
Turtle competitions in MicroWorlds EX

Sergei Soprunov, *logo@int-com.ru*

Logo Team, Institute of New Technologies in Education, Moscow, Russia
Dorodnicyn Computing Centre, Russian Academy of Sciences, Moscow, Russia

Elena Yakovleva, *logo@int-com*

Logo Team, Institute of New Technologies in Education, Moscow, Russia

Abstract

Creating games of all kinds is a traditional genre of Logo-programming, one of the most fascinating, motivating and important in terms of development of problem-solving skills. (Papert, 1996). The most challenging and most enlightening programming experience is provided by the situations when there is no explicit “best” or winning strategy or describing such strategy is too complicated. (Harvey, 1997). In such cases the programmers are constrained by the necessity to implement just a “good” algorithm instead of the “perfect” one. This gives a very good opportunity for comparing algorithms (more precisely, the programs implementing them) by making the programs to compete, just like it happens between LEGO robots.

Logo and MicroWorlds EX in particular presents a very natural environment for such competitions. Each participant creates a Logo turtle “trained” to follow its own strategy to play a particular game. These turtles are put then in a specially created environment where they compete “on their own”, without human intervention.

The latest version of MicroWorlds, called MicroWorlds EX, has several features that make creation the environments for competitions easy and natural. First, it is the idea of a turtle as an “all-sufficient” object. Each turtle has its own so-called “backpack” – a collection of shapes, properties, rules, procedures, media, etc. - all of its personal characteristics and knowledge. The turtle can be saved as a file along with all its “belongings”. Second, it is the new easy way of event-driven programming. It is implemented in the form of the list of rules kept in the turtle’s backpack. And the last but not least is turtles’ ability to communicate. Any turtle can send a message to a particular turtle in the project. Besides the text of the message, the message contains the “signature” of the sender, so the recipient always “knows” whose message is this. This feature allows to easily organise communications between the participants of the game. “Players”, “referee” and others just send each other messages and properly react to them.

From the other hand MicroWorlds EX lacks the important feature that would be extremely useful for competition projects: primitives regulating the access to the turtles, their procedures, rules and other “belongings”. Such kind of competition may provoke a human-competitor to use some kind of “hacker” tricks: try to control not only his own turtle but also the opponent’s ones and so on. We decided to enrich MicroWorlds EX with commonly practised public/private structure. A turtle can have some primitives and procedures declared public - just like in standard MicroWorlds EX, so everybody else in the project can ask this turtle to execute them. The primitives and procedures that are declared private for this turtle can only be used in this turtle’s backpack. They are not executed when called from outside the turtle.

To present the very structure of the competitions projects a very simple game of Tic-Tac-Toe is described in the paper.

Keywords

Competition, MicroWorlds EX, programming, private procedures, public procedures

Introduction

Have you ever seen robot races or other robot competitions? Here's how it happens.

Each robot is built and programmed by a human-competitor to perform a particular mission. In the course of the contest, people do not control their models, the robots act on their own. People just bring their creatures to the start position, turn them on, and step aside. Every player hopes his creature is smart enough to find a way to the finish and to make it there first. For a programmers competition between virtual creatures see, for example, <http://www.windowsforms.net/Terrarium/WhatIsTerrarium>

Why not have similar competitions for programmers between Logo turtles instead of robots. Relative to MicroWorlds, the idea of cooperative "living" the turtles created by different people in a common environment was expressed by E.Kabakov.

Enriched version of MicroWorlds EX

The latest version of MicroWorlds, called MicroWorlds EX, has some features that make creating the environments for competitions easy and natural.

1. We would like to mention three features. First, it is the idea of a turtle as an "all-sufficient" object. The turtle, which in previous versions of Logo already was "the kingpin", "the star" of the program, becomes in MicroWorlds EX an object that can live independently of the project. Now, each Logo-turtle in the project has its own so-called "backpack" – a collection of shapes, properties, rules, procedures, media, etc. - all of its personal characteristics, knowledge and belongings.

You can save a turtle as a file and, for example, e-mail it. When you get this "fish back into the water" (in a project), it will continue living its life on the new place.

