Chapter 4

"The worlds into which one steps through virtual reality are limited only by the imagination of the programmer"

- D. Powers, M. Darrow [5]

Implementation

In this chapter the implementation of the Intelligent Tiles Virtual Laboratory framework is presented. Firstly an overview of the implementation is highlighted, covering subjects such as the programming languages used for implementation and user interface design. Next system implementation design concepts are presented. Lastly an overview of the implemented system is presented with screenshots.

4.1 Overview of implementation

The **iTiles Ecosystem Virtual Laboratory** is a virtual reality application implemented using the iTiles framework and concepts presented in Chapter 3. This iTiles system is composed of three connected and dependent tools: the **iTiles Workbench** tool which is used for the authoring of an iTiles world, the **iTiles World Flow** tool which is used for specifying the behaviour of an iTiles world, and the **iTiles Virtual World** tool that presents a simulation of the authored iTiles world. These tools share system specific iTiles concepts and are used as independent tools in the process of authoring and simulation. Each iTiles world developed using the iTiles Ecosystem Virtual Laboratory can be considered an independent virtual laboratory.

The implementation of the iTiles framework has been accomplished using open source libraries. Open source initiatives support portability and applications developed using open source libraries are most likely to be able to be ported to other operating system environments

(e.g. Macintosh, Linux). The iTiles Ecosystem Virtual Laboratory application has been implemented on a Microsoft Windows environment. The object oriented programming language used in implementation is C++. The standard C++ header libraries are used in conjunction with the Standard Template Library (STL) in implementation. Section 4.1.3 describes the programming approach used in more detail.

4.1.1 Application programming interfaces (APIs)

The Open Graphics Library (OpenGL) and Open Audio Library (OpenAL) APIs are the open source libraries used in implementation of the iTiles framework. The key concepts of these libraries are presented in this section.

OpenGL is a widely accepted API for interactive 3D graphics rendering and 2D imaging. It provides device-independent support for common low-level 3D graphics drawing operations such as polygon specification, basic lighting control, transformation specification, and framebuffer operations like blending and depth-buffering. It also provides mechanisms for sending and retrieving 2D images to and from the framebuffer, and integrates 3D graphics with 2D imaging through texture mapping. While other low-level graphics APIs have provided similar functionality to that of OpenGL, OpenGL takes a novel approach in the presentation of many of its features. In addition to providing a simple model for combining 3D graphics with 2D imaging, OpenGL makes a clear separation between high- and low-level functionality, stresses fine-grained control and feature orthogonality, and is designed for excellent performance from basic PCs to high-end graphics workstations. OpenGL also specifies a fixed rendering pipeline that both provides a model for implementations and a clear foundation for adding new functionality. [42]

The OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of different windowing systems [41]. The GLUT API simplifies the implementation of programs using OpenGL rendering and requires very few routines to display a graphics scene. The GLUT API, like the OpenGL API, is stateful, and the initial state is reasonable for simple programs. [43]

OpenAL is a joint effort to create an open, vendor-neutral, cross-platform API for interactive, primarily spatialized audio [45]. The Open Audio Library provides a cross-platform API that is as useful and powerful for 3D audio as OpenGL is for 3D graphics [44]. OpenAL supports spatial audio where sounds can be played in a way as to reflect the position of objects in the

virtual environment. OpenAL, similarly to OpenGL, has an abstracted interface through a utility toolkit called ALUT.

4.1.2 Hardware and software used in implementation

The iTiles Ecosystem Virtual Laboratory application is aimed for use on a desktop VR solution. The application has been implemented on a Microsoft Windows 2000 environment on a Pentium III 733Mhz processor with a 3D accelerator, the ASUS GeForce 256 GPU with an nVidia chipset.

Various software tools have been used in implementation:

- *Graphic application.* For texture creation of the tile set and the application's 2D interface (Corel Photopaint 9.0).
- *Integrated Development Environment (IDE).* A programming interface to program and compile C++ code (Microsoft Visual Studio 6.0).
- *3D modelling.* A shareware tool has been used for the modelling and converting of 3D models for use as world objects (Milkshape 3D, available from chUmbaLum sOft at http://www.swissquake.ch/chumbalum-soft/).
- *Sound editing.* For sound clips (Creative WaveStudio).

4.1.3 Programming approach

An object-oriented approach has been used in the implementation of the iTiles framework concepts discussed in Chapter 3. Objects are used to represent these concepts, by taking advantage of inheritance and polymorphism object-oriented concepts. The class diagram of the iTiles framework concepts is presented in Appendix A.

A singleton class is an object-oriented design pattern. An implemented class exists as one and only one instance in memory at run time. Using such a global object ensures that the instance is easily accessible and provides a global point of access [50]. In the implementation of the iTiles system the use of singleton objects has provided simplicity in the GUI application and system functions.

The vector and map standard template libraries (STLs) have been used in implementation, as these STL collections can be defined to store a certain type of object, and can grow and shrink in size. The collections STLs were used to handle some of the difficult memory management problems that arise when implementing in C++. The STL string class was very useful in storing collections of characters.

4.1.4 User Interface Design

Computing exists now as part of many peoples' cognitive strategies, helping create and develop new ideas. We test, we model, we try out, and we interactively simulate our conceptions through use of computers [10]. The art world has discovered the potential of virtual reality for visual innovation [5]. Virtual worlds have become more and move visually elaborate and emotive. Some even aim at an environment nearly indistinguishable from the real world. Transferring natural interaction and communication principles from the real world to cyberspace in a seamless fashion is a very challenging task. Here, an attempt has been made to implement a fun and friendly user interface by adhering to user-centred design principles. Research on human factors in human-computer interaction shows that users of a new interface demonstrate a consistent desire to "get started right away" [46].

Easy-to-use products do not just happen [47]. To create productive and enjoyable user experiences an approach such as user-centred design must be adopted. In [46], basic fundamentals of such user-centred design principles include:

- *Affinity*. Bringing objects to life through good visual design
- *Assistance.* Assist the user in performing a variety of tasks, through hints or system help
- *Availability*. Allow the user to use all objects within a view in any sequence at any time
- *Encouragement*. Make actions predictable by ensuring that an action produces the expected results
- *Familiarity*: Build on the user's prior knowledge of the system. A user-friendly system enables the user to learn new concepts and techniques from accomplishing one task and apply them to a broad spectrum of tasks
- *Obviousness*. Make objects and controls visible and intuitive. Use visual or textual cues to help users understand functions.
- *Safety.* Protecting the user from making errors, where the burden of keeping the user out of trouble rests upon the designer. Report results of actions immediately
- *Satisfaction.* Create a feeling of progress and achievement. Allow the user to make uninterrupted progress and enjoy a sense of accomplishment. Report the results of actions immediately. Communicate to the user in the event that a function cannot be performed
- *Simplicity.* Do not compromise usability for functionality. Keep the interface simple and straightforward. Minimise the number of objects and actions in an interface
- *Support.* Place the user in control.
- *Versatility.* Support alternative interaction techniques. Allow the user to switch between methods to accomplish a single interaction

The above fundamentals provide guidelines for implementing systems for different users with different abilities. They also play a vital role in user-centered design for young learners as users. However added simplicity and pictorial representations form a superior supporting user-interface.

4.2 Implementing the iTiles Ecosystem Virtual Laboratory

In this section major implementation specifics of the iTiles Ecosystem Virtual Laboratory are highlighted. Some of the implementation classes are mentioned in this section: see Appendix A for better understanding their relationships with other classes.

4.2.1 iTiles system management

The *iTilesSystemManager* singleton class is responsible for all system specific functions for specifying which iTiles system tool is currently active, whether a new or existing iTiles world is being authored, and for loading the iTiles system interfaces.

4.2.2 Abstracting GLUT function calls

The classes implementing the different tools of the iTiles system are:

- A *display class* extending the interface of the *iTilesDisplay* class representing the visual display of the tool (e.g. *iTilesWorkBenchDisplay*); and
- A *logic class* for the 'behind the scenes' logic algorithms and the functionality of the tool (e.g: *iTilesWorkBench*)

The interface of the *iTilesDisplay* class is:

- virtual void keyboard(unsigned char key, int x, int y);
- virtual void special(int key, int x, int y);
- virtual void display2D();
- virtual void display3D();
- virtual void reshape(int newWidth, int newHeight);
- virtual void motion(int x, int y);
- virtual void mouse(int button, int state, int x, int y);
- virtual void idleFunction();
- virtual void updateFluidMotionValues();
- virtual void updateCameraFrozenPosition();

To simplify the functionality of the display classes of the iTiles system tools, the singleton class *iTilesDisplayManager* is implemented. It abstracts GLUT function calls and GLUT and OpenGL initialisation procedures. The *iTilesDisplayManager* class stores information on which display class of which iTiles system tool is currently active and passes on all GLUT call back functions to this currently active display class through the *iTilesDisplay* interface (as presented above).

The GLUT 'display' call-back function that the *iTilesDisplayManager* class implements extends GLUT functionality by firstly calling the display3D() function followed by the display2D() function for overlaying 2D on screen. The *iTilesDisplayManager* class is also responsible for abstracting some OpenGL drawing functions such as displaying text on a specific part of the screen and drawing textured rectangles.

4.2.3 2D GUI

Controls are the software elements, usually shown on a display, that you use to set preferences and make choices. These elements are necessary for interaction between people and computers, and are much like such hardware controls as knobs and dials, as they can be used to control many different things [48]. The iTiles system has a unique look and feel it terms of the way it captures user input and control. A custom widgets library is written for this purpose. The library has primitive widget components such as buttons, radio buttons and list boxes. These widgets are integrated and used to make input screens.

