Chapter 5 – Scaffolding different types of learning

5.0 Overview of the Chapter

The focus in this chapter is on the merits of EM for the development of educational software. In previous chapters, we have argued that the EM approach to model construction supports a wide range of learning activities based on a broad constructionist view. We now consider the potential of EM for the development and use of learning environments. We shall argue that the use of EM in developing learning environments is advantageous because the highly flexible and adaptable nature of EM allows for relatively easy customisation of learning resources through its support for a very broad definition of scaffolding. We discuss scaffolding in relation to three different types of learning: of fixed referents; of exploration of possibilities; and of learning languages. We illustrate these ideas with reference to EM case studies of learning environments.

5.1 Model use vs Model building

5.1.1 Constructionist learning environments

Up to this point in the thesis we have been concerned with the support for learning that is afforded by EM model-building activity. We have concluded that EM offers better support for learning than conventional programming due to its ability to integrate pre-articulate and formal learning activities. However, it is not always possible for users to create their own models, and therefore in order to provide a rounded picture of learning and EM we also need to consider the benefits of EM models in use.

There are many reasons why learners may not be able to, be allowed to, or wish to, create their own models. This is especially true in the educational context, where, for example:

i) school children generally do not have enough computing expertise to be able to construct models to meet all their educational needs.

ii) model construction following personal interests cannot guarantee that learning relevant to the curriculum occurs.

iii) teachers may lack the necessary skills or the available time to be able to construct models for pedagogical use.

We elaborate each of these points in turn.

In respect of the first point, Nardi [Nar93] has observed that the construction of programs by end-users may not be a realistic aim. This is apparent in relation to EM model construction with our current tools. To date, all the authors of models built using EM tools have had prior knowledge of the fundamentals of computers and programming. For instance, understanding functions, variables and parameter passing are at present an essential prerequisite to EM model creation. Our own experiments with 17 – 18 year old college students have exposed this problem. We found that students without any previous programming experience could not use the TkEden tool to create models because they lacked essential computing knowledge. However, students with programming experience succeeded in extending previously created models. When students do not have a good understanding of basic programming concepts they cannot develop their own models and are reliant on others to produce learning environments for them.

In respect of the second point, even if students can create their own models, there needs to be a degree of accountability where learning through model creation is concerned. Students are usually following a prescribed curriculum and if they follow their own interests when creating models they may be learning subjects outside their curriculum. Further evidence of the difficulties of accountability in constructionist learning is evident in Noss and Hoyles's idea of the *play paradox* where time spent at

the computer may not be being used for meaningful learning [NH92]. Even in the established practice of computer-based model building for learning, such as was introduced by Papert through Logo [Pap83], it could be argued that the need to learn computer programming skills detracts from domain learning. The disappearance of Logo from the United Kingdom National Curriculum in the 1990s has been cited as evidence of uncertainty about its educational merits [NH96].

In respect of the third point, although teachers have the educational knowledge required to develop useful learning environments they cannot put that into practise without the necessary programming skills. Ideally, teachers want to be able to customise educational resources to suit individual learning needs. This requires that small, but often unpredictable, changes to programs can be made with limited knowledge of their construction. Traditional approaches to programming favour development in which very high cognitive demands are placed upon the developer prior to programming, and do not lend themselves to unpredictable end-user customisation. As Nardi observes [Nar93]:

> 'While programmers can be called in to provide applications for minority areas, once the software is written, users are stuck with the applications given them by programmers, and the applications cannot easily be changed, extended, or tailored to meet the demands for local conditions.'

This is diSessa's motivation for proposing that teachers and software designers should work closely together with children to produce useful learning environments [diS97b].

Broadly speaking, educational software can be classified on a spectrum between instructionist and constructionist-based approaches (see Figure 5.1). This spectrum has historical significance in that Computer Assisted Instruction (CAI) preceded Intelligent Tutoring Systems (ITS), which in turn preceded Interactive Learning Environments (ILE). CAI uses computers to replicate the traditional school learning model that has been criticised by many [Fri70, Ill71, Pap93, Opp97]. CAI has often

been called the 'drill-and-kill' approach, whereby students are presented with a set of textbook style questions to answer. ITS, introduced by Hartley and Sleeman in 1973 [HS73], are an extension of CAI. In addition to providing exercises for students, an ITS system assesses what a student knows and what they should know, and generates new exercises based on this assessment. However there is no scope for a learner to take control of their own learning experience because the system designer has preconceived the material for delivery and the mode of interaction. CAI and ITS are instructionist approaches aimed at imparting and testing objective knowledge.

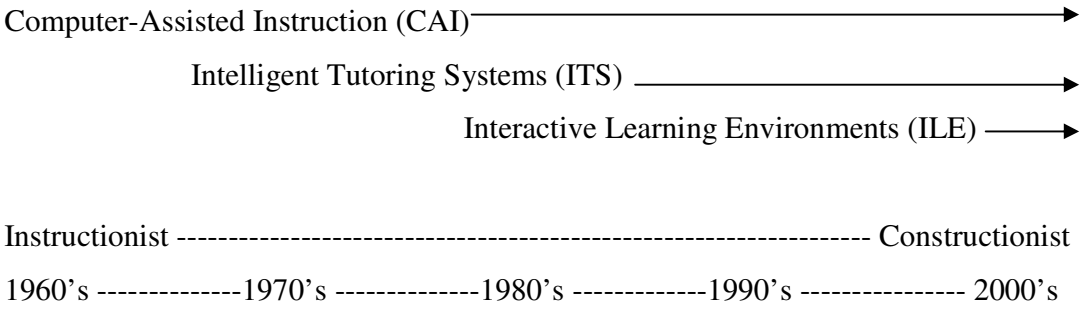

Figure 5.1 – A spectrum of learning perspectives

Constructionist software takes advantage of the medium of the computer to provide a qualitatively different learning experience. The essential difference between constructionist computer environments and CAI/ITS systems is that the learner has more control over their learning. As Soloway et al note, this necessitates a switch from user-centred design to learner-centred design [SGH94]. ILEs are constructionist because they emphasise the active role of the learner and are often called *microworlds*. A microworld is a small world within which students can understand concepts through active learning [Pap93]. For example, Cockburn's microworld to support the learning of Newtonian physics allows students to manipulate the parameters in physical laws and observe the resulting behaviour of objects [CG95].

The important requirements for constructionist software are that it should provide a learning environment in which:

1) a learner can explore the consequences of hypotheses whether or not they are correct.
2) learning objectives are situated in realistic domain contexts.
3) the designer or the learner can adapt or extend the environment to shape the learning process.

In the remainder of this chapter, we shall consider EM techniques for creating constructionist models that can support many types of learning activities.

5.1.2 Supporting different types of learning

In the 1980s, teachers had a high level of software ownership (as witnessed by the proliferation of small educational software companies often set up by teachers such as 4mation [4mat03] and Sherston [She03]) through being able to create software for their own particular teaching requirements. In recent years, software has become more powerful and more complex, but also less easily understood and adapted by its users [Joh03]. This motivates educational software that: can embrace a wide range of competencies; that is easily adaptable by both developers and users; and can provide teachers with resources that can be tailored to their particular pedagogical needs and context.

A standard approach to developing educational software that can be targeted at different learning scenarios is to expose simple concepts before more complex ones. This is similar in spirit to the HCI principle of *progressive disclosure,* which states that software should initially provide only the most commonly used features to a user, keeping more complex choices hidden in order to not overwhelm new users [PRS+94]. As a user becomes more competent, exploration leads them to find and explore the more complex features. In this way, the program is easy to learn for novices but still contains the powerful features that advanced users require. A simple

example can be found in expanding menu systems. For instance, in the Microsoft Office range of products, only the most commonly selected options from the menu are visible – to use other choices the mouse can be positioned over a double arrow to reveal all the possibilities.

The principle of progressive disclosure is not in general suitable for educational software. The purpose of progressive disclosure in application software is to hide the complex *features* of the software from novice users. In educational software, the aim is not to learn how to use a particular set of features, but to learn the *concepts* embedded within a learning environment. The exploration of progressively more complex concepts is associated with the idea of *scaffolding*. Scaffolding is defined as a technique for providing support to learners whilst they are learning a new task [Rog90]. EM gives support for scaffolding many different types of learning. For example:

i) **Learning as comprehension of a fixed referent** – a simple model of a specific referent is initially presented to the learner. This model is then gradually refined and extended by introducing more advanced concepts associated with the referent. The focus is on providing a computer-based model that accurately reflects its referent and level-by-level is guided by more precise observation of the referent.

ii) **Learning as in exploring possibilities and invention** – a simple model is initially presented to the learner. Although specific learning paths can be mapped out, the learner has discretion over how the model is extended at each layer, and different paths are associated with different referents. At each layer a learner is encouraged to interact as if in an exploratory laboratory.

iii) **Learning languages** – as a learner becomes more competent in a domain, their knowledge of domain specific language is progressively enhanced (cf. the music and rowing examples discussed in chapter 2). This can be reflected in the language used for interaction with the model of the domain.

Scaffolding of learning is analogous to the scaffolding that is used in constructing a building, which is removed when the building can stand by itself. As Soloway et al note [SGH94], scaffolding is provided to help a learner with a task they do not know how to do, and it gradually becomes less important as the learner becomes more competent. In educational terms, scaffolding is operating in Vygotsky's ZPD. The ZPD is defined as an area of domain knowledge, beyond a students' current comprehension, but which they can successfully navigate their way through with the assistance of their peers or an expert (such as a teacher) [Vyg62]. Soloway's Tools / Interfaces / Learner's needs / Tools (TILT) model [SGH94] is a classification of different types of scaffolding and their roles in the learning environment (cf. Figure 5.2). To scaffold learning tasks, software can coach a learner by providing helpful advice at appropriate points in the learning process. To support a learner's growing competence the tools in a model must be adaptable to the task in question (cf. bricolage). To support the learner at a communication level, the interface must provide different means of expression appropriate to the learners' competency.

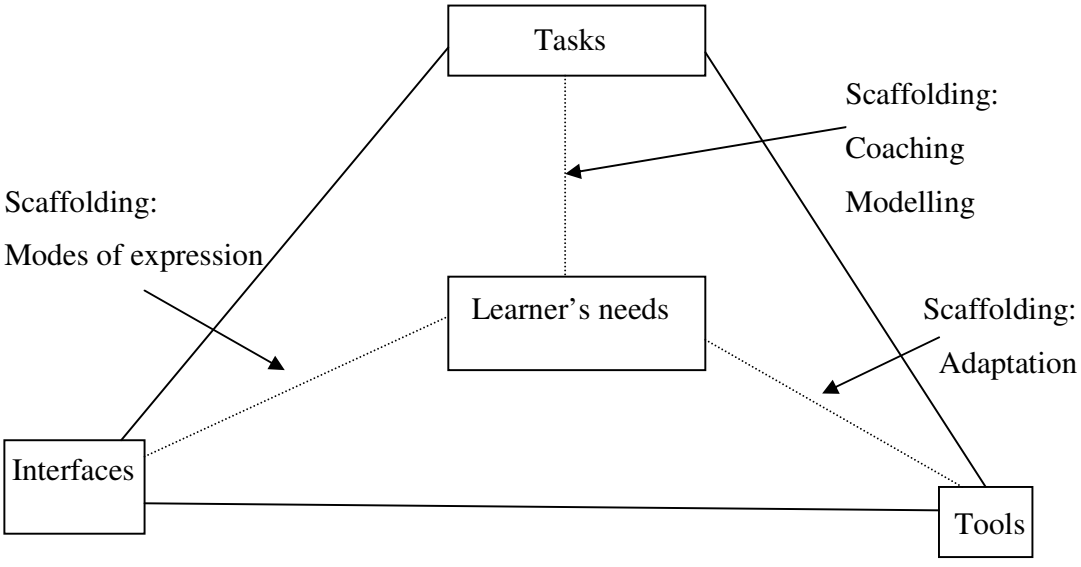

Figure 5.2 – Soloway's TILT model [SGH94]

The remaining sections of this chapter describe – and illustrate with reference to case studies – how EM learning environments can scaffold each type of learning identified in i), ii), iii) above. Our case studies also illustrate how EM could be exploited in different types of scaffolding similar to those identified in Soloway's TILT model, namely through scaffolding – for Tasks (section 5.2), cognitive layering – for Tools (section 5.3), and domain specific notations – for Interfaces (section 5.4).

