA actions can be used to trigger subsequent and additional actions.

Actions are Maya events that are triggered by the sensors. The following actions are available:

- Bind Camera:
  Switches the 3D display to the view from a specified camera. The camera may have a pre-choreographed animation, or it may be the camera associated with a viewer (see below) which allows the end user to have control of its position and orientation.

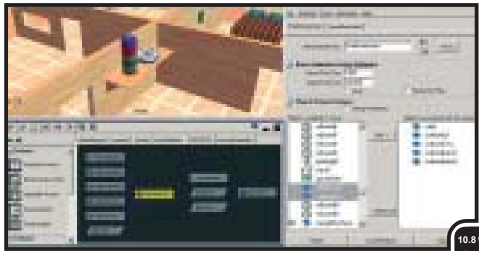

10.8

fig. 10.8
Maya interface with down left of the screen the RTA interaction editor.

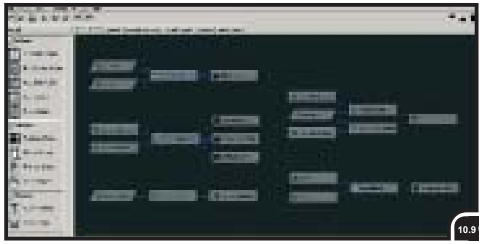

10.9

fig. 10.9
The RTA interaction editor.

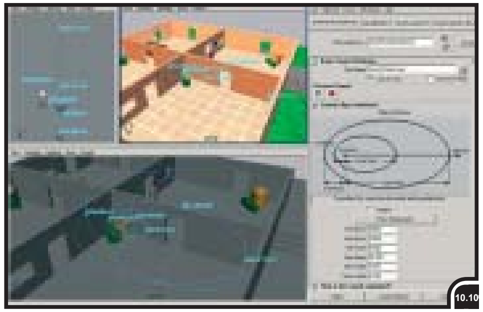

10.10

fig. 10.10
Visual editing of a spatial sound. Handles for min. and max. distance can be dragged in 3D perspective view.

- Timer:
  Starts a timer. The completion of the timer action is generally used to trigger subsequent actions.
- Animation:
  The animation action causes the motion of a specified set of objects over a specified range of frames to play. The completion of the animation can be detected and used to trigger subsequent actions. Animation actions can be looped.
- Sound:
  Plays a specified sound file. Sound can be ambient or spatial. Spatial sound is controlled by a Maya manipulator that depicts point of origin and extent in 3 dimensions. The completion of the sound action can be used to trigger subsequent actions. Sound can be looped.
- Hyperlink:

Launches a specified web page when triggered.

- Lingo Script:
  This is an action provided for use by Director Lingo programmers. It allows the Maya artist to put sensors in a Maya scene which are set to trigger Lingo scripts provided by the Director user.

Viewers are types of cameras and controls that can be placed into a scene and exported to Director and the Shockwave Player. The following viewers are available:

- Walk Viewer:
  The walk viewer provides first person 'game' type camera controls that allows a user to walk through a 3D environment. Control attributes provided to the Maya user are speed, step height, and collision distance. Speed is the speed with which the camera moves through the 3D scene. Step height is the height limit over which the camera will move. Above that height it will be blocked. Collision distance is the distance at which the camera bumps into obstructions.
- Tumble Viewer:
  The tumble viewer provides Maya type camera controls for viewing and interacting with a 3D object.
- Moded vs. non-moded Viewers:
  The option to make viewer cameras 'moded' or 'non-moded' is provided to accommodate a range of content complexity and a range of prospective users. The moded version can be used for more complex scenes. In these cases the user explicitly determines that the camera will be moved and hits a specified control key to enter camera control mode.
  The non-moded viewer option puts the camera immediately into control mode. As soon as the user clicks in the 3D scene, camera control is initiated. This moded viewer can be used with less complex 3D scenes.

Conclusion:
It is hard to describe advantages or disadvantages for Maya RTA without working with it. The only reason why it is described here is that in conjunction with Macromedia's Director it can be a good 3D interaction creation bundle. I only wanted to describe its functionality.
With Macromedia's Director on its own, it is hard to create 3D interactivity. All the interactivity has to be coded from scratch in Lingo (Director's own scripting language). In combination with Maya RTA all the modelling, texturing, animation and interactivity can be done within Maya. After testing the interactivity within Maya, all can be exported to Director to publish it from there for use on the web or as a stand-alone production. The output that is created by Maya RTA for Director is all in the Lingo scripting format. So these scripts are accessible within Director and can be adjusted and optimized. It would be nice if these adjusted and/or optimized scripts can be read back into Maya, but at the time of writing this is not clear if possible.

10.3 Other graphical creation tools

Note
The next two descriptions - Maya's Hypershade editor and Softimage XSI's Render tree - are not applications to build interactivity with, but parts of 3D applications. The reason they are described here is the fact that they both have a graphical interface for the workflow of connecting different building blocks (nodes) to create an objects material appearance (shader network).
This can be interesting for this project, because it can provide inspiration how to handle the workflow for the 'interactive window' within Blender.

10.3.1 Maya's Hypershade editor

As mentioned in the previous section, the Maya RTA 'interaction editor' workflow resembles very much the workflow of Maya's Hypershade. The Hypershade is the place within Maya, materials (shader networks) are created.

How does it work building these shader networks?

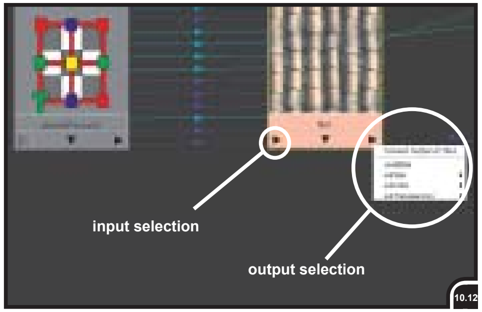


fig 10.112
Close-up of connected nodes within the Hypershade window.

First, some nodes (blinn, texture, bump map etc.) are dragged from the left list pane to he right into the 'work area' (see fig. 10.11, below). The nodes can be freely moved around in the work area and it is possible to zoom in and out.
After the nodes are dropped in the 'work area', they have to be connected to each other. To connect two nodes it is possible to middle-mouse-button drag a node onto another node, a window appears with a list of specific input attributes of that node it can be connected to. Another method is to click on the output selection arrow (see also fig. 10.12), choose which specific output of the node has to be connected and than click on the input selection arrow of another node to select the input it has to be connected to.
Most of the time it is enough to middle-mouse-button drag a node onto another and the connection is made automatically when there are no specific in- and output attributes. For instance, when a 2D (or 3D) texture node is dragged onto a material swatch (blinn, phong etc.) it is automatically connected to the color attribute of it. If more than one connection is made between nodes, more connection wires are visible. To change specific attributes of a node the 'attribute' editor has to be opened. This window is laid over the interface, when another node is selected the content of

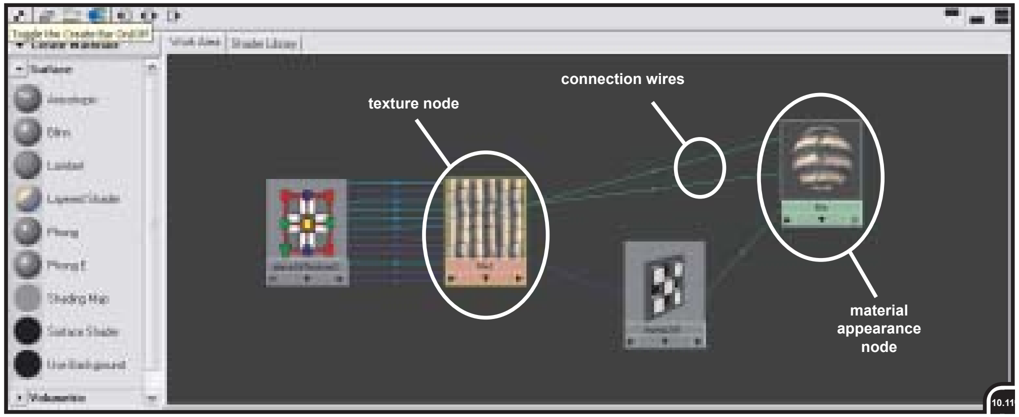


fig. 10.11
Maya's Hypershade window.

the ‘attribute’ window is changed according to the settings of the new selected node.

When creating a material is done, it is possible to clear the ‘work area’. Only the ‘material appearance node’ is visible in the ‘materials’ window. This window shows all the created materials. Now when a material has to be edited afterwards, it can be dragged again into the ‘work area’ in where it will unfold again, showing all the nodes instead of only the ‘appearance’ node as in the ‘material ‘ window.

10.3.2 Softimage XSI’s Render tree

Softimage XSI’s Render tree is just like Maya’s Hypershade. The graphical interface for creating the appearance of materials.
The Render tree is a tree of shaders (see fig. 10.13, below), where the output of one shader is connected to an attribute of another shader.
To create a Render tree, first the different nodes have to be selected and created from a menu within the Render tree window. Second the nodes have to be connected to each other. This is done by click-and-drag on the red circle next to the node name. A wire with a arrow on it appears. This wire has to be released over another node. When released a list appears with only the attributes it can be connected to. Select an attribute and the nodes are connected.
Another manner is to first unfold a node’s attribute list by clicking on the grey arrow next to the node name. A list with all the attributes of that node is shown below the node name (see also fig. 10.13). Now when a connection wire is dragged from another node, all the attribute names it can be connected to will have a green circle in front of their names. All the attributes to which the specific node cannot be connected to have a red circle in front of their names.

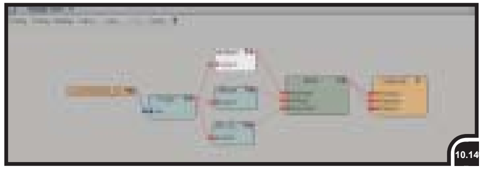


fig. 10.14
The Render tree window with nodes collapsed to only show the connected node attributes.

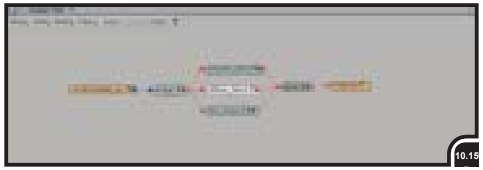


fig. 10.15
The Render tree again, but now only showing the connected nodes (collapse all).

Only one Render tree is shown at a time. The render tree is assigned to an object. Selecting another object causes the Render tree window to show the corresponding Render tree for the new selected object. A Render tree can be build first before it is assigned to an object or more objects at once. A finished Render tree can be saved and used later or in another project.
The make space within the Render tree window the different nodes shown can all be collapsed at once to only show the used attributes by selecting the proper command from a menu within the Render tree window (see fig. 10.14). It is also possible to collapse all the nodes at once to only show the node names of the connected nodes (see fig. 10.15).
Another functions is to ‘update’ the Render tree window, this will reposition all the nodes in a well-ordered way and so that all the used notes are visible at once.

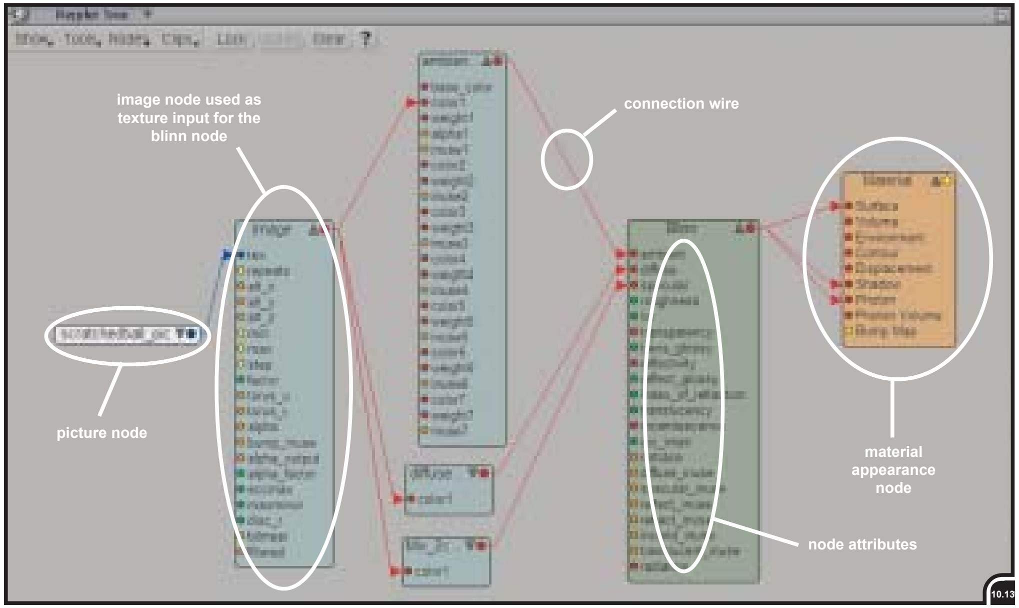


fig. 10.13
Softimage XSI’s Render tree window with different nodes connected with each other. All the possible node attributes are visible.

10.4 Overview of other interactive creation tools

In section 10.2 four different interactive creation tools are described. There are many more tools that do create interactive content in one way or another. As mentioned, the only reason why these four are described, is to obtain information and inspiration about means of creating interactive content.
But to give a better overview of the (many) more applications/tools a list is given with a short description, subdivided in ‘other 3D interaction creation applications’, ‘web 3D creation tools’ and ‘miscellaneous tools’, tools that cannot be placed along the other two categories, but are worth mentioning. It strictly has to be said that this list is not trying to be complete.

Other 3D interaction creation applications

Applications like Blender, MindAvenue’s Axel and Virtools’ Virtools Dev (as described in section 10.2).

- *Discreet’s Plasma*
  An application just like Maya RTA (see section 10.2.4), but instead of being a plug-in for Discreet’s own 3DStudio Max, it is a sort of stripped down version of 3DStudio Max with different features to build and create interactivity and, just like Maya RTA, to export to Macromedia’s Director Shockwave Studio to further adjust and refine the exported code (Lingo) and 3D world within Director.
- *Macromedia’s Director Shockwave Studio*
  Director is a little bit a special case. It didn’t start out as a creation tool for 3D interactivity, but this was added to version 8.5.
  With Director it is now possible to create interactive 3D media for the Web. There is one drawback, within Director itself it is not pos-

sible to create the 3D geometry itself. This has to be done within a dedicated 3D creation application.

- *Robolab (Lego Mindstorms)*
  Robolab is a template-based programming environment providing a interface to give life and function to Lego® robots (robots build with 'real-life' Lego® bricks). Robolab is designed as a string of icon commands. The strings visually describe the response and action of the inputs and outputs.

Web 3D creation tools
These tools have in common that they provide a way of exporting 3D for 'realtime' viewing within a web browser, but they do not have a kind of logic editor. 3D objects have to be imported from a dedicated 3D application. All the mentioned tools need their own web plug-in to view content on-line. These tools are 'perfect' for 3D product presentations on-line, but they are not meant for creating (small) games.

- Viewpoint
- Pulse3D
- B3D
- Cult3D

Miscellaneous tools
- jMax
  jMax is a visual programming environment for building interactive musical and multimedia applications. These applications are built by placing modules on a graphic surface and connecting these modules together with patch cords. The connections represent paths on which values or signal streams are send between the modules. These modules are either processing units (arithmetics, timing etc.), data containers (tables etc.), system inputs and outputs (audio, MIDI etc.)
- LabVIEW
  LabVIEW is a graphical software system for developing high-performance scientific and engineering applications. It consists of a front panel and a block diagram. The front panel (with knobs, switches, graphs etc.) represents the user interface, while the 'real' programming is done within the block diagram, which is the executable code. The block diagram consists of icons that operate on data connected by wires to pass data between them.

10.5 Conclusion

Now, what is useful? What kind of work methods, seen in these other applications/tools, are useful to remember, to use within the 'interactive window' within Blender?

After looking to these other applications one thing becomes clear. The use of sensors, controllers and actuators is not that bad at all. The applications that do use this kind of 'methaphors' do only make use of sensors and actuators (Axel and Maya RTA), but lack the controllers. But especially the controllers provide a higher level of control over the interactivity.

Having worked with Blender creating the interactive House (see section 5) and having compared the interactive part of Blender with other applications, the decision is made to stick with the sensors, controllers and actuators for creating the 'low' level interactivity. Besides the fact that this method competes very well, the fact that a wide user base is already using this system and is already used to it. This is in itself not enough reason to make this decission, but the current logic editor within Blender provides no means for creating 'higher' levels of interactiviy and efforts can better be put in finding an addition to the current logic editor to provide means of 'higher' level interactivity than in finding other work arounds for the lower level part. Of course, the current logic editor is not perfect at all and problems working with it have already been mentioned, but the decision is made to stick with the system of sensors, controllers and actuators.

Finally, providing a kind of ready made drag-and-drop 'behaviors' like in Virtools would be a nice addition to the logic editing. It could provide a means for the less advanced users to get started and to learn from these pre-build 'behaviors'.

11. Other GUI's

The look and feel of the GUI can be very important while working with an application. A 'good' GUI can persuade the user to use it, speed up the workflow and does not distract form the content worked upon. These are the reasons why this section is placed here.
What will follow now is a description of some 'good' GUI's (and what is good about them), on the basis of pictures of applications which do have a 'good' GUI according to the members of the content creation team of NaN.

11.1 Criteria for a good GUI

The surroundings of people working day in day out with 3D software and/or compositing applications is most of the time very dark, with the only light being the monitor. Often more than one monitor is used, posing the danger of dazzling the user.
The three applications mentioned in this section all use little or no color to distinguish buttons and the total color intensity is not too bright.
The reason for this is, probably, the dark surroundings described above. This said, it does make sense that the appearance of the application's interface will be 'dark'.

Figure 11.1 and 11.2: Discreet's Combustion
Combustion is a video effects and compositing application. As can be seen in the pictures, the global color intensity is very low. Everything, including the buttons, does have a dark gray color.
What is good about this GUI? Because of the dark colors used within the interface, the user is less distracted from the content worked upon. In this case the video sequence. All the buttons do have the same dark grey color as the background of the interface. Only buttons that are important at the moment, dependent on the selected tool, do get a distinct highlight to get noticed by the user (see blue color buttons in fig. 11.1 and 11.2).

Figure 11.3 (next page): Chrome Imaging's Matrix
The same as Discreet's Combustion. A dark grey color for the complete interface with the difference that the buttons do not get highlighted but do have a lighter grey color do distinguish from the background.

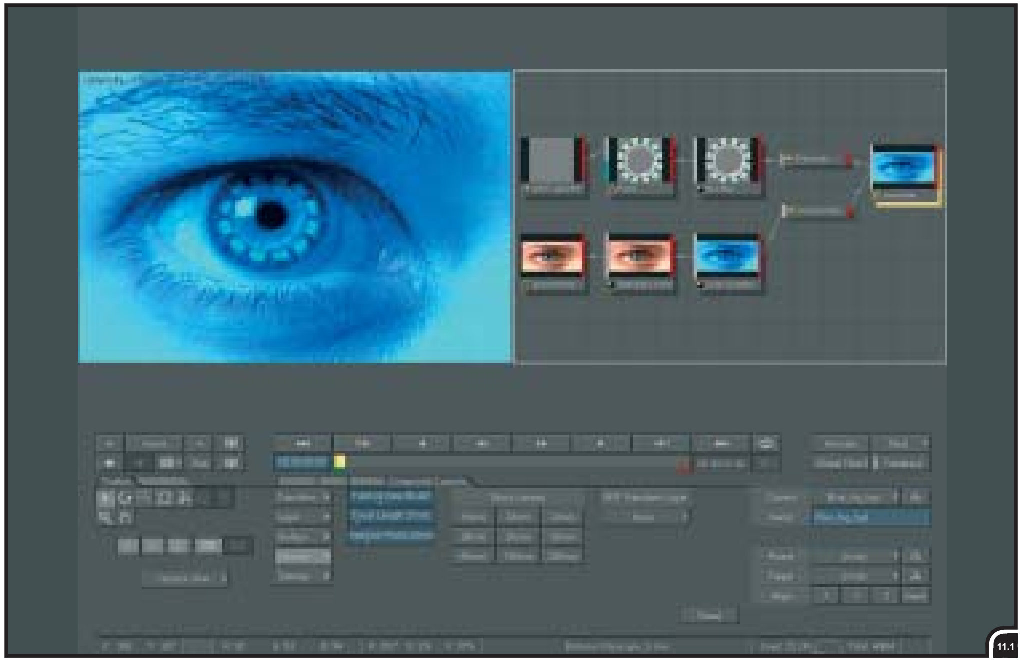


fig. 11.1
User interface or Discreet's Combustion 2 compositing application.

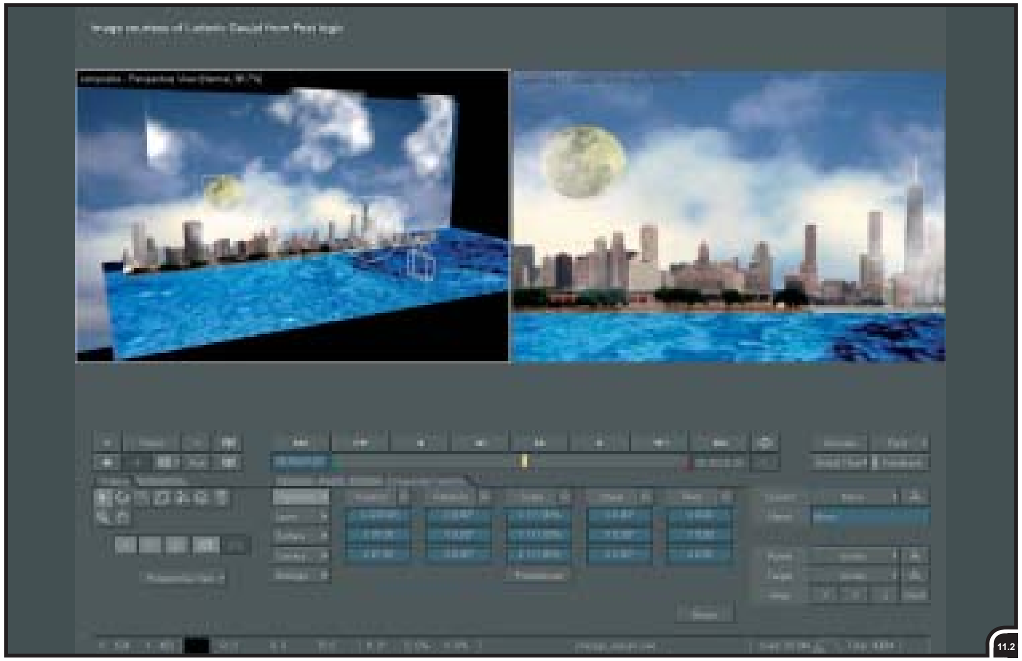
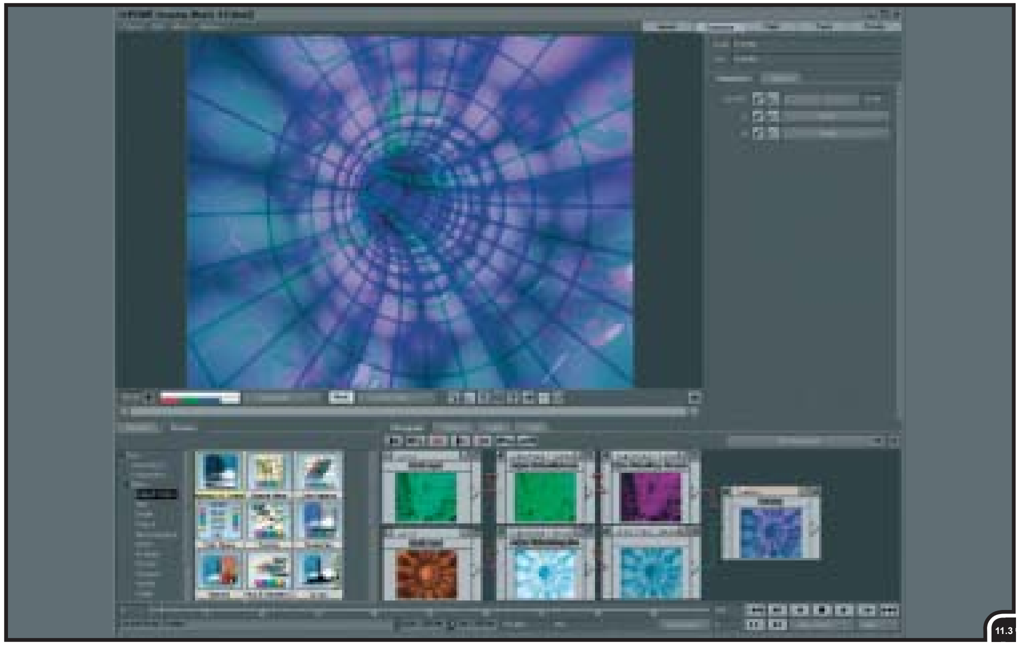


fig 11.2
Another screen of the interface of Discreet's Cumbustion 2.