The OpenGL co-ordinate system starts at the bottom right corner of a window, and the widget components are modified so that co-ordinates are given as if the co-ordinate system started at the top left. Appropriate reshape callbacks functions are implemented for correct resizing of screens, and information is passed on to the widget components affected by the resizing.

The widget components are divided into:

- *Standard components.* E.g. button, radio button, radio button group, list box, image button
- *Custom components.* The standard widget components are re-used and combined to create system specific components (e.g. plus minus control and world sound chooser)

The widget components are currently dependent on (closely linked to) the iTiles system. However some portions of code could be combined to form an independent library of widget components. The widgets are presented in their visual form in Figure 23.

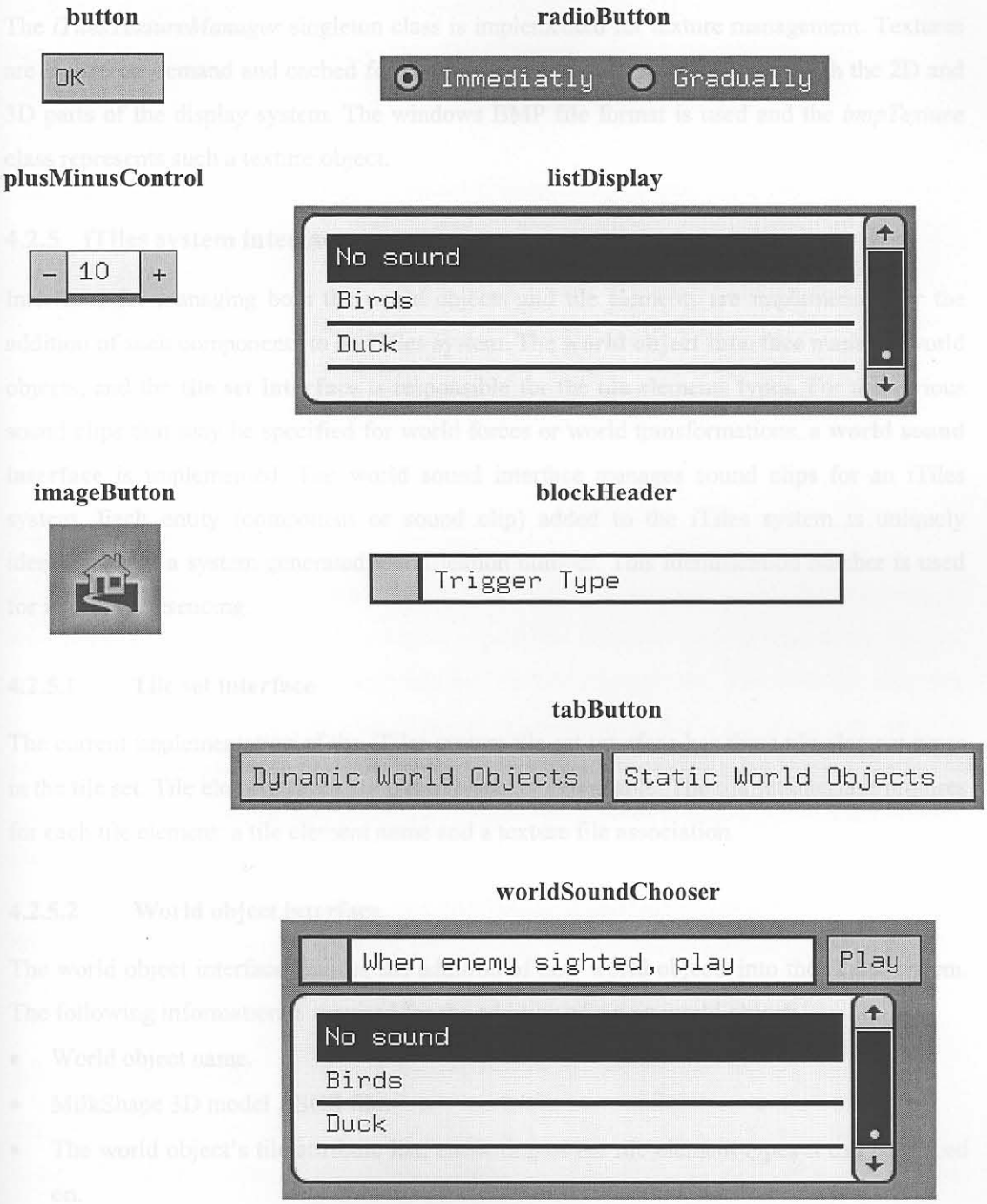

Figure 23. iTiles widgets classes visual representation

Overlaying a 2D user interface over parts of a 3D interface combines both interfaces to form a unique desktop VR experience. The 2D user interface consists of widget components that are used to interface with the virtual reality application. This combined interface is similar to an aeroplane cockpit interface, which is used in many flight simulators. This 2D overlay mechanism has been implemented in the iTiles Workbench and iTiles Virtual World tools presented in Section 4.3.

4.2.4 Textures

The *iTilesTextureManager* singleton class is implemented for texture management. Textures are loaded on demand and cached for future use. These textures are used in both the 2D and 3D parts of the display system. The windows BMP file format is used and the *bmpTexture* class represents such a texture object.

4.2.5 iTiles system interfaces

Interfaces for managing both the world objects and tile elements are implemented for the addition of such components to an iTiles system. The **world object interface** manages world objects, and the **tile set interface** is responsible for the tile elements types. For the various sound clips that may be specified for world forces or world transformations, a **world sound interface** is implemented. The world sound interface manages sound clips for an iTiles system. Each entity (component or sound clip) added to the iTiles system is uniquely identified with a system generated identification number. This identification number is used for internal referencing.

4.2.5.1 Tile set interface

The current implementation of the iTiles system tile set interface has three tile element types in the tile set. Tile element types are however easily extendable. The tile set interface requires for each tile element: a tile element name and a texture file association.

4.2.5.2 World object interface

The world object interface enables the addition of new world objects into the iTiles system. The following information is required for the addition of a new world object:

- World object name.
- MilkShape 3D model ASCII file.
- The world object's tile attribute list, consisting of the tile element types it can be placed on.
- Whether the world object is a static or a dynamic world object.
- Minimum and maximum scale of the object so that it falls in the visual boundary of a tile element.
- If the world object is a dynamic world object, the vision type (eg: herbivore, carnivore) and vision depth (number of tiles the dynamic world object can see).

Milkshape 3D is a shareware application, and has support for popular 3D file formats such as those models used in popular games such as Quake and Half-life. The MilkShape 3D

application is used for converting models in a 3D Studio format to the Milkshape 3D ASCII file format to be imported into the iTiles system. A MilkShape 3D model ASCII file loader is used for displaying a 3D model in an OpenGL 3D scene.

4.2.5.3 World sound interface

The world sound interface requires a sound clip name and sound file (WAV file) to add a world sound to the iTiles System.

The *worldSoundsManager* class, implemented using the OpenAL API and ALUT, has been tasked to load and cache sound clips. Furthermore it registers a sound source with a sound clip, and this sound source is programmed to play the sound clip spatially, at a certain point in 3D space.

4.2.6 Collision detection

A collision detection model is implemented in order for the dynamic world objects moving in an iTiles world simulation not to occupy or overlap the same 3D space as other world objects, or to occupy tiles with tile elements not in their tile attribute list. The collision detection model is simplistic in nature as a world object exists on top of a tile, and a tile may either be empty or a world object may be on top of it. If a dynamic world object wishes to move to another tile, it needs to enquire if the tile it wishes to move to is empty. The collision model also restricts dynamic world objects to move to tile elements in their tile attribute list. Another important objective of the collision detection model is to keep characters in the world, and restrict them to movement only within the world so that a character cannot move off the edge of an iTiles world.

4.2.7 Tile merging

The base terrain of an authored iTiles world is not visually appealing since the tile edges stand out and are rectilinear. This is overcome by adopting a tile merging technique from [40]. Using this technique the tiles that were specified in the base terrain of an authored iTiles world can be merged for the simulation of this iTiles world, to present a visually attractive environment. Figure 24 illustrates the tile merging process. Figure 24a represents an example of a base terrain of an authored iTiles world, consisting of four tiles of different tile element types. In the first step of the process, each tile in the authored iTiles world is divided into four quadrants, as illustrated in Figure 24b. In the following step, two extra rows and two extra columns are added to the iTiles world, as illustrated in Figure 24c. These newly created tile quadrants are then assigned the tile element types from their adjacent dominant elements, as

illustrated in Figure 24d. In the final step, nine new 'mega tiles' are assigned, each consisting of a group of four tiles, as illustrated in Figure 24e. The tile elements of these mega tiles are merged. The dotted lines in Figure 24e represent the merging of adjacent tiles in a mega tile. The resultant tiled world, as illustrated in Figure 24e, is somewhat larger than the original (containing 36 tiles overall), however it is more visually appealing since tiles are merged.