5.2 Learning as comprehension of a fixed referent

In some learning environments, the major aim is to allow students to gain an exact understanding of a specific referent. The typical application is in modelling 'real-world' situations. In such a context, the domain being modelled is presumed to behave reliably according to some well-defined rules. For instance, balls on a snooker table will, after being struck, behave in a definite manner when bouncing off the cushions, colliding with other balls and slowing down through friction. In order to give the leaner an appropriate construal of the real-world situation the balls, as modelled, should ideally behave according to similar physical rules. Pratt associates this similarity between model and domain with the ideas of *surface* and *cultural familiarity* [Pra98]. Surface familiarity is concerned with whether objects in the computer environment look like their real-world counterparts. Cultural familiarity is concerned with whether objects in the computer environment behave like their real-world counterparts. Where models have both surface and cultural familiarity, learners can leverage prior experience of the real-world situation and can successfully transfer knowledge gained from the computer-based environment back into the world. The ideas of surface and cultural familiarity are similar to Green's notion of 'closeness of mapping' in the Cognitive Dimensions framework [GP96].

The scaffolding principle suggests that the concepts in a learning environment should be layered and introduced only when the learner has a solid understanding of simpler concepts. For example, the benefit that can be gained from a fully functioning snooker model in learning mechanics could be limited because the complexity of the

complete model obscures the learning of simple concepts. Understanding the complete model requires comprehension of a number of inter-related concepts. In a snooker model for learning about mechanics, a first layer could consist of a single ball that bounces around a 2D table without ever slowing down. This could be used to explore how a ball bounces off a cushion. A second layer could introduce a concept of friction, so that the ball slows down over time. This could be used to explore forces acting on a ball. A third layer could introduce more balls and illustrate what happens when balls collide with each other. This could be used to investigate principles such as the conservation of momentum. A model constructed with these simple layers learners can serve as an exploratory environment for learning about mechanics (cf. [EMRep, billiardsMoissenkov1999] for a prototype implementation).

The notion of developing increasingly complex microworlds, where each microworld adds more complex ideas or tasks to perform, is a well-established educational strategy (cf. Burton et al [BBF84]). The various layers of the snooker model can be viewed as a series of graded microworld instances in the sense of Graci et al [GON92]. This means that each microworld builds on the concepts in the previous level to provide a more complete picture of the real-world situation portrayed in the model. As Graci et al note [GON92], these graded microworld instances are essentially increasing subsets of the functionality of the complete learning environment. These microworld instances provide the means for scaffolding the domain.

In the next section, we consider a case study that illustrates the idea of scaffolding in a practical EM learning environment, where the emphasis is on construing how the real-world domain of car racing works.

5.2.1 The racing cars case study

In this section, we describe an EM learning environment targeted at exploring factors that are important in car racing. Applying Pratt's principles to this real-world

situation [Pra98], the computer-based learning environment must be recognisable as a car racing environment ('surface familiarity'), and the cars should be set up to behave like their real-world counterparts so that learners are able to draw on their prior experience of the domain ('cultural familiarity'). The racing cars model was constructed by Simon Gardner in 1999 [EMRep, racingGardner1999] and takes the form of a series of increasingly complex microworlds. The final microworld contains two customisable cars racing each other around a partially customisable track. The full functionality of the model is implicit in each microworld, but only a subset of that functionality is exposed to the learner. There are seven microworlds in Gardner's model, and we discuss four that give a flavour of the increasing complexity of the concepts being introduced. Figure 5.3 shows the main concepts in Gardner's seven microworlds and highlights the four discussed in this section.

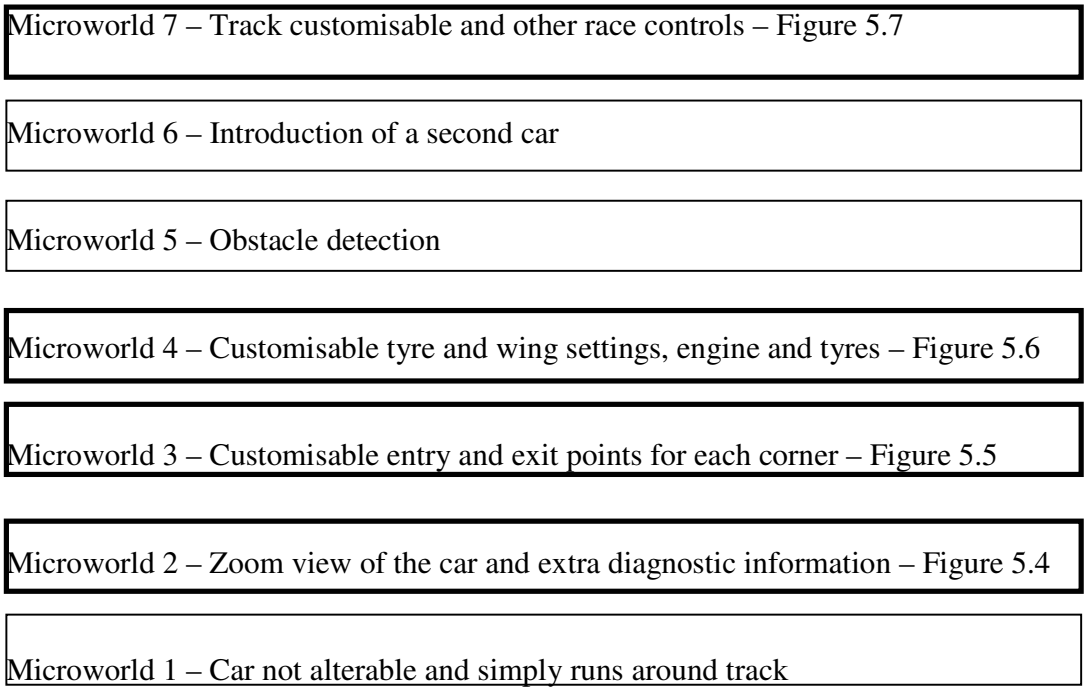

Figure 5.3 – The microworlds in the racing car model

Microworld 2 (see Figure 5.4) shows a car that is moving around a track. The learner is able to observe many significant attributes of the car, but there are no controls for

the learner to experiment with the car and its environment. The 'Car 1 status' table contains information about the car such as its acceleration, braking, friction and wind resistance. The acceleration and braking values refer to the change in speed that will occur in the next clock cycle if the car is accelerating or braking respectively. On the plan view of the track, the symbol '1' will move around the track. The zoom view on the left depicts the car and its neighbourhood in more detail.

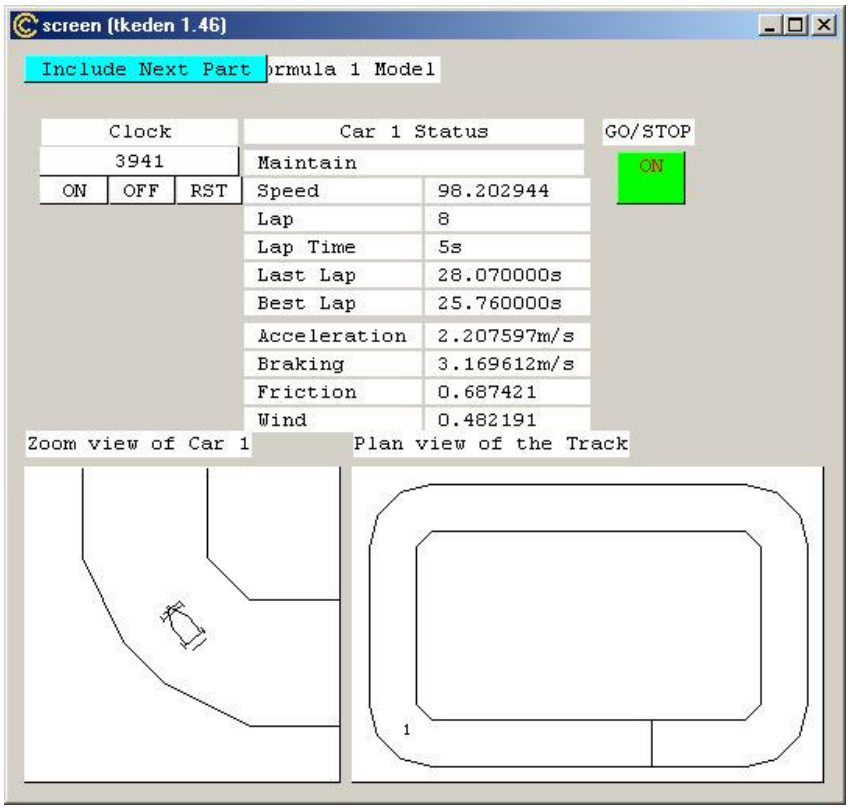

Figure 5.4 – Microworld 2 of the racing cars model

In this elementary microworld, learners can observe how patterns of acceleration and braking are correlated with the motion of the car and its position on the track. This can be used to gain a basic understanding of how the car accelerates and brakes in cornering, and how concepts such as wind resistance and friction are related to car speed. These concepts form the necessary background for exploring how to move the car around the track faster in the more advanced microworlds.

Microworld 3 (see Figure 5.5) builds on the previous microworld and introduces the concept of braking, turn-in and accelerate-out points for corners. These respectively refer to the key control points on the track where the car will begin to brake for a corner, where it will start turning inwards to take the corner, and where it will start to accelerate away from the corner. The set of crosshairs in the top right of the interface can be used to alter the significant points for each corner. Clicking on a different position in each rectangle will move the corresponding point on the track. This selection method restricts the key control points to a sensible region of the track. Changes to the points can be made at any time, even whilst the car is approaching the point being moved.

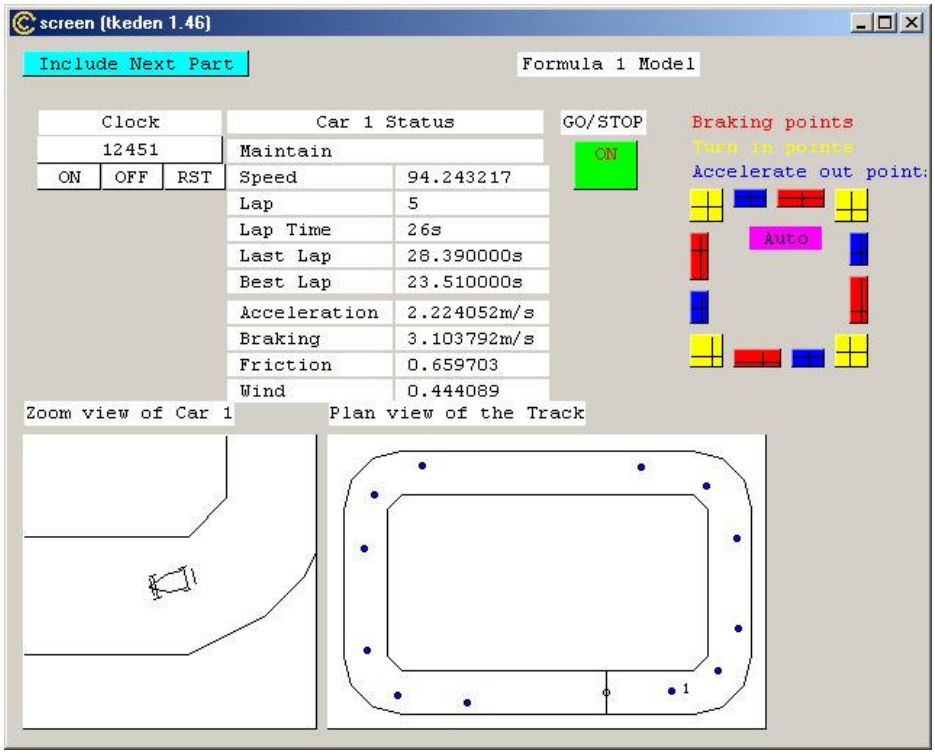

Figure 5.5 – Microworld 3 of the racing cars model

By manipulating the key control points and monitoring changing lap times, learners can explore the positioning of points required to achieve optimum car performance. If a car brakes too late for a corner then it will not stay on the track. In a stable situation, the speed at each position on each lap will be the same. If a car is going faster at the

same track position on each successive lap then eventually the car will miss a corner and leave the track. These characteristics of car racing can be appreciated by interacting with the model. Additional insight could be obtained from the model by linking it to an auxiliary model to plot graphs of speed against lap position and so more easily observe how the car responds to the key track points being changed. In this microworld, the behaviour of the car on the circuit can be explored, but only for a given car set-up. In reality, the car could be set up in many different ways. This additional complexity is introduced in the next microworld.