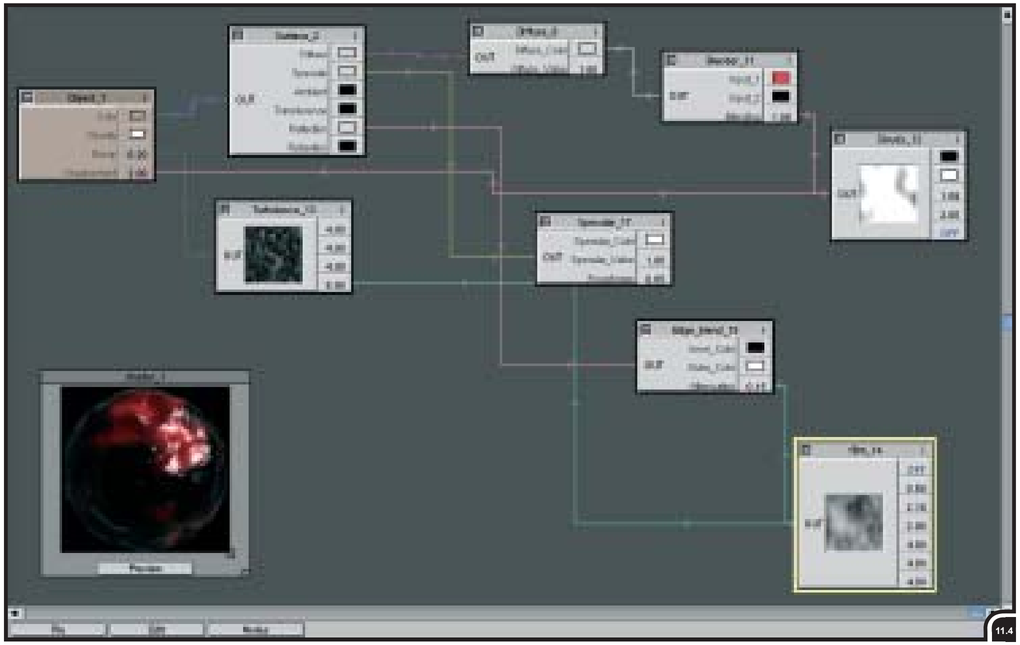


fig. 11.3
User interface of Chrome Imaging's Matrix compositing application.

fig. 11.4
Material editor Interface of Pixels' Pixels3D 3D application.

Figure 11.4: Pixels' Pixels3D
The connection wires between the different 'node' are all straight lines. This can be difficult to follow the connection wires from one node to the other. To ease this, the connection wires do have a different color.

11.2 Conclusion

What criteria will be useful to (re)use within the design of the 'interactive window'?

- First of all the overall color intensity has to be low. The background has to exist of a dark grey color, that doe not distract from the content worked upon.
- Buttons that are important for the user at a certain moment (depending on the selected tool, for instance) must have a different color to distinguish from the rest.
- Connection wires between different nodes, when straight or zigzag, can better be of different color to easier follow them.

12. Program of Requirements

In this section the program of requirements is listed. The requirements (demands) are not quantitative, but will be a sort of guidelines for the rest of this project the graphical interface for creating interactivity has to fulfill.
When 'interface' is mentioned in the requirements below, it means the Graphical User Interface (GUI) for designing interactivity, which will be designed in the next phase of this project.

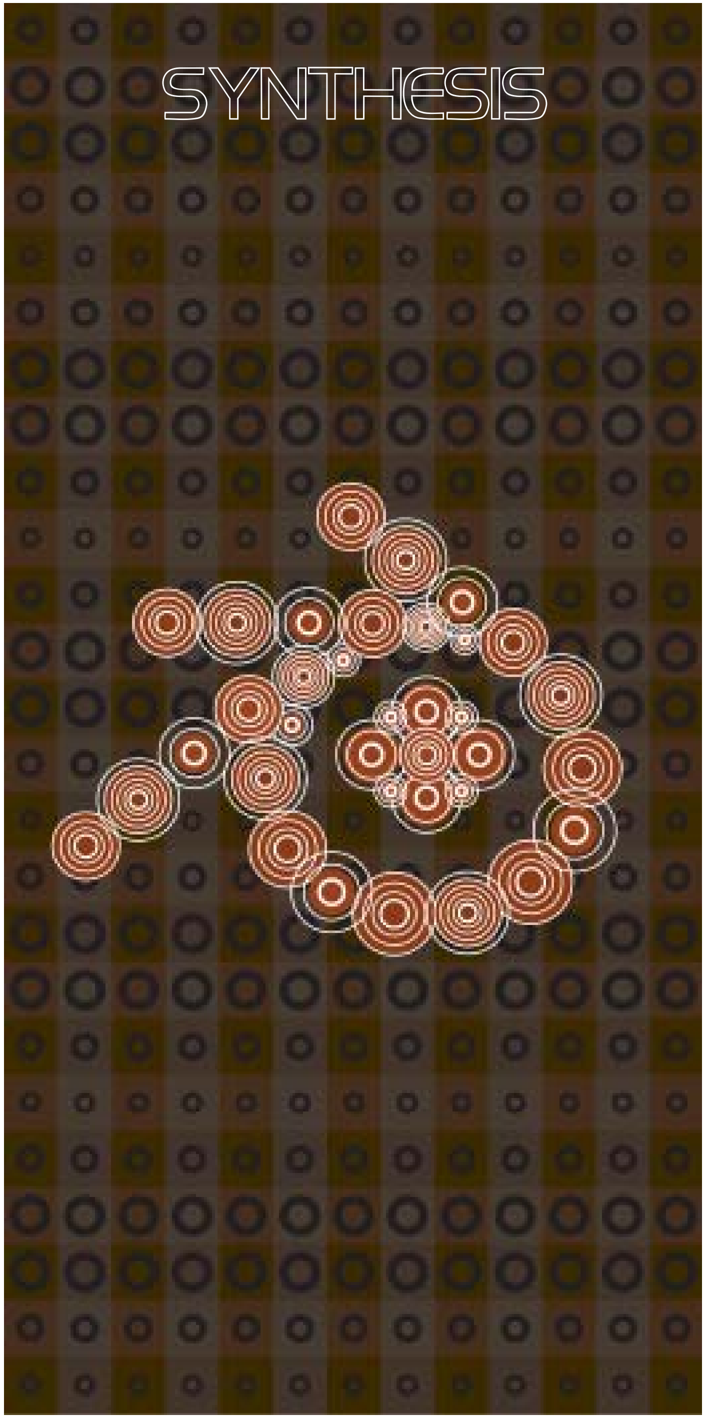
A.1 The 'interface' must integrate within Blender, it should not be a program within another program.
A.2 The 'interface' must support the workflow of the other parts within Blender. For instance, selecting a object in the 3D view provokes also the selection of the logic that goes with the selected object.
A.3 The 'interface' must - at least - contain the same functionality (same logic bricks) as the current user interface for designing interactivity.
A.4 The 'interface' should provide a flexible manner of creating interactivity. For instance, providing a order of creating interactivity that is not necessarily linear.

B.1 The working of all the elements within the 'interface' should be consistence (e.g. the timing).
B.2 Operations throughout the 'interface' must be consistent on all levels.
B.3 The look and feel of the 'interface' must be consistent throughout all the levels of operation.

C.1 The interface must allow users to group different elements of logic.
C.2 The 'interface' must support means to organize the logic.
C.3 The 'interface' must allow to reuse (groups) of logic. Therefore the 'interface' must allow the logic to be disconnected from objects: object independent.
C.4 There must be a way to get an overview (list) of all the elements (game pieces (assets), geometry, logic etc.) in a project and make selections with it.

D.1 All the related functions to the interaction 'interface' have to be able to fit in one 'subwindow' within the 3D view of Blender.
D.2 With different screen resolutions, all the important information provided by the 'interface' must be visible in one overview, whether it is by zoom-in/out, or other means.
D.3 The interface must support the different levels of interactivity: states, state engines and behaviour (as described in sections 9.4).

SYNTHESIS



13. Ideas

After finishing the first part of this project, the analysis, the second part, the synthesis, is described here. The analysis part existed of exploring the problem definition of 'designing interactivity with a Graphical User Interface (GUI)', laying down the needs for such a GUI and defining the field of solution.
The synthesis combines the information obtained during the analysis into a solution for the problem definition.

During the synthesis part all the ideas and concepts have to fulfill the different requirements formulated in section 12. Of course, the ultimate concept has to fulfill all of the requirements, but during the synthesis I will use three requirements (C1, C2 and D3; see section 12) as guides, because these three cover most the main problems of means of organizing logic and creating higher levels of interactivity graphically.
Based on the requirements the three most important questions to be answered during the idea and concept phase are:

1. How can logic be ordered?
2. How can logic be reused?
3. How can the creation of low-level and high-level interactivity be combined into one workflow?

The first outcomes of the three questions were some early ideas and are described in section 13.1. However, these ideas do not address the whole problem, but are starting point for the complete solution

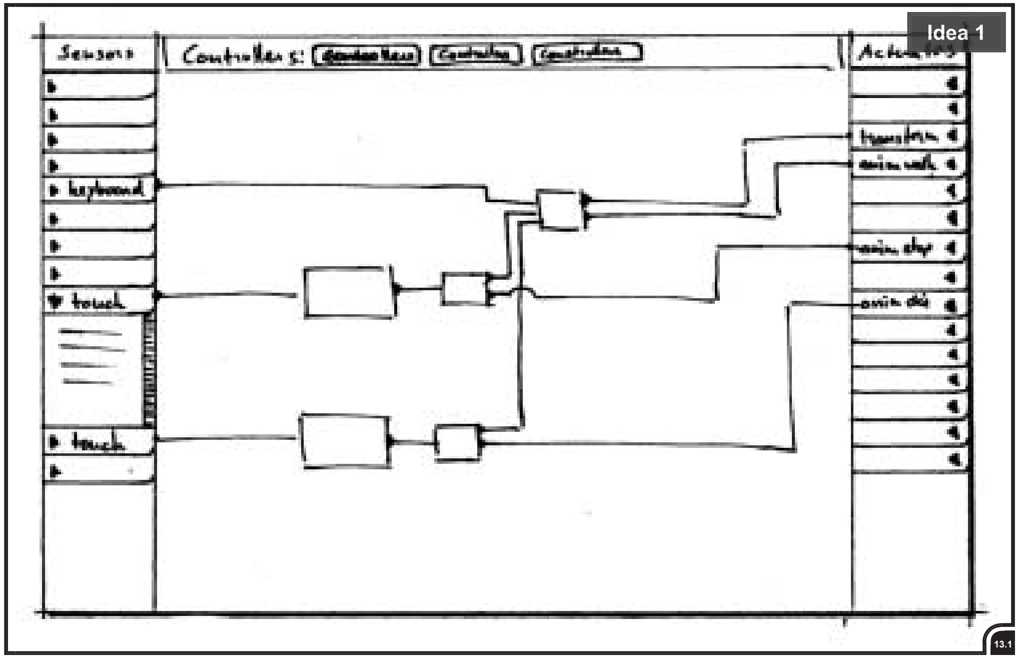
13.1 Result

What will follow now is a description of three different ideas and how they relate to each other.

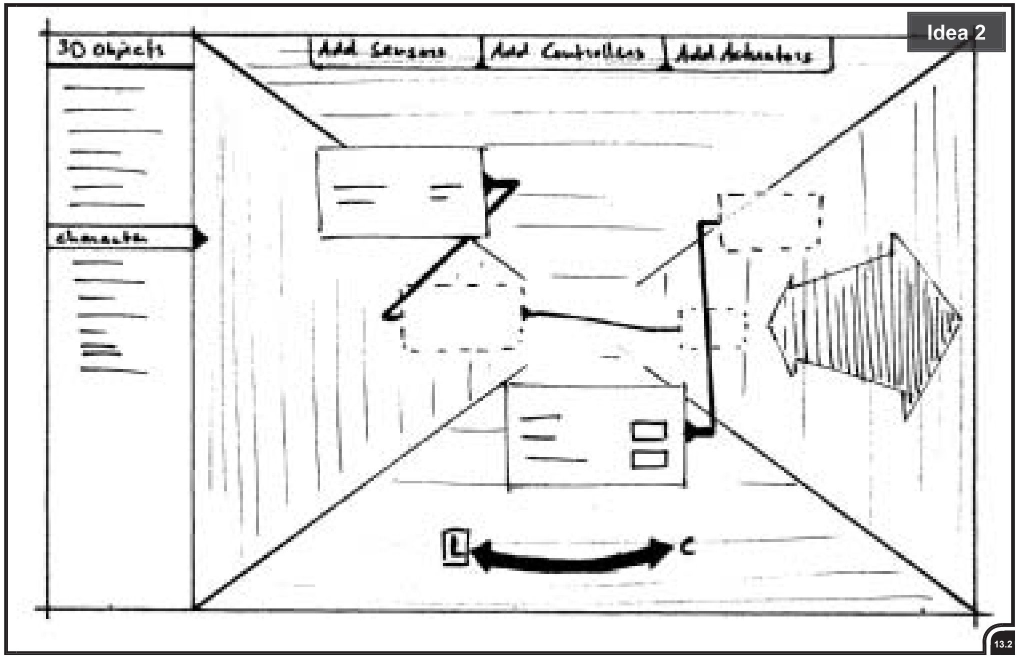
The three ideas described here try to find a solution for the three 'questions' mentioned above. Ideas 1 and 2 try to be solutions for questions 1 and 2: "How can logic be ordered?" and "How can logic be reused?". While idea 3 tries to be a solution to question 3: "How can the creation of low-level and high-level interactivity be combined in one workflow?".

13.1.1 Idea 1

Idea 1 (see fig. 13.1) exists out of two fixed col-


Idea 1

13.1

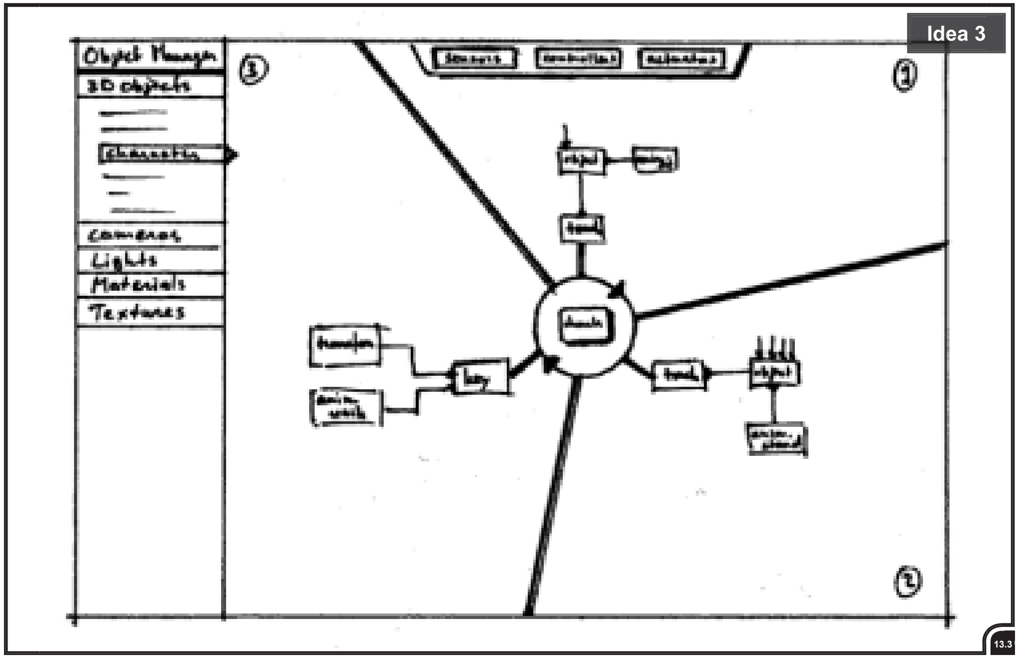

Idea 2

13.2

umns at the right and left side of the working area. In between is the work-space for creating and editing controller networks. The left column lists sensors and the right actuators.
The workflow is quite similar to the current Blender interface. The difference is that the user is free to add/link as many controllers as needed between sensors and controllers in series and/or in parallel. Second the user can move the different controllers freely around within the work-space.
To get an ordered overview of the used logic attached to an object, only the controller(s) and actuators connected to the *selected* sensor will be shown. All the other controllers and actuators will 'fade' away. The other way around works as well; when a actuator or controller is selected only the logic bricks connected to it will be shown. When nothing is selected all the logic bricks will be shown (see fig. 13.4, next page).
With idea 1 it is possible to reuse controllers within a logic network of another object or within a complete different project. To reuse a number of controllers the user is able to select, group and save them. When a set of controllers is grouped this group will be shown as a new controller 'brick' with all the in- and outputs visible and named for easy reuse without the need to open the group to see what the in- and outputs exactly are (see fig. 13.8, next page). These in- and output names are generated automatically or can be given manually.

13.1.2 Idea 2

Idea 2 (see fig. 13.2, previous page) focusses on a manner to seperate the creation of sensors, controllers and actuators from making connections between them. Within the 'first' window there are three layers. One for the creation and editing of the sensors, one for the controllers and one for the actuators (see fig 13.5, next page). These layers are located in front of each other on the z-axis. When clicking on the 'actuator-layer' icon (or on a actuator in the 'actuator-layer') when being in the 'sensor-layer', for instance, causes the window to 'zoom' into the depth (z-axis) to the 'actuator-layer', while passing the 'controllers-layer'. When being in one layer, other (visible) layers will be dimmed, so the user can create and edit, for instance, the sensors, without being distracted from the actuators and/or controllers.
Within the 'second' window the different sensors, controllers and actuators (created in the 'first' window) can be connected with each other (see fig. 13.6, next page).
The advantage of this idea is that the user can create, edit and connect every aspect in its own window/layer while having a more ordered look and overview of all the elements within a window.



13.3

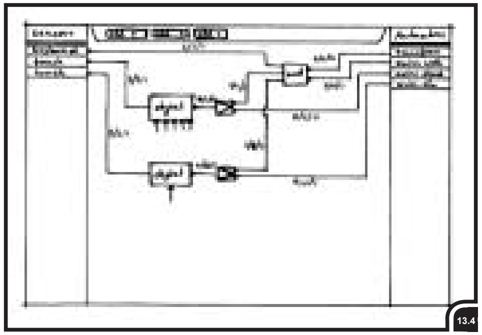
13.1.3 Idea 3

Idea 3 (see fig. 13.3) provides a means for dealing with higher-level interactivity. In the middle of the working-space is the a circle located. The circle is the starting point for all the logic created around it and it is the place-holder for creating and editing global (available for all objects) variables. The space around the circle can be split in as many smaller areas as needed. In every area a sensor 'brick' is connected to the 'starting' circle. And just like in the current workflow of Blender, from there on controllers and at last actuators are connected. The difference is, within every logic cycle (is the time that all the logic will be evaluated), that the order of evaluation is clockwise, starting with the logic within the area defined as 'starting logic'. The first advantage of this is that the user has control over the order of evaluation of the logic attached to one object. The second advantage is, dependent on what global variables are generated by other objects or by the same object within the previous logic cycle, only those 'logic areas' are evaluated that meet the current global variables. So depending on what 'state' (global variable) a object or the game is in, different logic will be executed.
The logic is attached to a object by selecting the object in the 3D-view and then starting to generate the logic bricks for it or by selecting the object from the 'object manager' at the left of the screen (see also fig. 13.7, next page). The 'object manager' makes it also easier to jump back and forward between different objects for comparing the attached logic, instead of selecting the object within the (cluttered) 3D-view.

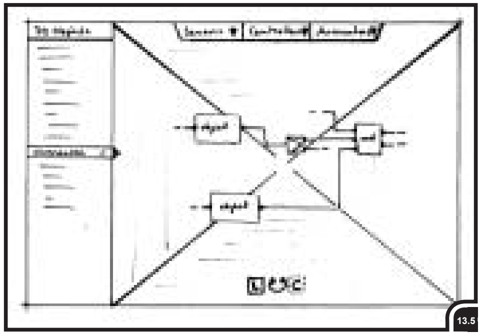
13.2 Conclusion

With these first ideas in mind, these 'small' solutions had to be put in place. The different ideas don't provide a solution to the three questions on their own. To find the right direction the ideas were submitted to the content creation team of NaN (see section 7.1).
These discussions with some members of the content creation team resulted in some considerations concerning the concept to be developed.

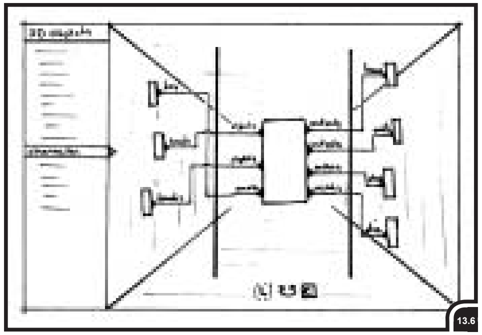
One thing was clear: idea *one* with two fixed columns for the sensors and actuators with a work-space for (more) controllers in between, for the creation of low-level interactivity was also 'approved' by the members of the content creation team of NaN, besides the fact that I person-



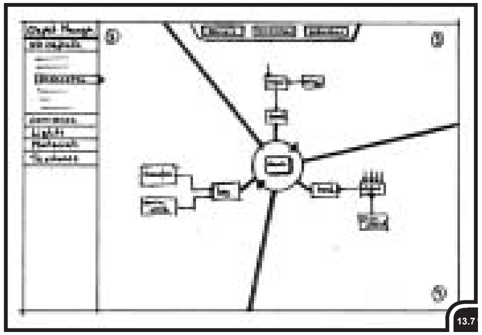
13.4



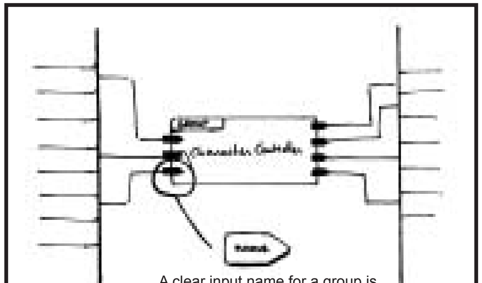
13.5



13.6

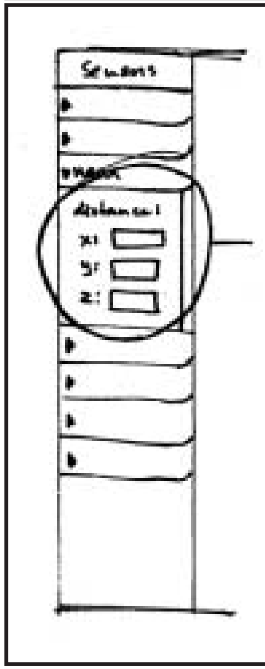


13.7



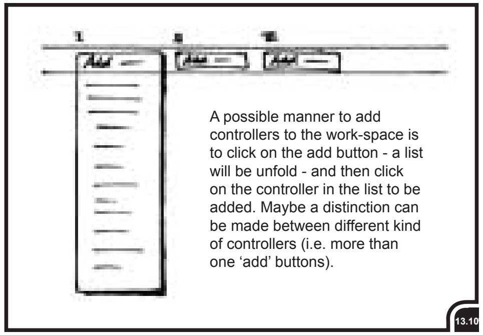
A clear input name for a group is given to make reusing it easier.

13.8



The properties of a (in this case) sensor and actuator can be fold open to edit. To gain space the properties can be collapsed all at once or individual per sensor/actuator.

13.9



A possible manner to add controllers to the work-space is to click on the add button - a list will be unfold - and then click on the controller in the list to be added. Maybe a distinction can be made between different kind of controllers (i.e. more than one 'add' buttons).

13.10

ally already made the decision to stick with them in section 10.5.

On the other hand idea *two* with the '3D-workview' and two different screens for creating and connecting sensors, controllers and actuators was - maybe - too ambitious. It is too much based on a gimmick instead of providing an ordered workflow.
Idea *three* proved to the content creation team that there is a way of providing a Graphical User Interface for creating interactivity on a higher level. The idea of providing a manner to decide which logic - attached to an object - should be evaluated and which not, depending on different generated parameters seemed a good idea to the content creation team. The side note to idea *two* was that it did not provide a ready-for-use solution, but only a starting point for a GUI for creating higher-level interactivity. In the next phase, the concept phase, this idea is worked out further.

14. Concepts

The ideas created within the previous section are further developed within the concept phase, described in this section.
The standard way of working within the concept phase is to provide more than one concept, all fulfilling the requirements (see section 12).
During this project of designing a Graphical User Interface for creating interaction, the workflow was a little different.
Instead of developing different concepts, one 'global' concept was developed out of the different ideas and the feedback. Instead of choosing between different concepts and then further detail the chosen one, the one concept was evolved step by step.
This 'evolution' of the concept was done within three steps. Step one was the development and evaluation of concept one. Feedback from this first evaluation was used to further develop the concept. This step of development and evaluation was repeated two more times to finally come to the 'final concept' described in section 16.

The three concepts are discussed on the basis of the workflow. The workflow clearly shows the difference between the concepts and what has been changed compared to the previous concept.

14.1 Concept A

The following is a description of the concept and its workflow of creating interactivity that emerged from ideas and discussions with the content creation team of NaN during the idea phase (see section 13).

First the different parts of the interface will be explained with the workflow in mind. This workflow is shown in the 'workflow-diagram' (see fig. 14.1) and shows which steps have to be undertaken to create the interactivity within the user interface of concept A.
For this concept their are two main screens. The state- and logic editor. How they work and relate to each other will be explained in section 14.1.1 and 14.1.2. After the explanation of the two main screens, a game example is given within the interface in sections 14.1.3 to give a better understanding of the workflow for a specific case. In section 14.1.4, the concept has been evaluated.

14.1.1 State-editor

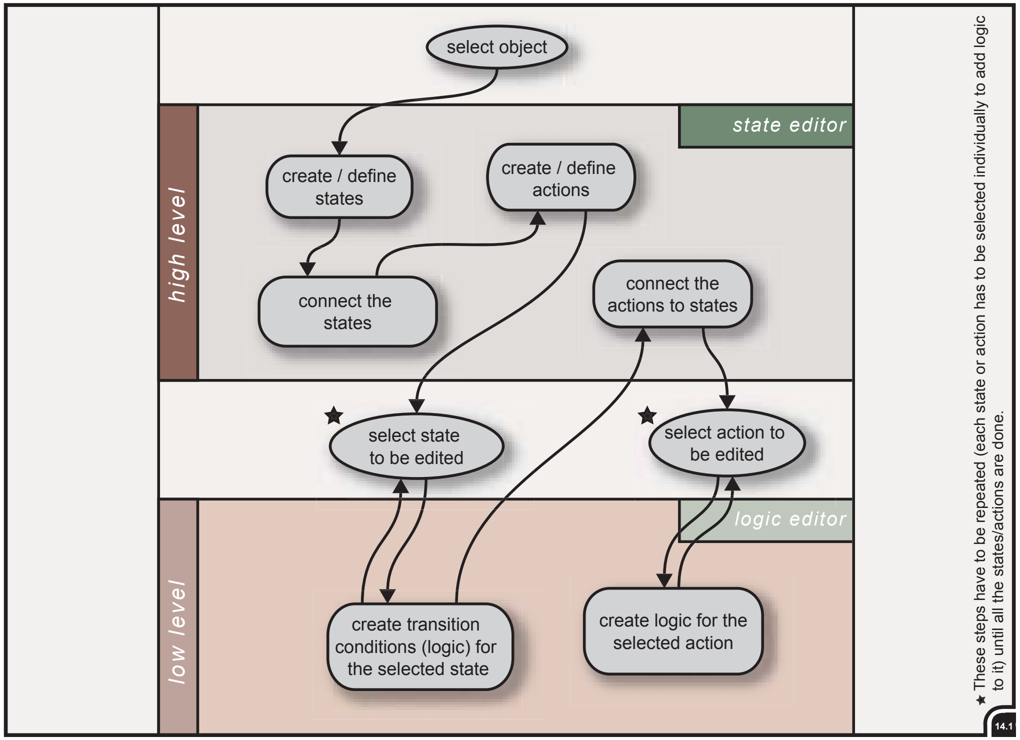
The state editor (see fig. 14.2, next page) is the place where the workflow starts after a object is selected to create interactivity for it. Within the state editor the user creates the states needed for that object. It depends on the user how many and what kind of states there will be created. For instance, for a 'smart' door you could think of two different states: 1) closed: as long as the player doesn't have the key, the door stays closed; 2) always open: the door will go open always and for everyone (also computer controlled opponents).

