Computer Programming for Noughts-and-Crosses: New Frontiers

W.M. Beynon, M.S. Joy
Department of Computer Science, University of Warwick, CV4 7AL

ABSTRACT

We relate the development of computer programs that play variants of noughts-and-crosses to an analysis of the perception and action involved in human play. We outline the construction of a computer model, expressed as a script of definitions, within which the perceptions and actions of agents such as the players and observers can be recorded. We briefly compare and contrast our concept of agent-orientation with alternative approaches to agent-oriented programming.

1. Introduction

This paper illustrates unusual principles of software development based on a particular kind of real-world modelling. The abstract ideas behind our method are discussed at length elsewhere [2], and the emphasis here is upon investigating a particular application in detail. To this end, we develop a model from which we can derive a class of computer programs for playing various games with a family resemblance to noughts-and-crosses, or *OXO-games*. For clarity, we shall refer to the computer model from which we develop our programs as the *OXO-model*.

The Concept of the OXO-game

For convenience, we refer to the family of games we study as **OXO-games.** Examples of commercial games in this family are:

- noughts-and-crosses
- 3-d dimensional noughts-and-crosses, as played on a 4×4×4 grid
- Go-moku
- Connect-5
- Othello

Each of these games is played by two players **X** and **O**, who in turn enter an X or O symbol respectively into a grid structure with a view to achieving a line of tokens of a certain length. In some case, there are special constraints on the positions at which symbols can be entered. In Connect-5, the X and O symbols are represented by black and white discs whose position in a vertical grid is constrained by gravity so that discs support each other. In Othello, the symbols are represented by discs, each of which has one black and one white face, that can be turned over in the course of play. In the context of this paper, we shall consider many other variants of these traditional OXO-like games. For instance, we may introduce other notions of *grid* and *line*, or modify the protocols and environment for play, as in playing blindfold OXO, simultaneous OXO, postal OXO, OXO with a time-limit on each play, OXO with cheating, etc.

1.1. Background to the approach

From a psychological perspective, the most interesting characteristic of our approach is that it connects software development with the analysis of processes of perception and action. In analysing the requirement, we uncover hidden assumptions about the way in which real-world activity is synchronised, or is deemed to be synchronised, and construct a partial model that captures these assumptions. The primary activity in the modelling process is the faithful representation of real-world states in terms of relationships between observations that can be determined by experiment. It is only when we have finally modelled the context for agent action that we introduce agents into the OXO-model. Even at this stage, as in a real experimental situation, there is scope to adapt system behaviour through introducing additional factors, such as introducing a timer, or another agent with the power to intervene in play.

1.1. Software Development as Engineering Design

In our development method, we regard computer programming as a special case of *constructing a physical agent with a particular capacity for autonomous response*. Viewed in this light, developing a program to play an OXO-game is an exercise in engineering design. What we must do is to identify a physical system (an OXO-game automaton) that can be observed in such a way that it is seen to encode the state of an OXO-game, that can be set up to alternate between two different kinds of state representing valid positions of an OXO-game in which either the automaton itself or the user (as OXO-game player) is responsible for initiating a state-transition that represents a valid play.

In designing a suitable physical system, we construct the OXO-model – an agent-oriented computer-based model that can represent both the OXO-game automaton and the human player. The purpose of the OXO-model is to specify the interaction between these two agents that are responsible for initiating significant state-changes in the course of play. The characteristic feature in our analysis is that we focus upon how each agent identifies changes in the state of its environment and how it can itself effect such changes. The results of this analysis are expressed in terms of *observations* that define the interface between agents in much the same way that we would account for the behaviour of a complex engineering system in terms of the observed values of significant values that could in principle be identified through scientific observation. By modelling the interaction between agents in the system in these terms, we arrive, through a process of incremental development and experiment, at a prescription for an OXO-game automaton that can itself be interpreted as an executable OXO-game program (cf [8], where we illustrate the application of similar principles in the context of engineering design). The OXO-model includes information that would be just as relevant to the design of an OXO-playing robot as a OXO-playing program, but, in the context of this paper, the OXO-game automatons with which we are primarily concerned are computers executing appropriate programs.

2. Principles of the Analysis

Our approach to the modelling task described above can best be understood by analysing perception and action in OXO-games between human players. In the ordinary way, we take the observation and interpretation involved in playing OXO-games for granted. This is to neglect implicit assumptions underlying the interaction between players, concerned with what each player can see, how each player can act, what each player knows and the nature of their environment. To disclose the hidden assumptions that make it possible to play an OXO-game, we shall consider the implications of perturbing the context for a traditional OXO game – in much the same spirit that a scientist might perturb parameters in an experimental context.

