Chapter 8
RECORDING GAME PLAY

1. Why record play?

Our main objective in developing a recording scheme was to assist game developers in analyzing game play. In the course of developing a game we'd play it several times, then find ourselves unable to remember in detail how the game went. After playing, we'd propose and discuss modifying the rules to improve the game, but we needed to know what actually had occurred. We tried recording play by photographing the board after each move using an instant camera, but this proved difficult and distracting. We considered using a videotape recorder but this, also, seemed to involve too much extra effort, both in recording the play, and again later, in viewing the tapes. Videotape might be a good way to record the games if one wanted to observe all events—including those not directly pertaining to game play. But we merely wanted to record the moves made.

Games that forbid players from moving or removing pieces once placed build, in effect, a partial record of game play. The board at the end of play contains every move that was made. Tic-tac-toe is a simple example. In such games it may be possible to reconstruct the sequence of play, or at least plausible sequences of play, just by looking at the board at the end of the game. But in our concept design games it is usually impossible to reconstruct the sequence of moves from the final state of the board. Hence our need for a means to record moves as the game is played.

From the start, we thought to construct the game recorder as a computer program that could accept descriptions of game moves and display game states and histories on the screen. It was logical, therefore, to consider using the computer to develop and play the games as well as to record game play. Initially, however, we decided to develop our games outside of the computer. We wanted to play with physical pieces on a physical site. We wanted to ensure that our games could also be played without a computer. We thought of the game recorder as a tool in a larger game development process.
2. Recording and Reviewing

The game recording task is minimally construed as keeping track of pieces placed: added, removed or moved. This includes recording:

- what piece,
- where it is placed,
- who placed it, and
- when it was placed.

Recording just this information would be useful for post-game analysis. In an extended version of the game recorder we may also want to record why the piece was placed, check whether the move is legal, and perhaps to keep score. Further, we may want the game recorder to keep track of rules, agreements, negotiations made among the players. Eventually we would develop the game recorder towards an environment for game play and game development.

The purposes of recording histories of game play are:

- to view and review previous game states, sequences of states, and entire histories of game play;
- to restore previous game states;
- to replay games from previous states.

A history of a single instance of play is simply a sequence of moves and board states. When a game is replayed from an intermediate state a branch is added from that state—the sequence becomes a tree. The history now represents a set of alternative ways the game was played; with every replay, the history tree becomes bushier.

3. Notation

Chess notation offers a model for our recording scheme. Three column-inches of newsprint suffices to record all the moves in a game. Chess notation describes a process, the sequence of moves from opening to endgame. The two players' moves are listed in order. Each move names a piece, and describes its final board position: P - QB4, or, in the case of a capture, it names the two pieces that engaged: P x P. Chess notation is sparse, as measured by the number of characters needed to record a move, or a game. It conveys each move using a small number of symbols.
In part, the elegance of Chess notation derives from assuming the reader understands the game rules. For example, to interpret the notation P x P the reader must know how pawns capture in order to determine which two pawns might be involved. If the notation did not presume this knowledge, it would need to describe the pawns' initial locations, and this would require more symbols. Thus if we devise a different game using the Chess board and pieces, most likely we cannot use Chess notation to record play in our new game.

This last observation relates directly to our problem. We wanted to devise a means to record play across a variety of different games. As mentioned above, in some of our games the rules change, are left unstated, or are invented as a part of play. Unlike Chess notation, our recording scheme cannot rely on knowledge of the game rules.

4. Layout Languages
From notation we came, therefore, to the idea of a layout or picture language. A layout language offers a means to describe and manipulate a two or three dimensional configuration of physical elements as a symbolic code. Once we have represented the layout in symbolic form, we can apply standard list-permuting and pattern-matching techniques, either to transform the layout, or to recognize, parse, and classify its features. By using a layout language to describe the physical configuration on the game board we avoid integrating the notation system with the game rules.

Layout, or picture languages have been widely applied. Physicists devised picture languages to describe patterns of atomic particle scatter [1]. Layout languages are also used by VLSI designers in the layout phase of automated integrated circuit design [2, 3]. The formal properties of layout languages have been studied [4]. A picture language can be recursive and purely functional [5]. The shape grammar formalism used in some studies of the built environment [6, 7] to represent thematic variation in built environments is also related.