Strictly, these *states* created by the user do not contain anything. They do not contain any logic so far or - when created - provide any kind of visible interaction.
After creating the individual state, they have to be connected. These connections represent the possible inter state changes. So, when two states are connected with each other, it is possible to change from the one state to the other.
Creating connections between states does provide one thing automatically, it creates a 'change-state' actuator within the logic editor. Creating the logic for a state and defining the conditions (parameters) for a state change is the next step. But before a start is made with the logic for every state, the actions have to be created.

Actions are on a less high abstraction level than the states. Actions define things like 'walk', 'shoot' and 'jump', while states define 'fighting' and 'fleeing'. In other words, changing parameters causes a state to change from one to the other - states are variable - while these parameters do not effect actions. Actions only define the states. One state can exist of different actions, while one action can be connected to more than one state.
Just like the states, when an action is created it contains nothing, no logic or code. By connecting actions to states the user defines which actions can be executed within that state.
After setting up all the states, actions and connections the next step comes in: creating the logic for every state (transition) and action within the logic editor.

14.1.2 Logic editor

The logic editor is the place where the logic is cre-

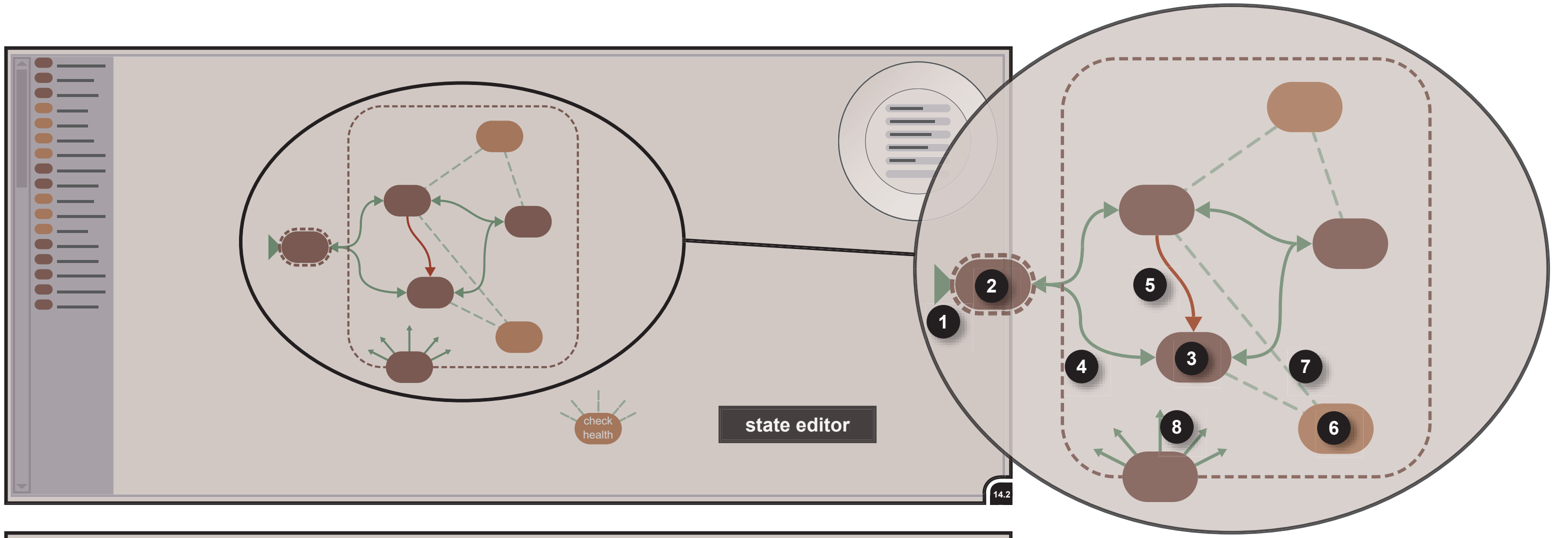

select object
state editor
high level
create / define states
create / define actions
connect the states
connect the actions to states
select state to be edited
select action to be edited
logic editor
low level
create transition conditions (logic) for the selected state
create logic for the selected action
14.1

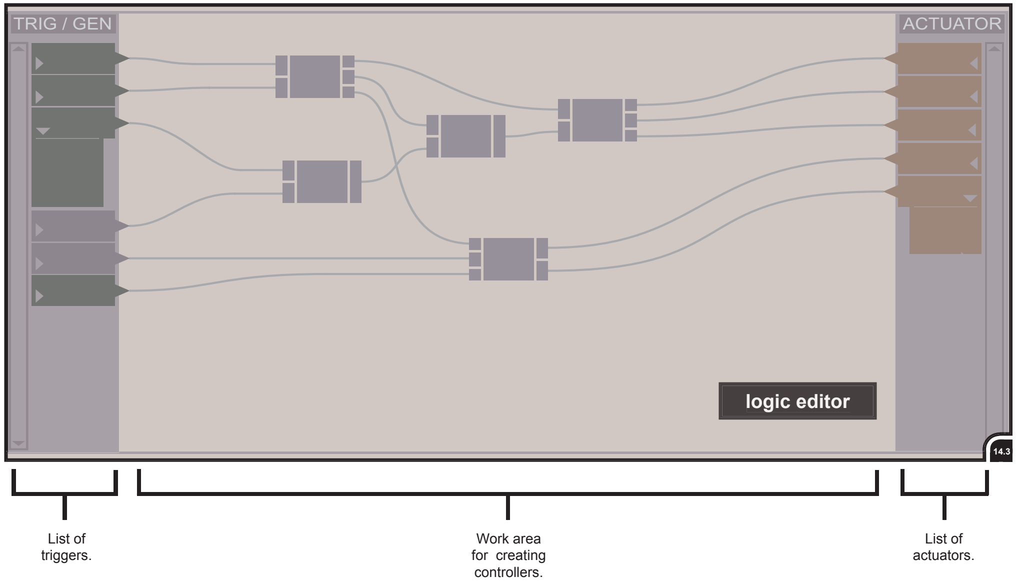
★ These steps have to be repeated (each state or action has to be selected individually to add logic to it) until all the states/actions are done.

fig. 14.1
Workflow diagram of concept A.

ated and edited for every state and actions. As said in sections 13.2, the logic editor has remained its current types of logic bricks: the triggers (formerly known as sensors, see note 1), controllers and actuators. But the interface didn't remain the same. As shown before in first ideas, instead of having three fixed columns, there will only be two fixed columns left and one 'flexible' (controllers are freely movable) column in the middle for the creation of controllers. The two fixed columns are for the triggers (on the left) and for the actuators (on the right), see figure 14.3.
The big difference with the current interface is that there will be more different kinds of controllers available to use. Besides, it must be possible to connect more than one controller in series and to have more than one in- and/or output per controller. These additions have to provide more functionality to the logic editor.

(1) During a content and development meeting it has been decided that from now on sensors will be called 'triggers'. The term 'trigger' covers the functionality of this kind of logic bricks better than the term sensor does.





State editor close-up:

1. Defines the start state.
2. General state brick; the 'dashed line' determines that the logic within this state is used within every other state. The 'dashed line' can be assigned to every state, but only to one at the same time.
3. State brick.
4. Two way connection between states. The state change can go either way.
5. One way connection between states. The state change can go only one way. So when the state change has occurred there is no way back to the previous state. For instance, when player is dying.
6. Action brick.
7. Connection from action brick to state. Determines which actions are possible during which state.
8. A lot of arrows coming out of a state brick, meaning that this state is connected to every other state that's within it's region (brown dashed line).

14.1.3 Example concept A

To give a better overview of concept A an example is given within the concept. This example exists out of a 'computer controlled character' within any FPS (First Person Shooter). First, it tries to hunt down the player and then to kill him.

Computer controlled character

This opponent is a character fighting against the player's character. While hunting and killing, the computer (the game) has to make decisions on what it should do next or what it should do depending on things that are happening around the computer controlled character.

The computer opponent can have the following states:

- Idle/living:
  health is between 100% and 1%
- Dying:
  health is 0%, computer opponent will fall at the floor and struggle for his life, but stays down at the floor
- Hunting/tracking:
  looking/finding the player and as soon as the player has been found switch to fighting
- Fleeing:
  when the health level becomes below a certain point, the opponent is going to flee
- Fighting:
  trying to kill the player, shoot at the player

The computer opponent can execute the following actions:

- Shooting
- Walking
- Running
- Jumping
- Fall down

Result

In fig. 14.4 the state diagram is shown. The states have a dark brown color and are connected as indicated by the green lines with arrows. The idle state is the 'general' state with the logic that is used trough all of the states.
Actions have a lighter brown color and are connected to states with light-green dashed lines.
The many red lines going to the dying state indicate two things. First, the red line means the transition can only go to the dying state and not back to the previous state. Second, many lines going to a state indicate that it is connected to all states. This is faster to set up than connecting the dying state to every state individual. This is also applied to the shooting action, but with the difference that there is a light-red dashed connection with the dying state. This indicates that the action is connected to all the states, except the dying state.
The green arrow attached to the idle state indicates that the idle state is the 'starting' state.
The list shown at the left of the state editor window is for saving and reusing states and actions. For instance, when a user wants to reuse a state he simply drags a newly created state into the list and give it a name. With another object or even project, the user can drag the saved state from the list to the work space and the state is there with all the logic in it.

In fig. 14.5 the logic for the walk state is shown.

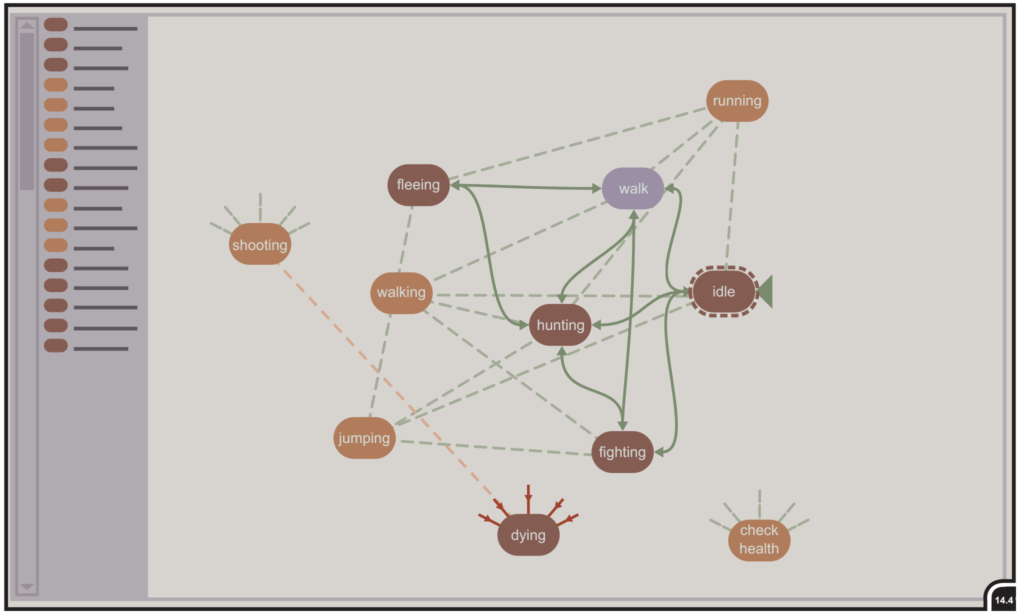
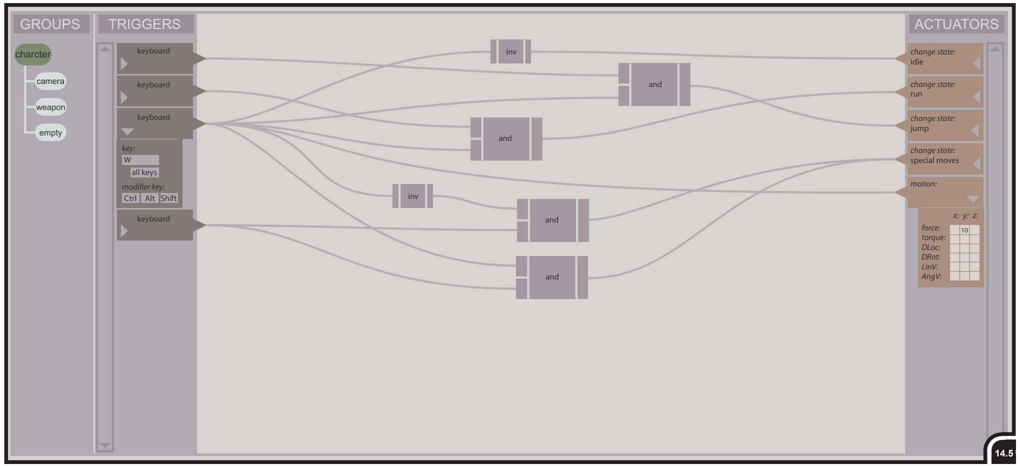


fig. 14.4
Example of concept A .
The 'state editor' for the main character.



key: W
modifier key:
change state: idle
change state: run
change state: jump
change state: special moves
motion:
force: 10
torque:
DLoc:
DRot:
LinV:
AngV:

fig. 14.5
The 'logic editor' for the main character and in this case the logic for the 'walk state'.

14.1.4 Evaluation concept A

It is hard to create an example yourself if you do not have enough experience creating games. Submitting the concept to the content team had to provide feedback and a different point of view and would most certain give a very different implementation of the game example compared to the one I did (try) myself.
The details of the concept were explained without showing the example described in the previous section. After the members of the content creation team did understand the idea behind the concept, they were asked to create the example of the computer controlled character within the given canvas. This was submitted individual to three members. The most useful information is shown on this page (see fig. 14.6 and 14.7).

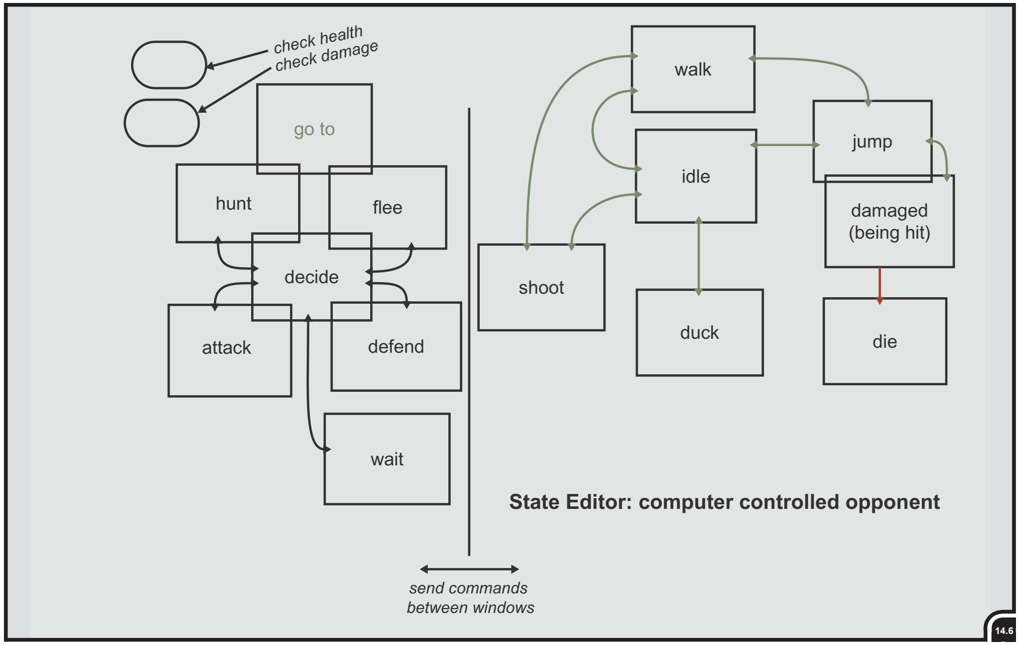
What can be said about the outcome of this evaluation? The conclusions are described on the basis of the three questions (see section 12).

How can logic be ordered?
This question was not emphasized, because the creation of high level logic was at that moment a part that still needed much more attention than a means of organizing logic that in one way or another could be 'fixed' at a later stadium.
This was the reason that there was less feedback on this aspect, but the members did like the way the higher-level states and actions did provide a way of ordering the logic. Instead of having a long list with all the triggers, actuators and controllers, they are divided into smaller lists within a state or action.

How can logic be reused?
This aspect was neglected with this evaluation, because it was not relevant at this point. This can be integrated at any point in different ways.

How can the creation of low-level and high-level interactivity be combined into one workflow?
This question took the most attention during the evaluation. Two aspects came to attention:

1. The idea of states and actions was accepted very well, but there should be made a clear distinct between the two. These should be in different windows, instead of being in one window together (two members putted a dividing line between the two; see figures).
2. Every member created a kind of 'all-embracing' state (a global state) to be able to check health and/or damage, but these 'states' were not connected to any other. It can be concluded that these kind of 'global' states do need a place of their own and in where they are able to influence all the other (object) states.



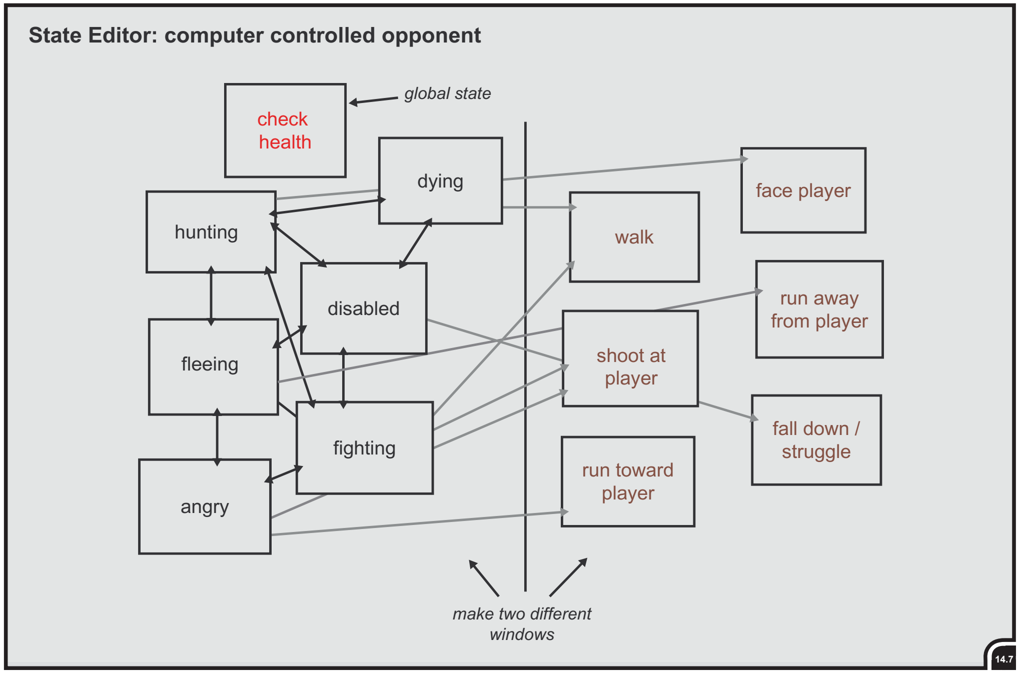


fig. 14.6
Evaluation of concept A: the state editor filled in by member one of the content creation team of NaN.

fig. 14.7
The state editor filled in by member two.

14.2 Concept B

Concept B followed after the evaluation of concept A. The most important change that followed out of this evaluation was the fact that states and actions had to be created in two different windows. So, the emphasis within the state editor could be placed on the connections, while within the action editor only the different actions could created and could be assigned to the different states.

The workflow diagram (fig. 14.8) shows that a new step has been added compared to the diagram in figure 14.1, the action editor.

What will follow now is description of the state and action editor and a short description of the logic editor (remained almost identical to the one of concept 1). The descriptions give only the aspects that changed compared to concept A.

14.2.1 State editor

The state editor (see fig. 14.9, next page) remains the place to create and edit the states. After creating all the individual states, the states have to be connected. The connection is enhanced with a “transition node”. This transition node is the actual object the logic is assigned to. As can be seen in figure 14.9 (next page), all transition nodes have two arrows. These two arrows represent both direction of a transition. When an object is in one ‘state’ certain conditions will make it transit to another, while it is in the other ‘state’ the conditions are (probably) different to change back to the state it came from. To define the conditions (parameters) a transition will take place, one of the arrows has to be selected (color will change; indicating it is selected) and then these conditions have to be created in the logic editor.

In addition a red arrow has been added to indicate that a state is the ending state of the state diagram. An ending state can be a ‘dying’ state, for instance, at the end of the dying state the game will restart.

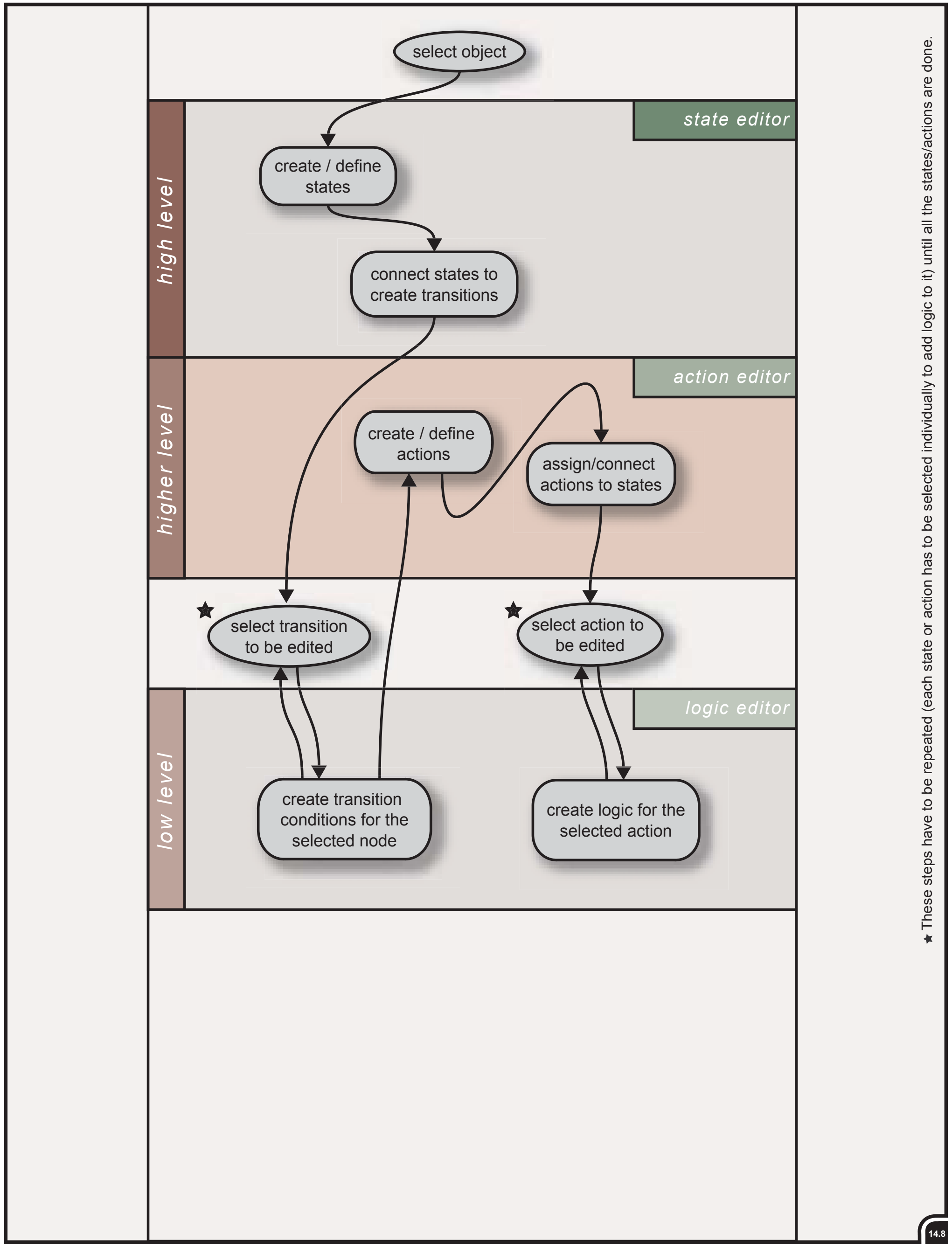
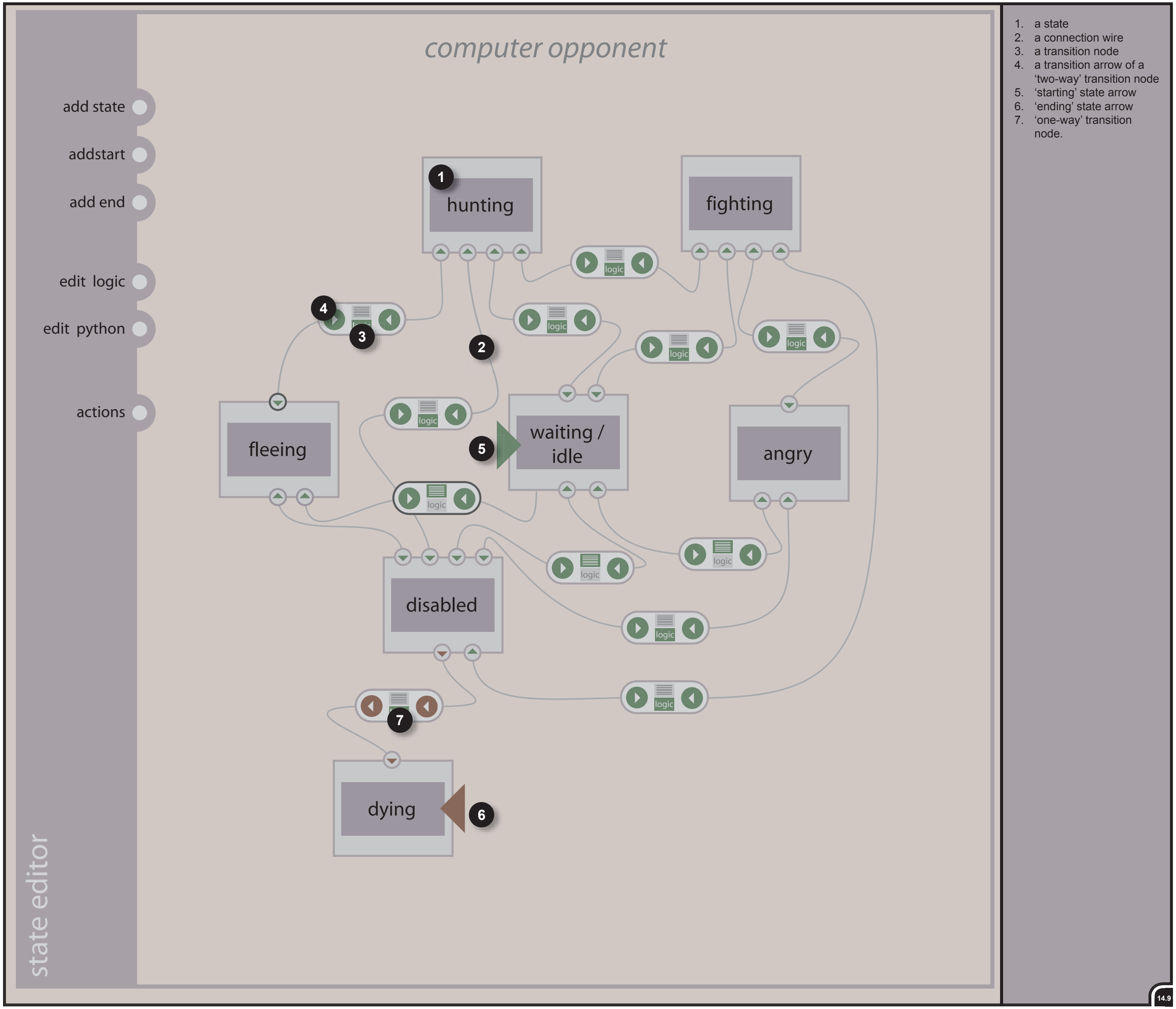

select object
state editor
high level
create / define states
connect states to create transitions
action editor
higher level
create / define actions
assign/connect actions to states
★
select transition to be edited
★
select action to be edited
logic editor
low level
create transition conditions for the selected node
create logic for the selected action
★ These steps have to be repeated (each state or action has to be selected individually to add logic to it) until all the states/actions are done.
14.8

fig. 14.8
Workflow diagram of concept B.