In a traditional game of OXO, the two players interact via a diagram that depicts the state of the OXO game, as in Figure 1:

```
X | O | X
  | O | O
  | X |
```

The players are the primary state-changing agents of interest in this application. Following the principles set out above, we must consider how, in the course of playing OXO, they *identify* changes in the state of their environment and how they can *effect* such changes. There are two aspects to this process: a relatively low-level aspect, concerned with the physical capabilities of the players, and a higher-level aspect, concerned with interpreting the situation in the OXO game. In appreciating the nature of this activity, it is helpful to think in terms of skills that a young child progressively acquires in becoming an OXO-game player, and of the difficulties that would be encountered by adult players disadvantaged by the loss of faculties through accident or illness.

The low-level activity of the two players involves identifying the locations of Xs and Os in the diagram, over which they must be in agreement. They have also to be capable of entering an X or an O in a space at an appropriate time.

The high-level activity depends upon abstract knowledge of rules of the game. For instance, it concerns: Who is to play next? Which player is X and which is O? What is a legitimate play? Is the game over? Has X or O won? Issues that require yet more sophisticated interpretation of the position are concerned with deciding the best play.

2.1. Synchronisation and action

Analysis of the relationships between observations made by OXO-games players exposes subtle assumptions about the medium through which the game is played and the way in which action and interpretation are synchronised. The approach we adopt to detecting some of these assumptions is illustrated in this section. The result of this analysis is to

identify features of the OXO-game context that have to be captured in our computer model.

The process of writing an X or an O takes a finite time, during which the interpretation of the OXO diagram is suspended – there is no meaning attached to a diagram in which part of an X has been entered into a square. A variant of OXO might stipulate that player X must alternate between writing an X as \ followed by / and as / followed by \, and that player O must alternate between writing an O clockwise and anticlockwise – a violation of this rule to lead to an X being deemed an O and vice versa. This particular OXO-game relies upon special characteristics of the playing medium, and the modified rules could not be applied if the game were played by placing O and X-shaped tokens on a 3 by 3 board.

Even when each play is regarded as an indivisible action, synchronisation of play can be problematic. In a game of postal OXO with an inadequate protocol for correspondence, it is conceivable that a player's moves might be received in the wrong order.

Synchronisation is implicit in determining how a player's perception of an OXO position relates to the *actual* position. We are accustomed to play OXO whilst looking at the diagram, and take it for granted that there is no discrepancy between the pattern of Xs and Os that is objectively seen on the diagram and what we as players perceive it to be. Some deep issues lurk behind this seemingly innocent presumption. What is the pattern objectively? A blind person, placing Braille symbols on an OXO board, might not trust the perception of a sighted person unfamiliar with the Braille alphabet; conventional OXO players might not trust the perception of a blind referee. The significant concern here is that there are typically several scientific observations that relate to the same object, and that these can be interrelated in very subtle ways. As a simple illustration, consider the variant of OXO in which only O's are visible to player X, and vice versa, and a player who selects an illegal play loses.

The characteristics of an OXO game are strongly affected by nature of the communication that informs a player about the actual position. Contrast looking at a diagram, reading a board with tokens bearing Braille symbols, and interpreting the sensation of hot (X) and cold (O) tokens placed on the palm of your hand. Inspection of a diagram provides a view that is "continuously" (or at any rate very frequently) refreshed, whereas interpreting a Braille board requires a tactile survey of each square. The distinction would be relevant in a variant of an OXO-game in which a limited form of cheating was deemed to be legal. The player with the move might be allowed to interchange a pair of X and O tokens on an OXO-game board prior to making a move, subject to the constraint that the game would be lost if such cheating was correctly alleged by the opponent on the following move. It would be exceedingly difficult for a blind player to play such a variant of Go-moku, as the number of tokens can be exceptionally large.

Yet another kind of synchronisation is concerned with linking interpretation of the OXO-game to the current state of the board. When a particular play leads to the introduction of a line of 3 identical tokens, a

physical event (placing a token on the board) is synchronised with a conceptual event (the winning of the game). In this context, synchronisation is linked with knowledge of the rules of the game, and is subject to similar concern about agreement over interpretation between players and an objective observer (such as a referee) as was discussed above.

Interpretation of the board also operates at a simpler level, in the apprehension of the geometry of the board. Playing an OXO-game depends upon being able to perceive a line in the grid, and determine its length.