2. Another MicroWorlds EX feature that appeared rather convenient for our purposes is the new easy way of event-driven programming. Each MicroWorlds EX turtle "carries" in its backpack a list of rules. A rule consists of two parts: the description of an event and the list of instructions to run in case the event happens. There's a number of pre-set types of events that happen in turtle's life most often and you can teach it to react to. They are called:

`onclick` – the turtle runs its instruction when clicked by a mouse

`oncolor` – ... when comes across a particular background color

`ontick` – the turtle runs its instruction repeatedly after a preset time interval

`ontouching` – ... when it collides with another turtle

`onmessage` – ... when it gets a "message".

Besides that, you can describe as many events and turtle reactions to them as you wish – and all this is saved in turtle's backpack. So when you place the turtle in another project, it already "knows" how to act in different "situations" and starts acting immediately.

3. And one more brand-new MicroWorlds EX feature that we'd like to mention. It is turtle's ability to communicate. Any turtle can send a message to a particular turtle in the project. The primitive used for that is `tell recipient message-text`. The message contains the "signature" of the sender, so the recipient knows the sender's name.

From the other hand MicroWorlds EX lacks the important feature that is useful for competition projects. In such projects we need to create turtles that not only are complicated but also have a restricted access to them. For example, in the case of football game the turtle playing the role of the ball has to accept rather restricted set of the commands. Otherwise a turtle-player can order the ball to move to an arbitrary place of the game field – for instance, directly score a goal! In the case of the chess game a player shouldn't answer the question "What is your move in this position?" if it's asked by the opponent, but has to answer if it's asked by the referee.

Of course, it's possible to rely on competitor's honesty or check his or her programs manually to make sure that nobody's cheating, but we are playing another "type of a game" – we'd like to have an environment where everything happens automatically. So we decided to enrich MicroWorlds EX with commonly practiced public/private structure. A turtle can have some primitives and procedures declared public - just like in standard MicroWorlds EX, so everybody else in the project can ask this turtle to execute them. The primitives and procedures that are declared private for this turtle can only be used in this turtle's backpack. They are not executed when called from outside the turtle.

We implemented two dual primitives `private` and `public` to set a list of private/public commands and reporters for a current turtle. For example after execution

```
turtle1, public [shape pos]
```

everybody is allowed to ask `turtle1` about its current shape and position, but all other requests will be rejected. Nobody in the project would be able to order `turtle1`, for example, to move or to change its shape.

Now everything is ready to develop the environment for the type of competitions that you like: races, football game, going through the labyrinth – just about anything.

Sample competitions: Tic-Tac-Toe

Let us consider as an example the game of Tic-Tac-toe. Actually, this game is not extremely interesting for organising real turtle competition. We've picked it for an example just because it is quite simple and "good for show".

For real turtle competitions better suite the games that do not have obvious winning strategy or this strategy is hard to program. In this case, actual competition is possible. There's no predetermined winner like in the game of Nim. The players could explore the strategies that are not "best" – and find out experimentally which of them are "better" and which are "worse".

Tic-Tac-Toe is not like that but the main features in the implementation of this game are the same as for more complicated and profound games.

Just like in the robotics competitions, there should be a "game field" and a set of game rules. The field for a game in our case is actually a Logo project, prepared in particular way. For the game of tic-tac-toe, the project includes the 3 by 3 board for placing "ouths" and "crosses" and some kind of "creature" which will supervise the game. This will be, of course, a turtle. We'll call it referee.

The referee would control the game process:

- give a turtle-player a command to make a move,
- get player's move
- check if the move is legal
- and if it is, the referee would accomplish the player's move by "drawing" an X or an O on the board.

The referee communicates with the players via messages, as described above. There is no use in having public commands/reporters for the referee, so all referee's primitives and procedures are private. We launch the game by clicking the turtle-referee, the game is over when the referee announces somebody's win or a tie.

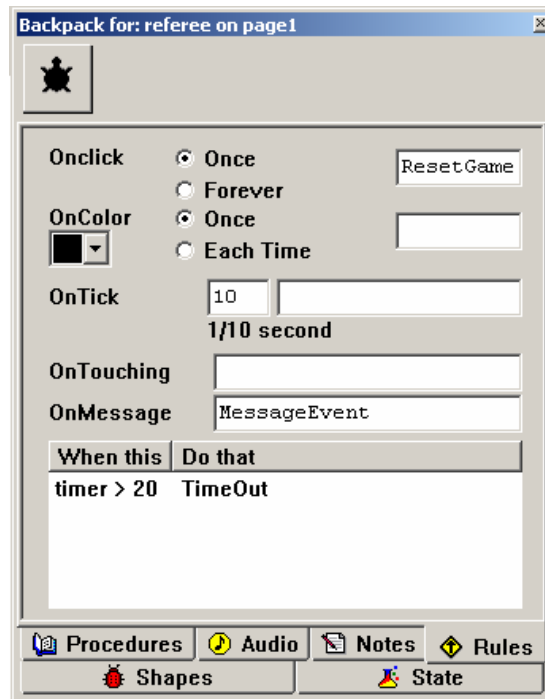


Fig1. The Rules tab of the Tic-Tac-Toe Referee's backpack.