1. a state
2. a connection wire
3. a transition node
4. a transition arrow of a 'two-way' transition node
5. 'starting' state arrow
6. 'ending' state arrow
7. 'one-way' transition node.

fig. 14.9
State editor example of concept B.

14.2.2 Action editor

The action editor (see fig. 14.10) is an addition to concept B (compared to concept A). The action editor provides an interface for creating actions (and after creation) connecting them to states.
The listed states at the left of the screen are all the states created within the state editor. When a new state is created within the state editor, the state list, within the action editor, is updated automatically with the new state listed.
It is possible to view all actions at once or individually by selecting one state in the states list causing only the connected actions to be showing clearly, while other actions fade away (see fig. 14.11). Selecting groups of states is also possible.

A great advantage of the action editor (and one of the reasons of adding it) is the fact that actions function as an ordering tool for the logic. Instead of having a long lists of logic it is broken apart into small packages within the actions.

Creating the logic for an action within the logic editor is done by selecting an action and then switching to the logic editor. The selected action will change color to indicate that the user is creating/editing the logic for that action, just like with the state transition 'nodes'.

14.2.3 Logic editor

The logic editor (see fig. 14.12, next page) remained almost the same, except that the new inteface style has been applied. A minor enhancement is the addition of a python script editor. This editor can be accessed by clicking on the 'python' button located on the python controller. The python controller itself is much clearer now. The in- and outputs are all laid out well, so the user can see clearly which in- and outputs the python script calls.
Within the python script editor it is possible to load an external script or to save the current script for reuse. The same is true for controller bricks. These can be saved or loaded as well. This can be one single controller (f.i. a python controller) or, more likely, many controllers in a group. The save and load actions are located at the top (middle-left) of the screen. When clicking on the save action the selected controller or group of controllers will be saved. Saving and loading can increase the speed of working with logic. Logic does not have to be created twice, but can be reused from another project.

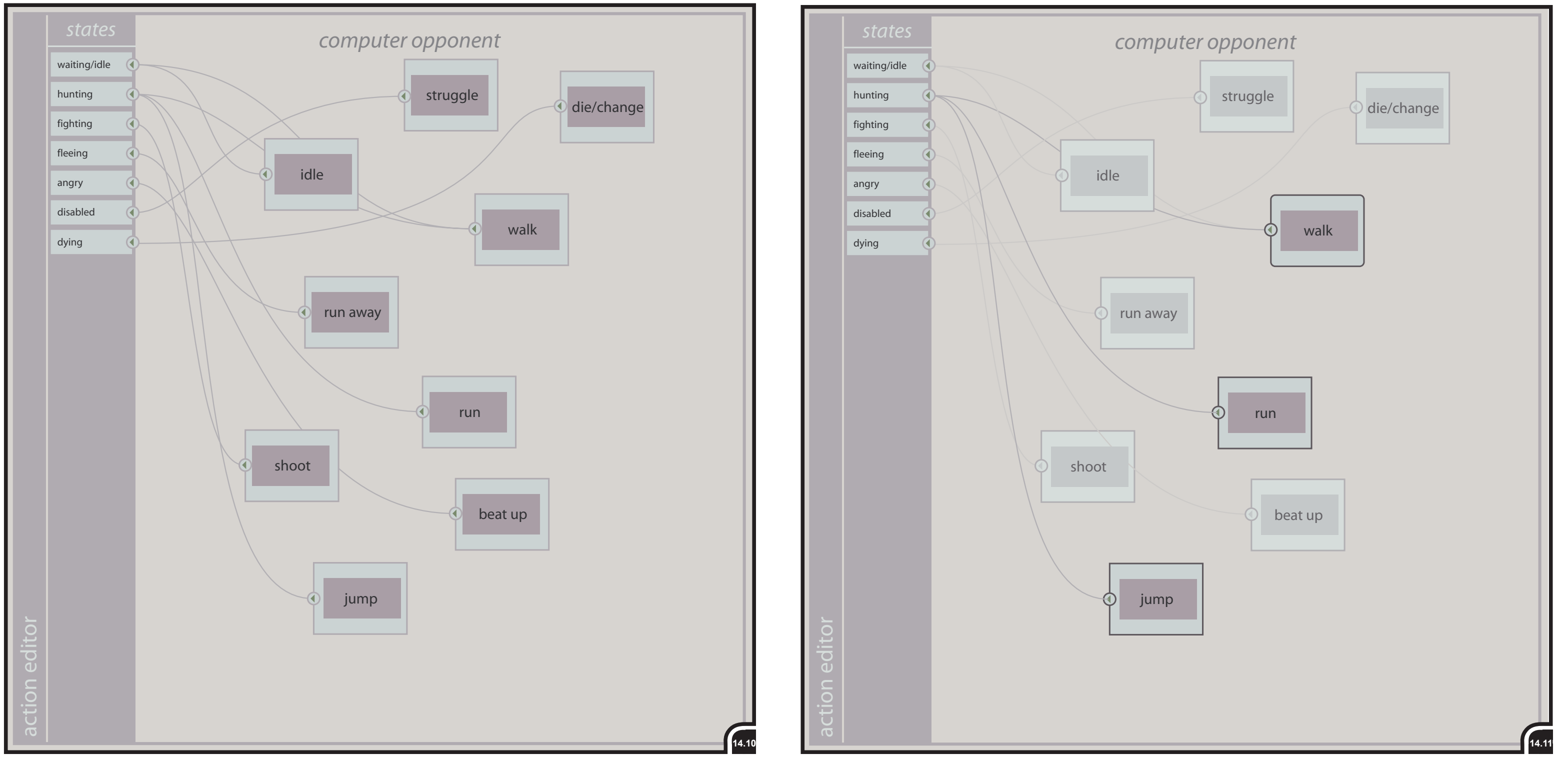
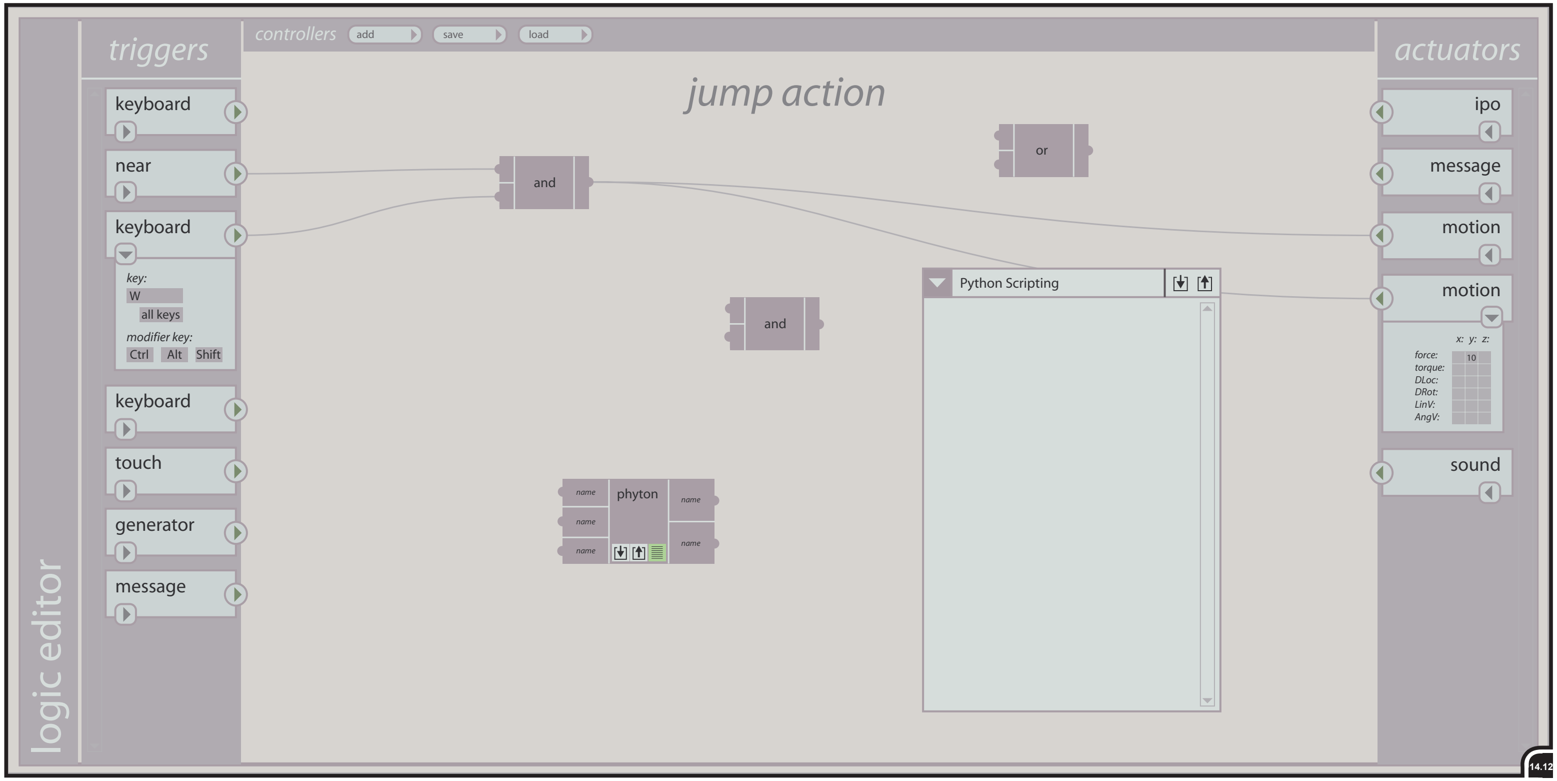


fig. 14.10/14.11
Action editor window,
showing some actions
connected to states
listed at the left.



14.2.4 Evaluation concept B

This second evaluation existed of conversations with two members of the content creation team of NaN. Only the changes compared to the previous concept were made clear to them.

After an explanation, it was apparent that it seemed immediately clear to them.
However, one important modification was proposed. This modification was the fact that the logic (transition conditions) for the transition nodes had to be created in the logic editor, while conditions are on a very different level than 'normal' logic. This seemed odd, so the conclusion was made that there had to be a kind of 'transition' editor at the same level as the state editor with the same functionality as the logic editor. This was the basis for the next concept, concept C.

fig. 14.12
The logic editor. Left and right fixed columns for the triggers and actuators and in the middle a work pane for the controllers.

14.3 Concept C

An additional step has been added to the workflow as can be seen in figure 14.13. This step is the creation of transition conditions for which a state transit will occur. These conditions are created and edited in the transition editor.
Because the addition of the transition editor is the only change compared to the previous concept that this is the only part that is described here. The other parts, like the state, action and logic editor remained the same.

14.3.1 Transition editor

The transition editor (see fig. 14.14, next page) is the place to define conditions for a state transition, as mentioned above. The addition of a separate transition editor makes the editing of the states and transitions an action on its own.

As can be seen in figure 14.14, in principal, the difference with the logic editor is small. The only big difference is that there are no actuators on the right anymore. Instead, there are 'change state' bricks. When an arrow on a transition node (see fig. 14.9, on page 57) is selected, the 'change state' actuator is added automatically. The left side of the window (with the available triggers) remains the same as in the logic editor. All possible triggers can be used to 'trigger' a transition, but message triggers will probably be used most of the time. These message triggers 'scan' for changing values of a 'property', such as 'health', 'damage' or 'score', for instance. These property changes are the most likely candidates for a transition. But as mentioned, every kind of trigger can be used.

The 'quick switch' button on the top right of the transition editor pane makes it possible to switch fast between the two directions of a transition node, instead of having to click the other green button on the transition node in the transition editor. The transitions between two states are bidirectional by default, unless the user explicitly defines a one-way transition (see fig. 14.9 as well).

14.3.2 Evaluation concept C

The transition editor is part of the state editor window (see fig. 14.15, next page). It is not a separate window like the action and logic editors. It has a strong connection with the state editor. States without transition conditions do not mean anything. This is the (main) reason to put it at the same level with the state editor (even within the state editor), although it resembles the logic editor very much. As can be seen in figure 14.13,

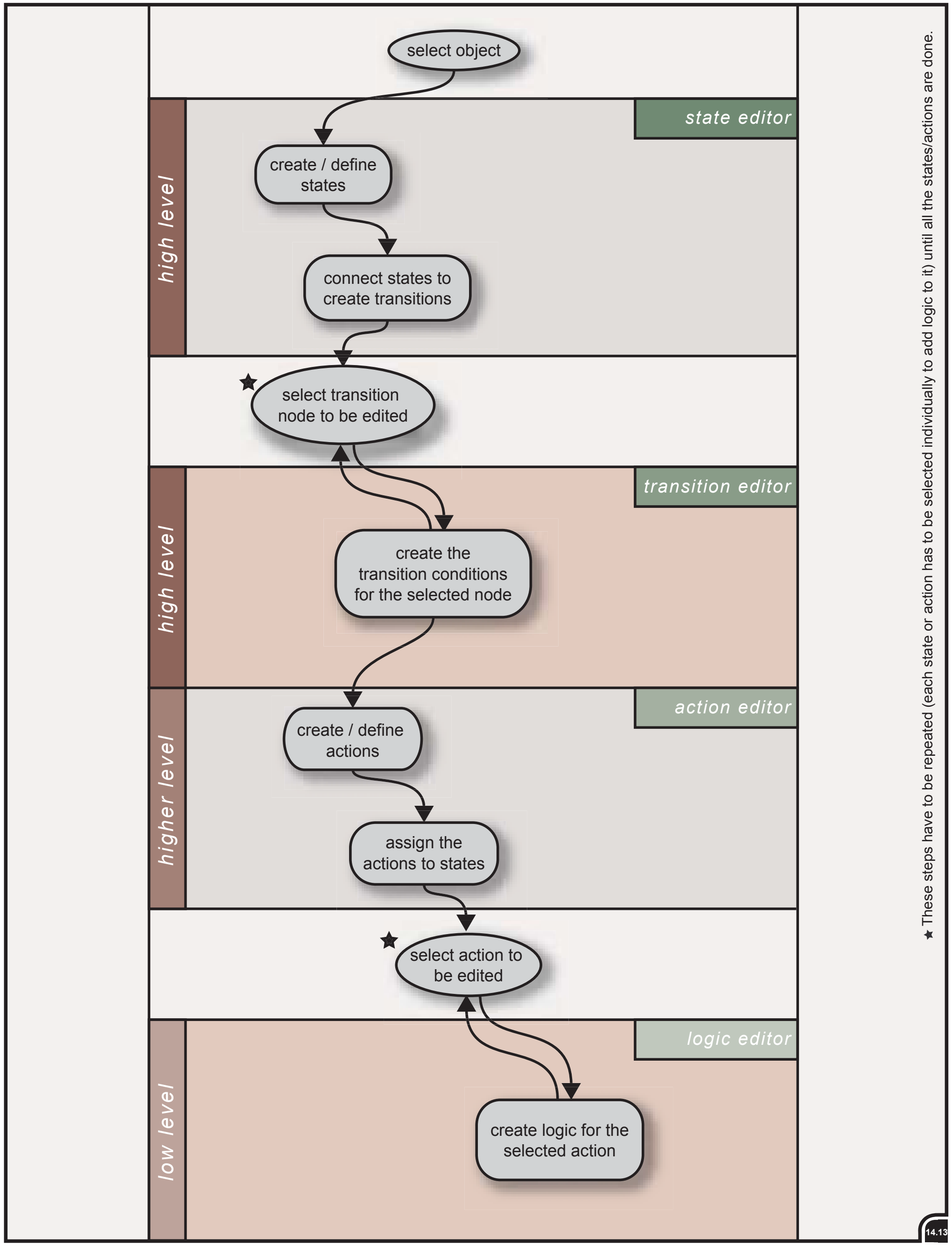
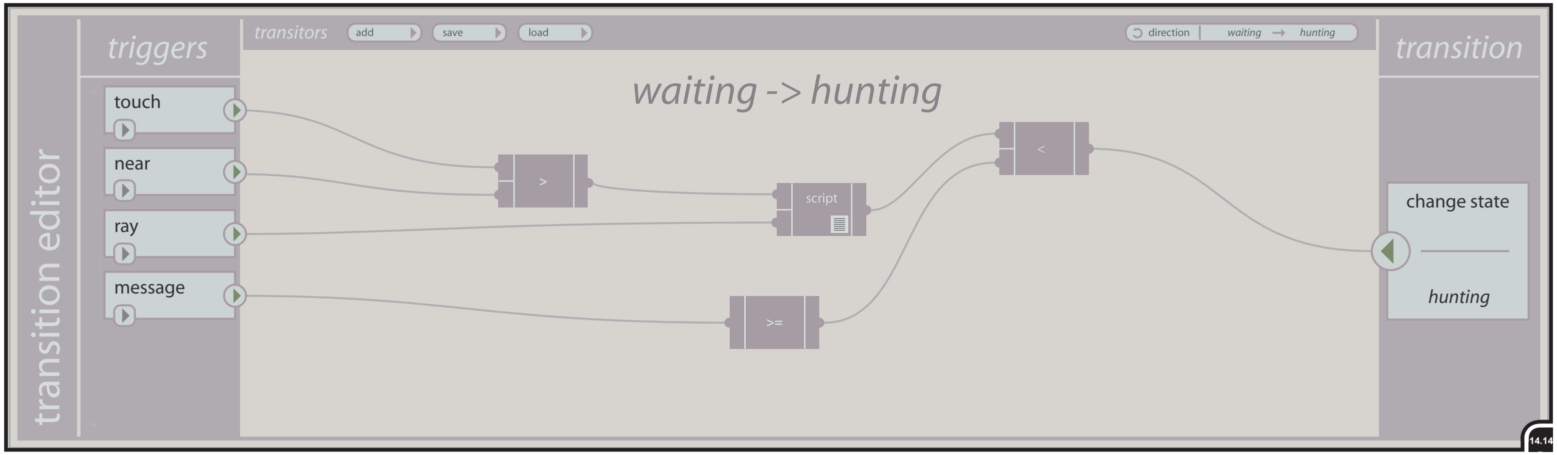

select object
state editor
high level
create / define states
connect states to create transitions
★ select transition node to be edited
transition editor
high level
create the transition conditions for the selected node
action editor
higher level
create / define actions
assign the actions to states
★ select action to be edited
logic editor
low level
create logic for the selected action
★ These steps have to be repeated (each state or action has to be selected individually to add logic to it) until all the states/actions are done.
14.13

fig. 14.13
Workflow diagram of concept C.



the workflow is much ‘smoother’ now than with the other two concept diagrams. The workflow is now top to bottom, instead one of going down and up again. Anyway, it has to be said that even when the graphical representation of the workflow is linear now (from top to bottom), it does not mean that the workflow is restricted. The user can jump between the different editors anyway he likes, but the line of thought (mental model) is now a linear one. This can ease the learning curve and make it easier to understand the structure behind logic editing.

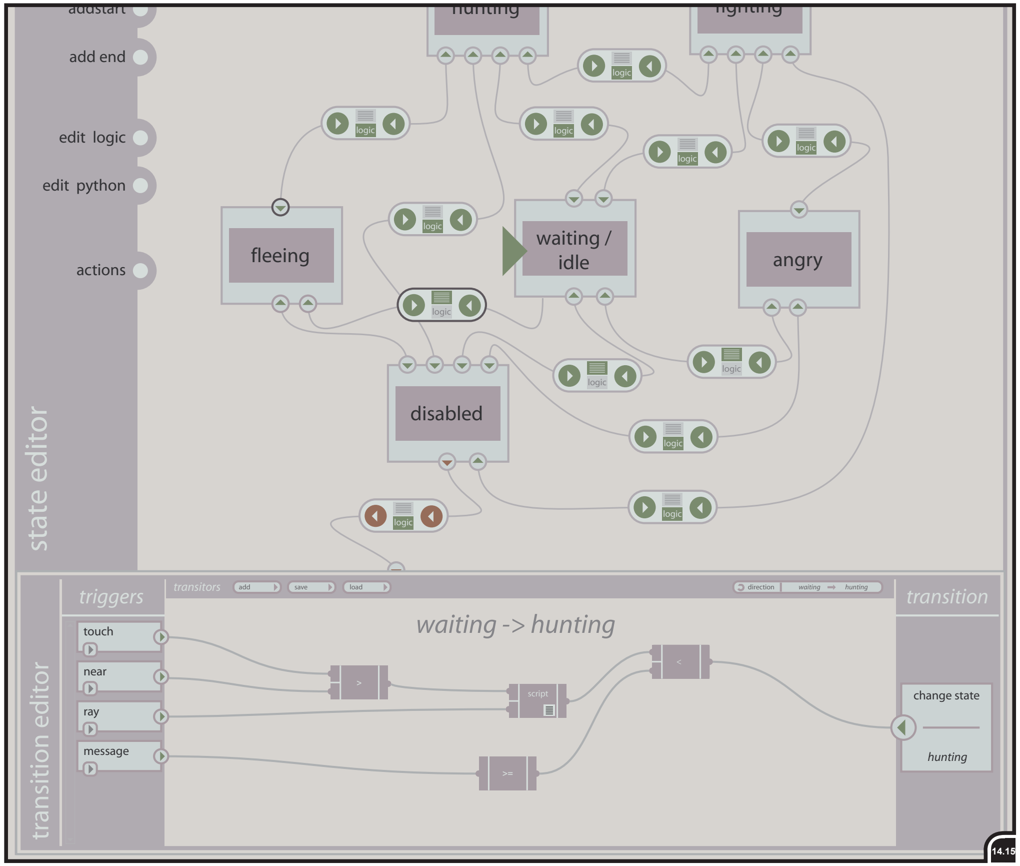


fig. 14.14
The transition editor as proposed in concept C.

fig 14.15
The location of the transition editor within the state editor window.