The purpose of our analysis of OXO-games and their derivatives is to express the common knowledge of physical properties of real-world systems that is required to appreciate what is happening in the process of playing the game. This knowledge is expressed in terms of agents that are responsible for changes of state (such as the players), the observations of the system that account for the perceptions of agents (such as the position of the tokens, the geometry of the board, the rules of the game), and the way in which changes in these observations are synchronised in the process of playing the game.

The observations to which we refer need not of course be observable in the colloquial sense. Observations are values that in general would have to be determined by experiment, and their diverse nature reflects the range and subtlety of phenomena to which agents in a real-world system can respond. For instance, as the discussion of how players apprehend the board illustrates, it may be appropriate to distinguish between visual inspection of the board and other modes of identifying the location of tokens.

In analysing an OXO-game, synchronisation is relative to what agents are acting, and how we decide to constrain, observe and interpret their actions. In postal OXO, we do not normally consider the possibility of subversion through third-party intervention. In the 2-person OXO games we have considered, we are not concerned with how long a player takes to select a square. The playing protocol prevents players from modifying the diagram simultaneously, and from playing out of turn. It is on this basis that we can regard the update of the board as synchronised with apprehension and interpretation of the resulting position by a player; no action can intervene between one play and the next. This assumption would not apply to accelerating OXO, in which each play had to be made more rapidly than the previous one.

3. Synthesising the model

Our method of developing an agent-oriented model for OXO-games is based upon representing the significant relationships between observations identified in the analysis above. The OXO-model (as outlined in Listing 1 below) contains such generic information about OXO-game playing that it can be applied in a variety of ways. For instance, it can be used to study the implications of varying the rules of OXO and the perceptions and capabilities of the players in many different ways, and trivially adapted so that the role of one or both players is automated. In

particular, our OXO-model has been implemented in the special-purpose programming language EDEN that we have developed, and in this way serves as the source for a family of EDEN programs to play OXO-games. Full details of the method are omitted from this abstract, but the essential principles will be briefly outlined.

Our model is developed incrementally through a sequence of simple modelling steps, each of which leads to the construction of a script of definitions (or *definitive script*) that is to be interpreted with reference to a protocol for redefinition. Each script captures assumptions about synchronisation of observations such as we have identified above (cf [8]). Each script can also be regarded as representing a particular aspect of the design of an OXO-game playing program, associated with a highly specialised view of the OXO-game playing process. The correspondence between analysing perceptions and actions in OXO-game playing and synthesising an OXO-game playing program is very precise. As an illustration of this, in our approach, the natural way to develop an OXO-game playing program is first to build the display interface (what does the board look like?), then to express the relevant geometric structure of the board (what are the lines in the grid?), then to apply the rules of OXO to interpret static positions (is this position won or drawn?), then to consider naive interpretation of positions in play (is it my turn? what is a valid play?), then to consider matters of strategy (what is the best play in this position?). This hierarchical organisation of the design reflects the hierarchy of perceptions and actions underlying OXO-playing, ranging from low-level capabilities to see the board and apprehend geometric patterns to high-level abilities to interpret the position and devise a playing strategy.

The correspondence between aspects of perception of OXO-game playing and different perspectives on the design of an OXO-game automaton is illustrated by considering one of the definitive scripts that make up the OXO-model – the DISPLAY script. The observations referred to in this script are concerned with how the appearance of the diagram (as a pattern of light rays on the eye) is related to the location of Xs and Os on the OXO-diagram. The display script comprises definitions that express the way in which the appearance of the board is indivisibly linked to the disposition of Xs and Os in a conventional OXO-game. In this context, definitions are used to express conceptual indivisibility that is justified by the assumption that the speed of light is orders of magnitude faster than the speed of OXO play. On the other hand, the exact nature of these definitions reflects the faithfulness of a player's image of the position – we would choose a different set of definitions to model OXO as played with a board and symbols so well-used that Xs and Os could no longer be distinguished, and the players had to remember where they had played.

Viewed from the perspective of OXO-playing program design, the DISPLAY script connects an abstract internal model of an OXO position to its visualisation (cf [1]), and can be written in different notations to reflect different visualisation requirements. During the development process, we can use the DISPLAY script in conjunction with the internal representation alone to investigate the appearance of the board on the display. Notice that

the issues we are concerned with in this aspect of the design correspond closely to the low-level issues of perception in OXO-game playing, as when I ask my opponent "can you see the diagram?". In this context, it doesn't matter whether the board we depict represents a valid position in an OXO-game: a designer interested in the visual effect is free to experiment by realising the board in many different configurations, whether legal or not.