Microworld 4 (see Figure 5.6) builds on the previous microworlds and allows exploration of how factors associated with the design of the car, such as braking efficiency, engine torque (power), tyres and wing settings affect the behaviour of the car on the track. For instance, an increase in braking efficiency means that the car can brake later for each corner. Likewise, an increase in engine torque means that the car will accelerate faster so that the braking point for each corner must occur further in advance of the corner.

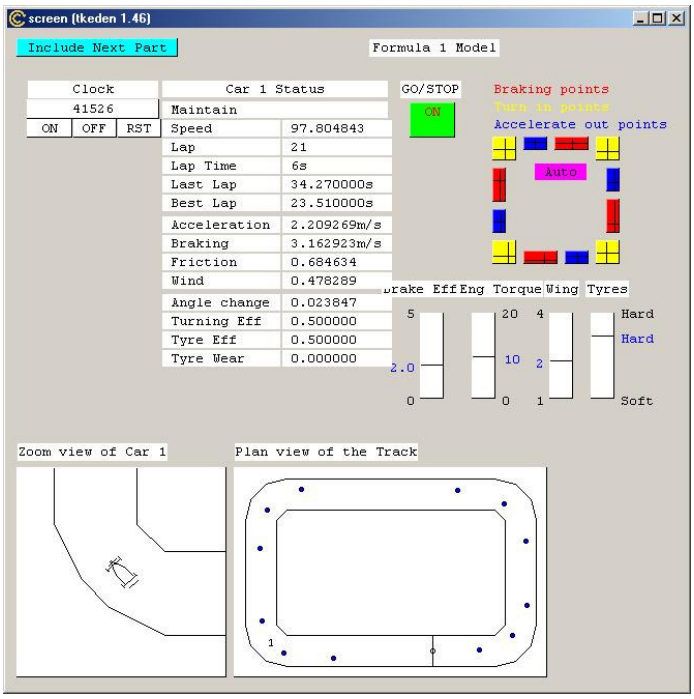

Figure 5.6 – Microworld 4 of the racing cars model

There is a trade-off between the tyre setting and the wing setting. This trade-off can be explored through experimenting with different settings and observing the effect on lap times. Empirical investigation plays an essential role in learning about this trade-off and forms a large part of the experimental testing undertaken by Formula 1 teams. In Figure 5.6, tyre efficiency, tyre wear, turning efficiency and turning angle are also displayed in the 'Car 1 Status' table. Microworld 4 addresses learning objectives for which modelling is essential, such as finding the optimum route around the track for the current car specification, and the optimum set-up of the car for the current track.

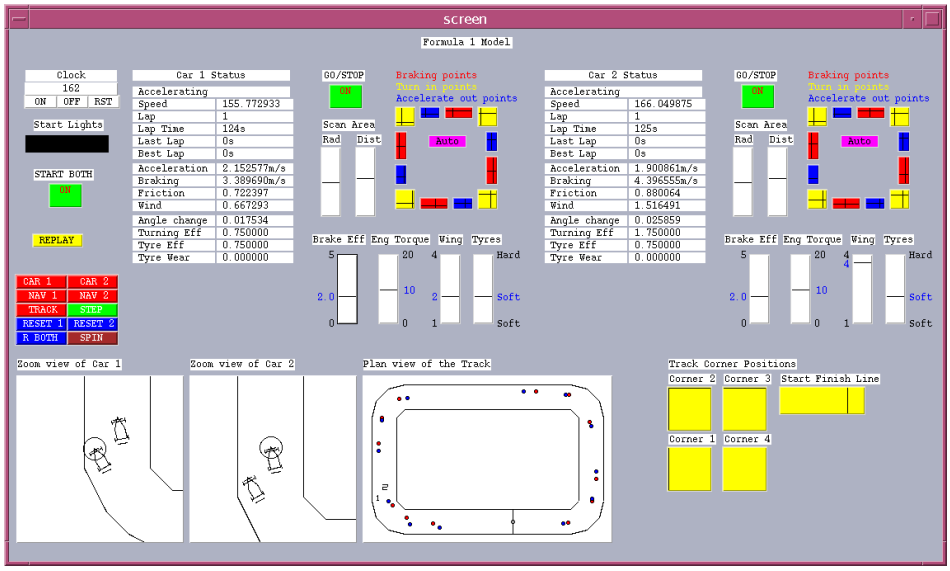

Figure 5.7 – Microworld 7 of the racing cars model

Microworld 7 (see Figure 5.7) shows two fully customisable cars that can be raced around a partially customisable track. In this final microworld each car has a means for obstacle detection: each car avoids obstacles within a scan area of specified radius and distance from the front of the car. There are also facilities for editing the corners of the track and the position of the starting line. One interesting area for exploration is trying to change the set-up of one car to beat the other around the track. Microworld 7 reflects the construal an expert has when exploring car racing situations. If all the set-up options in Figure 5.7 were available in an initial microworld, a learner would be likely to be overwhelmed by the complexity of the model. The object of constructing

the racing cars model is to enable a learner to come to appreciate this level of complexity.

The racing cars environment is constructionist because the learner is free to experiment and is not given a set of questions to answer. From a personal viewpoint, the experience gained through interaction with the model proved useful to me in developing my cornering technique the first time that I went go-karting. However, in such an environment, there is a limit to the level of creativity and invention that a learner can exhibit because the fixed referent constrains the construal that is being explored. In the next section, we consider more abstract microworlds where meaningful interaction is not constrained by a fixed referent and there is learning benefit in open exploration.

5.3 Learning as exploring possibilities and invention

Where the layering of microworlds is constrained by a fixed referent it is inappropriate to allow learners to interact in ways that subvert their emerging understanding. In other contexts, it can be beneficial for a learner to explore unrealistic scenarios and invent scenarios of their own. This is typically the case where the learning activity is directed towards design or invention rather than mere comprehension. For instance, in designing a new board game, learners can benefit from tinkering with the rules of an existing game in an *ad hoc* way that reflects the open-ended nature of experimentation in the world. Developing learning environments for this purpose requires a different style of scaffolding for which we have introduced the term *cognitive layering*.

5.3.1 Cognitive Layering

Scaffolding in educational software is analogous to scaffolding for building houses. This type of scaffolding is sensible if the learning environment is targeted at a learning objective that can be attained through a well understood progression of

stages. In such a context, the form of the scaffolding is itself shaped by the prior knowledge of the overall structure of the learning task. Not all learning tasks can be so structured or have such clear objectives from the start. A different type of scaffolding is required for these tasks. To appreciate this, imagine that, during the construction of a building, its plans were to be changed. This might well mean that the building itself would become impossible to construct with the existing scaffolding.

The term 'cognitive layering' describes an approach to scaffolding microworlds that takes the fact that learners can benefit from open exploration into account [RB02]. To support this open exploration, it is essential to offer the learner less restricted access to the underlying data model than is afforded by closed interfaces such as can be found in the racing cars model. Such open interaction is supported in our principal EM tool TkEden through the specification of redefinitions in the input window. Open interaction is necessary because the designer cannot preconceive the possibilities that a learner may want to explore. With conventional scaffolding, the complete model is preconceived and the learner only has access to a partial subset of its functionality. In cognitive layering, future layers are not preconceived, and can be flexibly adapted in any direction. Figure 5.8 depicts this essential difference between scaffolding and cognitive layering.

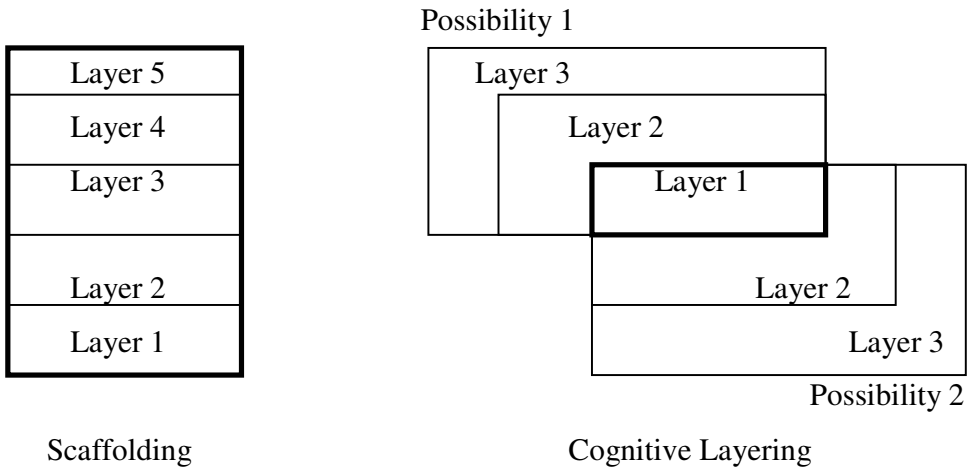

Figure 5.8 – Differences between scaffolding and cognitive layering

We now describe an EM case study in the form of a laboratory for investigating noughts-and-crosses style games.

5.3.2 The noughts-and-crosses case study

Noughts-and-crosses is a simple two-player game. Players take turns to place counters on a 3x3 grid aiming to make a straight line containing three of their own counters. From a strategy perspective, noughts-and-crosses is simple because of the limited number of different games that can be played. However, for children the game can be a challenge, as evidenced by Lawler's research into children's learning that used noughts-and-crosses as a case study [Law85]. In this section, we describe how a series of microworlds to investigate the game of noughts-and-crosses illustrates the idea of cognitive layering in practice. The scope of this investigation embraces a whole family of noughts-and-crosses style games to be referred to generically as OXO games.

The EM OXO model has been developed by a number of people over the past 10 years. The initial model ran in a textual interface and was developed by Meurig Beynon and Mike Joy in 1994 [BJ94]. Simon Gardner added a graphical interface in 1999 [EMRep, oxoGardner1999]. I adapted the model to create a 3D version of OXO using the Sasami notation in 2001 [EMRep, 3doxoRoe2001]. The OXO model that is illustrated in this section is Gardner's 1999 version.

The OXO model is a layered series of four microworlds, each of which introduces concepts not in the previous layer. This is in contrast to the racing cars model described in 5.2.2, where the concepts of tyre compound and wing settings were present in every layer of the model, but were initially inaccessible to the learner. In the OXO model, each microworld embellishes the situation by building upon the previously introduced concepts. Successive microworlds specialise the OXO model so that it more closely resembles the game of noughts-and-crosses [BJ94]. The layers

of the OXO model are depicted in Figure 5.9. These layers reflect: the layout and geometry of the board; the placing of pieces on the board; rules governing the playing of pieces; and strategic play.

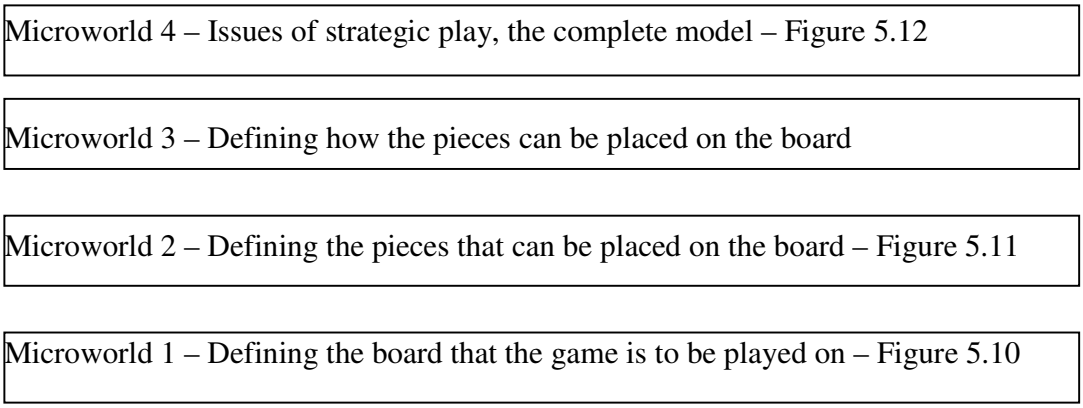

Figure 5.9 – The structure of the OXO model

At all times, the learner can alter aspects of the OXO model by redefining relevant parts of the model using the TkEden input window. In the OXO model, a specific learning path directed towards learning conventional noughts-and-crosses has been mapped out by the model designer. At every layer, redefinitions allow the learner to deviate from the mapped out path to explore variants in the OXO family. The discussion of the layers that follows is illustrated by examples of experimental redefinitions. It is important to note that the experiments that can be carried out are only limited by the learner's imagination.