15. Final concept

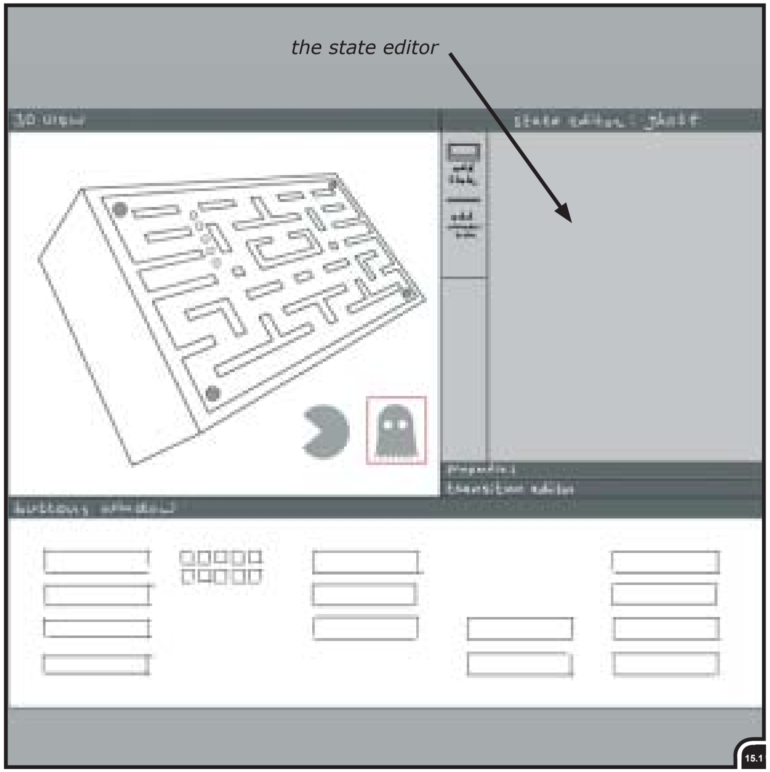
After completing 'different' intermediate concepts, the 'final' concept is presented here. The concepts, described in previous sections, have evolved with the feedback of the members of the content creation team of NaN, to this final concept.

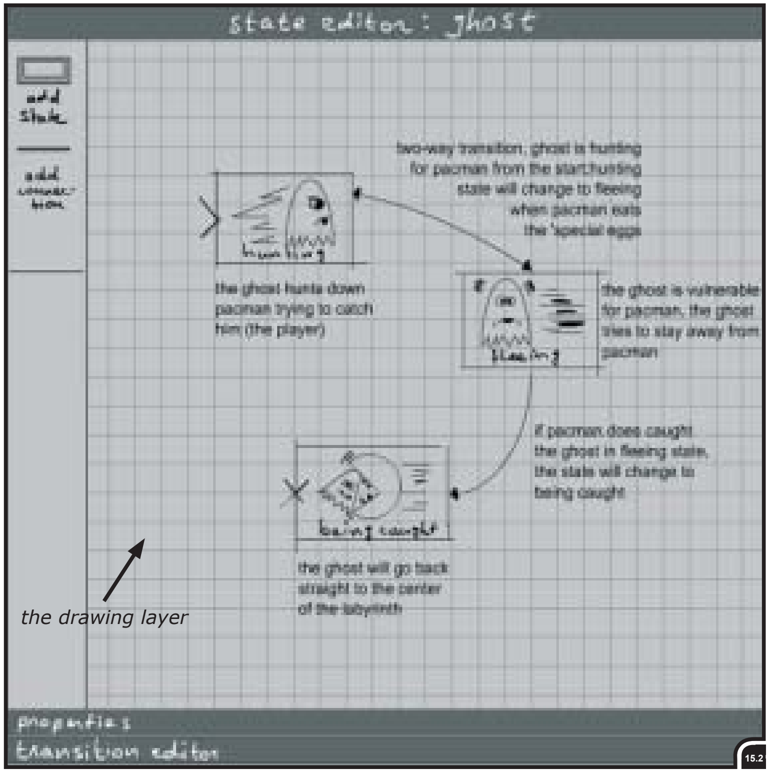
The final concept is described with the evaluation in mind. The concepts described in the previous sections did have a more predefined graphical look. The final concept has a more 'sketchy' look. It is presented in this way, because during an evaluation the subjects of experiment have to pay attention to the different steps (actions) that have to be taken into account to complete a task, instead of being distracted by the graphical look of the interface. By keeping the interface pictures as 'sketchy' as possible, there will be no discussion about the looks.

The final concept is divided in nine different sections. These sections represent a different part of the interaction creation process.
Next, the final concept is presented using the sections.

15.1 Conceptualizing States

Conceptualizing states will often be the first step when creating the interactivity.
The state editor has to be opened in one of the 3D views (see fig. 15.1). After opening, the drawing 'layer' can be turned on in the state editor. On this layer the user can sketch (with mouse or drawing tablet) the state engine for the selected object. After sketching, the user can add text to explain the design a little bit more, if needed (see fig. 15.2).
These steps have to be repeated for every object that has a state engine.
When finished, the user can save the Blender file containing the 'concepts' and hand it over to colleagues, who can give it feedback on it. The feedback can be given digital as well. The colleagues can sketch and type just like the initial creator did. Every colleague giving feedback does this on a seperate 'layer'. After getting back the file, the initial creator can turn these different 'feedback' layers on or off to see what each colleague has commented on the different state engines.
Another possibility is to print the different stage engines on paper. One, two or four state engines can be put on a single paper. Some people may prefer to write down the feedback on paper instead digitally. After writing down the feedback the papers can be returned to the initial creator.
These comments can be used to change or improve the initial setup of the state engines.
A big advantage of working with state concepts is that ideas about how different logic has to be created can be put on 'paper' fast and feedback can be given by many colleagues at the same time. Feedback is saved for review in later phase of the project.


the state editor


the drawing layer

two-way transition, ghost is hunting for pacman from the start.hunting state will change to fleeing when pacman eats the 'special eggs

the ghost hunts down pacman trying to catch him (the player)

the ghost is vulnerable for pacman, the ghost tries to stay away from pacman

if pacman does caught the ghost in fleeing state, the state will change to being caught

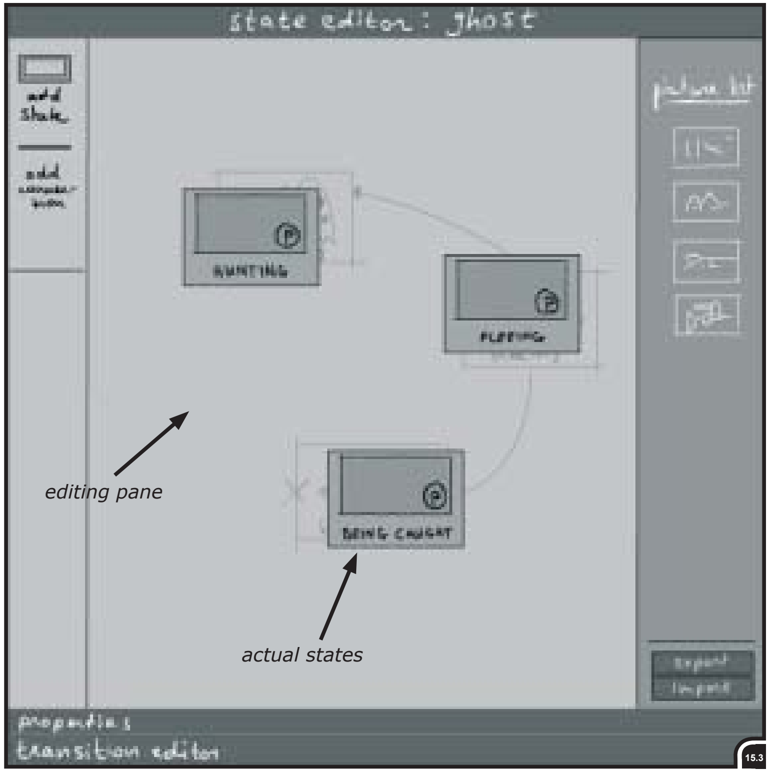
the ghost will go back straight to the center of the labyrinth

15.2 States

After the conceptualizing step is done, the actual creation of the states can start. The drawing 'layer' can be locked. When locked it cannot be edited anymore, but in this way it can be used as a guide for creating the actual states.
To create a state, a new state can be dragged from the 'creation' pane on the left onto the editing pane in the middle. The newly created state can be given a name to clarify its content. This has to be repeated for every state needed (see fig. 15.3).

To make it easier for another person to 'read' and understand the state engine at a glance, it is possible to include a picture for every state (see fig. 15.4). These pictures can be extracted from the drawing layer or can be dragged from the picture list (favorites) onto a state. This picture list can exist out of pictures created earlier on the drawing canvas or pictures that are imported and created in another program or by another person. In figure 15.4 can be seen that pictures are assigned to the different states.

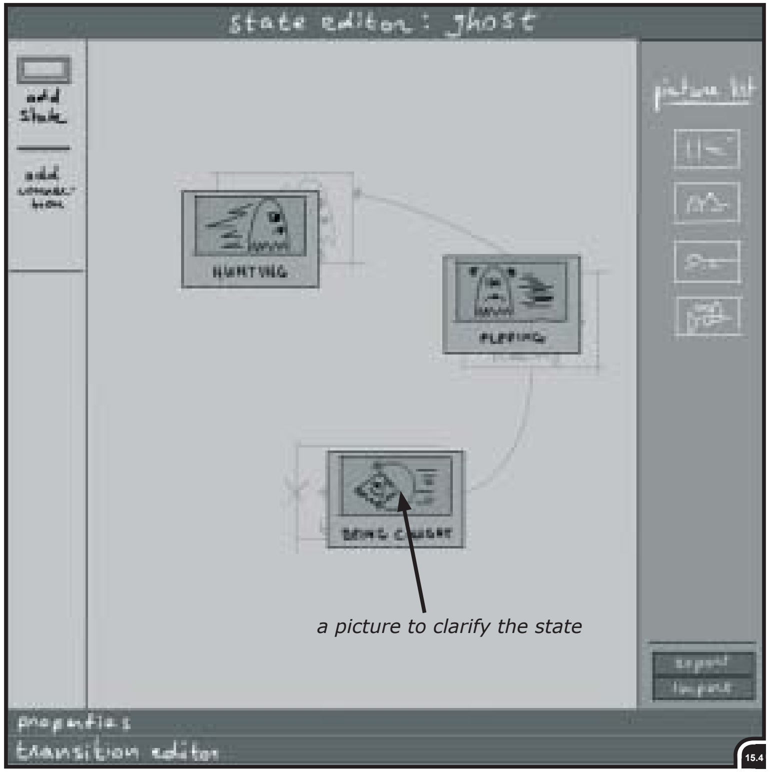
When the drawing layer is not needed anymore or when all states have been created or when the layer distracts from the actual states, the drawing layer can be turned off.
It is also possible to go back to the drawing layer at any given time to add extra components or change existing ones. This can be useful when a user has to finish something at a later stage. He can write/sketch it down on the drawing layer beneath the states, so the next time he is reminded to finish it.



editing pane

actual states

15.3



a picture to clarify the state

15.4

15.3 Transitions

States have to be connected with each other to define the state transitions. To do so, the user has to click drag on the border of one state and drag it, then release it on top of another state. After release, a connection is created. In between these states, halfway the connection wire, a 'transition node' can be found (see fig. 15.5).

A transition between two states is 'two-way'. This is the reason two arrows can be found on the transition node. One containing the transition conditions for state A to B and the other for state B to A. By default both arrows are greyed out. Indicating that no conditions have been defined. When an arrow has a green color it is indicating that conditions have been defined. When an arrow is selected half a side of the node is green, indicating which transition direction selected.

To edit the conditions for a transition the transition editor has to be opened. In figure 15.5 the transition editor has already been opened. The transition editor is located at the bottom of the state editor window. The transition editor is placed within the state editor, because these are closely related to each other. States without transition conditions do not make sense.
When the transition editor is open, a transition arrow has to be selected to be able to create a transition condition.

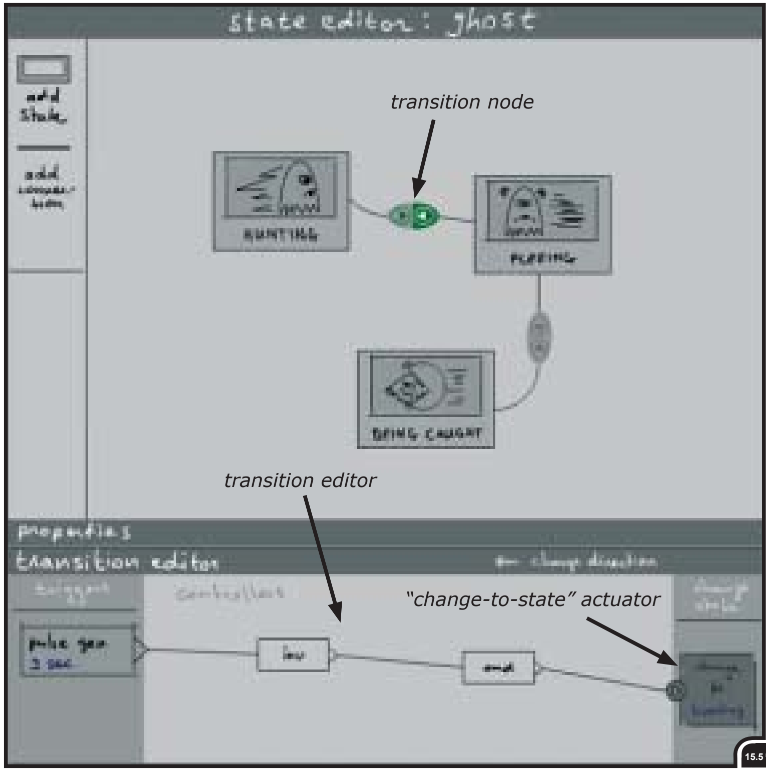
The transition editor resembles the logic editor (see section 15.6) closely. The only difference is that there is only one actuator available. This actuator is the "change-to-state" actuator. When creating the conditions for the transition from state A to B, for instance, the "change-to-state B" actuator is automatically added.

15.4 Global States

Global states are identical to 'normal' states with the difference that they are not assigned to an object, but apply to the entire game.

Global states can be created in the same manner as normal states in the state editor. When nothing is selected, the state editor automatically shows the global states. As shown in figure 15.6, global states are most of the time used for start screens, level management and scores.

The transition editor is also used for editing transitions between global states.

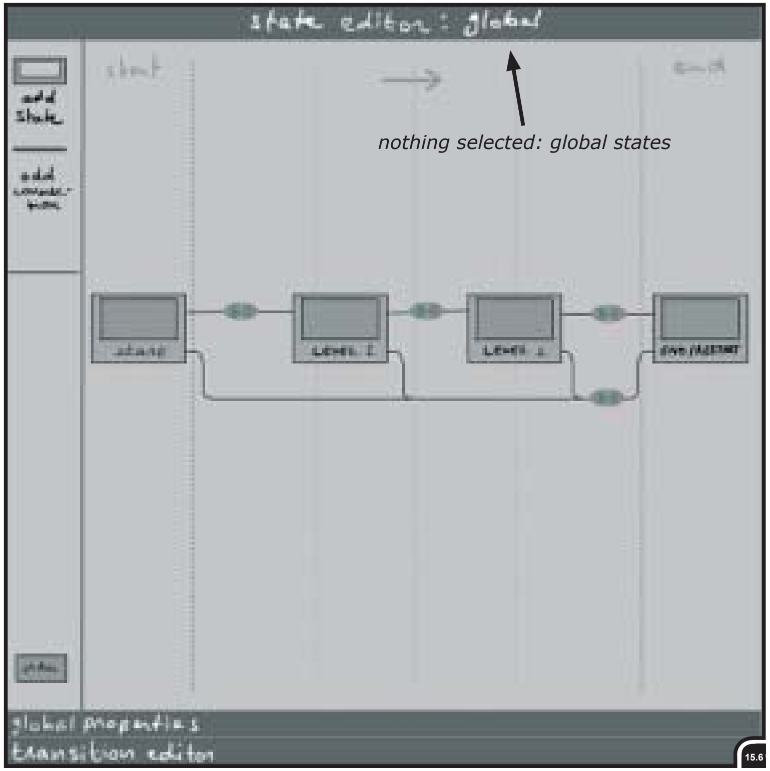


transition node

transition editor

"change-to-state" actuator

15.5



nothing selected: global states

15.6

15.5 Actions

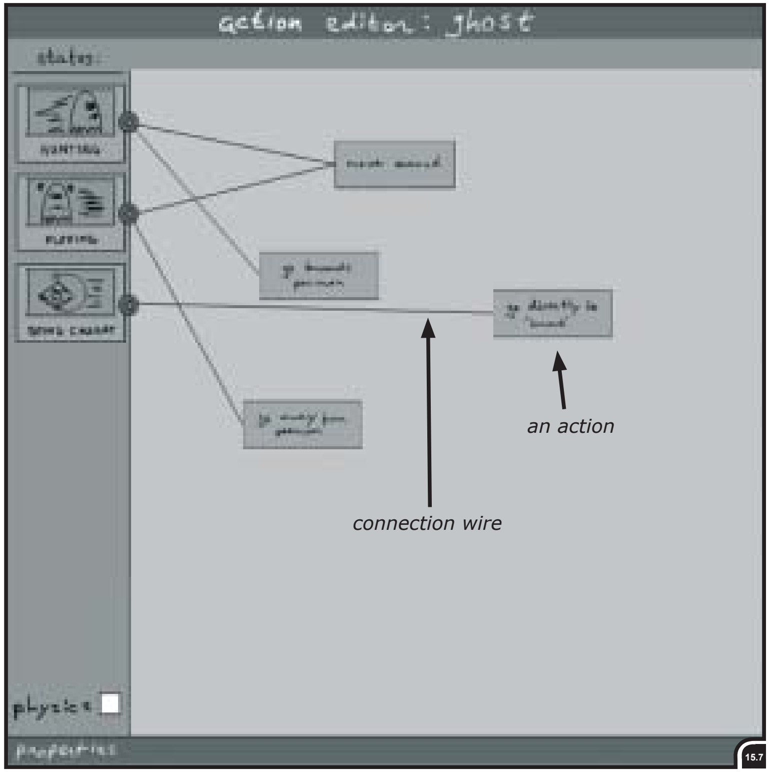
Actions are elements of a state. Actions are placeholders for the actual logic. Instead of creating the same piece of logic twice for two different states (when they need a part of the same functionality), one action containing that piece of logic can be connected to both of the states. One action can be shared by different states.

The action editor is opened in a 3D view just like the state editor. It will not happen often that the state and logic editor are open at the same time. A list with all the created states is displayed at the left pane of the action editor. After creating the actions needed and giving them a name, the actions can be connected by click-dragging them on a state. A connection wire will be then created automatically. Logic within an action will only execute/evaluate if the action is connected to the current active state of the object.

Another advantage of actions is that they provide a way of organizing the logic. Instead of having one long list of triggers, controllers and actuators, the logic is divided in different actions (see fig. 15.7). For instance, a "run" action contains all the logic to make a character run. A character can be running in more than one state. Instead of copying "run" logic, the "run" state can be connected to multiple states.

15.6 Logic

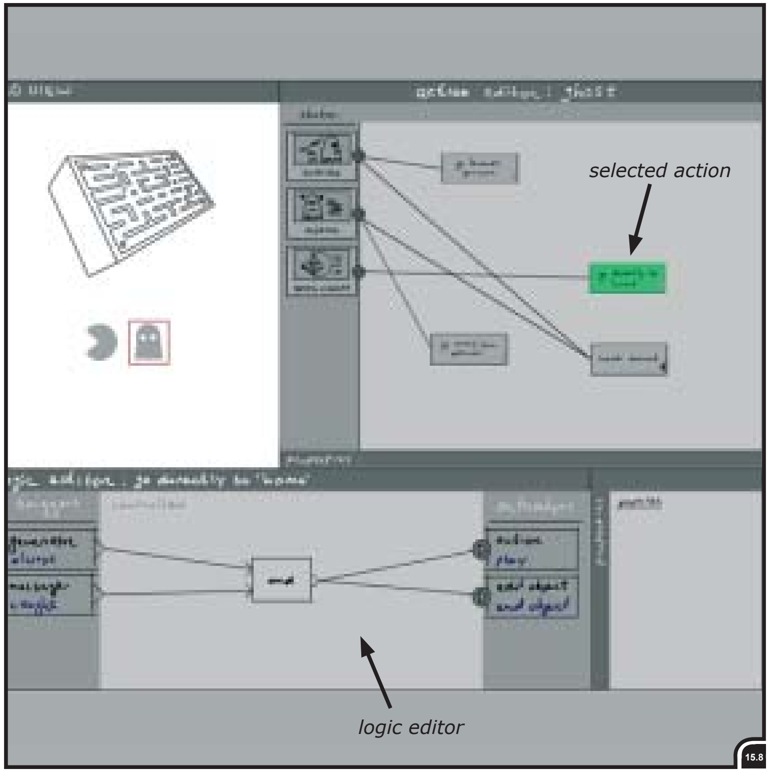
An action on itself is not very useful. As mentioned above, it is a container (placeholder) for logic. To create logic for an action, an action has to be selected in the action editor. A selected action has a different color (see fig. 15.8).
The logic editor has to be opened. In contrast with the state and action editor, the logic and action editor are preferably opened both at the same time. This makes it possible to select an action, create and edit the logic, select another action and so on without opening and/or closing windows. The logic editor has to be opened in the 'buttons window' area at the bottom of the Blender interface (see fig. 15.8).
The logic editor works identical to the original 'interactive window' within Blender. At the left the triggers (sensors), in the middle the controllers and at the right the actuators. The difference with the original version is that it is possible to add as many controllers between a trigger and an actuator as needed. For a more detailed description of the logic editor see section 14.1.2.



an action

connection wire

15.7



selected action

logic editor

15.8

15.7 Physics

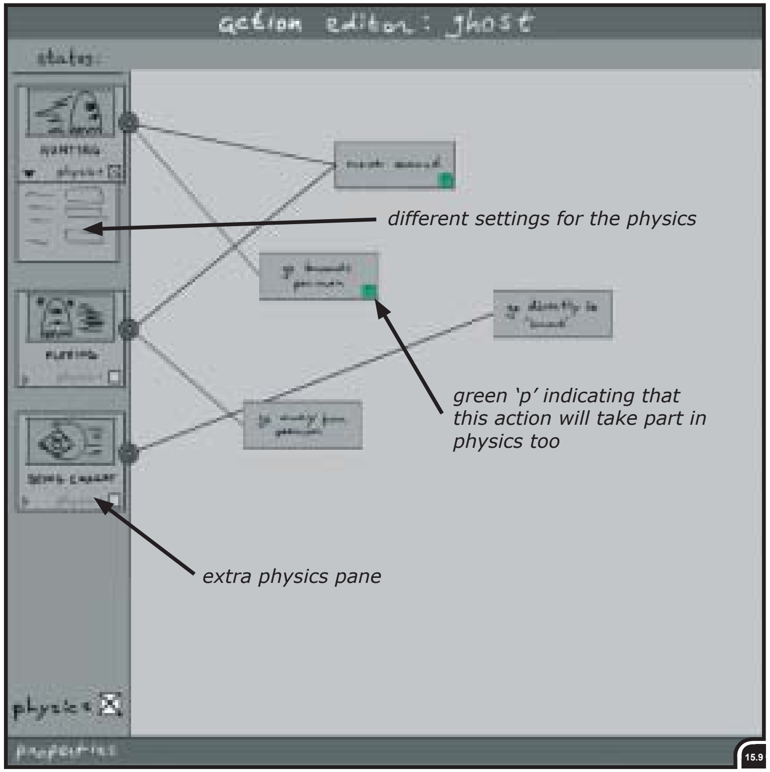
Assigning physics to an object is done at the level of states. The reason why it is done at the level of states is that this makes it possible for an object to take part in physics when it is in one state while being in another it does not have to take part in physics.
By clicking in a checkbox in the state, physics can be activated or deactivated. When activated an extra pane is added to the states (see fig. 15.9). Within this pane the specific state can be set to physics or not. By default a state is set to not take part in physics. When a state takes part in physics the connected actions will take part automatically too, off course. To indicate that a action takes part in physics a green 'p' is added to the action.

15.8 Reusing

Reusing states, actions and logic for another object or even for another project (file) is possible. States or complete state engines can be selected and saved as external files without any other data (such as geometry or animation curves). These can be imported within another project and exchanged with other persons for use within their own projects.
When saving states the user can choose to save only the states (as a framework for a new project), states with transitions or states with transition, actions and logic.
When saving actions the logic is as always included with the external file.

15.9 Properties

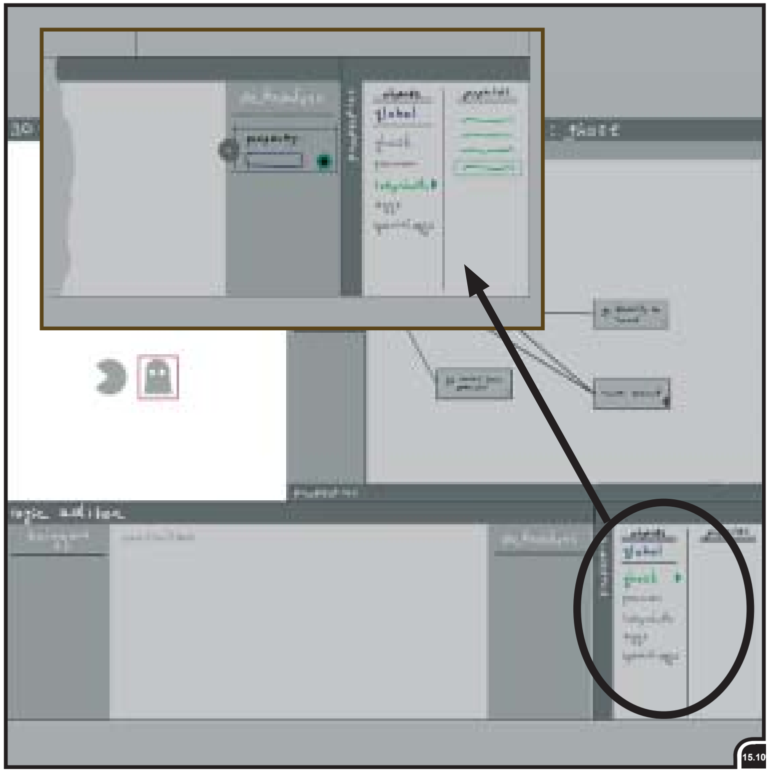
The properties pane is located next to the logic editor within the same window (see fig. 15.10). This pane makes it possible to browse to all the objects and their assigned properties.
A message trigger 'listens' to a certain property that is send by another object's actuator. Within this message trigger the property to be listened to has to be filled in by the user. Most of the time the user has forgotten the exact name of this property. When entering the wrong name, the field is cleared automatically when playing the game. Left wondering why the game is not working, the user has to find out that the name was typed wrong and has to look up the correct name.
The property pane solves this problem. As mentioned, the user can select an object, which has a property he is looking for, in the left column and see in the right column what properties are assigned to that object and select the right one to be entered within the property field of the trigger (or actuator) he is working on.