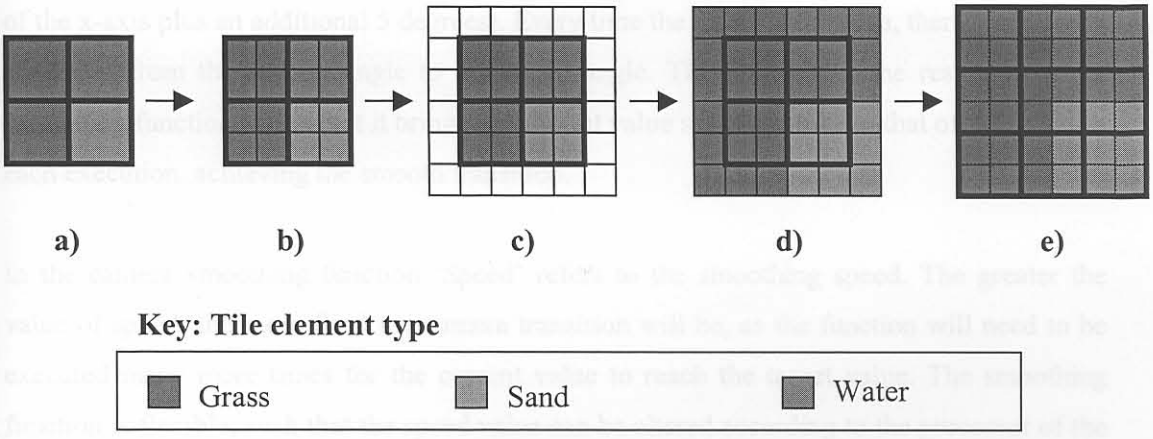

Figure 24. The tile merging process

4.2.8 Camera

The iTiles system provides multiple viewing points of a scene of an iTiles world. Functionality is also provided for zooming in and out of a scene, and rotating or shifting the scene on different axes. Additionally, the iTiles system provides a smooth transition between the different viewing points, including the zooming, rotation or shifting processes. This is achieved by implementing a camera smoothing function that makes use of OpenGL camera techniques. In general, the smoothing function is useful for any angle or position in 3D space, such as an x, y or z co-ordinate or angle.

For each angle or position in 3D space, two storage variables are needed for using the smoothing function:

- *Current*. The current value of the angle or position in space.
- *Target*. The target angle or position in space to which the current angle or position must adjust.

In each GLUT idle call-back function the following function is executed:

$$Current = Current + ((Target - Current)/Speed)$$

Consider for example, that every time a frame of an OpenGL scene is drawn, the scene must firstly rotate to a certain x-axis angle. Now consider a change of view in the scene, where the x-axis is rotated by positive 5 degrees. When using the smoothing function, when such a change of view occurs in a scene the camera does not move directly the new x-axis angle. Instead the camera rotation always references the current x-axis angle variable. When the change of view occurs, the target angle variable for the x-axis is calculated (the current angle of the x-axis plus an additional 5 degrees). Every time the frame is redrawn, there is a smooth transition from the current angle to the target angle. This is because the resultant of the smoothing function call is that it brings the current value slightly closer to that of the target in each execution, achieving the smooth transition.

In the camera smoothing function 'Speed' refers to the smoothing speed. The greater the value of speed, the more fluid the camera transition will be, as the function will need to be executed many more times for the current value to reach the target value. The smoothing function is flexible, such that the speed value can be altered according to the processor of the underlying PC hardware (i.e. a smaller value for a slower processor and larger value for a faster processor).

4.3 Overview of the iTiles Ecosystem Virtual Laboratory

The iTiles Ecosystem Virtual Laboratory is a virtual reality application that is used to author ecosystem type virtual laboratories using the iTiles framework. In this section an overview of the application is presented. The authoring process is firstly presented, starting from how the iTiles system interfaces are populated with tile elements and world objects, followed by how an iTiles world is authored using the iTiles Workbench. The iTiles Workbench tool, for specifying behavioural properties for an iTiles world is presented next. Lastly the iTiles Virtual World tool that presents a simulation of the authored iTiles world is presented. These iTiles system tools are discussed according to their functionality and implementation. Screenshots are also presented.

4.3.1 Populating iTiles system interfaces

How a user populates the iTiles system interfaces will greatly influence the use of the iTiles Ecosystem Virtual Laboratory. Since we are aiming to simulate an ecosystem, animals from the animal kingdom and plants from the plant kingdom are used. To create an ecosystem world, we should be able to represent grass, sand and water areas on the ground, and various trees and animals that would inhabit the world. Using the iTiles concept of world components, the grass, sand and water elements are represented as tile element types, and the

trees and animals are represented as world objects. These tile element types and world objects are added to the iTiles system using the tile set interface and world object interface respectively.

Figure 25 shows the textures used for the creation of the grass, sand and water tile element types. The tile set interface has been populated with these tile element types.

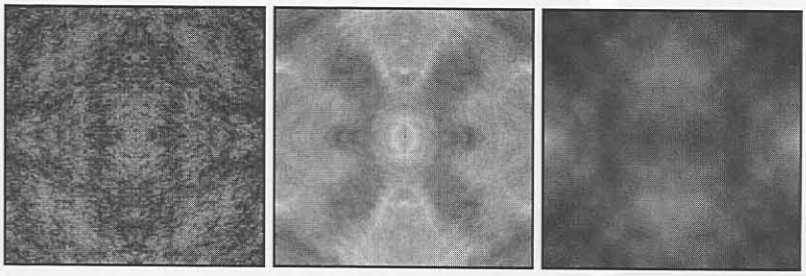

Figure 25. The grass, sand and water tile element textures

A rich variety of trees are modelled and imported as world objects, as shown in Figures 26, 27, 28 and 29. These trees are static world objects, but animals are dynamic world objects. Several animals are modelled and included in the ecosystem, such as a duck, elephant, pig, mouse, turtle and dog (see Figures 30-35). In Table 1 and Table 2 the attributes specified for these world objects (for the world object interface) are presented.

In Table 3, the sound clips that have been added to the iTiles system using the world sound interface are presented. These sounds clips are intended as sounds the animals of the ecosystem may make.