Microworld 1 specifies the geometry of the board and the concept of lines upon it (see Figure 5.10). There is no presumption about the desired functionality, except that there is a regular geometric board with significant lines. When learning to play a new board game, our attention is initially directed to the geometry of the board and its important features. In our OXO model, the significant lines are highlighted through animation as displayed in Figure 5.10.

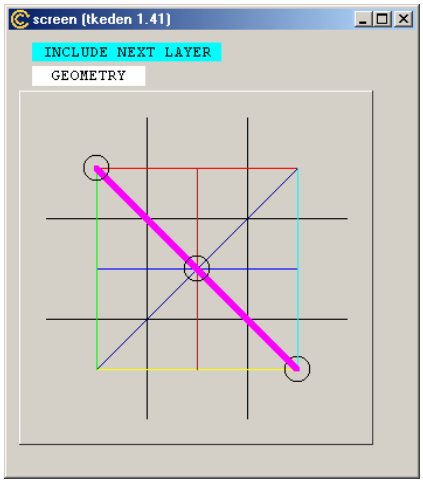

Figure 5.10 – Microworld 1 of the OXO model

In keeping with the theme of learning through exploring possibilities, changes can be made to the board. For example, the significant lines on the board can be redefined individually, or as a whole. The example redefinition:

```
lines[1] = [1,8,3];
```

will replace the horizontal line across the top row of the board by a new 'line' that contains the top left square, the bottom middle square and the top right square. The significance of such a redefinition is not apparent in this microworld, but in later microworlds this would have an impact on winning conditions and good strategic play. In the EM OXO model, the scope for adaptation of this nature is not preconceived. In contrast, educational software designers usually preconceive the useful adaptations that a learner can make in order to guide the learner to explore the possibilities that are deemed important by the designer.

Microworld 2 introduces the concept of placing pieces on the board (see Figure 5.11) and the criteria for a winning position. There are no restrictions on where pieces can be placed or on the order in which they should be placed. This might reflect a situation where players have yet to decide upon the rules of play. The interface in Figure 5.11 displays information that can be ascertained about the state of the board from a static analysis of a position. This includes the number of pieces of each type

and whether the board is full. These can be determined by observing the current state of the board.

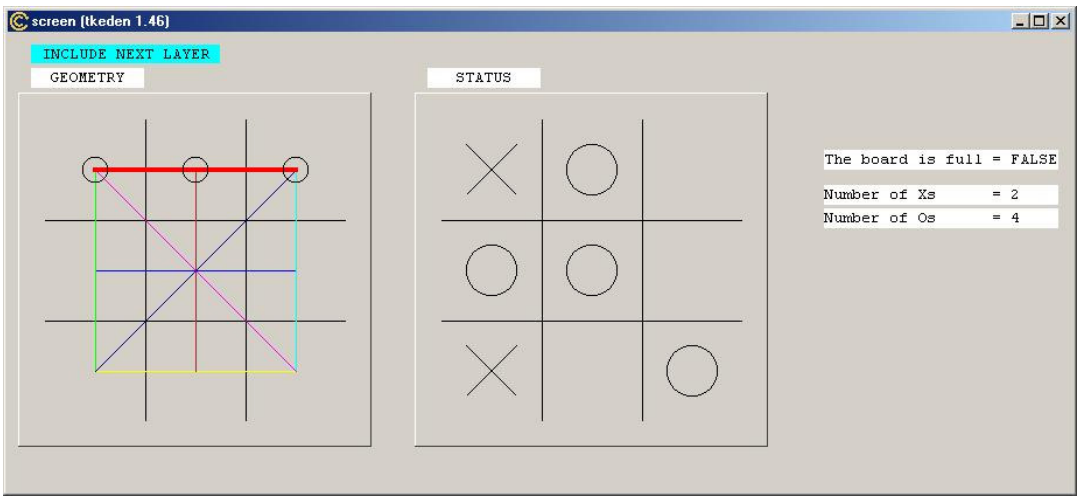

Figure 5.11 – Microworld 2 of the OXO model

This microworld resembles a laboratory where learners can explore the placing of pieces without adhering to any rules of play. It can be used, for instance, to experiment with OXO strategies, or to devise new OXO-like games. For example, learners can experiment with different ways of placing pieces, since the model imposes no restriction on the placement of pieces. Such unrestricted interaction would be outside the scope of a conventional environment for learning about noughts-and-crosses.

Microworld 3 (figure not shown) introduces the concept of playing rules that are characteristic of an OXO-like game. For example, in noughts-and-crosses the rules are simple; players take turns to place pieces and a piece cannot be placed on an occupied square. This microworld is the first layer in which there are two players who are constrained to play according to the current rules. Note that, in keeping with our aspiration to develop an open learning environment, this microworld can reflect the extraordinary variety of ways in which playing rules can be enacted in the world. For instance, we might imagine that players take turns to throw a piece onto the board and forfeit their turn if their piece does not land on an empty square.

Microworld 4 introduces the issues of strategy required to construct an automatic OXO player (see Figure 5.12). This involves two aspects: evaluating the squares on the board and deciding on the 'best' move to make. The value of individual squares is dependent on the state of the board, the rules of the game being played and the evaluation function being used. For example, in noughts-and-crosses the value of a square is dependent on the number of lines that pass through it and the number of pieces already in each line. In this OXO microworld, the learner can investigate the factors that are important in the positional evaluation of OXO boards by tinkering with the scores for each type of line identified.

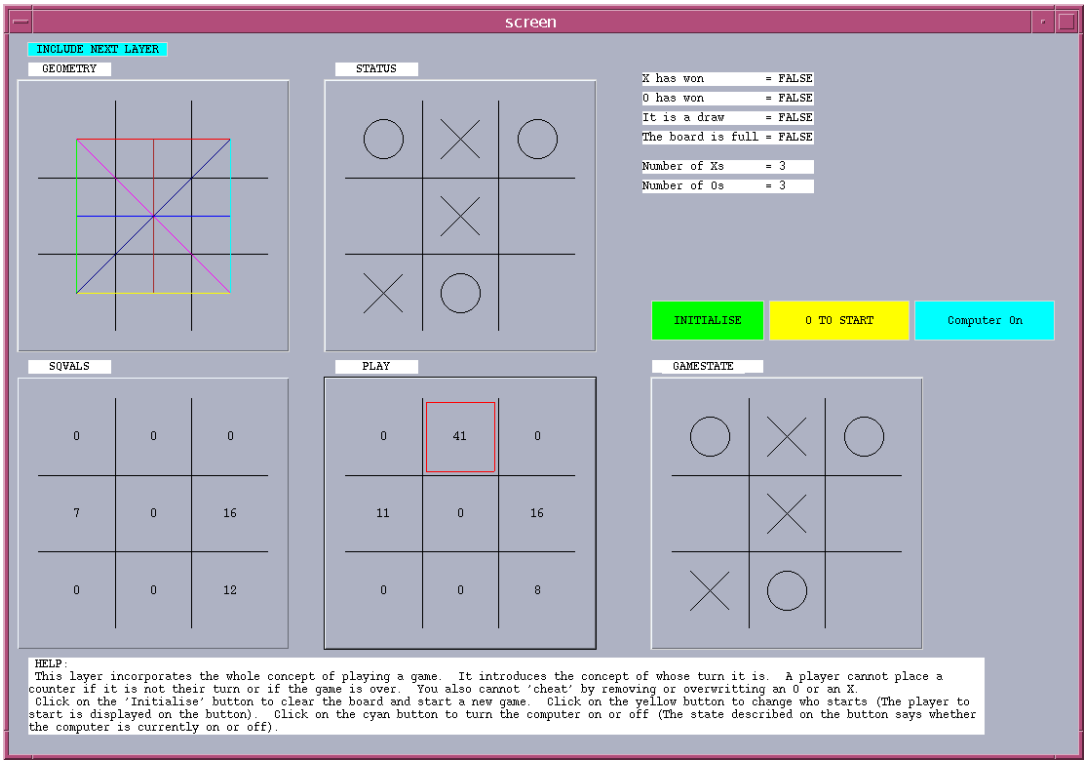

Figure 5.12 – Microworld 4 of the OXO model.

5.3.3 Case study – Adapting layers to form a family of models

The previous section considered the benefits of cognitive layering of microworlds from the perspective of the learner. There are also advantages in using cognitive

layering for the developers of models, since microworlds can be extended in different directions to create a family of models. In the EM OXO model discussed above, each successive microworld constrains the model to more accurately resemble the game of noughts-and-crosses. Adding a different set of rules creates a variant of noughts-and-crosses. In this way, games in the OXO family can be created by reusing some of the original microworld layers. In this section, we give examples of variations that can be introduced at each layer. This leads to a tree of possible models, as depicted in Figure 5.13.

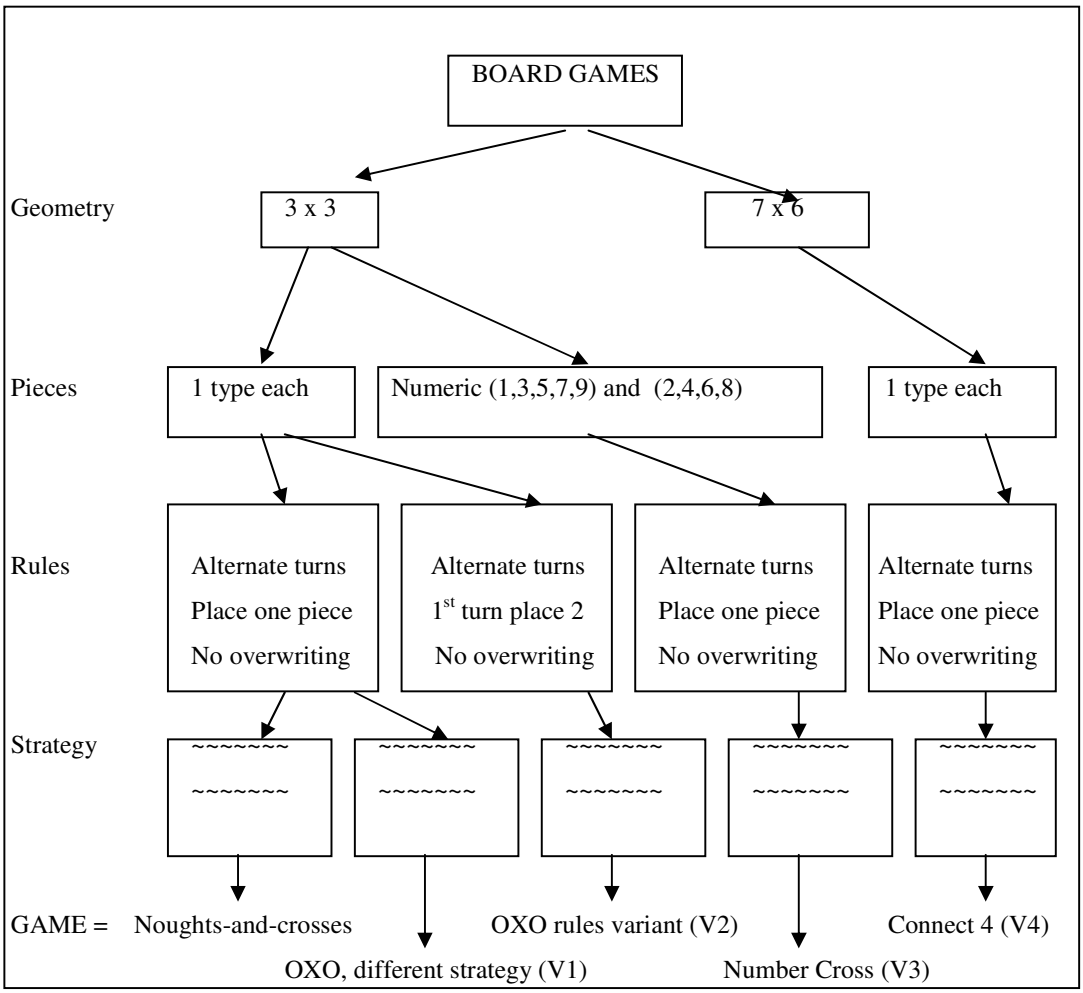

Fig 5.13 – A tree of possible models based on the cognitively layered OXO model

In Figure 5.13, four variants (V1, V2, V3, V4) are created by reusing some of the original OXO model. If any layer is altered, then the subsequent layers will, in general, be different. For example, changing the rules of noughts-and-crosses will probably mean that a different strategy is required. The variants outlined in Figure 5.13 are illustrative of the kind of adaptation that a teacher might want to carry out in order to customise learning resources.

