A noughts and crosses Java applet to teach programming to primary school children

J. Paul Gibson
Computer Science Department
National University of Ireland, Maynooth
Maynooth, Co. Kildare
Ireland
pgibson@cs.may.ie

ABSTRACT

We report on a continuing study into teaching programming to *pre-teens* school-children, with some as young as seven years old. As part of the study we aim to test childrens' *algorithmic understanding* through their ability to solve puzzles and play games; and to turn this understanding into working code. We review a project in which children have programmed (in Java) AI players for the game of Noughts and Crosses. This code is then incorporated into a 'programmable' Java Applet for use as an educational tool in primary (junior) schools.

1. INTRODUCTION

This paper is primarily a vehicle for supporting the following positions (views) central to our research into learning and complexity: children as young as seven years old can learn to program, Java is a good language for young children to learn to program, puzzles and games combine fun and education in an attractive problem domain for learning how to program, and Java applets provide an ideal technology for supporting the distribution of such programming case studies.

2. BACKGROUND AND RELATED WORK

2.1 Theoretical Background

Piagets theories [8] continue to be central to primary school education and its curriculum. His theory identifies a final key period in a child's life which concerns children older than 11. He argues that at this stage, and not before, children become capable of full logical and mathematical deduction.

Learning how to program is difficult. A major contribution to this difficulty is that programming relies on an implicit understanding of the concept of an algorithm. It has been suggested (by followers of Piaget - in the domain of child psychology and education) that children younger than eleven are unable to understand algorithms. This paper aims to support our argument that children as young as seven can demonstrate algorithmic understanding through writing programs. Although there is not enough empirical evidence from our small case study, this paper suggests that a larger study could lead to significant results.

We propose following Thorndike's theory of Law of Effect[11] which argues that learning is accomplished through satisfaction as goals are achieved. His research led to the idea of reinforcers and punishers in the learning process. Clearly, we have to be careful how we 'reward' children for 'good' algorithmic reasoning; and how we 'punish' them for 'poor' algorithmic reasoning. Fortunately, games provide a natural environment in which winning and losing are appropriate reward and punishment.

2.2 Java as a teaching language

Java is becoming the predominant language for teaching programming. Its success as an instructional language is directly related to its success as an object-oriented programming language in the 'real world'. There is a vast amount of on-line information concerning Java and education, including Sun's own SunEd site[9]. One of the earliest independent sites focusing on Java as a teaching language was created by Doug Lea[5]. Common arguments given for using Java as a teaching language are: it is a more pure object oriented than other languages like C++, its strong typing allows the compiler to find many problems that in other languages would result in run-time errors, its garbage collection means that programmers do not have to handle the complex task of managing memory, and in the real world Java developers are in high demand.

None of these reasons explains why we chose Java as a programming language for young children: Java code (embedded in applets) can be executed on all PCs that support browsing the web, installation of the JDK is simple enough for school teachers (and most of the children), it is freely available, we wanted to use a real programming language, we could easily put the childrens' code on the web so that their family and friends could play with it from their own homes, and the object oriented structures in Java meant it was easy to integrate the childrens' code with our own.

PPPJ 2003, 16-18 June 2003, Kilkenny City, Ireland.

It should be noted that we briefly tried out the *Karel the robot* approach to teaching Java[1]; but it did not match well with our need to test the childrens' algorithmic understanding.

2.3 Java applets as teaching tools

Java applets have been successfully deployed as teaching tools in all areas of children's education - Kahn[3] comments on the use of this and other technology with particular emphasis on the interactivity, fun and graphics. Our approach is novel in the way that the applet allows the children to incorporate their programs into the code; and allows them to see how they go from their high-level design - based on sequences of rules - to Java code.

2.4 Teaching programming skills to pre-teens children: existing languages and environments

LOGO [6, 7] was the first programming language designed for children. Using commands and programming language constructs, children programmed a turtle to move around the screen. AlgoBlock [10] and RoboLogo [4] are popular variations often found in schools. Apart from LOGO, many different languages and environments have been developed to help children program. These range from environments where children program graphical simulations to environments where they program using a video game metaphor - a typical example is ToonTalk[2]. However, none of these environments offered the advantages of working directly with Java; and our applet tool would address the disadvantages normally put forward as arguments against directly exposing young children to a real programming language.

3. APPLET REQUIREMENTS

Primarily, the applet arose out of the needs of the authors for automated teaching support when visiting schools to teach programming. After a number of schools had been visited, a pattern emerged:

1 - Identify the rules for play - Firstly, the teacher plays noughts and crosses against the children. The teacher is encouraged to 'cheat' - write over characters already played on the board, play twice in a row, place strange characters (neither an 'X' nor an 'O') in the board, continue playing after the game has been won, etc ... The children can stop the cheating only by communicating the rules of the game which forbid such behaviour. In general, it takes only a few minutes for the children to agree on an adequate set of rules.