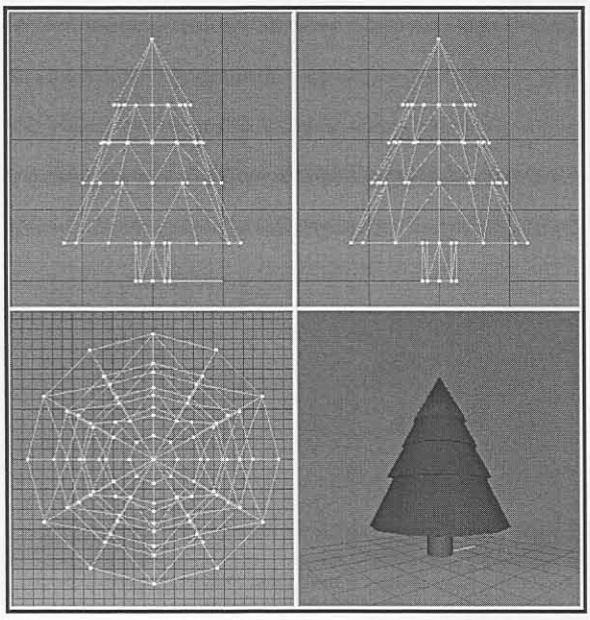

Figure 26. Fir tree

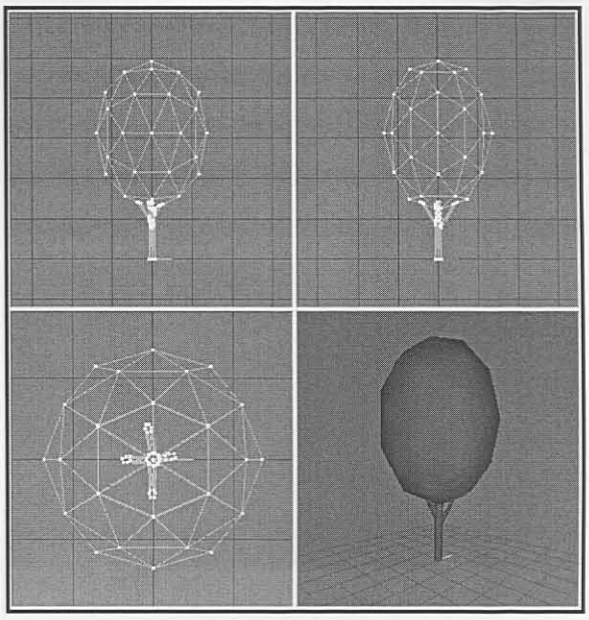

Figure 27. Broad tree

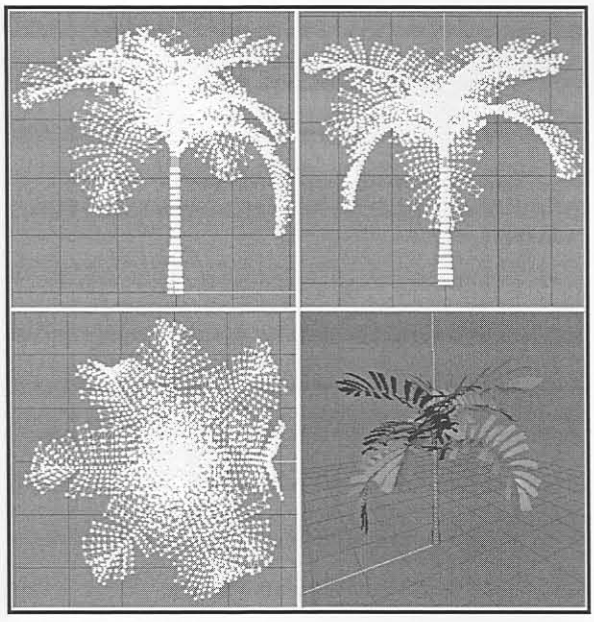

Figure 28. Palm tree

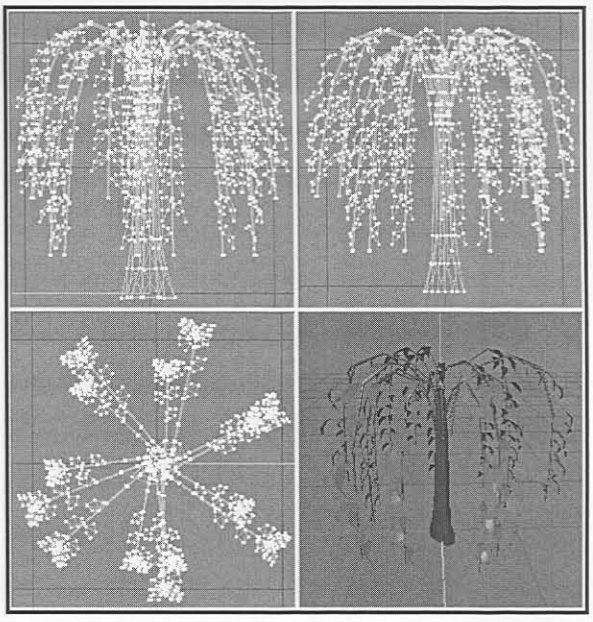

Figure 29. Willow tree

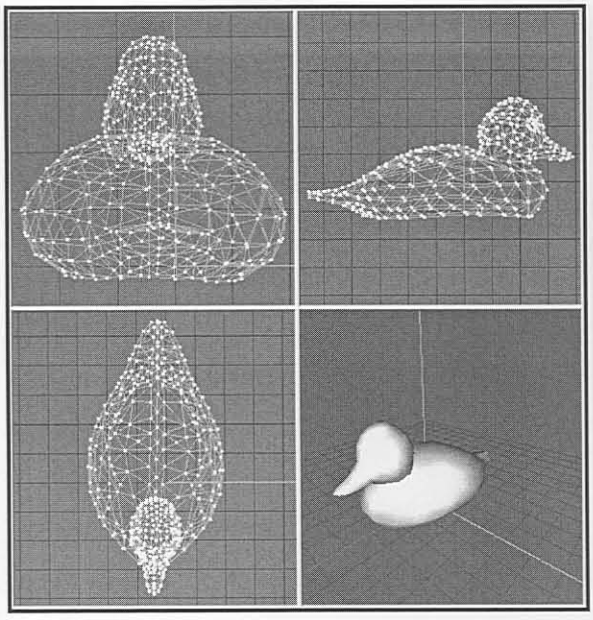

Figure 30. Duck

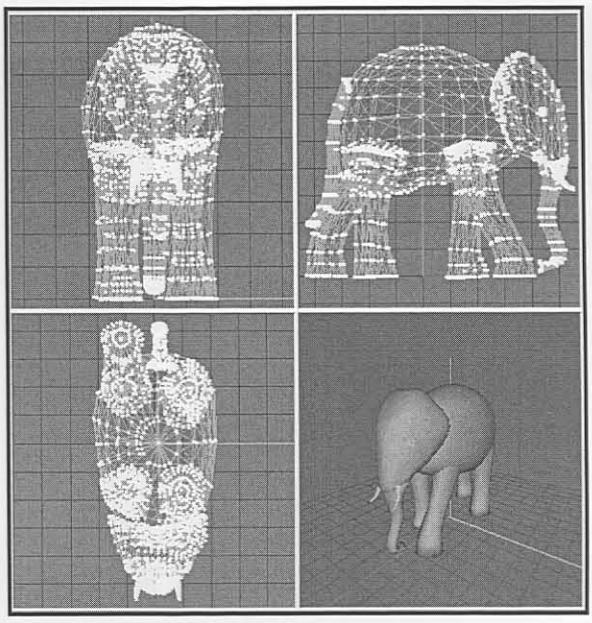

Figure 31. Elephant

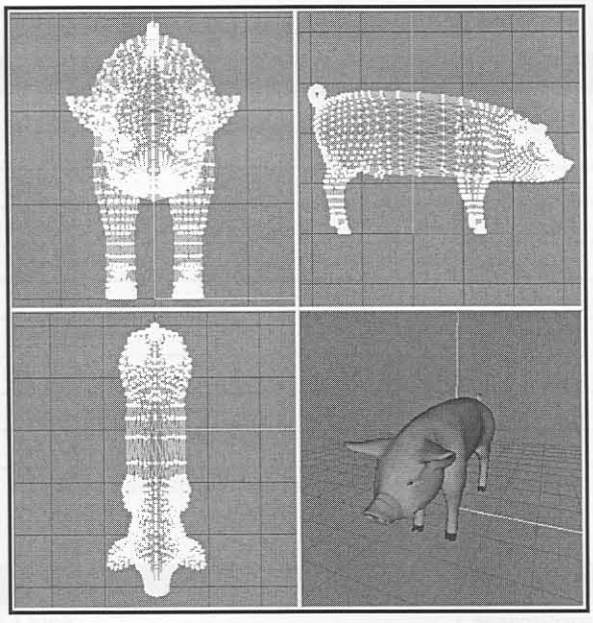

Figure 32. Pig

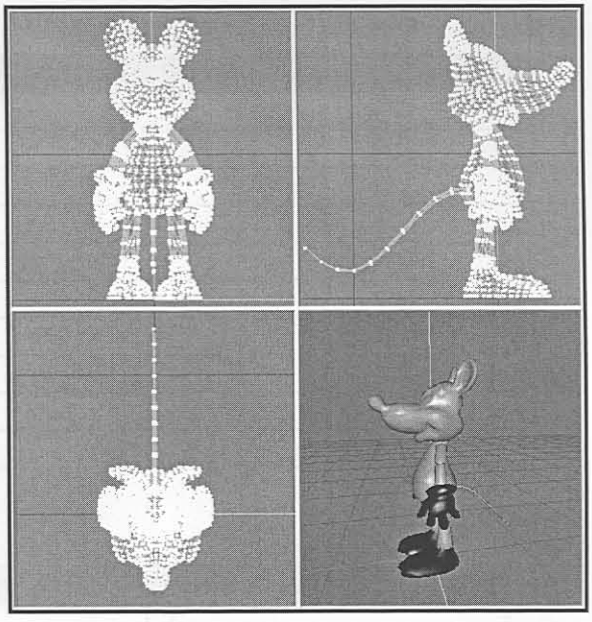

Figure 33. Mouse

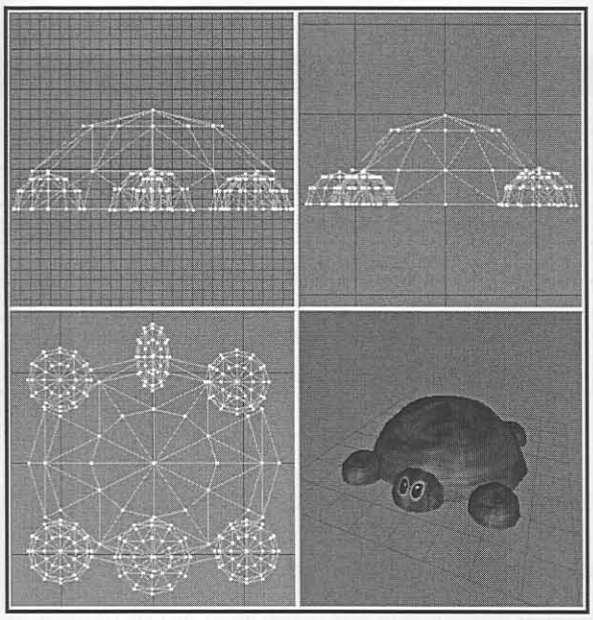

Figure 34. Turtle

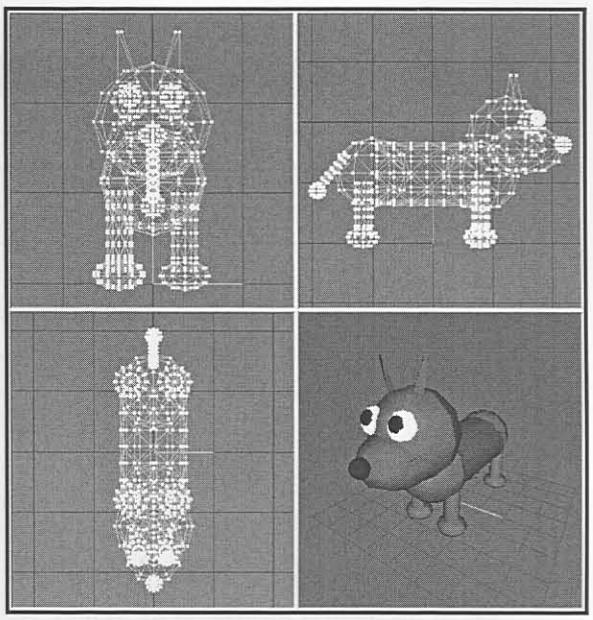

Figure 35. Dog

World Object	Grass	Sand	Water
Duck	×	×	✓
Elephant	✓	✓	×
Pig	✓	✓	×
Mouse	✓	✓	×
Turtle	×	✓	✓
Dog	✓	✓	×
Fir tree	✓	✓	×
Broad tree	✓	✓	×
Palm tree	✓	✓	×
Willow tree	✓	✓	×

Table 1: Tile attribute list of world objects

Dynamic World Object	Vision type	Vision depth
Duck	Herbivore	2
Elephant	Herbivore	4
Pig	Herbivore	3
Mouse	Carnivore	3
Turtle	Herbivore	2
Dog	Carnivore	4

Table 2: Dynamic world object vision properties

Filename	Description of sound
birds.wav	Birds chirping
dogbark.wav	A dog barking
duck.wav	A duck quacking
elephant.wav	An elephant trumpeting
pig.wav	A pig snorting
water.wav	Splashing of water
magic.wav	A magic sound effect
music.wav	A magical melody

Table 3: World sounds

4.3.2 The introductory screen and main menu

The introductory screen of the iTiles Ecosystem Virtual Laboratory is the screen that is presented to the user when the application is launched. Figure 36 presents a screenshot of the introductory screen. The application provides space for five iTiles worlds to be created. A sphere of the earth represents an already existing iTiles world, and an empty sphere indicates a space to include a new world. Existing iTiles worlds are easily accessed by a mouse click on a sphere. When a sphere is selected, it is highlighted with a yellow circle, and the iTiles world is selected. A user may remove an existing iTiles world making space to include a new world, by selecting an existing world and clicking on the 'delete world' image button.