V1 – Altering the computer strategy.

The computer OXO player described in section 5.2.2 contains a serious flaw because a particular pattern of opposition moves is guaranteed to lead to a win. In this variant, we adapt the computer player to eliminate this defect in its play. In the original model, the computer player does not use a minimax algorithm (see [BB96]) but simply analyses the set of lines incident with each square to compute its value. The value of a square is dependent on the number of pieces in the line, and these values are summed to give each square an overall value (see Table 5.1). In EM terms the values `weight1`, …, `weight5` can be regarded as observables for the computer player and can be changed to alter the way the computer plays.

Condition	Observable	Value
X X _, X _ X, _ X X	`weight1`	100
O O _, O _ O, _ O O	`weight2`	40
X _ _, _ X _, _ _ X	`weight3`	10
O _ _, _ O _, _ _ O	`weight4`	6
_ _ _	`weight5`	4

Table 5.1 – The evaluation strategy for player X in OXO

Using this evaluation routine, the computer player would respond to the game situation in Figure 5.14 by playing in the bottom left, as indicated by the highlighted square. The value of 22 attached to this square can be construed as the result of a

particular 'mode of observation' on the part of the computer player. The threatened winning diagonal from bottom left to top right contributes 10 to the value of the square (cf. `weight3`). The blocking of the left and bottom lines each contributes 6 to the value of the square (cf. `weight4`). The opponent can respond by playing in the top right square thereby blocking the potential diagonal winning line and creating two winning squares for their next move.

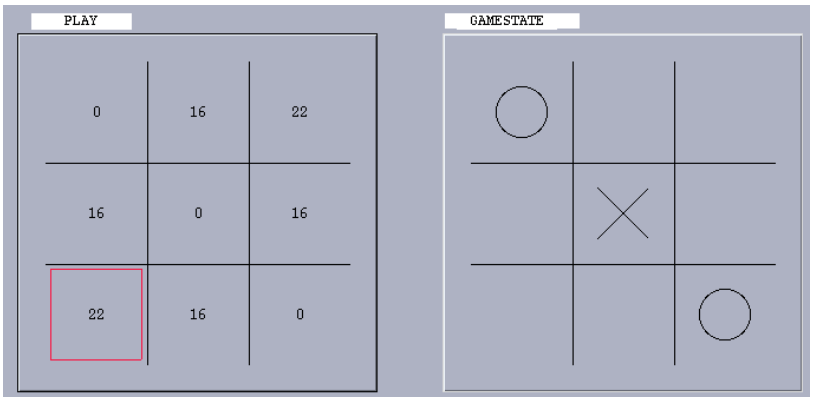

Figure 5.14 – A problem situation for the OXO computer player.

The problem with the existing evaluation routine is that the computer player does not observe situations where the opponent can introduce a *fork*: a situation where the opponent can make a move that sets up two independent ways to win. An extra condition to recognise fork situations, together with a change to the evaluation routine, changes the strategy of the computer player to avoid the trap in Figure 5.14.

This example is fairly trivial due to the simple nature of noughts-and-crosses. In more complex games such as chess, changes to the computer player could alter its strategy to play defensively, to attack, or to try and control particular important squares. The OXO variant described in this example uses the same board, the same pieces and the same rules as noughts-and-crosses. The only difference is in the strategy of the computer player. EM principles are well-suited for making changes of this nature, which involve changing the way in which the computer player is construed to observe the state of the game. The above example shows how our modelling principles enable

the computer strategy to evolve through experimental interaction. Papert has observed that children use a similar style of development when writing computer programs to play noughts-and-crosses [Pap93]:

> 'rather than following strictly in the path of the so-called "knowledge engineers" who build expert systems, children followed in the path of psychologists who deliberately construct a series of "inexpert" systems that made the computer act like a "novice" and then pass through a progression of levels of increasing expertise'.

It would be of particular interest to adapt the computer player so as to model human strategies employed in learning to play noughts-and-crosses. Understanding of good strategic play emerges from experience of the game. Learners, especially children, cannot initially expect to fully understand how to play a good game. Lawler's extensive study of how an individual child learnt to play noughts and crosses supports this claim [Law85]. His study led him to recognise four stages of comprehension in playing noughts-and-crosses, namely [Law85]:

i) Naive comprehension – the learner's play is guided by individual inclinations but with no idea about how to achieve particular outcomes. They typically move anywhere for obscure reasons.

ii) Fragmentary comprehension – the learner acts on the basis of highly specific knowledge of one or two games. They typically respond in a rigid way, independent of the opponent's strategy.

iii) Procedural comprehension – the learner can recognise situations in which victory can be forced. They typically know when they are going to win before their opponent plays their last piece.

iv) Systematic comprehension – the learner is familiar with all the possible game situations and appropriate responses (cf. Lawler's comprehensive classification of noughts-and-crosses games – such classification is only possible for simple games). They typically make the optimum move at all times.

In applying EM principles to model these particular stages in learning we would adapt the computer player to reflect the observation and construal of the child at their current level of competency. This would also be a way of providing an appropriate opponent to scaffold the child's learning of noughts-and-crosses at each of Lawler's stages.

V2 – Altering the rules of the game

Variant 1 only differs from the standard OXO model of noughts-and-crosses at the strategy layer. Variant 2 differs from the standard model at the rules layer; the board and the pieces placed on it are the same as for the game of noughts-and-crosses. In variant 2, the standard rules of noughts-and-crosses have been changed so that, on their first turn only, each player can place two pieces. In the OXO model, there are definitions for whose turn it is to play. To make the simple change to the rules specified above, it suffices to replace these definitions. The new definition for player x is shown in Listing 5.1.

```
x_to_play is (!end_of_game) &&
  ( /* X is about to make their first move */
      ( (( nofx<2)&&(nofo==0)&&(startplayer==x)) ||
      ((nofo==2)&&(nofx<2)&&(startplayer==o))
  )
||
  ( /* X is about to make a subsequent move */
      ((nofo>nofx)&&(nofo>1)) || ((nofo==nofx)&&(startplayer==x))
  )
);
```

Listing 5.1 – The new definition that describes the state of the board for when player x should play.

In general, a change to the rules of a game also requires a change of playing strategy. The best playing strategy will depend on the specific features of the game, the board, the pieces and the rules.

The sample rule change described above is one of a number of possibilities that is limited only by the imagination of the learner. Experimentation with different rules gives the OXO model the flavour of a 'rule laboratory' in which the implications of different rules for the game can be explored. Many changes can be made: rules can be added to an existing set; existing rules can be redefined; or rules can be removed.

Educational arguments have been made in favour of acquainting pupils with the notion of devising and adapting rules because of the significant part this plays in social behaviour [BFP+03]. On this basis, experimentation with rules is perceived as an important educational activity in programming games within the Pathways programming environment [GKN+01]. In Agentsheets (see section 2.3.3), the creation and manipulation of agents' rules is the main programming activity [RRP+98]. Both Agentsheets and Pathways are rule-based programming systems and the rules are obligations to agents to behave according to the specified rules. Rule-based programming is recognised to be a problematic way of specifying behaviour since changes to rules are liable to lead to instability and incoherence [Akm00]. For instance, the requirement to bind rules to agents can lead to difficult issues relating to object-orientation (cf. [GKN+01, section 6.1]).

From an educational perspective, it is important that the semantics of rules in programming should conform as closely as possible to that of rules in the world. In EM, rules are implemented in conjunction with dependencies in such a way as to guarantee that the integrity of state is preserved: the maintenance of dependency is not itself the product of user-specified rule-based action (cf. the discussion of Agentsheets in section 2.3.3). This makes it much easier to imitate the semantics of real-world rules inside EM models. For instance, the rules in the EM OXO model place a constraint on the moves that can be made and so shape the referent. This shifts

the emphasis from using rules to maintain the semantic relation α to using rules to maintain the semantic relation β (cf. section 2.2.2).

V3 – Altering the pieces that are being played

This variant of the OXO model only re-uses the board layer. The second layer of the OXO model describes the pieces to be used. We will refer to the game in this section as *number cross*, a game based on noughts-and-crosses. In noughts-and-crosses, each player has just one type of piece. Number cross uses the same board as noughts-and-crosses but uses the numbers 1...9 as the pieces. The aim of the game is to complete a line of three numbers that sum to 15. Figure 5.15 shows the number cross model. At this layer, the rules of the game are as yet unspecified.

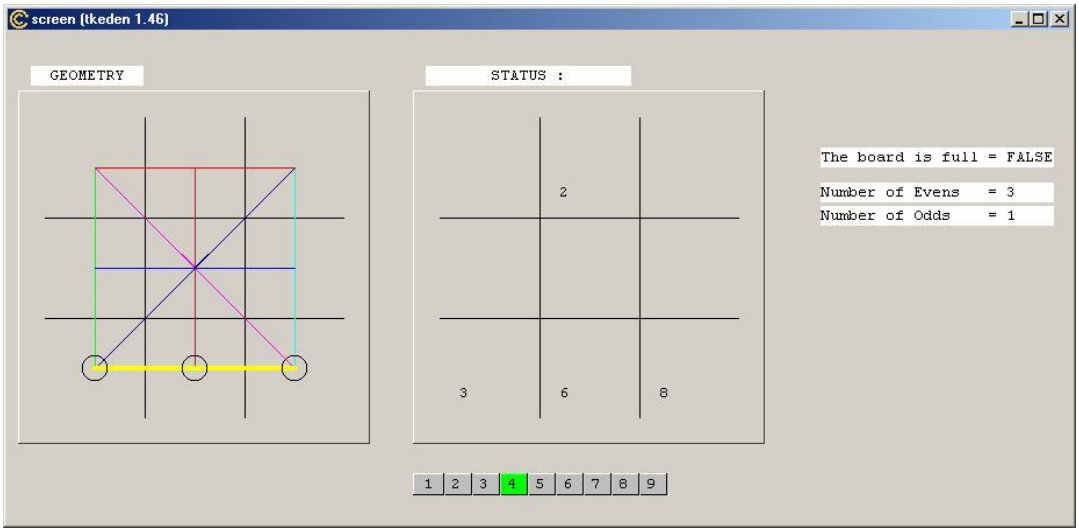

Figure 5.15 – The board and pieces of the number cross model

The interface in Figure 5.15 allows each player to select a piece to place on the board – a feature that is not required for noughts-and-crosses. There are no restrictions on where a number may be placed, or any rules governing who can play a particular number.

In number cross, the players take turns to place pieces. The player who starts ('odd') can only place odd numbers and his opponent ('even') can only play even numbers. Each number can only be used once and may only be placed in an empty square. These rules are introduced into the model at the next layer. Figure 5.16 shows the game with the rules layer added.

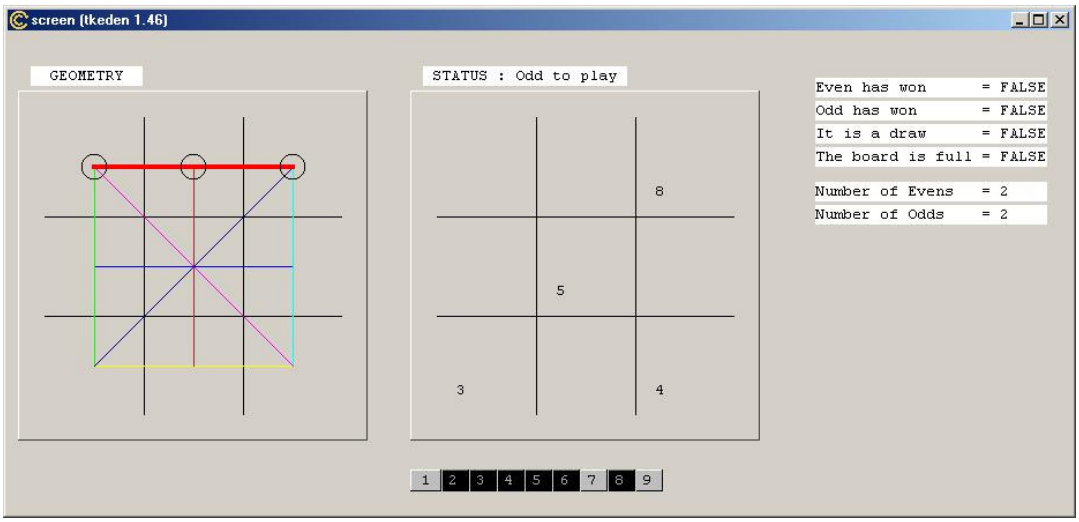

Figure 5.16 – The game of number cross with the rules present.