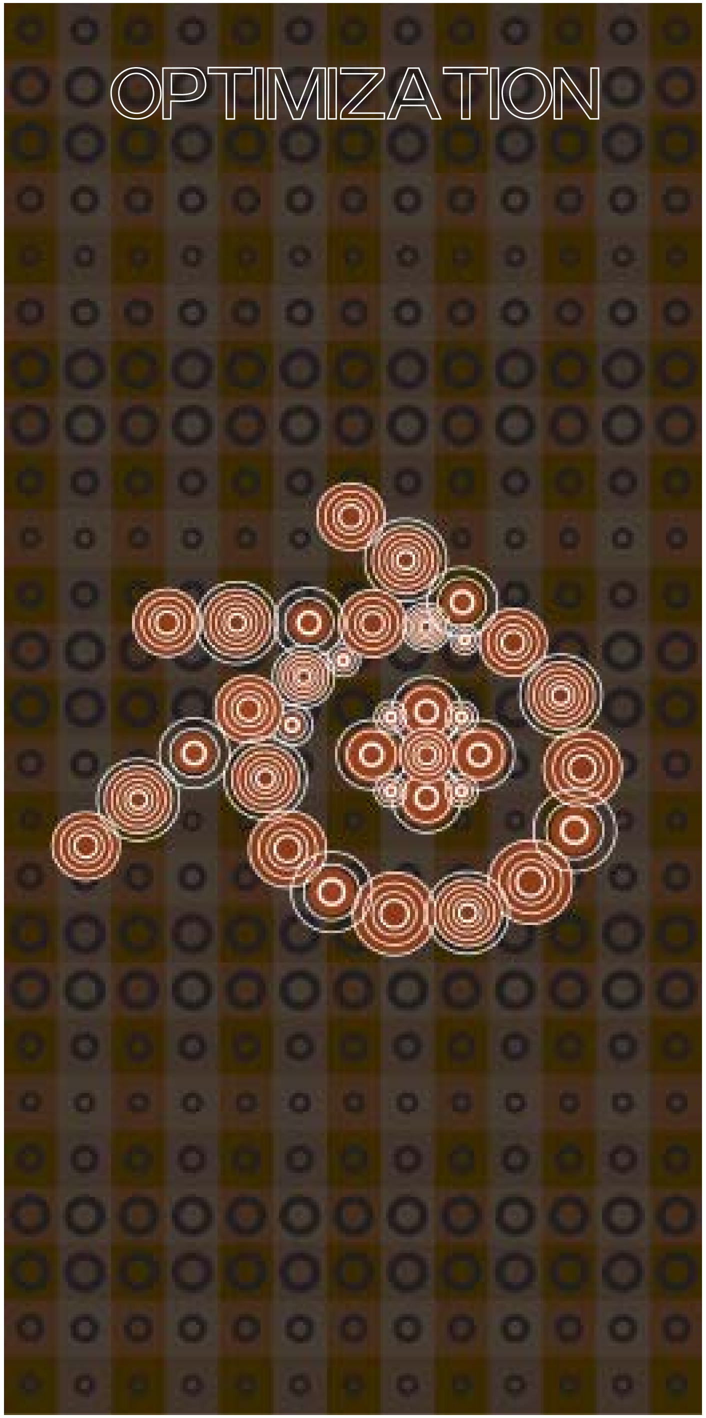
different settings for the physics

green 'p' indicating that this action will take part in physics too

extra physics pane



OPTIMIZATION



16. Evaluation final concept

The layout of this phase is as follow. Section 16 starts with the approach and the goal of the evaluation. Secondly, the process is described, results (findings) are listed and finally conclusions are made.
Section 17 describes the final design, a more concrete proposal for the GUI, with some conclusions of the evaluation already processed. Finalizing this report with recommendations, future developments and a process evaluation.

16.1 Approach

The goal of the evaluation was to find out if the final concept could come up to expectations and needs of creating interactivity.

The final concept (see section 15) was used as a framework for the evaluation. The final concept has nine sections. These sections were used for the evaluation too. For every section a few slides were created representing the steps needed to complete a section (1). During the evaluation these steps were shown one by one in successive order. In between every section some questions were asked. When all the questions were answered, the next section was shown and so on.

The questions used for every section were as follow.

Conceptualizing states

- Would you use a sketching/exchange tool like this?
- What's your idea of doing this?
- How would you like it to be done?
- Would you sketch on the screen or would you rather use pen and paper?"
- Are the tools for sketching on the sketching layer enough?
- Is it clear how to use the tools (simple sketching and typing text), or do you need more sketching tools?

Creating states

- Do you already use the idea of 'states' (per object)?
- Can you explain how 'states' are used in your projects at the moment?
- What do you think of assigning pictures to states?
- What kind of pictures would you use? Would you use something like the 'picture list'?
- Do you think the option of importing and exporting pictures to use within the states is useful and would you use it?

Creating transitions

- Do you think this second step, creating the transitions, was a logical step after creating the states?
- Was it clear to you that the (by default) grey arrows are the empty (no logic) ones? A transition can be two-way. Was this clear and logical within the design?
- Does it make sense to you to have a special transition editor? Else, how would you create the transitions (conditions)?

Global states

- What are 'global' states according yo you?
- Do you already use the idea of 'global' states and how do you use them?
- Could you reach the same effect with this concept?
- Is it clear if nothing is selected (objects) that the state editor is automatically set to 'global'?
- Does this meet the expectations you did have of 'global' states?

Creating actions

- Is it clear that you can subdivide states in actions?
- Do you think this is a 'clever' way of working? Or do you see other means?
- Do you think it is needed to provide a way of defining a order of evaluation of the actions?
- Why do you think so and can you give an example?

Assigning physics

- The physics are assigned to the states. What do you think of this?
- Is this the best way of doing it?
- What do you think of assigning physics to actions, or assigning it on the logic level (physics actuators)?
- What can be the advantages and disadvantages of the three means of assigning physics?

Creating logic

- The logic is subdivided by means of the actions. This is done to provide a way of organizing the logic (instead of a long list of triggers, controllers and actuators).
- You can't see all the logic at once, do you thinks this is a disadvantage?
- Can you imagine situations you want to see all the logic at once?
- Do you need a manner of editing logic that's divided over more than one action at the same time?

Reusing

- When saving state(s) engines, do you think it is useful to only save the state (placeholder) or do you only want to save all (states, transitions, actions, logic)?
- To reuse within a project there's the save/load option. To reuse a state engine within another project there's the export/import option. When exported, no geometry and animations are included.
- Do provide these two means of reusing enough functionality?

Properties

- Properties can be edited within every level (states, actions and logic). Is this needed?
- Global properties can only be assigned and edited within the 'global' editors, while it's possible to assign and edit object properties at every level within every object. Is this a useful way of working with properties?
- Do you oversee problems?
- Within the transition and logic editor it is possible to assign and select a property (of another object) for a trigger or actuator by browsing to that property. Do you think this is useful?

16.2 Process

The evaluation was done with two subjects, both of which were members of the content creation team of NaN. They were chosen because of their experience with games in general and their several years of experience with Blender and with the logic system.

The duration of the evaluations lasted for about 1 1/2 and 2 1/2 hours respectively. The difference in duration was mainly a result of discussions about related and not related subjects, but not (directly) important for the evaluation and conclusions of final concept itself.

The evaluations took place on two separate days. The second evaluation the day after the first. The first evaluation took place at the NaN office in Amsterdam, while the second took place in the office of the subject's home.

(1) The complete test program can be found on the accompanying CD

Both evaluations were done behind a computer screen, because the evaluation was presented with slides in Macromedia's Flash, as mentioned before. Besides the Flash screens, the subjects also got the different screens on paper.

Afterwards, the plan of the evaluation can be considered as succeeded. The questions asked (see previous page) were almost all answered. Most of them were not answered directly. Many questions were answered stepping through the slides, before the actual slide with questions came along. The questions slide served mainly as support for not forgetting to ask about certain aspects. Many questions do not have straight answers, but came up for consideration stepping through the slides.

16.3 Findings

What will follow now is a summary of the findings of the evaluation. The findings from the two different subjects are in such a way different that the findings of them will be described separately. When there is no subdivision made between the two subjects, it means that the two almost think the same about that specific part of the final concept.
The findings are order according to the subdivisions made in sections 16.1.

Conceptualizing states
Subject 1 (S1):
A means of conceptualizing states within Blender itself is 'out of scope'. S1 would rather use a white board or a existing application like Adobe's Photoshop or Microsoft's Visio. It would take to much effort to incorporate a 'feature' like this within Blender.

Subject 2 (S2):
S2 thinks that a 'feature' like this would be very useful.
However, the drawing tools should not be too extensive. Not necessary within Blender. It does not have to be a second 'Photoshop'. Simple drawing tools are sufficient.

The conceptualizing tools would also be very useful for story-boarding.

The conceptualizing tools (drawing tools) could also be incorporated through out all of Blender's creation tools (parts).

There could be a kind of interface (tabs, for instance) to get a overview of how many and to which objects a 'state concept' is assigned. This would make it possible to click different tabs and see all the 'state concept' in one overview, instead of having to click-select all the different objects within the 3D-view to look if there might be a 'state concept' assigned to.

Creating states
States are already used by members of the content creation team, but at present only within Python scripts. Blender does not provide a way of using states in a different way.
A Python script can actually be seen as one big 'state engine', according to S1.

Creating pictures for assigning to states will be too much. The two subjects would not do it. A description or a name in textual form would be sufficient enough. The team and projects are too small at NaN to use such a feature.

Subject 2 (S2):
The use of assigning pictures to states would (maybe) be useful when working with a team of 15 members.
Within a small team there is (most of the time) only one person who does create the interactivity for a game. It is not necessary to exchange files.
But a picture would tell more in a glance then words.
So, when files have to be exchanged and more people are working on it concurrently, it could be useful, but that has to be found out in practice. But this was exactly the purpose to add this feature.

It would also be useful for inexperienced users to get to get grips with the creation of interactivity and 'state engines' in special.
Providing ready-made states and 'state engines' with clear and simple icons (pictures), can explain and clarify these in a simple manner. Besides new users can learn how a 'state engine' should be used when these are provided ready-made within Blender.

Creating transitions
Subject 1 (S1):
There should definitely be a way of creating conditions for state transitions, but the location of the transition editor within the final concept is not the right place, according to S1.
The transition conditions can be created within the logic editor as well, with the addition of the 'change-to-state' actuator. When created within the logic editor it is also possible to add other actuators to be executed at the same time a state transition will take place. Likely candidates are 'property' actuators to set different properties for speed, health or whatever according to the state a object transits to.
S1 thinks that this is a more convenient for creating transitions, instead of creating the transition conditions in the transition editor and then creating the properties to be set to be executed immediately when entering the new state within the logic editor.

Subject 2 (S2):
The place of the transition editor (within the same pane as the state editor) seems like a good place for it. The transition conditions and the states belong closely together.

When a transition node is explicitly a 'one-way' transition, it has to be possible to 'turn-off' the direction you do not want to use. This makes it possible for someone else to see that it is meant as an 'one-way' direction and not an incomplete transition.

It would be nice if it is possible to 'turn-off' transitions to evaluate/test the different states independently from each other. This would speed up debugging. Instead of 'playing' through all the states a object has, the creator can test and debug different states independently.

Global states
The use of 'global states' is clear to both of the subjects. Global states contain logic and properties that is object independent.

To create 'global logic' within the current version of Blender, dummy objects are often used. For instance, invisible small planes which serve as a placeholder for logic. A big disadvantage of this is that somebody else is at search for a long time, clicking on a lot of objects before finding the one with the global logic assigned to it.

The means provided within the final concept for creating and editing global states/logic is a good and conveniently manner. Creating and editing global states by deselecting, meets the expectations of both of the subjects.

Global states are seen as a way of creating and defining the different scenes (levels) of a game. Every state within the 'global states' editor can be seen as a different level. Every state can turn layers visibility on and off within a certain level and will set the right properties for a level.

Concerning the 'one-click' (automatic) ordering of a state engine (within the state editor), both of the subjects can only say that a feature like that would be nice, but that it can be done in different ways. Maybe the user needs more control on how a state engine has to be ordered to get a better overview, without having to drag all the states to the 'right' position himself.

Creating actions

Subject 1 (S1):

The actions do provide a way of organizing the logic. S1 thinks also that it would be very useful when it is possible to define the order of evaluation of the actions. There are some situations that action A has to set a property before action B may be evaluated. If this property is set after action B is evaluated (when action B is evaluated before action A), action B will then be evaluated with the 'right' property not until the next game cycle. This could cause annoying delays in reaction time of some game elements. When it is possible to define the order of evaluation of the actions this can be prevented.

Subject 2 (S2):

The first response of S2 to the action editor was that it could be skipped. That it was an unnecessary step. Why not immediately go to the logic editor to create the logic for every state?
As a reason for this first reaction, S2 mentions the fact that he has learned it the 'wrong' way with the current version of Blender. Within this version all the logic is placed in a long successive list of triggers, controllers and actuators.
One reason that the actions are there in the final concept is to provide a means of organizing the logic. Instead of having one long list, there will be only small 'blocks' of logic divided over the different actions. When this became clear to S2, he thought this way of working provides a lot of advantages.

The use of actions serves the overview of the logic and makes it more convenient to reuse logic. Only an action has to be 'saved' to store all logic included. The name of the action provides at the same time a name for the included logic when saved.
Actions provide a way of ordering the list of logic attached to an object. If needed, the logic can be spread over more actions to reduce the logic bricks within one action.

Actions also provide a means to design higher levels of interactivity which is not possible with 'standard' logic bricks. Previously, this kind of higher level interactivity would only be possible to create with (very difficult) hand coding in Python. Now, it can be provided with 'special' actions. These special actions can include 'path finding' and 'terrain reasoning', for instance. The 'special' actions can be made available as ready-made modules, which can be downloaded.
The actions provide a good way of incorporating these kinds of higher level interactivity modules within the workflow available in the final concept. Besides the logic editor can function as a 'how to use' window. Instead of showing all the logic bricks, the space available within the logic editor can be used to show information about the module, about how to use it and by whom it is created etc.
S2 wonders why there are no icons (pictures) used for the actions as well. It would be inconsistent using icons for states, but not for actions.

In many cases the user will only use one action per state. Most objects do not have very complicated states, so one action provides enough 'space' to fill in the logic. In this light it would be useful when every state has one action connected to it by default. When a new state is created within the state editor one action (with the state name) is created automatically within the action editor.

Why not subdivide actions into 'brain', 'movement' and 'special' actions. 'Brain' actions would be placeholders only for logic that concerns properties, decisions etc., while the 'movement' actions would contain only logic concerning movement such as translate, IPO etc. actuators. The 'special' actions would contain the higher level interactivity, as described above. This can advance the overview, reuse and overcome problems assigning physics to states (see next topic "assigning physics").

Assigning physics

Subject 1 (S1):

S1 would not assign physics at the level of states. States need to be something more 'abstract'. By using them to decide if a object takes part in physics during a state or not, does make them too 'specific' again. That is not the way S1 wants to see the use of states.
S1 would assign physics at the level of actions. A 'special' physics action would do the job. This physics action could be connected to every state that has to take part in physics.

Subject 2 (S2):

S2 thinks it is a good way assigning physics at state level. He can not think of a better way of doing it.

A few things should be added. When a state takes part in physics, all the actions assigned to that state have to take part in physics to overcome problems. So one action can only be connected at the same time to states that do take part or to states that do not take part in physics. When one action (with logic only working when being part in physics) is connected to a state that takes part in physics and at the same time to a state that does not take part in physics, feedback has to be given in the form of a connection wire flashing red, for instance, indicating that it might be a problem or not working at all. A solution would be duplicating the action or making both of the states to which the action is connected to taking part in physics.
As mentioned above (see topic "creating actions"), different kind of actions can address this problem. 'Movement' actions can only be connected to states that do take part or do not take part in physics, but not to both at the same time. While 'brain' and 'special' actions do not have this 'drawback', because they do not contain movement related triggers and actuators, which are effected by physics.

Both of the subjects would not assign physics at the level of logic in the form of a special actuator, for instance. This meets my own expectations, because physics does not belong at the (lowest) level of interactivity.

Creating logic

Both of the subjects do not see any problems creating logic. Mainly, because the interface resembles closely the current interface for creating logic.
Both of the subjects think also that the close relationship between the action and logic editor will benefit the clarity of arrangement. Selecting an action in the action editor will show the logic within that action in the logic editor immediately, provided that both of the windows are open at the same time.

Reusing

The operations of saving and loading is not needed according to both of the subjects. When the Blender file is saved, everything is saved, including states, actions and logic. To 'load' a state, for instance, for use with another object the user only has to add a existing state with the 'add' button.

The possibility to export (and import) logic to a special file format or to the standard Blender format, but without all the meshes, animation curves, materials etc. would be useful agreed both of the subjects.

The possibility to ex- and import is only needed at the state and action level. Not at the logic level. When exporting or importing an action, all the logic included will be exported as well. So there is no need to ex- and import at a lower level.

When exporting one state at state level all the attached actions, logic and related properties have to be exported by default. When exporting two or more states (or even a complete state engine), the transitions have to be included as well.

Properties
Using and assigning properties is clear to both of the subjects. The possibility to 'browse' through all properties of all objects within a project to be able to select the right property to paste within a subject field of a trigger or controller would be very useful.
In current Blender interface, when the wrong property name is typed, the name is automatically cleared. Instead, the user has to select the right object with the property he is looking for and remember the right name and spelling, then select the object he is working on and type in the correct name.

Subject 1 (S1):
The subjects would like to see that a sort of spreadsheet will be added to Blender. Within this spreadsheet could also all the properties be edited. The spreadsheet would show all the used properties and which object are using them. With a view clicks (without selecting objects one by one) a whole list of (object) properties could then be changed, removed or added.

16.4 Conclusion

The goal of the evaluation was to find out of the final concept could come up with the expectations and needs of the users (represented by two members of the content creation team).

Did the subjects think that the design would suit their workflow and mental model?
It is clear that the concept is an improvement over the current logic editor. This can be concluded from comments of the subjects.
However, many aspects of the final concept can only prove their use when actual implemented within Blender. According to the subjects, it is hard to forecast if all the aspects will be used as long as they are not used for 'real'.

Are the subjects willing to change their current workflow in favor of the new editor(s)?
They are very used to the current workflow, the final concept has to prove itself when actual implemented. One of the subjects thought aloud, during the evaluation, that they (the members of the content creation team) did learn it the wrong way, because Blender did not provide any other way. In other words, it is hard to oversee if they will use the new editors of the final concept, because they are not used to it (yet), but the subject in question could say that the state, transition and action editor and modified logic editor would be very useful additions to speed up the workflow and ease the implementation of higher level interaction without (or with minimal) coding.
The final concept allows to skip these three parts to create the complete game only in the logic editor, (almost) just like it is done within the interactivity window (current logic editor) now.

It can be concluded that the design of the new state, transition and action editors and the modified logic editor is successful. Together, they provide means to implement higher level interactivity and to order logic.

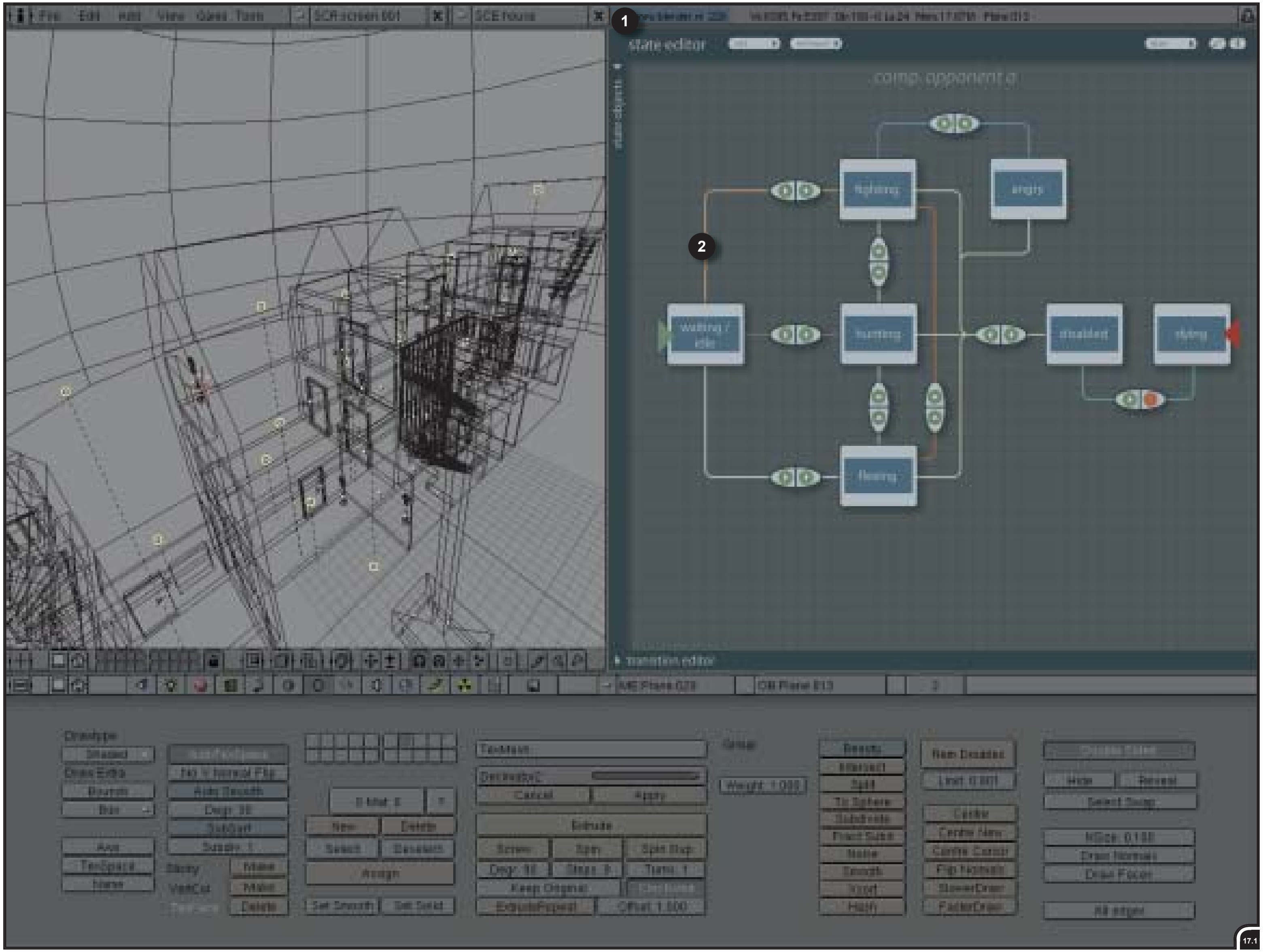
17. Final design

The following pages describe the 'final design'. Compared to the final concept (described in the previous chapter), the final design has been improved by adding some of the changes proposed in the evaluation and by applying the graphical style used in the concepts in chapters 14 and 15.

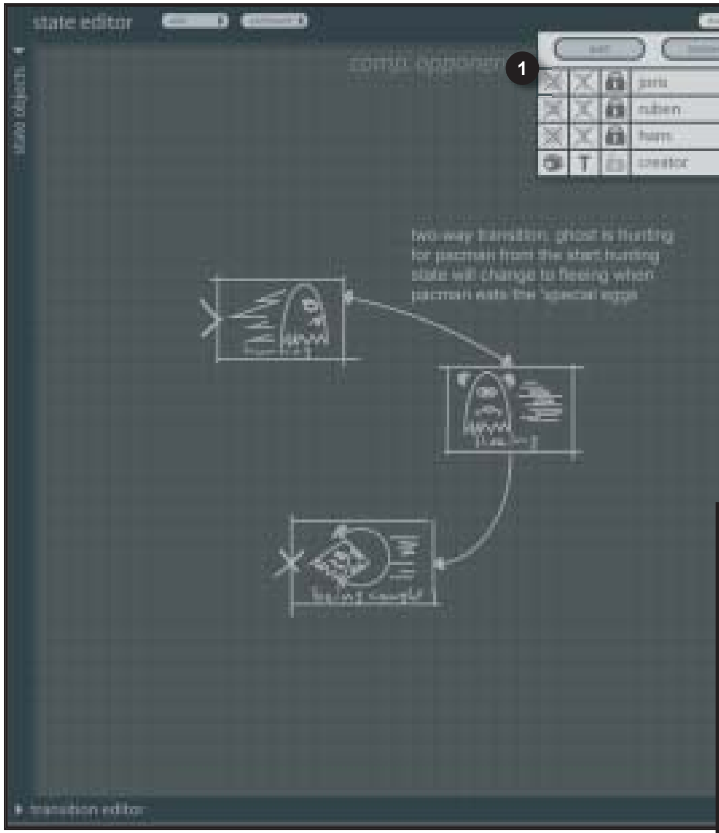
Some of these changes (such as the draw pane on page 74) are not yet tested. However, the expectation is that these are improvements since they were proposed by the subjects in the evaluation and they are not radical changes to the design.

The appearance of the GUI is dark, the reason for this is given in section 11 (page 43). The style of the interface elements is flat instead of the semi-3D look most modern computer applications have. This is a personal preference but it has been topic of discussions with several members of the content team and they prefer this style as well.

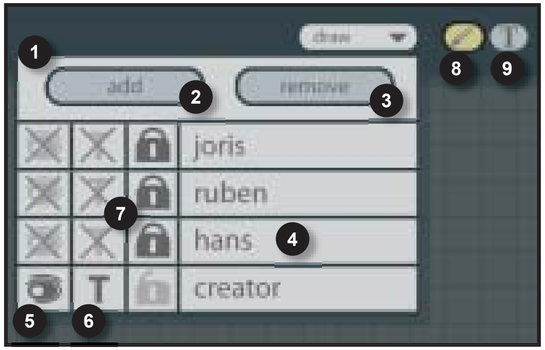
The state editor window (1) next to a 3D-view in Blender. A dark color scheme is used and the connection wires (2) have different colors to better distinguish their path, as was described in section 11. The connection wires have rounded corners to ease the process of following their path.