The application provides a mechanism for controlling the order in which the iTiles system tools may be launched for a specific iTiles world. This mechanism translates to the three steps of the iTiles framework, as described in Section 3.1.1. An iTiles world firstly needs to be authored using the iTiles Workbench tool before specifying the behaviour of that iTiles world using the iTiles Work Flow tool. Only once an iTiles world has been authored and the behaviour specified, may the iTiles Virtual World simulation tool be launched. The icons underneath the spheres are image buttons used to launch the iTiles system tools. These image buttons may either be enabled or disabled according to the state of the selected iTiles world (i.e. the iTiles world may not have an iTiles World Flow defined and therefore the iTiles Virtual World tool will not be able to be launched). To launch a tool that is enabled, a user can simply click the image button for the corresponding tool.

Figure 37 presents a screenshot of the iTiles main menu. The main menu is accessible at any time during program execution by pressing the ESC key on the keyboard. With this menu a user can launch an iTiles system tool for the selected iTiles world. The menu will also not display any iTiles system tool that cannot be activated for the selected iTiles world. If one of the iTiles system tools is currently active (not the introduction screen), the menu allows a user

to go back to the introductory screen, by selecting the menu option 'choose another world'. Functionality is also provided to activate a full screen mode if the application is running in a window, activate a window mode if the application is running full screen, and to exit the iTiles system. The menu also allows a user to alter the camera smoothing speed (as discussed in Section 4.2.8).

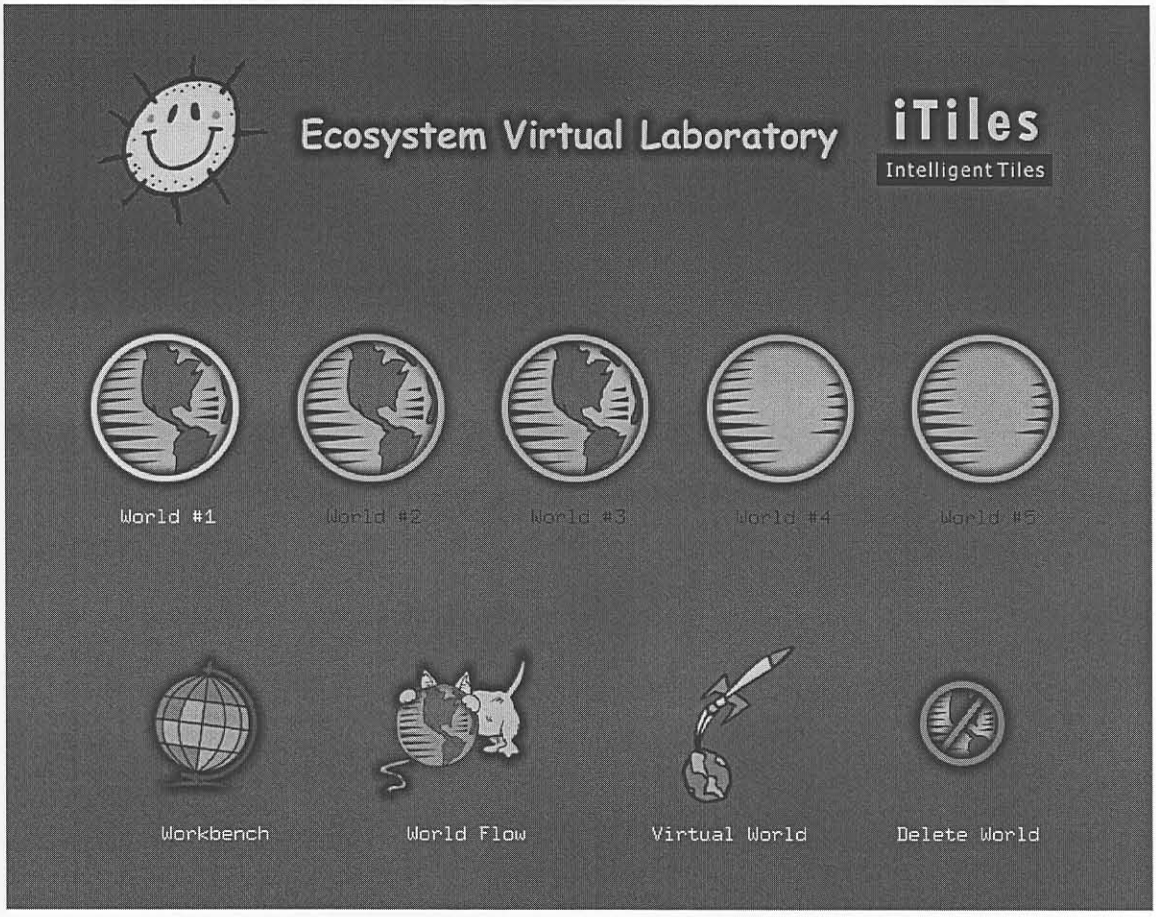

Figure 36. The iTiles Ecosystem Virtual Laboratory introductory screen

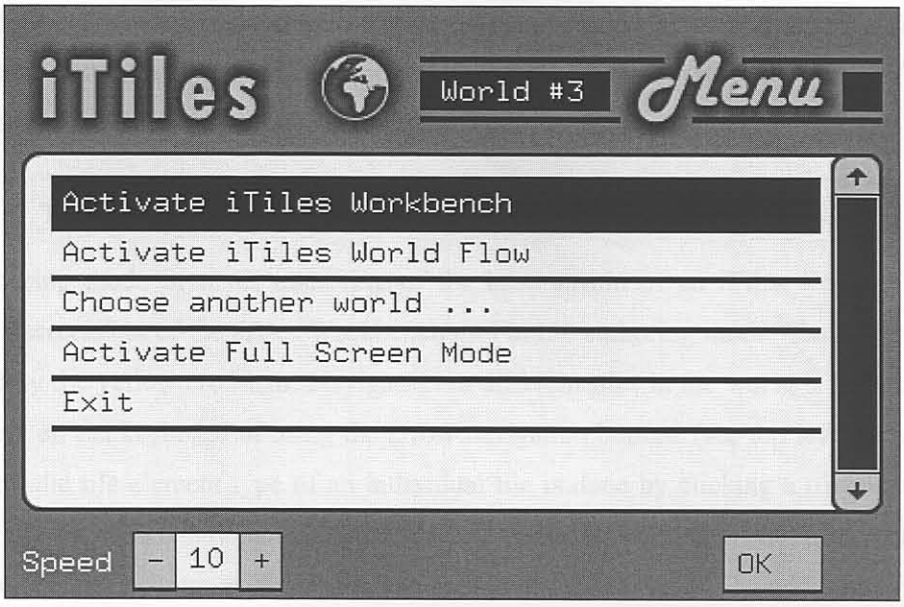

Figure 37. The iTiles main menu

4.3.3 The iTiles Workbench tool

The iTiles Workbench is the authoring tool for an iTiles world and allows teachers to author an iTiles world for use in the presentation of a lesson. With the iTiles Workbench tool a new or an existing iTiles world can be authored. Authoring of an iTiles world is a visual process that involves the procedures of authoring the base terrain of tile elements, and specifying which world objects will inhabit the world. These authoring procedures have been divided in the iTiles Workbench tool as two different modes: *tile authoring mode* and *world object authoring mode*.

In both modes of the iTiles Workbench, functionality is provided for navigating the individual tiles of an iTiles world and for camera rotation, shifting and zooming functions as described in Section 4.2.8. The image buttons appearing in the top right of the screen are used to for these camera modes (see Figure 38 or 39). Consider these image buttons be called $button_1$, $button_2$ and $button_3$, as appearing from left to right, top to bottom. $Button_1$ is used for activating a camera rotation mode, where the iTiles world can be rotated on different axes. $Button_2$ is used for activating a camera mode where a user can zoom in and out of an iTiles world, and shift the world on each of the axes. $Button_3$ is used for locking and unlocking the camera. When the camera is locked the current view of the iTiles world will remain unchanged. However, when the camera is unlocked, the selected tile will always appear in the middle of the screen, and the view of the iTiles world will continually shift in relation to this selected tile.

The iTiles Workbench tool is generic authoring process since it allows a user to choose tile element types of the base terrain and world objects when authoring an iTiles world. Therefore an infinite number of possible worlds can be created.

4.3.3.1 Tile authoring mode

Tile authoring mode involves authoring of the base terrain of an iTiles world. Figure 38 presents a screenshot of the iTiles Workbench tool in tile authoring mode. The selected tile is indicated by the yellow crosshair. Navigation to different tiles in the world is done using the arrow keys on the keyboard or using the arrow navigation buttons (see top left in Figure 38). Specifying the tile element type of an individual tile is done by clicking a tile element type from the tile set, which is depicted by the horizontal 'traffic light' (see bottom right in Figure 38). Functionality to grow or shrink the base terrain is also provided. To add more tiles in all directions to the base terrain (to the perimeter), a user can click the plus button (see left, middle in Figure 38). To add tiles in a specific direction a user can click an arrow button surrounding the plus button, in the corresponding direction. Removing tiles from the base terrain is similarly done by clicking the minus button, and arrow buttons surrounding the minus button.

The larger the base terrain of an iTiles world, the more computational dependencies arise. In this implementation of the iTiles system, an iTiles world has been restricted to a base terrain of the maximum dimensions of 10x10. This also restricts the number of world objects that can be placed in the world. The more dynamic world objects present, the more the simulation engine will need to accommodate in terms of processing time and storage for the movement and behaviour of characters.