The other scripts constructing in developing the OXO-model are

- STATUS
  relating the board state to the game status;
- GEOMETRY
  recording the incidence relations between the squares and lines;
- SQVALS
  relating a square location to the advisability of playing there;
- PLAY
  selecting the best square in which to play;
- GAMESTATE
  relating the state of a board in play to player privileges.

Each such script represents a particular perspective on OXO-game playing, so that for instance someone unfamiliar with the rules of OXO might nonetheless be able to identify the incidence relations in the GEOMETRY script, and a player involved in an OXO-game would take turns according to observations recorded in the GAMESTATE script.

The complete OXO-model, as specified in outline in Listing 1 below, is synthesised from the family of definitive scripts developed from our analysis of perception and action in OXO-games. This involves integrating scripts that represent many such views into a single script. The OXO-model can be turned into a computer program by supplying the automonous actions and view management operations that are required to transform the resulting script into an OXO-game playing program. The integrated family of scripts may be regarded as supplying a model of the environment for interaction between the players; a model that is then complemented by the introduction of actions associated with (one of) the player agents. The most significant features of the design method are:

- the development of each script is a relatively simple programming process, resembling the specification of a spreadsheet through introducing defining formulae for cells and experimental redefinition of values;
- it is easy to reconstruct the environments encountered during the design process, and adapt scripts incrementally and retrospectively for debugging, in re-use, or to meet a new requirement;
- the final step of animating is trivial in as much as it involves delegating (to the computer) privileges to change the system state that the programmer can already exercise.

At present, the methods we use to represent the evolving script – based on the EDEN programming language – are relatively informal (cf [8]). This is often convenient, in as much as the program developer enjoys very strong

privileges to modify scripts at every stage of the design process, but the method has limitations. An important technical issue concerns the proper representation of the synthesis of views that make up a final specification – a task for which the Abstract Definitive Machine is our chosen model [3].

4. Agent-orientation vs object-orientation

The relationship between our approach and other methods of software development is subtle. There are several ways in which our approach can be compared with varieties of object-oriented development (cf [5]) – these are the subject of current collaborative work with the Software Development Laboratory at IBM Warwick. Some key points of comparison are considered here.

We adopt a philosophy of *programming as modelling* that was the primary motivation for object-oriented methods [4], but our emphasis is upon modelling observations of an entire system of interacting agents rather than constructing models of objects in isolation and integrating these within a message passing framework. An important advantage of this emphasis is that we can specify synchronised changes in observations that propagate across object boundaries. Even more significant is the fact that in the process of developing our models we are concerned with formulating relationships between observations that are provisional, in the same sense that the expectations generated through scientific experiment can be confounded. This means in particular that our model can be viewed as a formal specification of an object or program only after an appropriate commitment to faith in the validity and reliability of the relevant observations.

In practice, the problems of distributing the work of building large software systems have influenced the development of object-oriented techniques quite as much as concern as the *programming as modelling* perspective. In its modern usage, object-orientation is centrally linked with synthesising software systems from modules that can be independently developed. The agent-oriented method we use exhibits some of the virtues claimed for modular development in an object-oriented idiom. For instance, the explicit specification of dependencies makes it possible to carry out independent development of scripts that make up the OXO-model. It is also easy to re-use fragments of script, or to revise or replace components of the model to suit a new requirement. To illustrate this, we set up an environment in which the script that specifies the GEOMETRY component of the OXO-model can represent 3-d OXO and OXO over a classical finite projective plane. We also replaced the representation of the board as a character string in Listing 1 by a more sophisticated graphical display taken from another source – a substitution that involved no change to the other constituents of the model.

In the construction of large-scale software, the division into independent modules is recognised to be one of the most difficult tasks [6]. From an object-oriented perspective, minimising the dependency between modules is essential, because of the problems of maintaining the consistency of distributed data. Careful examination of Listing 1 reveals complex data

dependencies between the constituent scripts even in such a relatively simple model. In simulation from the OXO-model, synchronised updating of variables via definitions plays an essential role in managing this complexity. On the other hand, the rich data dependencies established by definitive scripts can make it hard to grasp the model conceptually. More work is needed to determine whether there are better ways to organise a definitive script such as Listing 1 so as to localise dependencies. Such concern is directly relevant to the issue of effective object-oriented modularisation, and also echoes concerns that have been widely expressed about tracing dependencies in large spreadsheets.