17.1



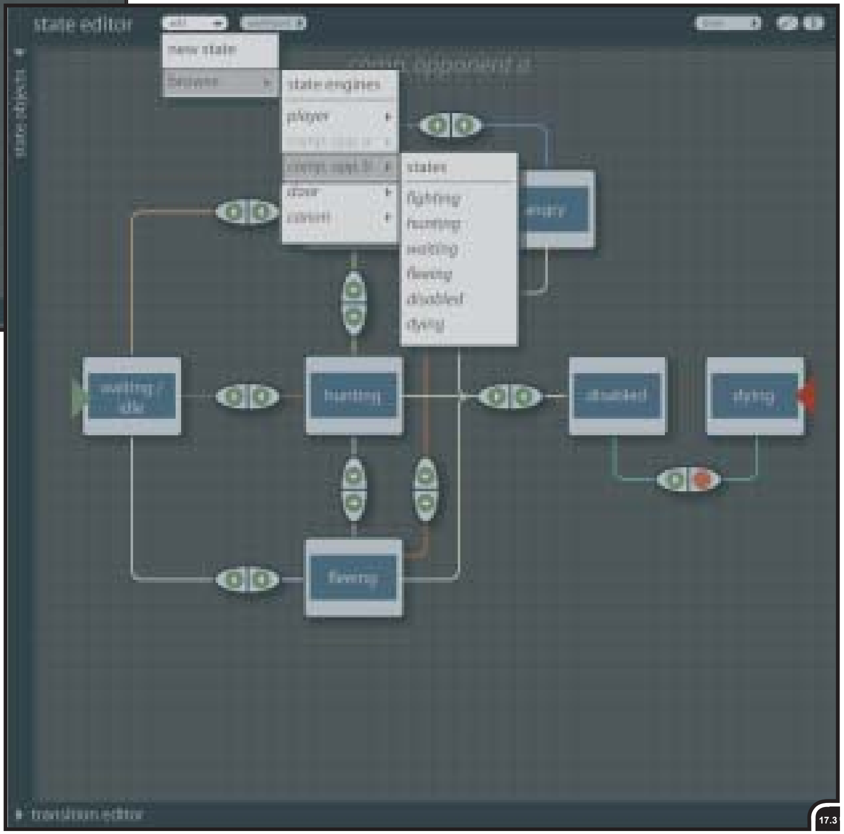
two-way transition, ghost is hunting for pacman from the start. hunting state will change to fleeing when pacman eats the 'special eggs'

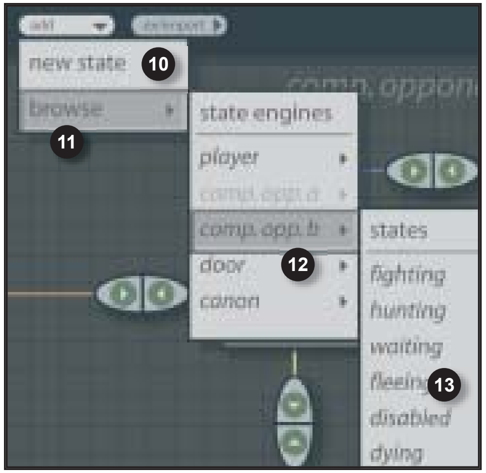


◀▶ The drawing (conceptualizing) tools (1) can be opened by clicking on the 'draw' button. The draw pane exists of an add button (2) to add a new layer for comments and a remove button (3) to remove a selected layer. A name (4) can be given to each layer to indicate who has added the comments.

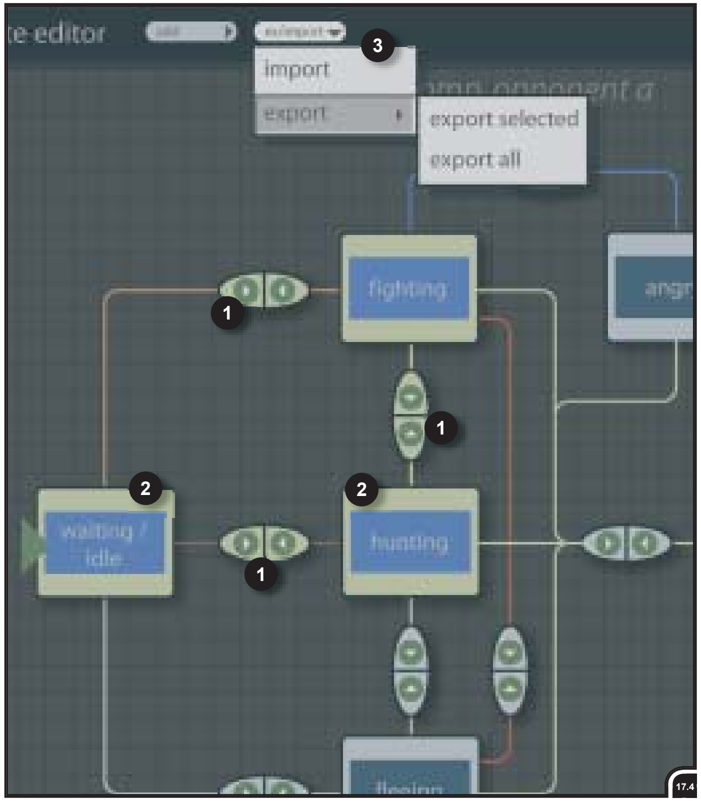
Visibility of drawings within a layer can be turned on and off (5). Text visibility within a layer can be turned on and off as well (6). To protect content of a layer from being altered, a layer and all its content can be locked (7).

The user has two tools at his disposal. The (simple) drawing pencil (8). In this example this tool has been selected (light yellow color of the button). And the type tool (9). With this tool selected, the user can type anywhere on the canvas.



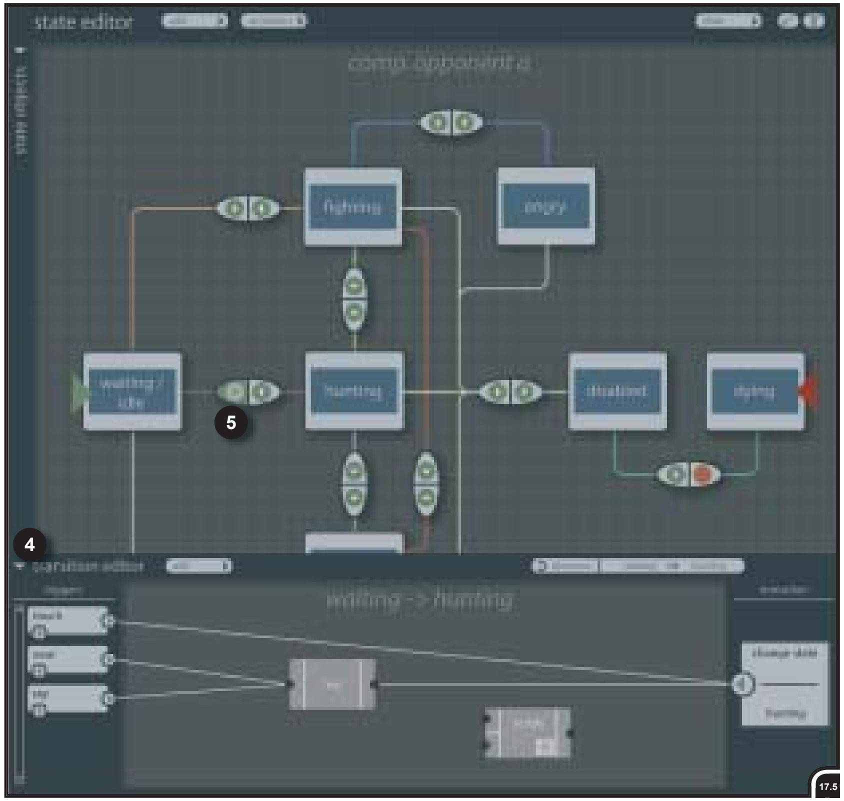


◀▶ Adding 'new' states to the work pane can be done in two different manners. A new (empty) state (10) or a finished state or (complete) state engine can be added. To add a state engine, the user has to browse (11) for the object with the state engine to be reused (12) and click on its name. Only the object names with a state (engine) are listed. To add a single state (including, of course, all the connected actions and logic), the user has to browse one level lower, to the list of states of an object's state engine (13) and click on one.



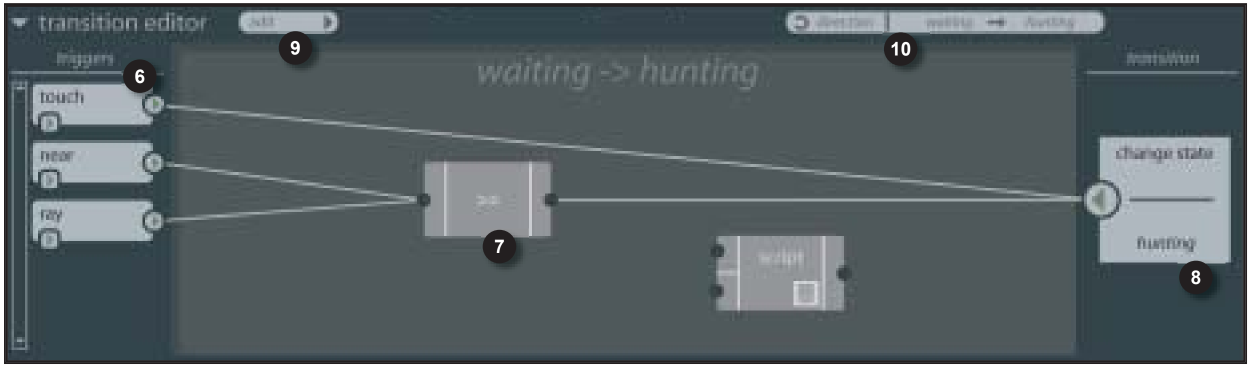
▲

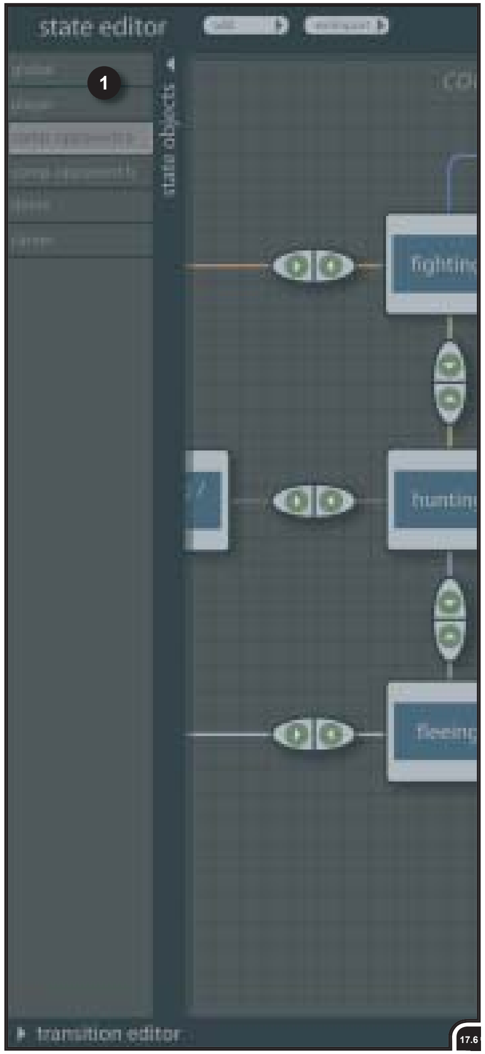
States (2), the transition nodes (1) in between, the included actions and logic can be exported to be used by another person and/or within another project.
To export states (state engines), first the states to be exported (2) have to be selected, indicated with a light green color. When two or more states are selected and they are connected together, the transition nodes will be selected automatically too.
With the 'export selected' function (3) the selected states (and transitions) are exported. Instead of exporting only selected states, a user can also 'export all' (3). With this function the complete state engine (states, transition, actions and logic) is exported.



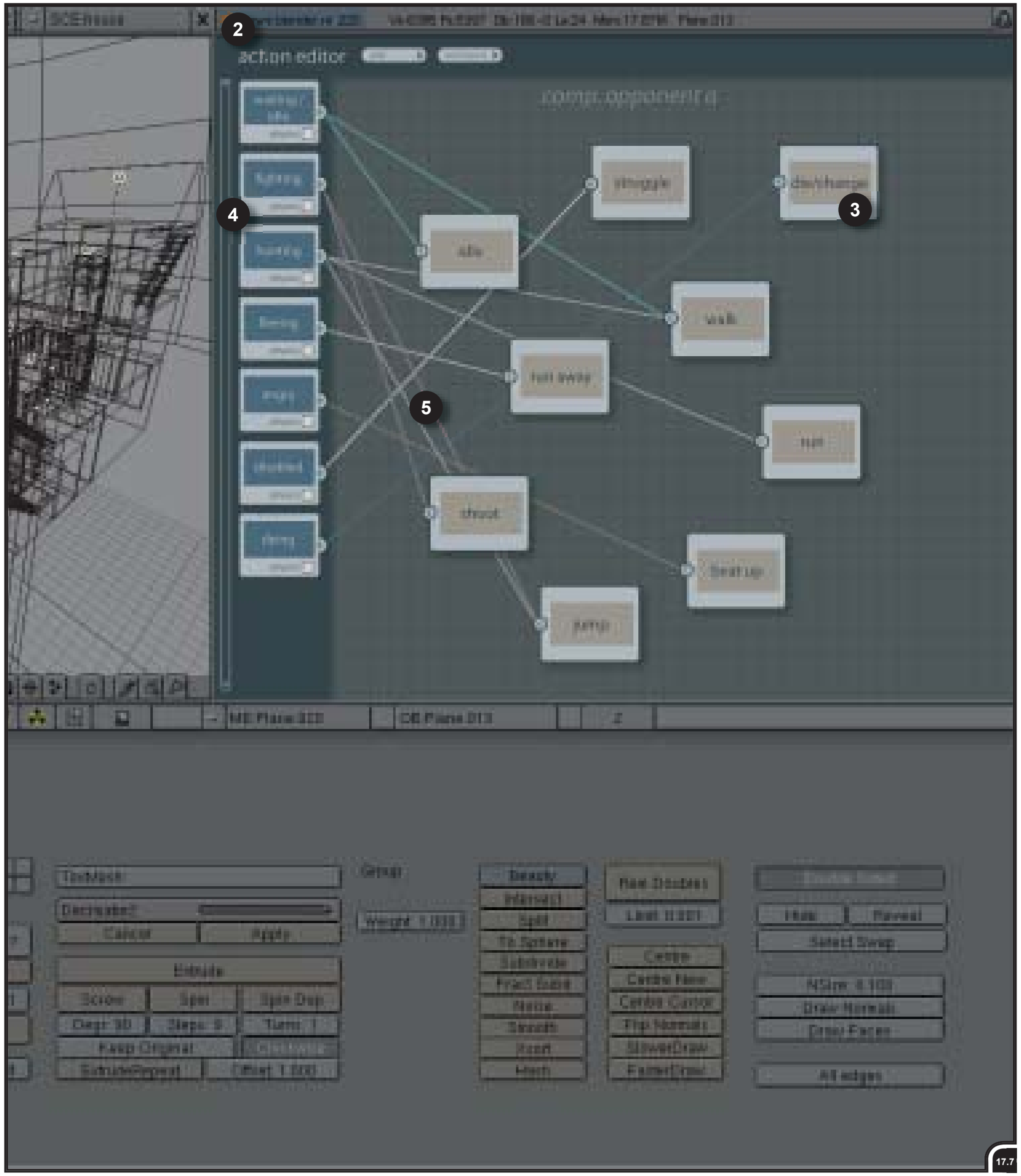
▲ The transition editor (4) has been opened within the state editor window. In this example the transition editor is showing the (already created) transition conditions for the selected transition arrow (5), from the 'waiting' state to the 'hunting' state.

▼ The transition editor pane showing triggers (6), different controllers (7) and the (special) 'change state' actuator (8). Adding new triggers and controllers (9) is analogous to adding states (previous page).
Instead of having to select (and find) the opposite transition arrow (5), a quick 'change direction' button (10) can be used.

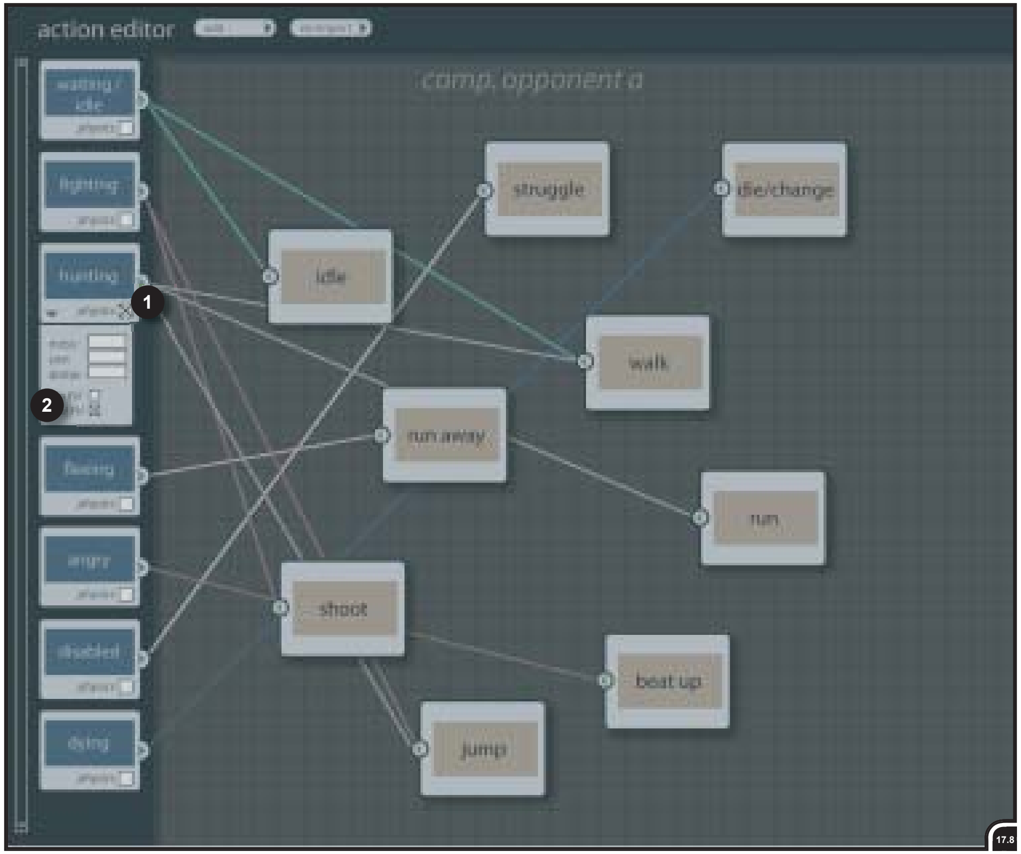


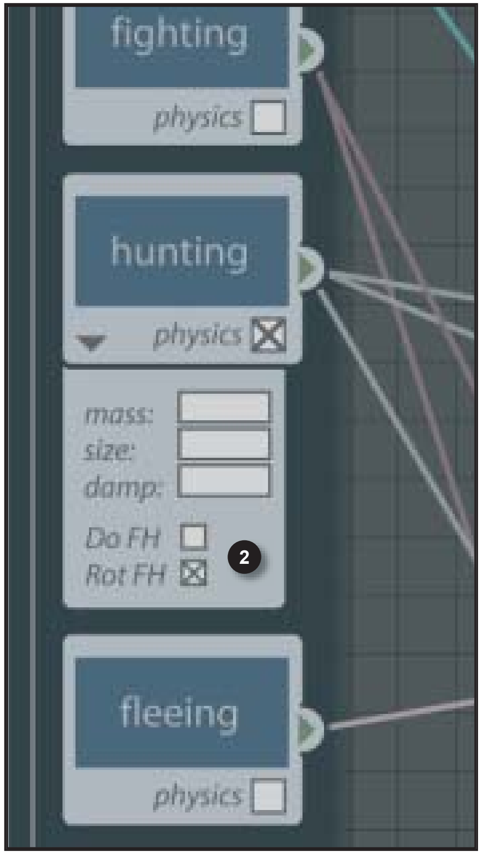


When states are being conceptualized and the file is shared with colleagues to be approved and to get feedback, all the objects with state concepts are listed (1) to 'browse' fast trough all the state concepts, instead of having to search and click trough all the objects within the 3D-view to see if there might be a state concept created for. This object list speeds up the workflow and makes it also possible to switch fast between objects (with states) when editing logic.



The action editor window (2) next to a 3D-view in Blender. Action and state editors can be opened at the same time if necessary. The different actions (3) can be moved freely within the action pane. The actions are connected to the states (4). All the states created within the state editor are listed within the action editor too. They are only listed and can not be edited within the action editor. The connection wires (5) do have different colors to better distinguish their path. All the connection wires connected to the same state do have the same color. The different colors are automatically and randomly assigned to the wires.





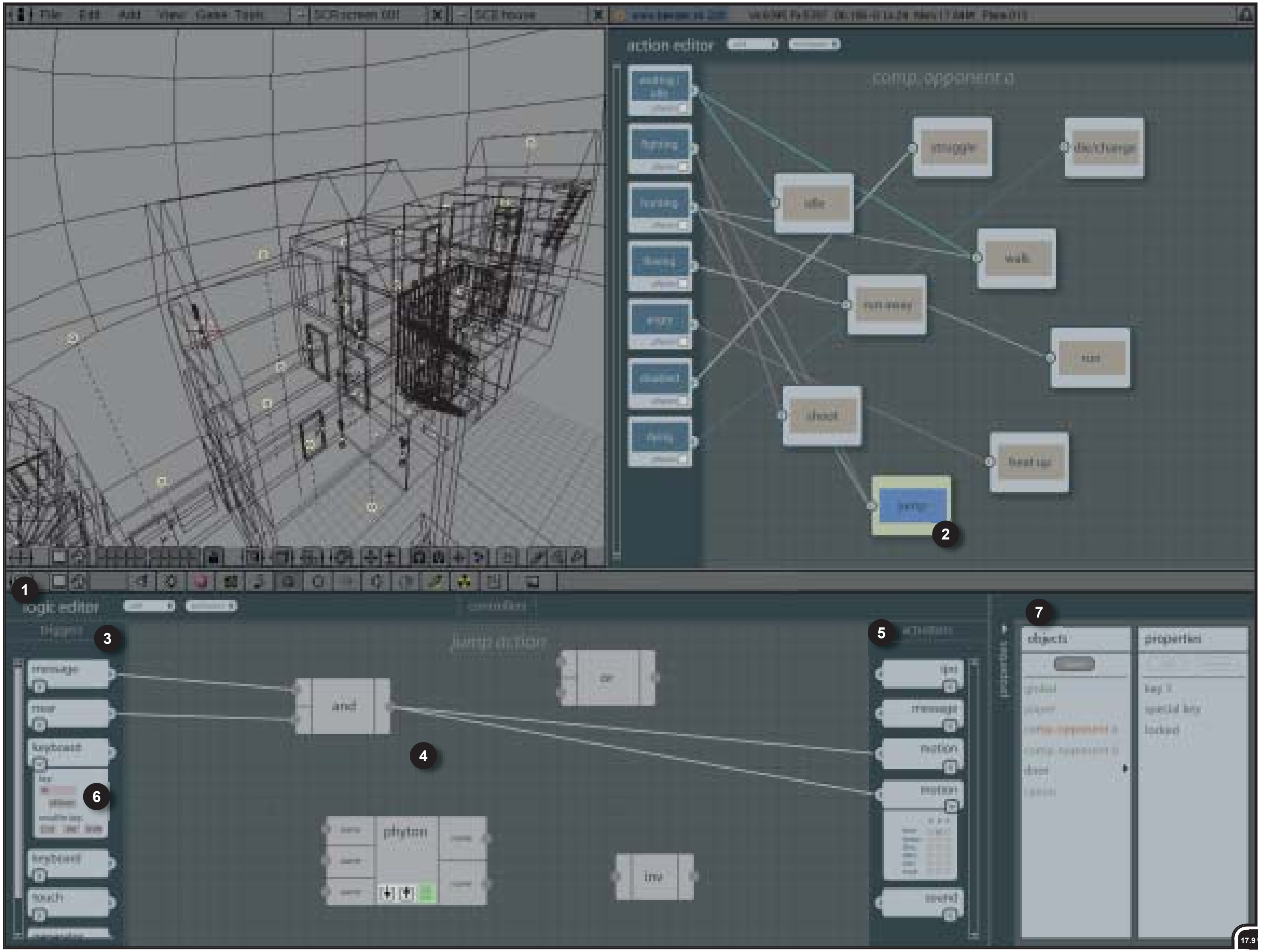
fighting
physics □

hunting
physics ☒

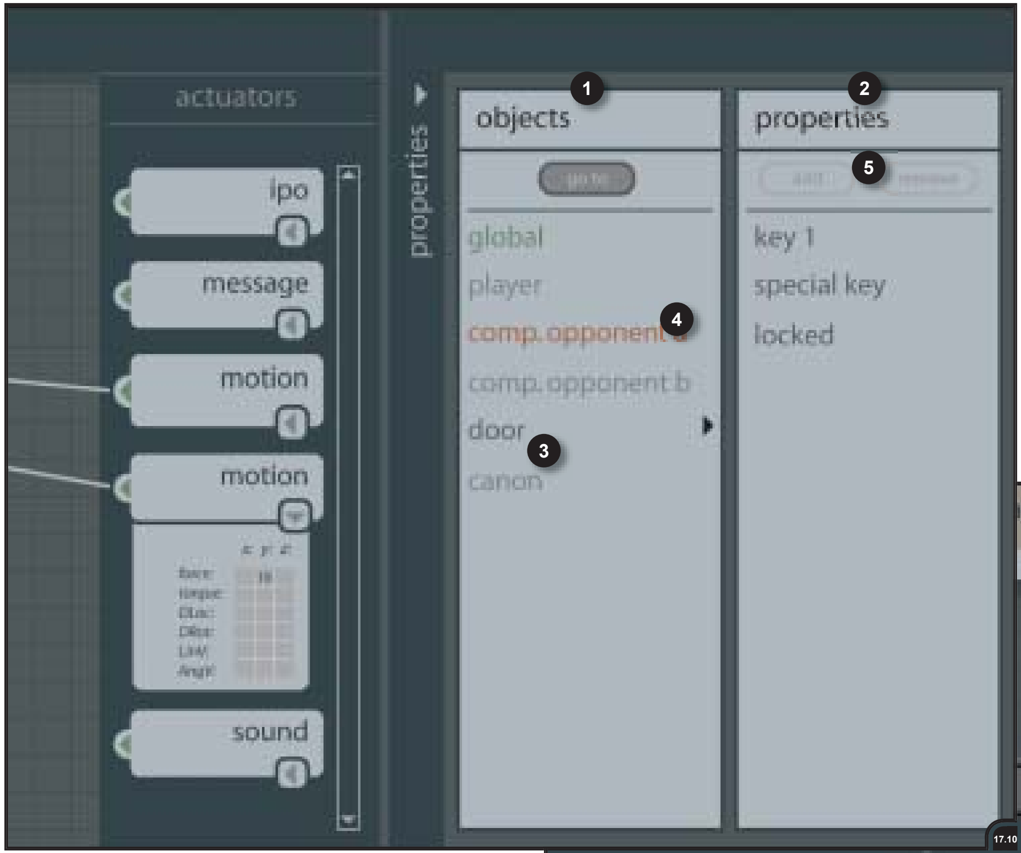
mass:
size:
damp:
Do FH □
Rot FH ☒

fleeing
physics □

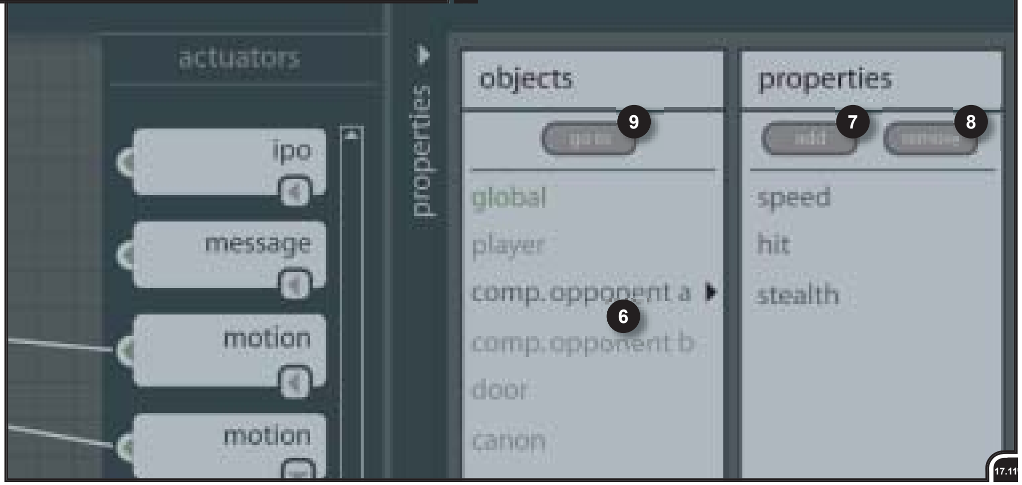
Assigning physics is done by checking a checkbox (1) on a state itself within the action editor. By checking this checkbox (every state has) an extra pane with the physics properties becomes available (2). Within this pane properties like ‘mass’, ‘size’, ‘damping’ etc. can be filled in and/or edited. Making a state part of physics also means that connected actions will take part in physics.