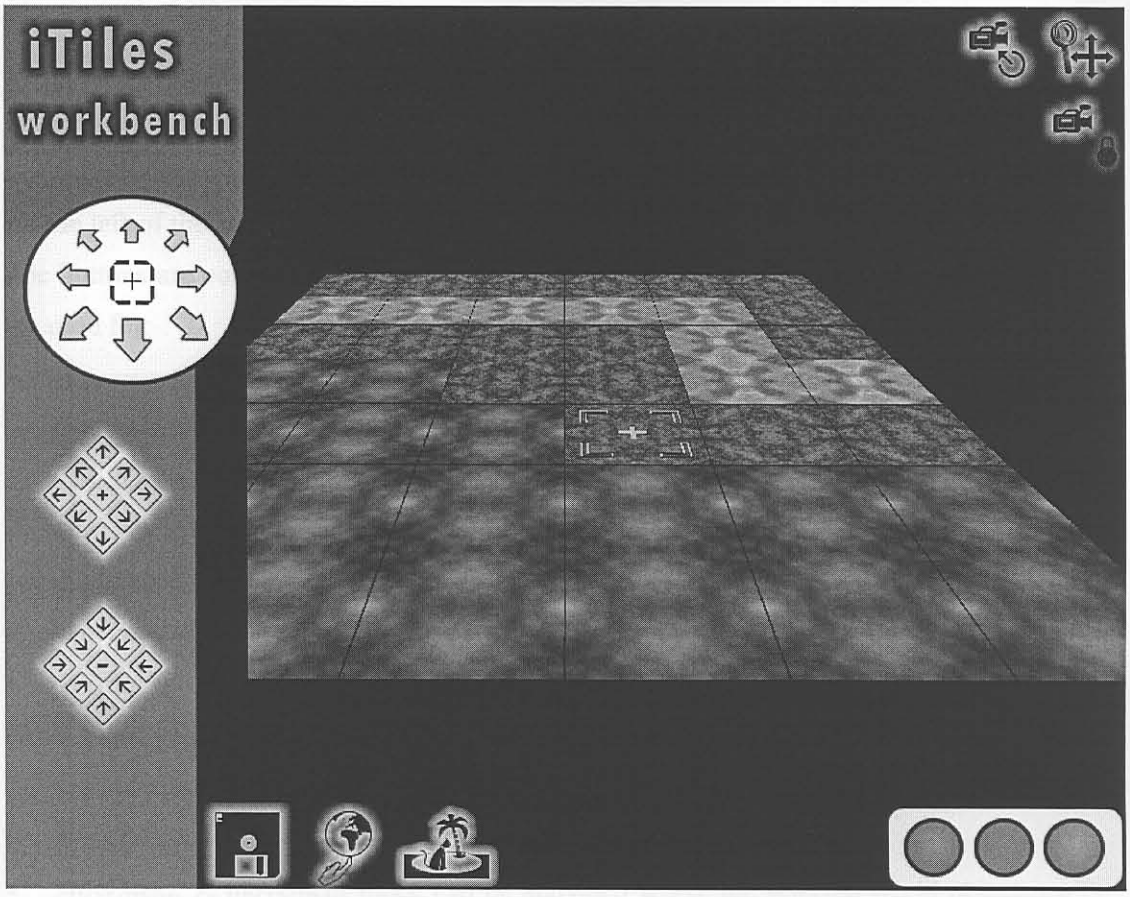

Figure 38. A screenshot of the iTiles Workbench tool in tile authoring mode

Clicking the image button with a 'globe spinning on a finger' is used for activating the tile authoring mode. Clicking the 'cat on an island' image button is used for activating the world object authoring mode, which is described next.

4.3.3.2 World object authoring mode

Figure 39 presents a screenshot of the iTiles Workbench tool in world object authoring mode. In the Pokémon™ craze in South Africa (January 2001 - June 2002), the South African potato chip manufacturer Simba (Pty) Ltd, included circular plastic discs of the pokémon characters called tazos in packets of potato chips. These tazos[1] indicated the skills and abilities of a pokémon character, and were intended for children to collect, and used in a game children could play. The circular shape at the bottom right of the screen uses this tazo metaphor for iTiles world objects. When a world object is selected from the list of available world objects (see left part of Figure 39), information of the world object is displayed in the tazo. The tazo indicates whether the world object is a dynamic or static world object, and displays the world object's tile attribute list, for indicating what tile element types the world object may be

[1] Tazo is a trademark and copyright of Nintendo.

placed on. A wire frame cube is also used to indicate which world object is selected. In the screenshot in Figure 39, the dog is selected. It is a dynamic world object, and may only be placed on grass or sand tile element types.

To the left of the tazo, are buttons providing functionality for the authoring of world objects. The buttons are coloured green when enabled, and red when disabled. These image buttons are used for:

- *Placement.* Placement of world objects involves the addition (clicking the plus image button) and removal (clicking the minus image button) of a world object on a tile in the base terrain. World objects can only be placed on tiles in the world according to their tile attribute list. The plus image button will be coloured red if a user cannot add a world object to the selected tile, or if the tile is currently occupied by another world object. This is because only one world object may be placed on an individual tile.
- *Orientation.* A world objects can be rotated so that it faces a certain direction.
- *Scaling.* A world object can be scaled (up or down) to a certain percentage between 0% and 100%. These percentages are linked to the world object's minimum scale and maximum scale, so that the world object retains visual conformation.
- *Movement.* A movement mode can be activated where the selected world object can be moved to another tile in the world that it may be placed on.
- *Search.* When a certain world object is selected, a user can search if such a world object exists in the world by clicking on the eye image button. If found, the world object will be selected. If more that one object of that type exists, clicking the button again, will navigate to the next found world object.

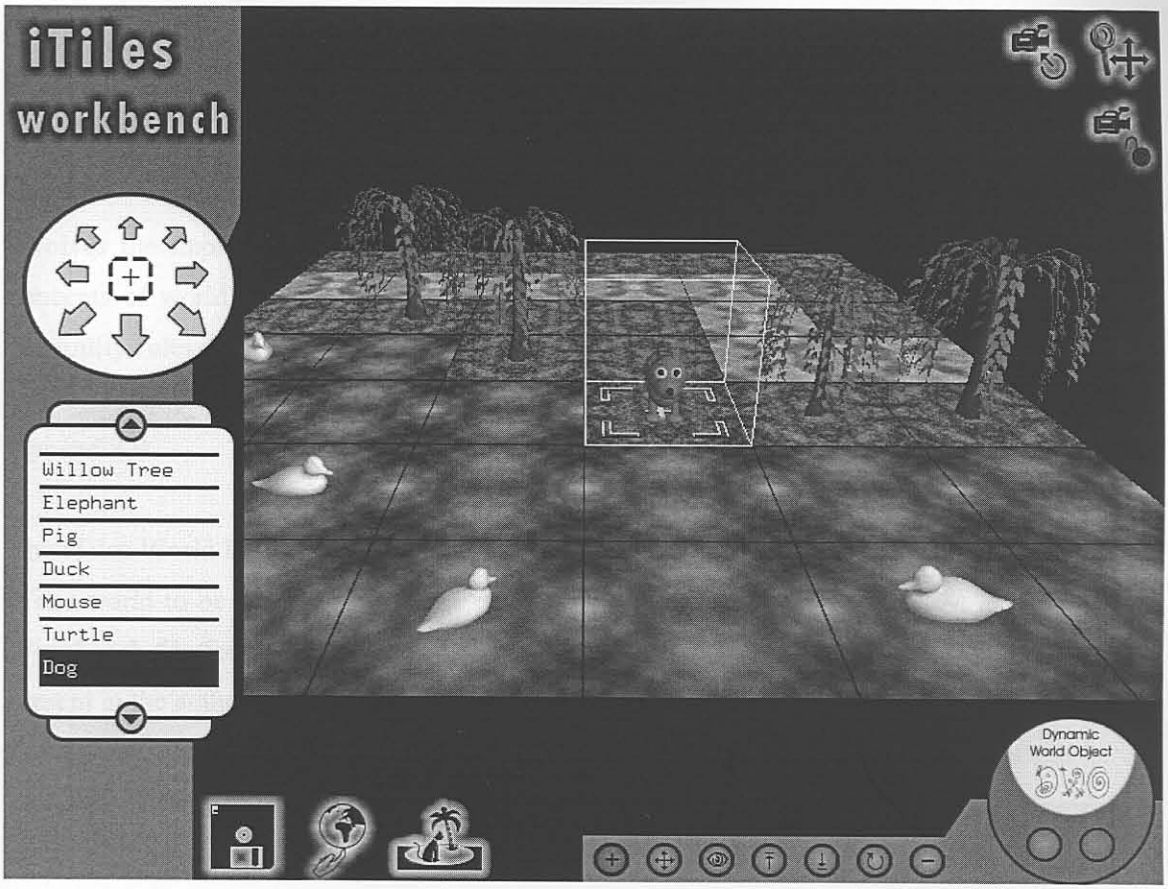

Figure 39. A screenshot of the iTiles Workbench tool in world object authoring mode

A user can save the authored iTiles world at any time by clicking the diskette image button (see Figure 38 or 39). Figure 40 summarises the information stored in a file for an iTiles world authored using the iTiles Workbench tool. This file contains information on the width and height of the base terrain, the tile element type of each tile in the base terrain, and the world objects that were placed in the world. The position, direction and scale properties specified for these world objects are their initial properties in the simulation of the iTiles world. Once an iTiles world is saved the behaviour of the iTiles world can be specified using the iTiles World Flow tool, which is described next.