5. Writing Form
We call our layout language "Writing Form", or "WF" for short [8]. The game recorder uses WF to represent board configurations. It is implemented as an embedded language in an object-oriented dialect of Common Lisp [9]. WF nouns refer to material and space elements; WF verbs refer to operations for arranging these elements in space. WF's primitive arrangement operation strings elements along imaginary lines. Each WF sentence denotes a configuration of material and space elements; the most elemental description of configurations are given in terms of edges and distances.
For example, the letter "E" can be described in WF as a horizontal string of two elements: a vertical edge (e1) and a vertical string of three horizontal edges: two identical top and bottom edges (e2) and a middle one half as long (e3), with a distance (d1) between each pair of horizontal edges (see figure 8-1). First we define these edges and the distance, then we string them together to make the "E" configuration:

```
(define e1 (edge vertical 30))
(define e2 (edge horizontal 20))
(define e3 (edge horizontal 10))
(define d1 (distance 10))

(define E
  (string horizontal e1
    (string vertical e2 d1 e3 d1 e2)))
```

The names of the simplest elements -- the edges and distances -- are shown in the diagram. Note that each element is named uniquely, even the two instances of edge e2 that form the top and bottom horizontal edges of the E.

Writing Form users can extend the language in two ways. First, WF may be extended by defining new element types. Element types defined within WF are first-class WF objects: they may be composed and configured in the same ways as the built-in edge and distance elements. Second, users can define new operations for positioning elements relative to one another. These operations are independent of the elements they may be applied to. For example, players may define alignment, centering, equidistant spacing as well as more complex means and routines for positioning elements and configurations using parameters, conditionals, and iteration.

We have built two interpreters for the WF language. The first--we call it the WF interpreter--translates WF expressions to layouts for display. The second--the WF reader--translates layouts entered with a graphic editor into WF expressions. For our present purposes it is sufficient for the WF reader to generate any WF expression that describes the layout. (Here we are not concerned with simplest or canonical forms).
6. A Game Recorder

The game recorder appears to the players as a graphic editor. Players record moves by directly manipulating pieces; they need not learn the WF language that the game recorder uses internally to represent board states. The game recorder screen is divided into three parts (figure 8.2). One part contains a supply of elements of various types. Each available element type appears as an icon; selecting the icon generates a new element instance that can be placed on the board. Another part of the screen contains the game board and the pieces in play. A third part of the screen displays the game history.

Players take turns making moves: selecting elements from the supply and placing them on the game board, moving elements already on the board, or removing elements. The game recorder, using the Writing Form reader described above, translates these moves into its internal representation and stores them in its history. Players can view game states by selecting items on the history and also explore the consequences of making alternative moves by playing games over, beginning at intermediate states.
The game board at any time is described, internally, by a WF expression. As we have seen, the physical configuration and the symbolic WF representation correspond directly. It follows that their transformations also correspond directly. Adding, deleting, and moving pieces around on the board corresponds to (lexically) adding, deleting, and replacing symbols and subexpressions in the WF expressions. The "current board state" expression may also be viewed and edited in a text buffer. Changes made textually will alter the layout, and this change will be recorded as a move.

7. Extending the game recorder

So far the game recorder only records players' moves: the sequence of board states and transformations. Now we want to extend the game recorder to take account of rules. In particular we want it to "know about" constraints on the selection, placement, and dimensions of game pieces. Such constraints determine the "behavior" of the game pieces. It must be possible for players as well as the game developer to write these constraints. Here are some examples of the sorts of constraints we want the game recorder to be able to handle:

"washers only on grid crossings";
"pegs always perpendicular to nails";
"a washer must occur wherever a nail and a peg meet";

In games where a well-defined technical universe dictates allowed and forbidden arrangements of game pieces, we want the game recorder to check players' moves for validity. In games where players introduce rules during play, the game recorder should keep track of when rules are entered, rescinded, applied, violated. Once the game recorder can be taught rules, perhaps it could also suggest possible moves.

To achieve these goals, we are recasting the game recorder and the WF layout language within a constraint-based computing environment. Following well-known work on constraints [10, 11, 12], we have developed a constraint language [13, 14] and connected it with a package of interactive graphic routines. The constraint language interpreter maintains, manages, enforces, solves relations among variables that describe attributes of a system whose behavior we wish to simulate. We can use constraints to describe both the geometry of our pieces and the rules about piece placement.
For example, we can define "peg" as a special class of "rectangular-physical-object", with fixed dimensions, and certain position constraints with respect to other similarly defined objects, for instance, nails. We might require of pegs that they must always be connected and perpendicular to a nail. If we place a peg parallel to a nail, the game recorder can 1) report the violation and reject the move; 2) report the violation but accept the move, 3) rotate the peg 90 degrees in order to satisfy the constraint, or 4) offer alternative ways to satisfy the constraint (rotate the nail instead, turn the peg the other way, turn both peg and nail 45 degrees, etc.).

The constraint-based version of the game recorder is still under development. Initial experiments seem promising. In games, as in real-life designing, rules and constraints play an important role. Therefore, constraint-based computing seems to offer an appropriate technology for research in design theory and methods. However, much work remains to be done, both to advance constraint-based computing techniques and to demonstrate that design rules can be expressed in these terms. We have found our concept design games a useful vehicle for both these aims.

REFERENCES