The new logic editor (1) at the bottom. The logic editor is showing the logic for the selected action. In this case the 'jump' action (2). The logic editor has three columns. These columns exist of triggers (3), controllers (4) and actuators (5). The triggers and actuators are listed and do have a fixed location, while the controllers can be moved around freely within the bounds of the controller column. Properties of triggers and actuators can be edited in a properties pane (6). This properties pane can be expanded or collapsed. At the right of the logic editor the properties pane (7) is located, presented in the next page.



The properties pane consists of two columns. An objects (1) and a properties (2) column. In the properties column all objects that have properties assigned are listed. In the properties column all properties are listed for the selected object in the objects column. This makes it easy to find and copy a property from another object.
The dark grey object with the arrow (3) is the currently selected object, showing its properties. The red object (4) is the object for which the logic is being edited.



If the object selected (6), within the objects list, is the one for which the logic is being edited, the 'add' (7) and 'remove' (8) buttons become usable. Properties can now be added and removed. To switch between objects to edit and add or remove properties, an object from the objects list can be selected and the 'go to' button (9) can be used. Clicking this button will cause the content of the logic, action and state editor (if open) to be changed to the new 'selected' object.

18. Conclusions

The final concept/design has shown that the design fulfilled the requirements and wishes of the users (presented by two members of the content creation team of NaN during the evaluation).
However, some aspects can be improved. These aspects are described in the recommendations. Other aspects do need to be explored yet and are described in the future development section.

18.1 Recommendations

Many aspects of the final concept (and design) that were not touched on, but emerged (mainly) during the evaluation and are useful to implement or to be improved in one way or another, are described below.
The aspects are described in the same successive order as the final concept (section 15) and findings (section 16.2) as long as there is something to say about.

Conceptualizing states
The method of conceptualizing states can be conveyed through more of the interaction creation parts and maybe in other parts of the Blender interface too. Within the interaction creation part, it can be included within the action and logic editor as well. Within the action and logic editor it could be useful in itself to only write down comments and draw attention marks, for instance.
To what extend the drawing (conceptualization) tools can and must be implemented within other parts of the inteface was yet to be explored.

Creating states
Concerning the states, the use of pictures to clarify the content and use is a bit tricky. In small projects, users will probably not use it, because it is too much work. So, besides the possibility of including your own pictures, a picture list of 'standard' and other used pictures (icons) can be included to speed up things and to help clarifying the use of states to new come users. The same idea could be used for actions. Why include the use of pictures within states and not within actions? Within actions it can be as useful as within states.

Creating transitions
Within the creation of transitions it must also be possible to set a transition node explicit as a 'one-way' direction transition. This clarifies to other users that this was intended that way by the first creator.

Creating actions
As mentioned in the findings, it should be a welcome addition to be able to define the order of evaluation of actions in certain conditions. So, this 'feature' would be good to include, but it has to be explored in what manner.
Subdividing actions into different kinds of actions, like 'brain', 'movement' and 'special' actions can be a good addition. It will bring more depth and versatility to the use of actions. But this has yet to be explored.

Assigning physics
It was chosen to stick with the solution given in the final concept (and design), despite the fact that both subjects did not agree on this subject. One of the subjects did think that this was the best solution, though, while the other subject thought that assigning physics directly to the states was an infringement of the higher abstraction level of the states. Instead, the subject would like to see a kind of 'physics' actions that could be connected to every state that does need to take part in physics.
My opinion is that this method is more or less the same as marking a checkbox on the state itself, saying "now you take part in physics" (as being proposed in the final concept).
Assigning physics with a kind of 'physics' action would infringe, on its turn, the idea behind the actions. Physics are on another level than actions. My choice is that it makes more sense marking a checkbox on the state, instead of connecting a special physics actions to the state.
It remains the question if this is indeed the best solution and would fulfil the expectations of users the most (a real choice can not be made on the ground of comments of two subjects).

Reusing
A special file extension for the exported logic is needed to distinguish it from the normal 'blend' extension.

18.2 Future development

After the completion of this project there are still enough aspects that may be explored. Aspects for which no time was left and/or were overlooked during the project, but do need attention before the design can actual be implemented within Blender. On the other hand, decisions about some other aspects can probably only be made when the design is actual implemented within Blender. These aspects do need 'real-world' testing within the 3D environment of Blender before suitable choices can be made.
Other aspects were left aside, because they proved to be not important enough to accomplish the two main goals (within the given time of the project) of creating higher level interaction and ordering logic.

Some aspects that can be explored and further developed will be listed below in arbitrary order.

Debugging:
Probably the most important aspect that needs attention is debugging. Debugging of logic networks. At the moment debugging occurs by playing the game and watching in what part it will go wrong. An addition of visual debugging the logic networks might be a real time saver if networks can be checked immediately within the (state, transition or logic) window itself. For example, by 'firing' color codes trough the connection wires, watching at what point a signal stops unintended etc. This is definitely an aspect worth exploring.

Ordering of state engines:
States can be placed freely within the state editor window. This freedom can be useful to a lot of users, but can also be confusing. Other users than the initial creator can have difficulties understanding a state engine. As mentioned before (final concept) there must be means of automatically ordering state engines. On what principals, algorithms and/or rules this has to be done has still to be explored.

Standard icons for states and actions:
It has to be explored what 'standard' states and actions are, before, clear icons (pictures) can be made. These standard icons (for states and actions) can prove to be very important to get to grips with the creation of (high level) interactivity in the first place and to encourage the use of the state, transition and action editor.

Triggers, controllers and actuators:
Many of these can be copied from the current Blender interface for creating interactivity. However, the current set needs to be reevaluated. The controllers are a bit of a different story. At the moment there are only four and in the final design the user has much more freedom with controllers. The user is able to connect more than one controller in a row between one trigger and actuator. The idea is to use many more, but 'smaller', controllers to build more complex interaction. How many and what kind of controller 'bricks' are needed is still open as well.

In my opinion these are the most important future development aspects of the design for creating interactivity within Blender.

Some smaller aspects are, for instance: In what order states, from top to bottom, within the action editor are to be listed? In what manner can a user see which and how many actions there are connected to a state within the state editor, without opening the action editor?
What must a spreadsheet editor look like to be able to edit multiple properties at once? Are special 'transition' triggers and/or controllers needed? If so, what kind of triggers and/or controllers?
What will be the order of listing of triggers and controllers within the logic editor? Must this be alphabetic or in order of creation?

There will be probably many more questions to be answered before the creation of (high level) interactivity works fine according to everyone's needs and wishes.

19. Process evaluation

I think I have to start first with the most 'sensational' event, during my graduate project. Five months after I started my project (three months before graduating) the company, Not a Number BV, went bankrupt. The atmosphere the last week before bankruptcy was not that good, understandably. It was inconvenient for my project, but it was the most inconvenient for the employees and Ton Roosendaal (founder of NaN) of course.
Fortunately, my project progressed so far that I did not need feedback every day/week anymore. I still had to perfect my concept, but the information gathering was finished. However, the drawback of working at home for three months was to keep on going and to get a fresh view on certain aspects.
Besides the personal inconveniences of the bankruptcy, there is also the fact that it is uncertain if something will be done with the outcomes of my project. It would be pity if this were not the case, but luckily there might be a change. At the moment of writing, it seems that NaN will continue in a different structure. Yet, it has to be seen what can and will be implemented.

As was mentioned in the introduction of section 14, the design process in this project deviated from the standard Industrial design process (especially in the concept phase). Usually, three or more concepts are generated, all fulfilling the requirements. Then one concept is chosen that fulfills the needs and wishes of users the best. It is a process with divergent and convergent phases.
The concept phase I completed was an iterative process. Just one concept emerged from the ideas and was further improved step by step. Feedback was given on each concept by members of the content creation team. This feedback was used to improve the concept and resulted eventually in the final concept.
Designing a user interface is complicated because many items have to be addressed in one design. It was therefore different to generate three or more complete different concepts. Instead, just one concept was generated and broken down in smaller parts that gradually improved in every new concept. This project is not the first user-interface project with this process: the same procedure has been reported in other projects at the faculty of Industrial Design.

The outcome of my project, the final design, can be a good solution within Blender. Before I started with this project I did not know much about this topic and I did not have experience with game creation at all. Although, I had some experience with 3D creation programs, starting fresh is a strength of industrial designers. Not knowing a lot about one aspect, but knowing enough about many aspects. So, I could tackle the problem(s) with a fresh point of view.
As mentioned above, I think the final design is a good solution for in Blender, but it still has to be seen to what extend it will be a solution in general to the creation of higher level interactivity. The final design has been created with Blender in mind and with feedback of professional users of Blender.

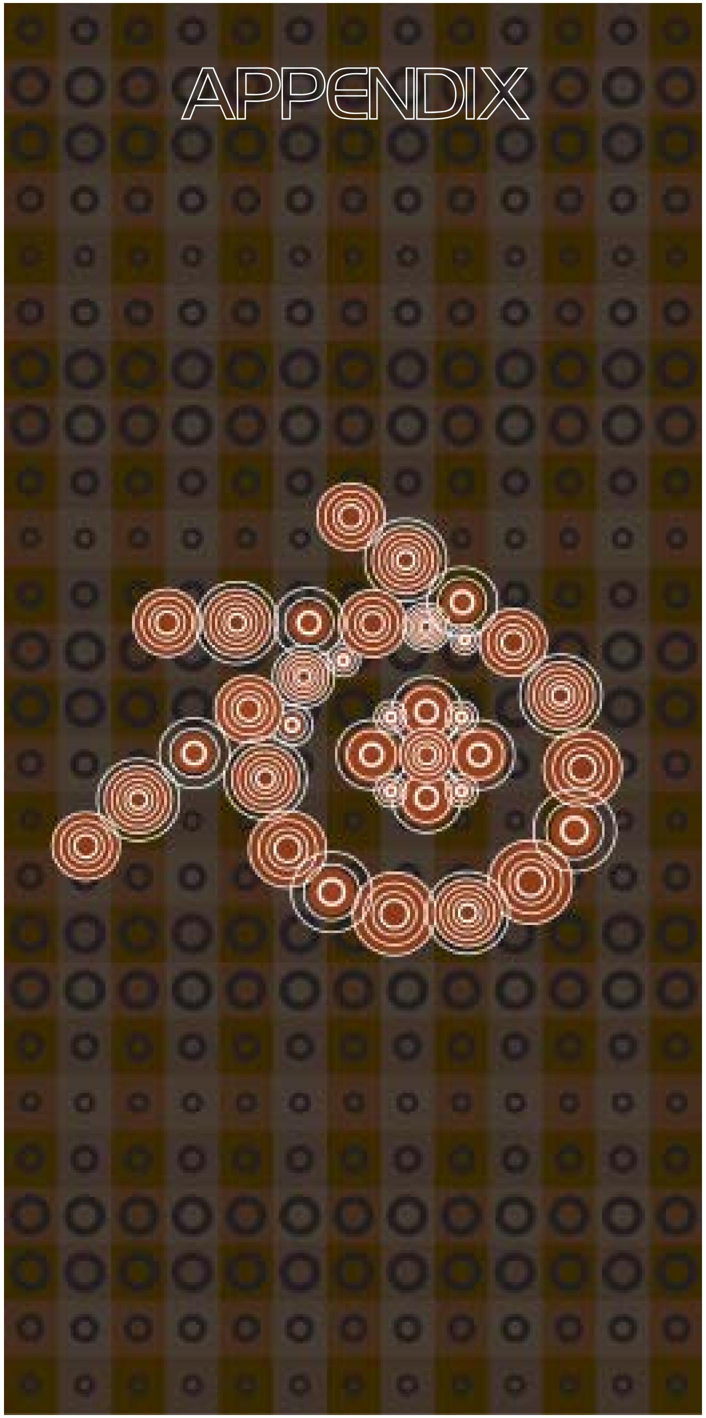
Being over at the NaN office once, in Eindhoven, I attended a meeting with both developers and members of the content team. The meeting was about triggers, controllers and actuators. It had to be defined by the members of the content team in what manner these had to function actually. This had to be discussed with the developers, because that were the people creating the code behind the functionality. Afterward, I did find out to my surprise that this was the first time (for a long time) such a meeting took place between users and developers of the Blender software. Here, another strength of an industrial designer appears. Being able to listen to people with totally different backgrounds and interests and then bringing together the requirements and wishes.

To conclude, I will quote Warren Spector of ION Storm Austin (creators of Deus EX and Thief III) in an interview with gaming magazine Edge (number 107, February 2002):

"Gaming's best hope for a grand future is to give people the power to decide how to interact with the game world."

So, the best way to do this is to give people the power to create their own games without coding......

APPENDIX



Appendix I: Blender features

Extensive feature overview, extracted from the Blender site (www.blender3d.com).

Features general
- Single integrated program: download size 1.6 MB!
- Modeling of polygon meshes, curves, NURBS, vector fonts and metaballs
- Deformation lattices and bones
- Animation with keyframes, motion curves, morphing and inverse kinematics
- Particle systems
- Rendering: solid, transparent, halo/lensflare
- Sequence editing of images and postproduction effects
- Real-time 3D engine integrated
- Embedded scripting language (Python) for advanced control
- Powerful object oriented data system
- File management included

Files
- Saves all work in a single file
- Other Blender files can be used as libraries
- Read/write image and texture files - format TGA, JPG, Iris, SGI Movie, IFF or AVI
- Import and export DXF, Inventor and VRML 1.0 & 2.0 files

Curves
- Bezier, B-spline, polygon
- Procedural extrusion and beveling
- Any curve can be used as a bevel
- Support for vector fonts allowing the use of 3D text

Meshes
- Uses triangle and square polygons
- Extrude, spin, screw, bend, subdivide, etc.
- Real-time 'vertex paint' to add vertex colours
- Subdivision Meshes
- Soft editing tools for organic modeling
- Radiosity solver for calculation of realistic 3D environments
- Python scripting access for custom modeling tools

Animation
- Motion paths: Bezier curves, B-splines or polygons
- Motion curves: XYZ translation/rotation/scaling
- Innovative 'Ipo' system integrates both motion curve and traditional 'key-frame' editing
- Vertex key framing for morphing
- Inverse kinematics and Bones
- Character animation editor
- Constraint system
- Animation curves for light, materials, textures and effects
- Python scripting access for custom and procedural animation effects

Game creation and playback
- Collision detection and dynamics simulation included
- Supports all OpenGL™ lighting modes, including transparencies.
- Defines interactive behavior of Objects with a simple buttons menu
- Python scripting API for sophisticated control and AI, fully defined advanced game logic
- Playback of games and interactive 3D content without compiling or preprocessing
- 3D Audio, using the Open Source OpenAL™ toolkit
- Multi-layering of Scenes for overlay interfaces
- Content can be saved in a single file, and played back with the Blender 3D Plug-in
- Full Animation features supported within the real-time engine

Render
- Rendering in foreground with direct output
- Can be initiated with a single key press at any level
- Three rendering layers: solid, transparent and halo's.
- Delta accumulation buffer with jittered sampling for perfect antialiasing
- Resolution up to 10000 x 10000
- Field rendering and pre-gamma correction for the best video output possible
- Panoramic rendering
- Soft shadow
- Plug-ins for textures and postproduction

Light
- Local lights, spotlights, hemispheres and suns
- Textured lights, spot halo's
- Shadow buffered system with volumetric effects
- Selective lighting for individual objects

Windows
- User configurable window layout
- 3D Window: wire frame/solid/OpenGL-lighted/rendered
- Animation curve/keys window
- Schematic diagram window
- Sequence editing window
- Action editor
- File selecting and file management window
- Text editor included for scripting and notes

Supported platforms
- Windows 95, 98, 2000, ME, NT (i386)
- Linux (i386, Alpha, PowerPC) glibc 2.1.2
- FreeBSD (i386) 3.4
- BeOS (i386) - no further development planned for this OS
- SGI Irix (6.5)
- Sun Solaris 2.6 (sparc)
- A Mac OSX version will be available Q4 2001

Appendix II: Questionnaire

Questionnaire about interactive design with Blender

- Does the work environment encourage you?

- How long do you already work with Blender?
- What kind of interactive productions have you made with Blender?

- When you make an interactive production in Blender, what is the order of working? Where do you start? Can you describe your design sequence? (Storyboard, character describing, interaction diagram etc.)
- What kind of material (paper, pencil etc.) do you use during the design process?

- Do you always work alone on a project or also with more people? In the latter case, with how many did you work on a project and how were the task divided?
- Can you describe what was different working with more people on one production compared to working alone?

- Can you show a production you made? How was your work process making this particular production?
- Where did you get your inspiration from for this project (and others)?
- Can you explain the difficulties you came along building this particular game?
- What did you leave from the initial plan during this project?

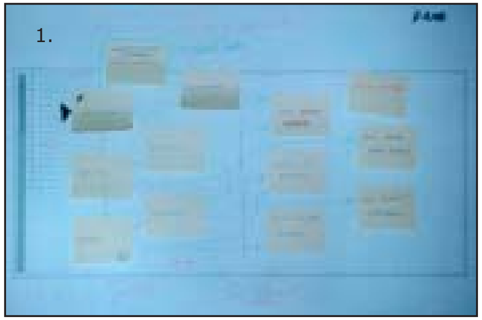
- Can you describe your feelings about making interactive content within Blender?
- What takes a lot time to do? On what task do you lose time?
- Do you use functions in away they are not meant for (figurative use)?

- Are you happy with the way Blender deals with interaction creation?
- What can (should) be changed or improved according to you?
- Do you have an idea how the interaction creation should look like?

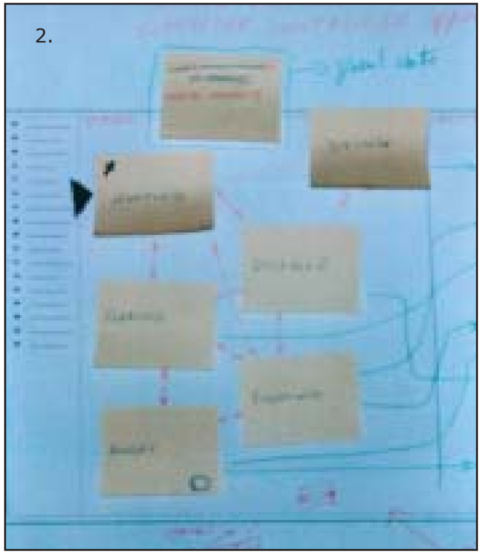
Appendix III: Evaluation concept A

Evaluation concept A

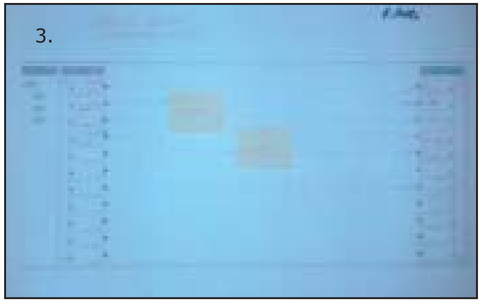
1.



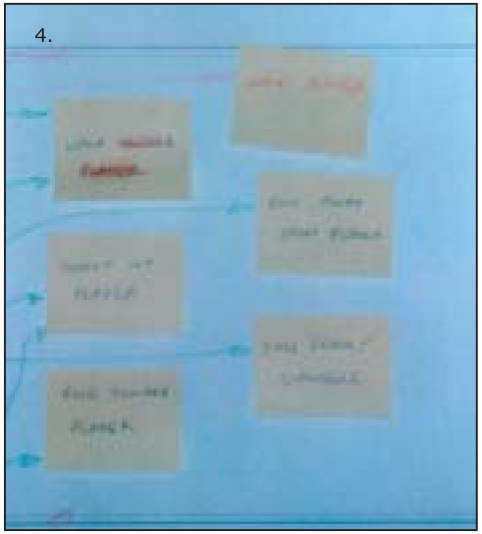
2.



3.

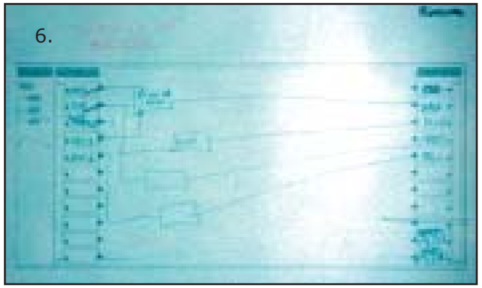


4.



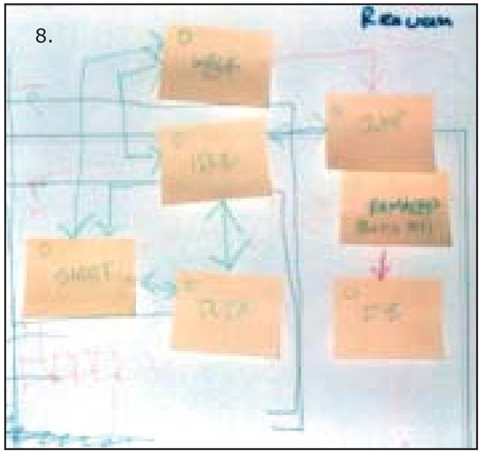
5.

6.



7.

8.



1. Filling in of the state-editor by content employee A.   2. Close-up of picture one: the *states*.   3. Filling in of the logic-editor for the hunting-state, by content employee A.   4. Close-up of picture one: the *actions*. 5. Filling in of the state-editor by content emplyee B.   6. Filling in of the logic-editor for the idle-state, by content employee B.   7. Close-up of picture five: the *states*.   8. Close-up of picture five: the *actions*.

Source listening

[1] Wartmann, Carsten; Game Blender Documentation (for Blender 2.21 Edition); September 2001

[2] http://www.pong-story.com/intro.htm

[3] Gamasutra
www.gamasutra.com

[4] Molyneux, Peter; Postmortem: Lionhead Studio's *Black & White*; June 2001 (http://www.gamasutra.com/features/20010613/molyneux_01.htm)

[5] Corry, Chris; Postmortem: Lucas Arts' *Star Wars Starfighter*; August 2001 (http://www.gamasutra.com/features/20010801/corry_01.htm)

[6] Smith, Brent; Postmortem: Poptop Software's *Tropico*; October 2001 (http://www.gamasutra.com/features/20011010/smith_01.htm)

[7] Imlach, Wayne; Postmortem: Muckyfoot's *Startopia*; October 2001 (http://www.gamasutra.com/features/20011027/imlach_01.htm)

[8] Biessman, Eric; Postmortem: Raven Software's *Soldier of Fortune*; September 2000 (http://www.gamasutra.com/features/20000927/biessman_01.htm)

[9] Dybsand, Eric; Game Developers Conference 2001: An AI Perspective; April 2001 (http://www.gamasutra.com/features/20010423/dybsand_01.htm)