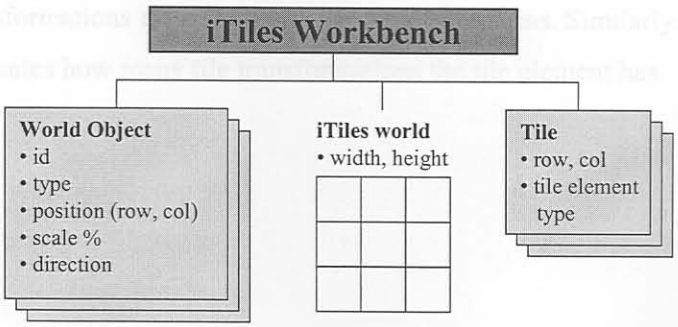

Figure 40. Information stored for an iTiles Workbench file

4.3.4 The iTiles World Flow tool

The iTiles World Flow tool is used for specifying rules that govern the simulation of an authored iTiles world. It gives a teacher the freedom to control the behaviour of the simulation of an authored iTiles world. The implementation of the iTiles World Flow tool involves the implementation of algorithms and screens to capture the iTiles World Flow concepts of world transformations and world forces, and adhere to the iTiles World Flow feasibility rules as described in Section 3.6.3. Many screens have been implemented for the iTiles World Flow concepts. A collection of screenshots of these screens is presented in Appendix B.

The iTiles World Flow tool provides for either a new or an existing iTiles World Flow of an iTiles world to be specified. When starting up, the tool uses information stored in an iTiles Workbench file for initialisation. From this file the tool determines what world objects were present in the authored world and constructs a list containing one kind (a specimen) of each of those world objects. This is because all world components (world object or tile) of a similar kind will share the iTiles World Flow properties as specified for that specific kind of world component in the iTiles World Flow tool. Therefore all world components (tile or world object) of a similar kind will exhibit the same behaviour in the simulation.

Figure 41 presents a screenshot of the main screen of the iTiles World Flow tool. The tab buttons at the top of the screen enable a user to navigate to a list of dynamic world objects, static world objects and tile elements. The lists of static and dynamic world objects contain those world objects from the constructed list. The list of tile elements contains a tile of each tile element type from tile set. In Figure 41, the dynamic world object's tab button is selected, and the duck dynamic world object is selected from the list. To the right of the list is the 'World Flow summary'. For a dynamic world object, this summary indicates how many positive forces, negative forces, world transformations, or movement forces are specified the selected dynamic world object. For a static world object the summary indicates how many world object transformations the selected static world object has. Similarly, for a tile element, the summary indicates how many tile transformations the tile element has.

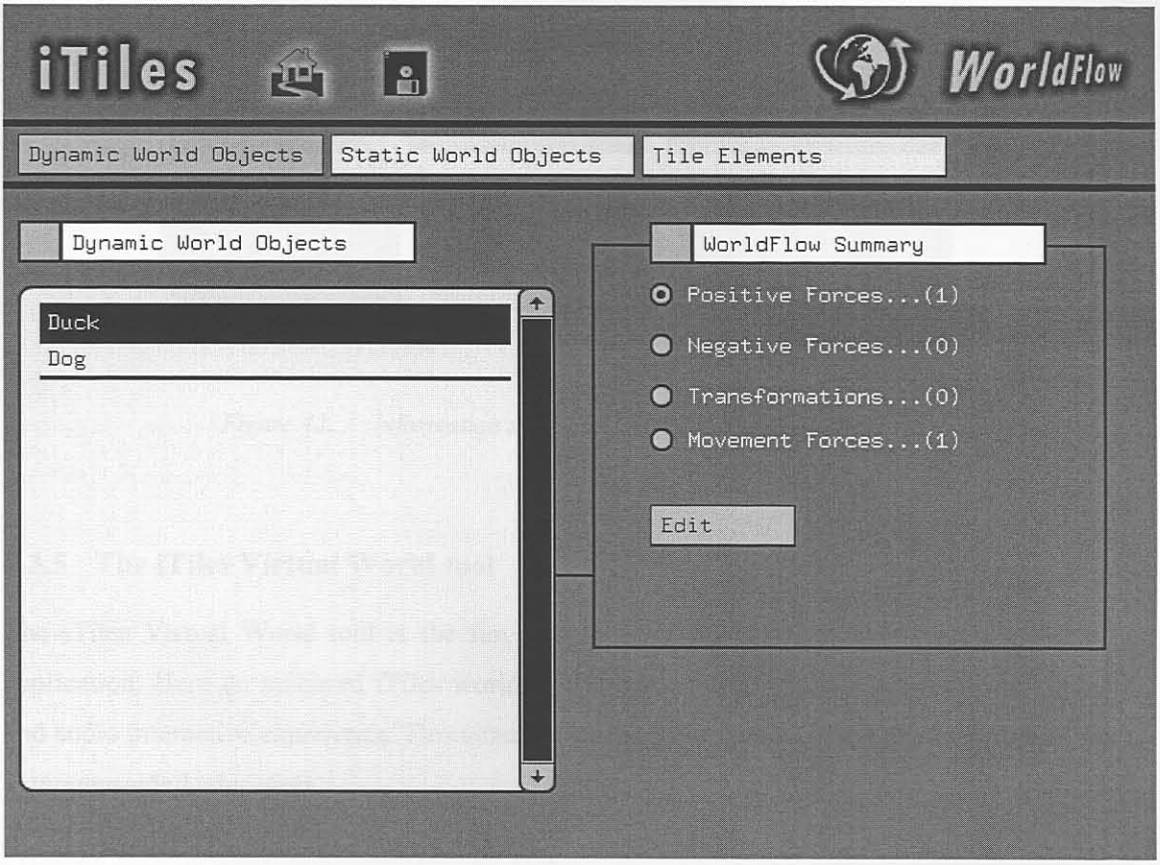

Figure 41. The main screen of the iTiles World Flow tool

To navigate to the positive forces of a selected dynamic world object, a user should select the 'positive forces' radio button in the World Flow summary, and click the edit button. A screen will be presented with a list of that dynamic world object's positive forces. A user can then add new positive force, edit an existing specified positive force or remove a positive force for the dynamic world object. Similarly, a user can add, edit or remove:

- Tile transformations of a specific tile element type.
- World object transformations of a static world object.
- Negative forces, movement forces or world object transformations of a dynamic world object.

The iTiles World Flow can be saved at any time clicking the diskette image button. Figure 42 summarises the information stored in a file for the iTiles World Flow tool. Once saved, the iTiles world is ready for simulation, and iTiles Virtual World simulation tool can be launched. This tool is discussed next.

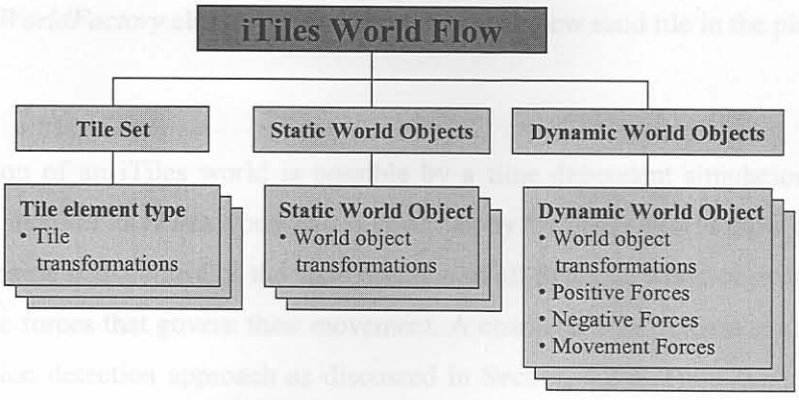

Figure 42. Information stored for an iTiles World Flow file

4.3.5 The iTiles Virtual World tool

The iTiles Virtual World tool is the fun part of the iTiles Ecosystem Virtual Laboratory application. Here an authored iTiles world is "brought to life" through a multimedia visual and audio interactive experience. This simulation is experienced by young learners to enjoy as computer-aided education.

The iTiles Virtual World tool combines information saved from the iTiles Workbench tool and the iTiles World Flow tool to present a simulation of an authored iTiles world. The *iTilesVirtualWorldFactory* class is responsible this initialisation procedure. This class uses the factory method, an object oriented programming design pattern, which is used when a specific class wants to delegate responsibility to one of several helper subclasses, which can instantiate objects with certain characteristics [50]. This concept is analogous to an industrial factory. The *iTilesVirtualWorldFactory* class is used to 'manufacture' world objects and tiles during the initialisation procedure of the iTiles Virtual World simulation tool. It integrates the information of world objects and tiles captured in both the Workbench tool and the World Flow tools, and combines that information as input into the Virtual World simulation tool. In this 'manufacturing process' each world object that has been specified in the iTiles Workbench tool is given the behavioural properties (world transformations and world forces), as defined for it's kind in the iTiles World Flow tool. Similarly, tile transformations specified for the different tile element types are also attached individually to each tile of the base terrain of the iTiles Virtual World. This 'manufacturing process' is needed because world transformation and world forces have numeric attributes that will change during the course of simulation. The *iTilesVirtualWorldFactory* is also used to 'manufacture' new tile objects during the course simulation. This will occur when a certain tile transforms to a different tile element type. For example, if a grass tile transforms to a sand tile, the

iTilesVirtualWorldFactory class will be able to produce a new sand tile in the place of the old grass tile.

The simulation of an iTiles world is possible by a time dependent simulation engine that monitors the status of an iTiles world and is governed by the iTiles World Flow concepts. The simulation engine is in control of the state machine of all dynamic world objects (see Section 3.6.2) and the forces that govern their movement. A character's movement is also monitored by the collision detection approach as discussed in Section 4.2.6. Time dependent updates such as the number of world beats that a dynamic world objects should remain frozen for, and increasing the force strength of incremental seeking positive forces of characters are implemented. The visual and auditory changes in the simulation in terms of iTiles World Flow concept of world transformations are implemented. The simulation engine is also responsible for the tile merging process as described in Section 4.2.7. Since the merged world is larger, the world objects positions are shifted accordingly when placed on tiles in the iTiles world during the initialisation procedure.