Our concept of *agent-orientation* has been developed quite independently of the large body of AI research referenced by Shoham in [9], and reflects an fundamentally different perspective. (In particular, we do not consider it appropriate to regard our concept of agent-oriented programming as a specialization of object-oriented programming [9], despite the superficial similarities.) In our framework, the concept of *agent action* is complementary to the notion of *indivisible change of system state* as expressed in definitive scripts. The development of agent-oriented modelling over definitive scripts originated with the design of the agent-oriented specification language LSD in 1986. As our discussion of the OXO-model illustrates, our work relates interaction between agents in the first instance to basic perceptions and only subsequently to more sophisticated issues of knowledge and belief. Recent unpublished work of Lam [7] strongly suggests that our approach provides an appropriate foundation for more sophisticated agent models of the kind referenced by Shoham, and leads us to propose the Abstract Definitive Machine as an appropriate computational model for an Agent-Oriented Programming paradigm [10].

References

1. Beynon, W.M., Yung, Y.P. *Definitive Interfaces as a Visualisation Mechanism*, Proc. GI'90, Canadian Info. Proc. Soc., 1990, 285-292
2. Beynon, W.M., Russ, S.B. *The Interpretation of States: a New Foundation for Computation*, Proc. PPIG'4, Loughborough Univ., 1992
3. Beynon, W.M., Slade, M.D., Yung, Y.W. *Parallel Computation in Definitive Models*, Proc. CONPAR'88, BCS Workshop Series, CUP 1989, 359-367
4. Birtwistle, G.M. et al, *Simula Begin* Lund, Studentlitteratur, 1979
5. Booch, G. *Object-Oriented Development* IEEE Trans. S.E. 12(2), 1986, 285-292
6. Brooks, F.P., Jr. *No Silver Bullet: Essence and Accidents of Software Engineering* IEEE Computer 20:4, 1987, 10-19
7. Lam, N.S. *Agent-oriented Modelling and Societies of Agents*, unpublished MSc project report, University of Warwick, Sept. 1993
8. Ness, P.E., Beynon, W.M., Yung, Y.P. *Agent-oriented Modelling for a Sailboat Simulation*, (submitted to ESDA'94, London July 1994)
9. Shoham, Y., *Agent-Oriented Programming* Report #STAN CS 90 1335, Dept of Computer Science, Stanford University, October 1990
10. Torrance, M., Viola, P., The AGENT0 Manual Report #STAN CS 91 1389, Dept of Computer Science, Stanford University, April 1991

Listing 1: An outline of the OXO-model

The variables and definitions that make up the OXO-model are as follows:

```
                                                                    GEOMETRY
allsquares is [s1,s2,s3,s4,s5,s6,s7,s8,s9]   //  list of all squares on the board
nofsquares is allsquares#                    // # denotes the length of a list
lin1 is [s1,s2,s3]                           // explicit enumeration of triples
lin2 is [s4,s5,s6]                           // that define the lines
.....
alllines is [lin1, lin2, lin3, lin4, lin5, lin6, lin7, lin8]
noflines is alllines#
linesthru1 is [lin1, lin4, lin7]             // explicit enumeration of lines through a square
linesthru2 is [lin1, lin5]
......
linesthru is [linesthru1, linesthru2, ..., linesthru9]
//geometry describes the relationships between squares that determine oxo lines
```

```
                                                                    STATUS
nofx is nofpieces(allsquares, x)
nofo is nofpieces(allsquares, o)
full is (nofo + nofx == nofsquares)
xwon is checkxwon(alllines)
owon is checkowon(alllines)
draw is !xwon && !owon && full
status is (xwon ? "X wins " :  "") || (owon ? "O wins " : "") || (draw ? "Draw " : "") || ""
// status  explains how a particular position is interpreted by an oxo watcher
```

```
                                                                    SQVALS
square is ....       // possible values are s1 / s2 / s3 / s4 /s5 / s6 / s7 /s8 / s9
availsquare is allsquares[square]==u
cursqval is sqval(linesthru[square])
// sqval() associates a value with a particular square in a particular oxo position
// sqvals evaluates squares (without lookahead) from the perspective of an oxo player
```

```
                                                                    PLAY
maxindex is findmaxindex(allsquares)
// findmaxindex determines the square with the highest evaluation in a given position
// play is derived from sqvals by "contemplating each square in turn"
```

```
                                                                    GAMESTATE
startplayer is ....          // possible values are o / x
x_to_play is (!end_of_game) && (startplayer==x  && nofo==nofx) || nofo > nofx
o_to_play is (!end_of_game) && (startplayer==o  && nofo==nofx) || nofx > nofo
end_of_game is xwon || owon || draw
// gamestate determines who (if anyone) is to move  in the current position
```