In order to model the game of number cross, the rules of noughts-and-crosses have been extended through adding game rules such as:

```
n1valid is isodd(player) && not_used(1);
```

to indicate that piece 1 can only be placed by the 'odd' player and it is not already on the board.

The strategy layer for the number cross model is quite different from that of the noughts-and-crosses model because winning lines can use opponent's pieces.

The example described in this section illustrates that different board games can be constructed with little revision. This ease of revision can be of educational benefit when teachers can adapt the model to take advantage of a particular learning situation. For instance, a related game to number cross is 'the game of 15' where

there are no restrictions on which pieces can be played. There is an intimate relationship between the game of 15 and noughts-and-crosses. If the numbers 1, 2, ... , 9 are placed on a 3x3 grid so that they form a magic square, then the game of 15 is equivalent to noughts-and-crosses. Learners and teachers can use the game of 15 as a base from which to explore simple properties of odd and even numbers and the mathematics of magic squares.

V4 – Altering the board

The above sections have illustrated how the EM OXO model can be adapted through systematically replacing layers. The purpose of introducing this final variant is to illustrate that the OXO model can serve as a template from which to construct more general board games. This level of adaptation is typical of what might be required in educational contexts. Each of the variants discussed so far has used some of the original OXO model. Even when we change the board, it is still possible to maintain the same layered structure of the model. By way of illustration, in a game of Connect 4, the vertical 'board' has 7 columns and 6 rows, and a winning configuration is one in which four pieces of the same type lie on a line of contiguous squares. The Connect 4 model has the same layered structure as the OXO model, but differs at every layer. A layered model of Connect 4 can be found in the EM repository [EMRep, connect4Roe2003].

The model development discussed in this section illustrates principles that offer advantages to model developers, teachers and learners. These can be summarised as follows:

- For the model developer – the structured design exhibited in the OXO model is an aid to reuse (cf. design patterns in OO programming [FRK+01]).
- For the teacher – cognitively layered models allow easy customisation to take advantage of opportunities offered by learning situations.
- For the learner – cognitively layered models allow issues in the neighbourhood of the original model to be explored.

In the next section, we consider how EM models can address issues of learning to communicate and represent our emerging understanding of a referent using language.

5.4 Learning languages

Talking about our experience has a fundamental role in learning about a domain. With reference to the EFL, language is associated with moving from pre-articulate interactions to objective knowledge. As the discussions of rowing and piano-playing in chapter 3 illustrate, language skills develop alongside our experience of a domain. The role of language in learning often transcends the typical use of formal language – meanings are personal, determined by situation and negotiated through interaction. In the learning context, knowledge of domain specific language develops incrementally with the learner's competency, and builds on their evolving experience and understanding of the domain.

Conventionally, computer-based languages are not well suited for adaptation to their context in use. Interaction languages in computer-based learning environments typically have their functionality fixed by a designer. A learner must interact with the language of the domain as specified by the designer. This is appropriate when the designer understands a language well, and it is not subject to change. Enabling full engagement with the learning agenda involves adapting the language to the needs of the learner. For this purpose, the language must be opportunistically adaptable: to take account of new concepts as they are encountered; or to promote new ways of interacting with existing concepts. This level of adaptability is not a feature of formal languages, which stand in a preconceived relation to the domain (cf. Figure 5.17). It is more characteristic of natural language, as when we use the same word to describe rowing on a static machine, rowing on a machine with slides and rowing in a boat (cf. section 3.2.1). The key issue is that the semantics of a formal language is abstractly specified and independent of its context of use, whereas that of a natural language develops with experience of use in context.

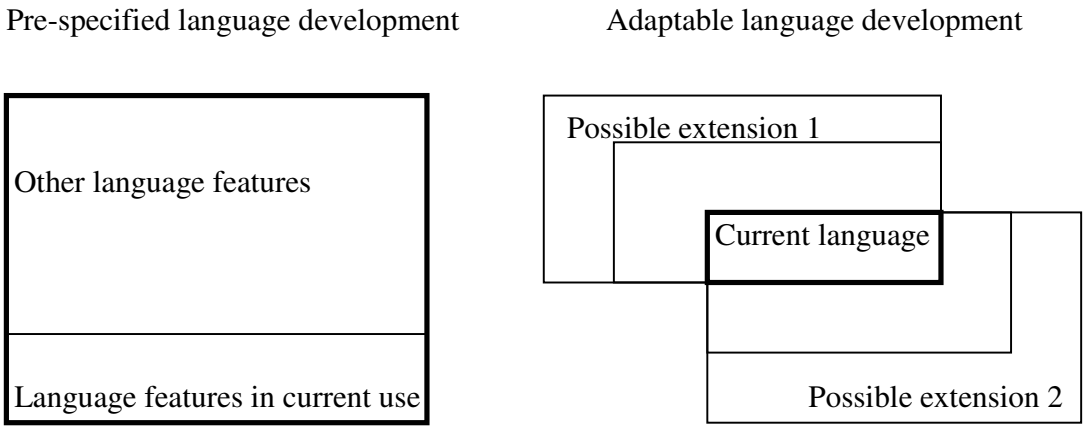

Figure 5.17 – The relationship between pre-specified and adaptable language development

In this section, we argue that learning environments need to exploit languages that can be framed on-the-fly and opportunistically extended to match learners' competencies. This is useful for teachers in customising learning resources to take advantage of particular learning situations. In the next section, we describe an approach to interactive parsing of adaptable computer-based languages, and then illustrate it with two practical case studies.

5.4.1 The Agent-Oriented Parser

The Agent-Oriented Parser (AOP) is a utility that can be used in conjunction with the EM tool TkEden, for interactive parsing of adaptable languages. The AOP utility was respectively constructed and refined by two final year undergraduates, Chris Brown [Bro01] and Antony Harfield [Har03].

The AOP differs from a conventional parser in many respects. Instead of parsing blocks or lines from left to right, the parser searches for the most salient features of a statement, in the way that we might derive the meaning of a statement by inspection. For example, when we look at the string '`a=b+c;`' we recognise that it is an

assignment by observing the = symbol, then expect to find a variable identifier on the left hand side and an expression on the right hand side. When parsing the string from left to right, symbols may have to be stored without knowing their semantic significance until the meaning of the entire statement becomes clear. The AOP also allows the parser itself to be modified on-the-fly, that is in such a way that the parsing conventions can be changed even whilst the interpreter is executing.

A full technical discussion of setting up a parser for a complete notation is beyond the scope of this thesis. Appendix B contains documentation from [Har03] that shows how an example calculator notation can be constructed. The following discussion assumes a basic level of familiarity with the parsing approach described in Appendix B.

Two key advantages of the AOP are its flexibility, and the way in which it generates parsers that can be adapted on-the-fly to suit particular learning circumstances, or to reflect a change in the language of interaction. Each AOP language contains a set of definitions that describes how the language should be parsed, together with the actions required to translate these statements into EDEN code for execution. For example, in the krusty notation (see section 5.4.2) the statement that recognises the `down` command translates this into a procedure call to move the clown in the maze (as shown in Listing 5.2).

```
krusty_statement3 =
     ["literal","down",
          ["action",
               ["later","move_clown(3,1);"]],
                    ["fail","krusty_statement3_2"]];
```

Listing 5.2 – The example command for the down operator in the krusty language described in section 5.4.2

In building an EM learning environment, the languages that are developed with the AOP are not preconceived, and can be introduced, refined or extended on-the-fly to reflect emerging understanding or emerging requirements of a teaching/learning situation. Conventional parsing could easily be used to construct a language functionally equivalent to an AOP language, once all refinements and extensions have been specified. What is important to stress however, is that there is no restriction in how a language developed using the AOP can evolve and be reconfigured in interactive use. This is beyond the scope of conventional parsing approaches such as are described in [GBJ+00].

We now discuss two case studies that have used the AOP to define domain-specific languages. The first, the clown-and-maze environment, shows how a simple language can be defined for young children to navigate a maze and how this language can be subsequently adapted and extended to suit different learning requirements and abilities in a way that was not preconceived. The second, the SQL-EDDI environment, is targeted at learning about relational database query languages. In this example, the languages are much more complex and were developed incrementally to suit the evolving educational objectives of an undergraduate database module as it was being taught [BBR+03].

5.4.2 Case study – A clown-and-maze language

The clown-and-maze environment [EMRep, krustyRoe2002] shows how learners can be scaffolded towards understanding the Logo language [Pap93]. In this case study, we illustrate an extensible notation for young children that initially allows them to express geometric concepts in a simpler way than in Logo. In the Logo language, a turtle is controlled by giving it commands such as `forward 50` or `left 90`, which move or turn the turtle appropriately. Commands can be combined into repeating blocks, or grouped into a procedure that can be referenced by name. Papert intended young children to use Logo to explore geometrical concepts. However, the

concept of 'angle', and even of 'turning', may be too sophisticated for young children. The clown-and-maze environment provides basic primitives – in the form of the krusty language – that can be used as a starting point for learning about directions and turning. The krusty language can be incrementally extended towards Logo, and in this way can provide scaffolding for understanding Logo. Figure 5.18 shows the relationship between the languages described in this section.

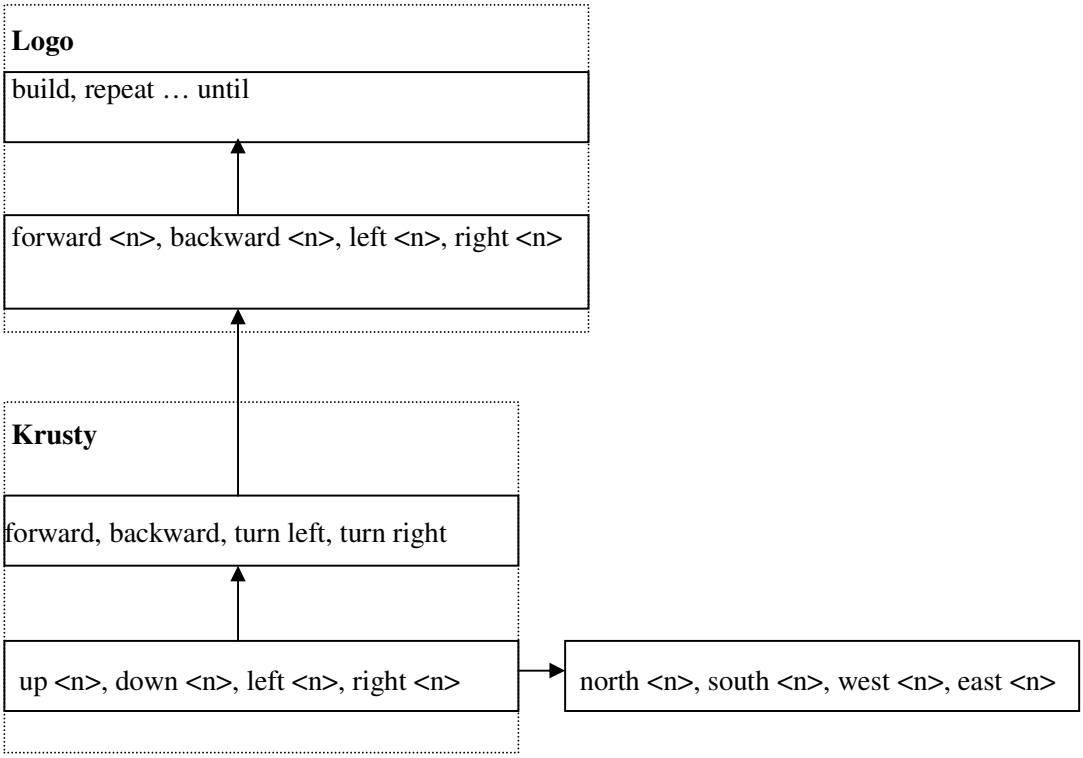

Figure 5.18 – The structure of the clown-and-maze languages

Within the clown-and-maze environment, the learner's task is to direct the clown to the treasure in the centre of a maze (see Figure 5.19). The maze is a 5x5 grid whose walls become visible as the clown visits the squares in the maze.