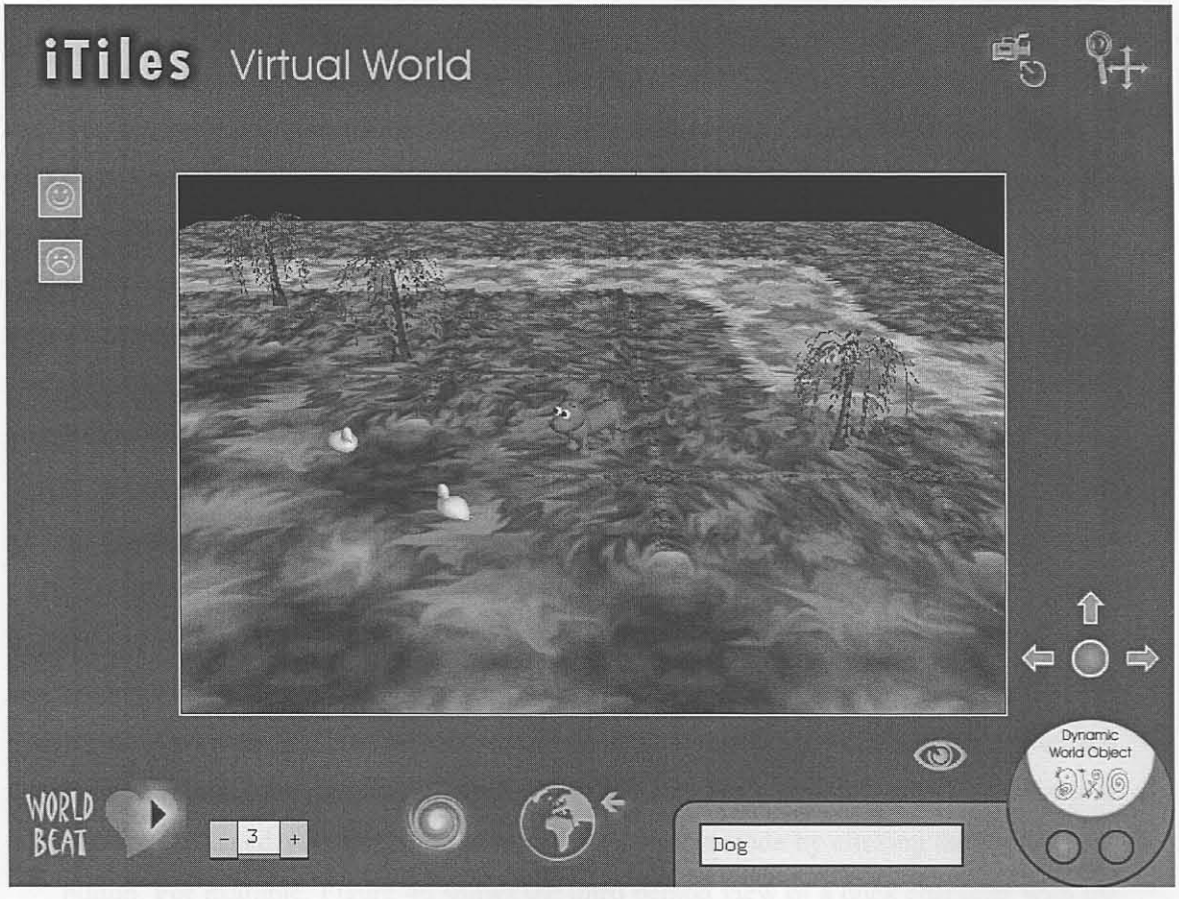

Figure 43. A screenshot of the iTiles Virtual World tool

Figure 43 presents a screenshot of the iTiles Virtual World tool. The iTiles world shown here is the simulation of the authored iTiles world as shown in Figure 39. The following functionality is provided by the iTiles Virtual World simulation tool:

- *Managing simulation time.* The simulation of an iTiles world can be started or stopped by clicking on the 'heart' image button in the bottom left of the screen. The simulation time can also be sped up or slowed down by specifying how many seconds each iTiles world beat consists, using the plus minus widget control provided.
- *World object selection and taking control of characters.* Navigation and interaction is implemented as discussed in Section 3.7.1. A user can navigate through the world objects of an iTiles world by clicking on 'swirl/portal' image button (see bottom part of Figure 43). This portal metaphor is used to symbolise an idea that a user is magically transported to the world object. The selected world object will always appear in the middle of the screen. Taking control of a character (i.e. taking on a virtual identity) and being able to move it through an iTiles world is also available. The 'circle' image button above the tazo (see bottom right part of Figure 43) is used to take control of a selected character, and the 'arrow' image buttons are used to control the character's movement in the world.
- *Identification.* Information of the selected world object is presented similarly to the iTiles Workbench tool. The world object's name is displayed in addition to the tazo, indicating the tile attribute list of the world object. Additional information of what world components (world objects or tiles) the character likes and does not like in terms of it's positive and negative forces can also be displayed as a list. These lists are displayed when a user clicks on the 'smiley' and 'sad smiley' image buttons (see top left part of Figure 43). If the selected world object is a static world objects, these 'like' and 'don't like' lists are respectively populated with characters that like and do not like the static world object.
- *View of an iTiles world.* Seeing an iTiles world through different perspectives and points of view is available. The camera rotation, shifting and zooming modes as discussed for the iTiles Workbench tool are also available (see top right part of Figure 43). However the camera mode that enables a user to lock or unlock the camera is not available. The camera remains unlocked in the iTiles Virtual World tool. The view types discussed in Section 3.7.1 are implemented. Figure 44 shows a third person view point of a dog, while Figure 45 shows the first person view point from the dog's eyes. A user can alternate between these view types by clicking the 'globe' image button. The inner vision mode is also implemented. A user can activate the inner vision mode by clicking the 'eye' image button. For example, Figure 46 shows the third person view of a duck character with the inner vision mode activated. This mode is also available in first person view.

- *Spatial sound.* The sounds that occur in the simulation are played spatially in relation to position of the selected world object. The user is considered at the position of the world object. For example in Figure 44, a user would hear a duck quacking on his left speaker or earphone.

Figure 44. Third person view

Figure 45. Virtual identity view

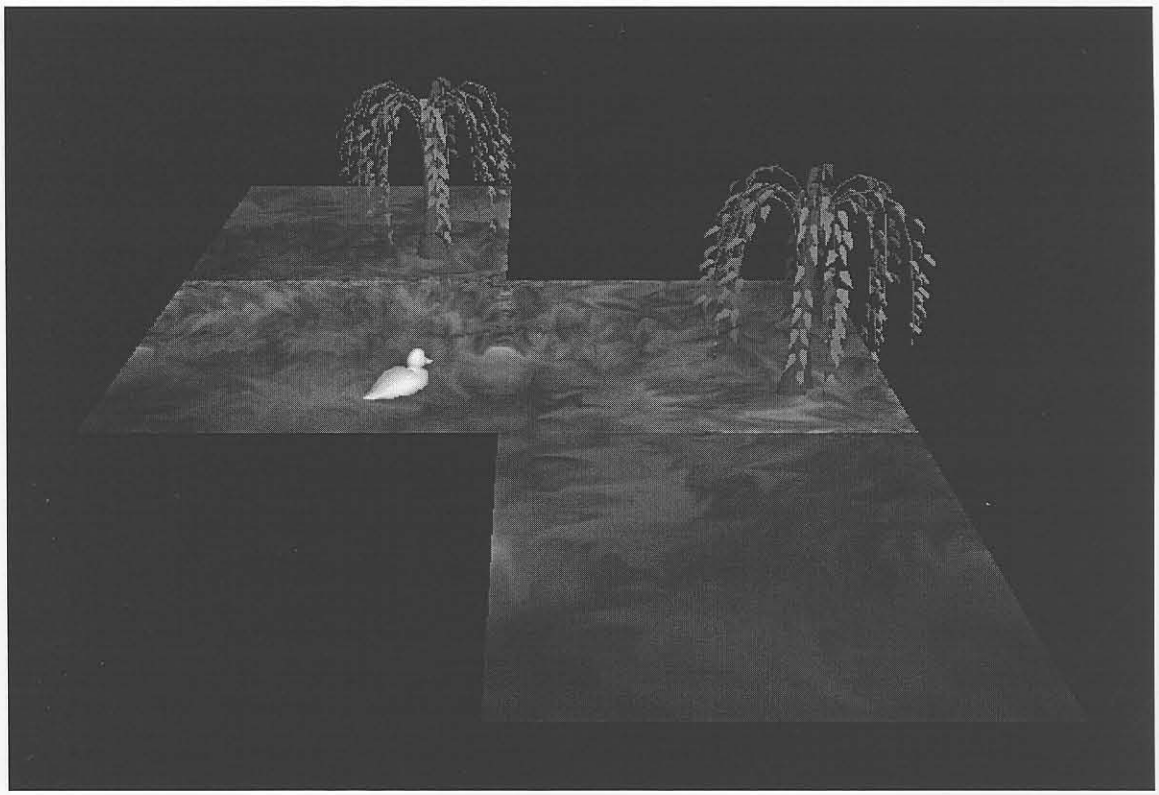

Figure 46. Inner vision view

4.4 Summary

This chapter presented the implementation of the Intelligent Tiles Virtual Laboratory framework. Firstly an overview of the implementation was given, covering subjects such as the programming languages used for implementation and user interface design. Next implementation design concepts were presented. Lastly an overview of the implemented system was presented with screenshots. In the next chapter an iTiles world produced with the iTiles Virtual Laboratory application is presented.