2 - Simulate random play following the rules - The children are then told that they must play the game; but they must now identify the rules for playing well. To illustrate the notion of playing badly, the teacher simulates a number of random games and asks the children to call out when one of the players makes a bad move.

3 - Add intelligence to the players by communicating new rules - The teacher then invites the children to compete against himself. However, the children must play randomly if they have no applicable previously identified rule. This stage becomes quite competitive and many rules are identified: 'take the middle if it is free', 'win if you can win', 'block the opponent from winning', 'corners are more important than sides', etc...

4 - Illustrate the importance of ordering the rules - The children quickly identify the need for multiple rules in order to play intelligently. The teacher shows them that putting them in order is vital - for example, chosing to block a win instead of taking a win is 'bad play'.

5 - Get each child to write their own program - The children are then told to write their own player 'program' as an ordered sequence of rules. There is then a competition between the children to see who has written the best program.

6 - Introduce the children to Java - At this stage, the children are told about programming languages and how the machine does not 'speak the same language' as they have used in their programs. They must learn Java in order to build their own players to run on the computer.

7 - Show the children some rules already programmed by other children in Java - This stage is crucial to the children wanting to come back to learn how to program in Java. They must be convinced that they can - after a few weeks of programming lessons - 'speak Java' well enough to write similar programs; and they must be motivated to actually want to do it.

In large classes, it is impossible for all the kids to play against the teacher in a short space of time - and thus some children get frustrated. By encoding the teacher's play in an applet, all the children can play against him at the same time! Furthermore, there are usually too many different players - defined by the children's rules - for the teacher to play them all off against each other. By letting the kids write their programs into the applet, the children can try out as many player variations as they wish.

By keeping a library of rules previously developed by other schools, the teacher can virtually guarantee that the children cannot think of a rule which is not already coded in the system. (One unpredictable child asked to play 'in order from top left to bottom right', and this rule had never previously been asked for! This resulted in the teacher frantically coding the rule in Java and recompiling the applet. Fortunately, this is now a rare occurrence.)

The required functionality of the applet is easy to state. It must allow children to play against a number of predefined players. It must allow the children to define their own player as a sequence of rules - and be able to play against such a player. It must keep score - so that the children's competitiveness reinforces the learning of good programming. The children need to be able to chose whether X or O starts, and whether to play as X or O. The applet must prohibit cheating.

The interface must be simple enough for children to use. It must be graphical and incorporate sounds (which can be switched off). There must be a help window.

As many schools have only older versions of Java installed, we chose to write the code using the older AWT libraries. The code must be structured to facilitate: easy integration of the childrens' own Java code for the rules, easy addition

of new rules *on the fly*, and re-use in similar applets for other games and puzzles.

4. APPLET DESIGN

We chose to divide the applet frame into four components. The top left was the play area for the noughts and crosses games. For a child to make a move it is necessary only for them to click in the play area at the appropriate position. The top right is the control area where the children can chose to be X or O, can chose to play first or second, can switch sounds on/off, and can chose to play against a player from a menu of predefined AI players (including the player which they themselves have programmed - known as "Rules") The bottom right is a help window with textual messages concerning their play. The bottom left is the programming area where the children can program their own AI player as a sequence of rules. This programming is done uniquely with the mouse so that no child ever has to type at the keyboard in order ot use the applet. This design layout is fairly typical of educational applets and there is nothing novel worth commenting on.

An AI player has default unintelligent behaviour, as a random player. Children can add rules, with each new rule added to the top of the list showing that it is currently the most important. To move a rule up or down (to change its importance) it must be first selected and then swapped up or down. At any time a child can remove all rules (to return to default random play). We first allowed rules to be dragged-and-dropped, but many children had problems controlling the mouse to the degree of accuracy required - so we just provided buttons for them to `select` and `swap`. Within a few minutes (sometimes with teacher assistance) children managed to program the AI players without any notable problems.

Internally, an AI player was designed to be implemented as a linked-list of rules. Starting with the most important rules at the head of the list, each move will correspond to moving through the list until one of the rules applies. We took a design decision that an empty list of rules - in the design - should correspond to a random player, whose only intelligence is in making sure that a move is chosen randomly from all possible valid moves.(We decided to implement this by always placing a 'play randomly' rule at the end of a list of rules. In effect, the constructor for the linked-list of rules guaranteed that one could not construct an empty list.)

This design has been adopted for other games - although we are making it more general by supporting conditional branching. Adding a rule (as programmed by any child) is simply a case of defining the child's rule as a new class which extends an abstract rule class (which must provide a method for applying the rule to any given game position).