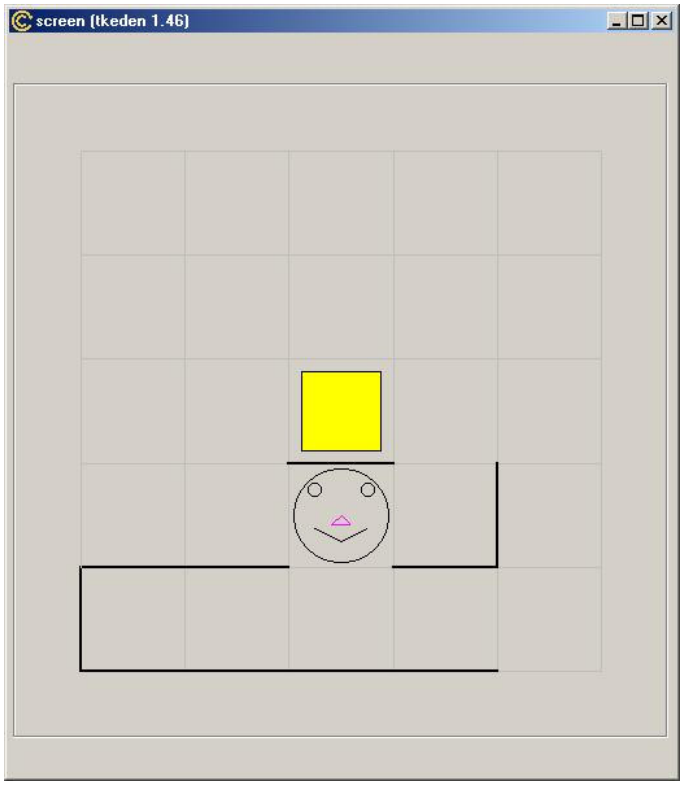

Figure 5.19 – The clown-and-maze environment.

Initially the clown can be controlled using the basic set of Krusty commands, `up`, `down`, `left` and `right`. Each of these directional commands can have an optional numeric parameter to move the clown that number of squares in the specified direction (e.g. `up 2`, `left 3`). This exposes the learner to the use of parameters, a concept required to use Logo. Krusty is a simpler interaction language than Logo because it is not necessary for the learner to take account of the way the turtle is facing.

Using the AOP, we can incrementally and interactively adapt or extend the basic krusty language to scaffold learning of more advanced manipulation languages. For example, compass directions could easily be substituted for the basic movement commands (substituting '`east`' for '`right`' etc), to satisfy different educational objectives. To scaffold Logo learning, an intermediate language that introduces the concept of turning can be introduced. Children who are in the process of learning to

distinguish 'right' from 'left' could benefit from a control language where the commands are `forward`, `backward`, `turn left` and `turn right`. Success in controlling the clown now depends on understanding the concept of turning. In Figure 5.19, the point of reference for the direction the clown is facing is the tip of its nose.

The next language layer introduces the concepts of distance and angle that are found in Logo. The new commands available in this layer are `forward <d>`, `backward <d>`, `left <a>` and `right <a>`. Controlling the clown using Logo is one way of learning about angles. The clown's nose is actually a Logo turtle, re-used from an earlier EM student project [EMRep, logoEdwards2000]. In moving the clown around the maze, the values for the angle `a` should be confined to multiples of ninety degrees and `d` to multiples of the square size, in order to keep the clown in alignment with the maze.

The clown-and-maze environment could be used to learn about more complex movements. For example, mazes could be irregular in shape, so that the learner would have to manoeuvre the clown through the maze using arbitrary angles and distances. This would refine the learner's concepts of 'angle of turn' and 'distance'. The clown-and-maze environment could also be used in significantly more advanced learning situations. For instance, notations and primitives could be designed to allow users to investigate and develop algorithms for maze solving.

The clown-and-maze case study illustrates how the learning of a domain-specific language for interaction can be scaffolded from a simple level through a number of competency levels. The AOP allows the construction of a flexible layered learning environment, where there is no restriction on how the interaction language at each layer can be extended or refined. It would not be difficult to construct a learning environment in which the interaction language would adapt dynamically to match the competency exhibited by the learner in moving the clown successfully around the maze. A more advanced illustration of the use of the AOP is described in the following section.

5.4.3 Case study – A learning environment for relational query languages

At the University of Warwick, a core 2nd year module is Introduction to Database Systems. The module aims to give students a basic understanding of relational database theory and practice. The practical component of the course comprises an introduction to SQL (Structured Query Language) and exposure to relational algebra, the mathematical language that underpins relational query languages. The objectives of the practical component of the course are to:

1) teach SQL as a relational database query language
2) get students to appreciate that relational query languages are based on relational algebra
3) get students to appreciate that SQL has a poor mathematical semantics because it is unfaithful to the relational model of query languages.

Objective 3 is the major focus of the learning environment discussed in this section.

In the past, the practical component of the course was taught exclusively using a commercial relational database system. Whilst students have undoubtedly benefited from this experience, it is not ideally suited for the learning agenda outlined above. In particular, commercial database systems are not designed for highlighting the flaws in the design of SQL and so give little support for learning objective 3. A special purpose environment targeted at this objective could show how the design of SQL deviates from the relational model it supposedly embraces [Dat00, DD00].

The SQL-EDDI environment [EMRep, sqleddiWard2003] was developed by Meurig Beynon from an original prototype developed by EM group members Chris Brown, Michael Evans and Ashley Ward. It allows learners to interact with tables and views using [BBR+03]:

- a pure relational algebra query language ("EDDI")
- a variant of SQL whose semantics is consistent with relational theory ("SQLZERO")
- a subset of standard SQL.

The main educational objective of the SQL-EDDI environment is to allow students to study the evaluation of relational algebra expressions, and relate these to the translation and interpretation of standard SQL queries.

The interpreter can be interactively changed so that SQLZERO is interpreted according to the evaluation conventions of relational algebra, or those of standard SQL. Figure 5.20 shows how the languages within the SQL-EDDI environment are related.

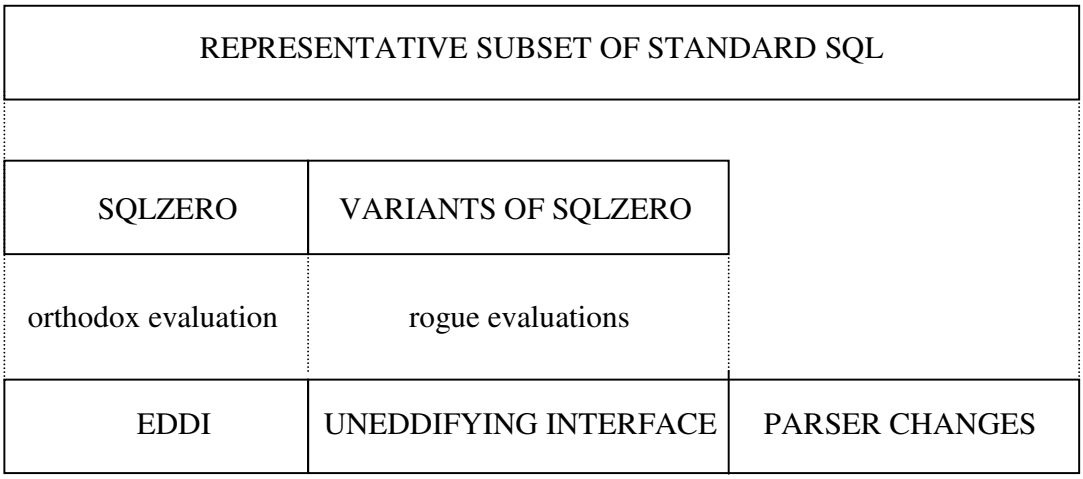

Figure 5.20 – The relationship between the query languages in SQL-EDDI

EDDI is a relational algebra language that allows users to create tables and interrogate them using the basic relational operators of union, difference, intersection, projection, selection and natural join. It is loosely based on Todd's Information Systems Base Language (ISBL) [Tod76], realising all of its functionality but adopting different syntactic conventions. The `eddi` interpreter is a front-end to the EDEN interpreter since commands are translated into EDEN for execution. Listing 5.3 shows some EDDI code to create a small example database. The line numbers are not part of each command and are only included for purposes of discussion. Lines 1-12 create the database by defining the tables and populating them with records. Lines 13

and 14 create views on the tables, whose current value is always kept up to date, and line 15 assigns the value of a relational algebra expression to a table.

```
1. %eddi
2. allfruits (name CHAR, begin INT, end INT);
3. allfruits << ["granny",8,10],["lemon",5,12],
                ["kiwi",5,6],["passion",5,7];
4. allfruits << ["orange",4,11],["grape",3,6],
                ["lime",4,7],["pear",4,8];
5. allfruits << ["cox",1,12],["red",4,8];
6. apple (name CHAR, price REAL, qnt INT);
7. apple << ["cox",0.20,8],["red",0.35,4],["granny",0.25,10];
8. citrus (name CHAR, price REAL, qnt INT);
9. citrus << ["lime",0.30,3],["orange",0.55,8],
             ["kiwi",0.75,5],["lemon",0.50,2];
10. soldfruit (name CHAR, unitsold INT);
11. soldfruit << ["cox",100],["granny",153],["red",70];
12. soldfruit << ["kiwi",23],["lime",15],
                 ["lemon",55],["orange",78];
13. fruits is allfruits % name;
14. popcitrus is (fruits.citrus % name).
                        (soldfruit : unitsold > 50 % name);
15. nonapplesncox = allfruits-
       (allfruits*apple%name,begin,end)+allfruits:name==”cox”;
```

Listing 5.3 – An EDDI extract illustrating the definition of the FRUITS database

Understanding of how standard SQL is related to relational algebra is scaffolded through the introduction of SQLZERO, an SQL-like notation. SQLZERO queries are translated into EDDI by using the `sqlte` interpreter, which displays the translation but does not carry out the evaluation. The evaluation strategy in EDDI can be interactively changed so that SQLZERO is interpreted according to the evaluation conventions of relational algebra, or in such a way that it mimics standard SQL. Changes to the evaluation strategy are made via the ‘Uneddifying Interface’ to be described below (see Figure 5.22).

When the orthodox evaluation conventions of relational algebra are adopted, SQLZERO is a variant of SQL that is faithful to Codd’s relational model [Cod70]. Through interaction with the SQL-EDDI environment in this orthodox evaluation mode, students can appreciate the intimate connection between SQLZERO and relational algebra. By interacting with the SQL-EDDI environment in other

evaluation modes (cf. Figure 5.22), students become aware of the flaws in the design of standard SQL. Though changing the evaluation strategy readily makes it possible to mimic the interpretation of simple queries in standard SQL, it becomes evident that much more is required to support the interpretation of more complex standard SQL queries. It was at this point in developing the SQL-EDDI environment that the flexibility for exploratory language development proved to be most significant; it was only through experiment that a feasible strategy for implementing a more representative subset of standard SQL emerged.

As stated above, the main educational objective of the SQL-EDDI environment is to provide a way of exploring how the design of SQL has deviated from the relational model and the implications of this. SQLZERO with the orthodox evaluation conventions differs from standard SQL in that:

i) `SELECT` is treated as a synonym for `SELECT DISTINCT`,

ii) type checking on constructing union, intersection and difference of relations takes account of both domain types and attribute names,

iii) `SELECT * FROM X,Y` is interpreted as a natural join of relations.

Figure 5.21 shows SQL and EDDI statements that can be used to highlight the flaws in the design of SQL that respectively stem from i), ii) and iii) above. Query 1b) is the nearest equivalent in relational algebra terms of the SQL query 1a). In EDDI, query 1b) returns a set of *distinct* fruit names. In standard SQL, query 1a) returns duplicate rows for the cox, red and granny tuples. We should expect both queries 2a) and 2b) to be equivalent to the relational algebra expression underlying the EDDI query 2c). In standard SQL, queries 2a) and 2b) return tables with the same contents but with different attribute names (cf. the output tables in Figure 5.22). In EDDI, query 2c) causes a semantic error when type checked. Much syntactic complexity in standard SQL could be avoided if query 3a) generated the natural join that is specified in EDDI query 3c) but in practice query 3b) has to be used to achieve this result. In standard SQL, query 3a) returns a table with six columns (allfruits.name, begin, end, apple.name, price, qnt), two of which have identical contents whereas query 3c)

returns the natural join of the two tables, namely a table with five distinct columns (name, begin, end, price, qnt). SQL query 3b) explicitly eliminates the duplicate column that is generated in query 3a).

Duplicate rows:

```
1a) SQL:  (SELECT name FROM apple) UNION (SELECT name FROM
      allfruits)
1b) EDDI: ?apple % name + allfruits % name;
```

Loose type checking in creating unions:

```
2a) SQL:  (SELECT * FROM soldfruit) UNION (SELECT name, qnt FROM
      citrus)
2b) SQL:  (SELECT name, qnt FROM citrus) UNION (SELECT * FROM
      soldfruit)
2c) EDDI: ?soldfruit + citrus % name, qnt;
```

Indirect and clumsy representation of natural join:

```
3a) SQL:  SELECT * FROM allfruits, apple
3b) SQL:  SELECT allfruits.name, begin, end, price, qnt FROM
      allfruits, apple WHERE allfruits.name=apple.name
3c) EDDI: ?allfruits * apple;
```

Figure 5.21 – Some example SQL queries and their EDDI equivalents

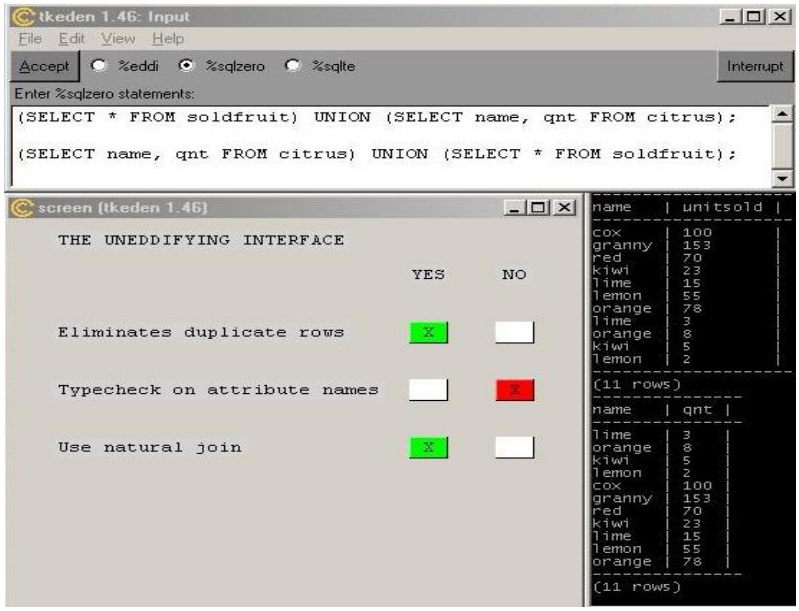

Figure 5.22 – The SQL-EDDI environment in use (cf. queries 2a and 2b in Figure 5.21)

EDDI queries obey the strict mathematical conventions of the relational model. In the SQL-EDDI environment, the interpretation of SQLZERO is changed via the 'Uneddifying Interface'; this adapts the evaluation so that (cf. the three logical flaws described above), it allows duplicate rows, typechecks on domains alone, and uses 'unnatural' join.

The design of the SQL-EDDI environment was not preconceived, and the pedagogical goals for the software emerged as the development was being carried out by Beynon on-the-fly in parallel with the teaching of the database module. The use of EM in the development of SQL-EDDI was significant in two respects:

- the flexible and organic nature of the development meant that it could proceed alongside the teaching.
- the adaptable language parsing offered by the AOP meant that incomplete languages could be developed and flexibly modified on-the-fly to support different teaching requirements.

By way of illustration, the eventual development of a parser for a more representative subset of standard SQL required changes to both the syntax and the evaluation strategy used in implementing SQLZERO – this could be effected by introducing small files comprising new definitions and redefinitions. This was not a conceptually simple process, free of error, or technically straightforward, but the entire activity of testing, modifying and debugging the environment revolved around interpretation through experimental interaction of the modification of small groups of definitions.

The EM development of SQL-EDDI was carried out within the same environment that the students were using for tutorial purposes. In principle, this process could be continued in extending the SQL-EDDI environment to address issues such as:

1. supporting a larger subset of SQL features, (e.g. more sophisticated data definition, integrity constraints and support for nulls).
2. implementing other relational query languages (e.g. QUEL [Dat87]).
3. incorporating an interface to study optimisation of relational database queries.

To further illustrate the concept of scaffolding we now show how the SQL-EDDI database environment can be tailored for use with younger age groups to introduce relational algebra operators as operators on tables.

The Relational Algebra Tutor (RAT) uses colour coding to suggest how the operators of relational algebra work. Each of the six relational operators (project, select, union, intersection, difference, join) has a different meaning and is applicable in different circumstances. Students will be unable to formulate queries in EDDI without a sound conceptual grasp of how these operators work.

Figure 5.23 shows the RAT in use. The interface is split into three sections:

- The top section shows the input tables that are generated from EDDI queries. These can either be individual tables or complex EDDI queries.
- The middle section contains a switching mechanism to change the currently selected operation, together with information about the currently selected operation. This information comprises the EDDI language statement that produces the output table from the input table(s), and the currently selected operator. The field for specifying parameters for a command is only required for the project operator (when it specifies the names of the columns to project) and the select operator (when it specifies the boolean condition used to select rows from the table). If an operation cannot be performed – for instance, if tables are not be compatible for union, intersection and difference – then this is reported in the error window.
- The bottom section shows the output table formed by the operator applied to the input tables. The rows and column headers of the output and input tables are colour coded to show how the result of the query is composed from the input tables. For example, in Figure 5.23, the current operation (union) is coloured yellow, the column headers are also highlighted in yellow, and the rows from the output table are coloured differently depending on the input table from which they have been derived.

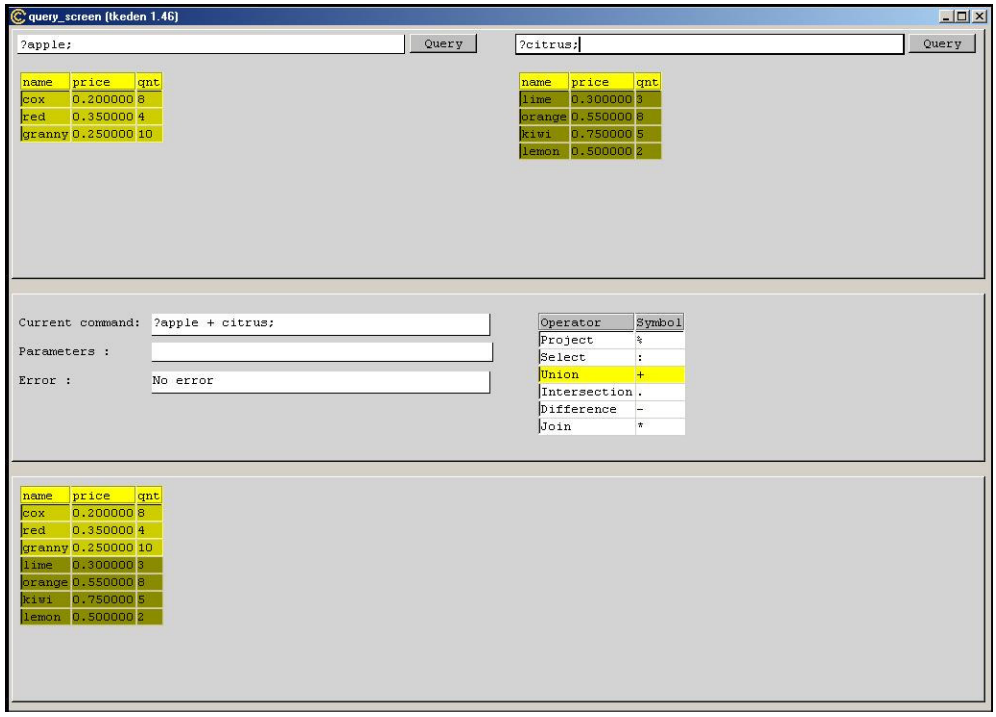

Figure 5.23 – Using the RAT to support understanding of operations on tables

The construction of the RAT environment illustrates a high degree of code re-use. The development time I required – as an EM expert – was about 2 days, but such development would be impossible for a non-computer specialist. The RAT uses spreadsheet grids (see section 2.2.1) to display the input table(s), the operators table and the output table, and uses EDDI to generate the output table by executing the command string built up in the 'Current command' window. The high level of re-use meant that the majority of the model was constructed from existing resources. The colour coding for the input and output tables is dependent on each individual operator and was implemented using simple search and matching routines.

With reference to the EFL, the purpose of the RAT is to allow learners to experiment with basic relational algebra operations on various tables to establish and reinforce their conceptual understanding of the operations on tables and the EDDI language. RAT provides the support for learners to gain the experience of interpreting symbolic relational algebra operators that is required to use EDDI successfully.

5.5 Chapter Summary: Scaffolding with Empirical Modelling

In this chapter, we have described EM case studies that have illustrated scaffolding operating in the zone of proximal development in a wide variety of contexts (cf. Soloway's TILT model, Figure 5.2). The analogy suggested by scaffolding – of rigid and predefined buildings – seems inappropriate to describe the rich ways in which the EM models described in this chapter have been flexibly developed and presented. In [NH96] Noss and Hoyles described three criticisms of the scaffolding metaphor in computer learning:

i) the notion of scaffolding suggests a structure being erected around the learner by an external agency. This may not take account of how learners structure their own learning.

ii) The idea of a 'zone' is a useful metaphor that suggests the idea of a bounded territory. It is important to leave open how it is defined and where its limits are.

iii) The idea of the scaffolding fading away with learning implies that, if the computer provides scaffolding, then it should be removed at some point. This is not always desirable.

Our case studies illustrate how EM can give more support for learning than the traditional scaffolding metaphor suggests. For instance they exhibit support systems for learning with characteristics that address Noss and Hoyles's criticisms outlined above:

i) Our case studies support the idea that the learner should control their own learning. For instance, in the racing cars model (see section 5.2.1), the learner is always in control over when they move on to the next microworld.

ii) The OXO family of games case study (see section 5.3.3) presumes no preconceived bounded territory within which learning is to take place, since the learner is always being encouraged to explore.

iii) The SQL-EDDI environment (see section 5.4.3) gives support to learners that remain accessible to the expert. For instance, the SQLTE translation interface can always be used to confirm relationships between SQL and relational algebra.

Noss and Hoyles propose an extension of scaffolding that they call *webbing*. This draws on the metaphor of the World Wide Web to convey that the learner accesses a support structure that they can draw upon and reconstruct as they learn. Webbing is distinctive because [NH96]:

i) It is under the learner's control.

ii) It is available to signal possible user paths rather than point towards a unique, directed goal.

iii) The local and global support structures are dependent on the learner's current level of understanding.

The support structures provided in the EM case studies described in this chapter give practical evidence of the use of webbing in learning environments. For instance, the specific OXO game is a possible path that a learner can follow, but there are many other games that can be explored 'in the neighbourhood of OXO'.

In chapter 2, we discussed how learning activity can be associated with the negotiation and elaboration of concepts (cf. section 2.2.2). The notion of scaffolding supports the negotiation of the semantic relation β but is limited in respect of elaboration. In the racing cars model, the concept of 'car racing' is gradually exposed to the learner. A learner understands the concept at a simple level before it is embellished. This leads the learner to embark on a process of negotiation of the concept through experimental interactions and making and testing hypotheses. When a learner is comfortable with the concept at a particular level of complexity they have the control to move on to the next level. However, the fixed nature of the referent limits the scope for investigative exploration around the subject. In this respect, scaffolding is limited with respect to elaborating the semantic relation β.

In contrast to scaffolding, webbing offers better support for learning through the elaboration of the semantic relation β. Since webbing is an extension of scaffolding it is natural to expect that it still supports negotiation of the semantic relation β. The analogy that underpins webbing – that of building connections in a flexible structure as in the Web – shows that elaboration of the semantic relation is represented in a webbing approach. The scope for using EM in 'building connections in a flexible structure' is illustrated in our OXO case study.

Learning is nevertheless much more than can be represented in terms of scaffolding or indeed webbing. Our previous discussions (cf. chapter 3) have shown how learning activities can be very diverse. This diversity cannot be represented within preconceived frameworks for presenting models to learners. Model use can be more varied than is represented in the case studies presented in this chapter. Model building can likewise take exceedingly diverse forms. In the following chapter, we discuss three EM case studies that illustrate a variety of different types of learning and ways of developing and interacting with models, and interpret them with reference to the EFL.