The key to our teaching programming using this tool is that the children learn to program at two different levels of abstraction (and using two different programming paradigms). The applet supports reasoning and (limited) programming using a rule-based approach. Then, as we see in the next section, the children progress to implementing the rules directly in Java. The rules can be thought of as abstract specifications, and the Java methods are the corresponding concrete implementations.

5. APPLET IMPLEMENTATION

5.1 Game data structure code

After the children have been taught about Java fundamentals they write (assisted by the teacher) their own Java code for representing the game. It is beyond the scope of this paper to report on the way in which the children learn enough Java to code up their logic rules as methods.

No two schools (or children) end up producing the same code; but we include a copy of a code fragment produced by children aged 9 years old. In this example, the code checks if a player has won the game:

```
\\ Check if 'X' has won
boolean rows = false;
boolean cols = false;
boolean diag = false;
for (int i = 0;i<3; i++){
      if ((board[i][0] == 'X')
         && (board[i][1] == 'X')
         && (board[i][2] == 'X'))
        rows=true;
      if ((board[0][i] == 'X')
         && (board[1][i] == 'X')
         && (board[2][i] == 'X'))
        cols=true;
    }
if ( ((board[0][0]=='X')
     && (board[1][1] == 'X')
     && (board[2][2] == 'X'))   ||
     ((board[2][0]=='X')
     && (board[1][1] == 'X')
     && (board[0][2] == 'X'))
   ) diag = true;
if (rows || cols || diag)
    System.out.println("X has won");
```

The code largely speaks for itself - the children have managed to correctly use a good mix of Java constructs in a manner which makes it clear that they have some algorithmic understanding of the game logic.

5.2 Putting the rules together

It should be noted that we do not expect the children (or their teachers) to be able to put the rules into the applet. This is currently done by the author. However, it is hoped that this construction process will be automated in the near future. The current composition mechanism, based on the use of abstract classes, is introduced below.

`XORule` is an abstract class with each 'rule' subclass having to implement the `apply` method: if the rule applies then play continues according to the rule and returns true; otherwise false is returned.

```
abstract class XORule {
 public abstract boolean apply(XOGame game,char ch);
           \\ ..
     }
```

Now we show how a concrete rule - playing in the middle - can be added to the system:

```
class XOMiddlePlay extends XORule{
public boolean apply(XOGame game, char ch){
  if (game.charAt(1,1)!=' ')
       return false;
  if (ch=='X')
       game.playX(1,1);
  else game.playO(1,1);
return true;
}
```

The code for putting the rules into sequence is a straightforward linked-list structure with additional methods for selecting a rule in sequence and for swapping it for rules on either side.

6. CONCLUSIONS AND FUTURE WORK

The noughts and crosses applet has shown us how Java itself is a useful tool for teaching about Java programming in schools. Children get to see two different types of programming - using rules and imperatively (using classes and objects). The applet structure is being re-used for other games, like connect-4 and draughts.

Recently, the internal object oriented design of the applet has been used as a case study as part of a second year undergraduate course. Students have been asked to re-engineer the code so that it better meets the original requirements. We hope to be able to report on this in the near future.

7. ACKNOWLEDGEMENTS

The authors would like to thank all the schools that have assisted in this case study. In particular, we mention the good-will of the teachers and the trust they placed in us through supporting our work - organising a group of preteens children is a key, time-consuming, skill which is not built into our Java applet (and is unlikely to be a requirement that can be met in the immediate future).

8. REFERENCES

[1] J. Bergin. "karel j. robot". http://csis.pace.edu/ bergin/KarelJava2ed/Karel

[2] K. Kahn. Toontalk - an animated programming environment for children. *Visual Languages and Computing*, 1996.

[3] K. Kahn. *Helping children to learn hard things: Computer programming with familiar objects and actions*. Morgan Kaufmann, 1998.

[4] M. Kamvysselis, J. Lueck, and C. Rohrs. "robologo: Teaching children how to program interactive robots". http://web.mit.edu/manoli/robologo/www/robologo.htm.

[5] D. Lea. "some questions and answers about using java in computer science curricula". http://g.oswego.edu/dl/html/javaInCS.html.

[6] C. Maddux. "the need for science versus passion in educational computing". *Computers in Schools*, 2(2-3):910, 1985.

[7] S. Papert. *Mindstorms - Children, Computers, and Powerful Ideas*. Harvester Press, 1980.

[8] J. Piaget. *The Jean Piaget Bibliography*. Jean Piaget Archives Foundation, 1989. ISBN:288288012X.

[9] Sun. "sun microsystems training, distance learning and educational online courses". http://suned.sun.com/.

[10] H. Suzuki and H. Kato. Algoblock: A tangible programming language – a tool for collaborative learning. In *The 4th European Logo Conference*, pages 297–303, 1993.

[11] E. Thorndike. *The Measurement of Intelligence*. New York: Teachers College Press, 1927.