There could be different ways to represent the Tic-Tac-Toe board. We use 9 turtles with the names "cell11", "cell12", ..., "cell33" for the board cells. Each cell can have one of the 3 shapes: **X**, **O** or empty. Everybody in the project can "see" what shape the cell wears, so the `shape` reporter is public for every turtle-cell. From the other hand we don't want to allow a player to directly change cell's shape, so all other primitives (including `setshape`) and procedures are private for the turtle-cells. Only referee can (by sending a message) give the cell the order to change a shape.

Certainly the players (humans) may know the full content of the turtle-referee and the cells procedures but can't change them. Though all these procedures are very simple and straightforward we cite the fragment of the referee instance to make the description a bit clearer.

On the **Fig1.** and **Fig2.** you can see two tabs of the referee's backpack.

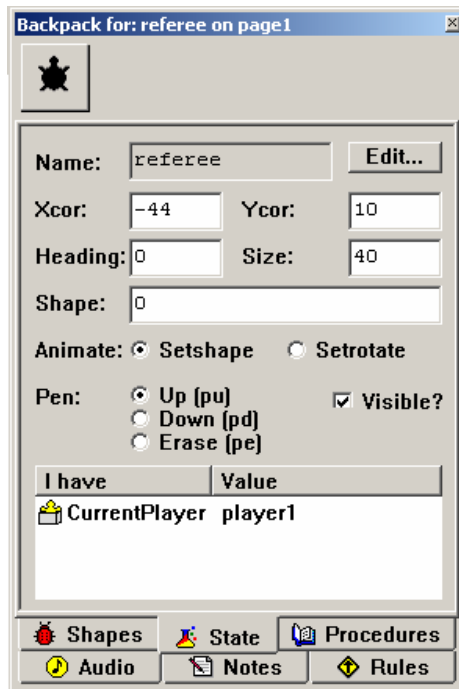


Fig2. The State tab of the Tic-Tac-Toe Referee's backpack.

The referee watches for two events: one predefined (getting a message) and one user defined (communication timeout) (see **Fig1**). The private variable `CurrentPlayer` keeps track of the turns (see **Fig2**).

Here is the fragment of the referee's backpack Procedures tab:

```

to ResetGame
CleanBoard ;CleanBoard sets all cells' shape to "empty"
SetCurrentPlayer "player1
AskForMove
end

to AskForMove
tell CurrentPlayer "your_turn
resett ;this command resets timer
end

to MessageEvent
if not equal? sender CurrentPlayer
    [announce "|The turn is not yours!| stop]
let [cell message]

```

```
ifelse CorrectMove? :cell ;CorrectMove? reports if the move is legal
  [tell :cell ResetShape
    if GameOver? ;GameOver? reports if the game is over
      [announce "|The Game is over| stopall]]
  [announce "|This move is not legal.|]
SetCurrentPlayer ifelse equal? CurrentPlayer
  "player1 ["player2]["player1]
AskForMove
end

to ResetShape
op ifelse equal? CurrentPlayer "player1 ["X"]["O]
end

to TimeOut
  announce (se CurrentPlayer "| seems to fall asleep. His opponent
  wins. The game is over.|)
stopall
end
```

The text of three procedures is missing here: `CleanBoard`, `CorrectMove?`, and `GameOver?`

The latter two are reporters:

`CorrectMove?` reports `false` if the player tries to make a move in the cell that is already busy and `true` in all other cases.

`GameOver?` reports `true` when somebody wins (three X of O in a row) or if all cells on the board are already busy.

The content of the `turtle-player` is almost completely at the programmer's discretion. The player could use just any strategy for playing the game. One, simple strategy, is to put an **X** or an **O** to a randomly chosen out of the still empty positions. Another strategy is to get each cell to have some "weight" and pick an available position with highest weight. There are other strategies possible. There is just one restriction for player's procedures, and it's about the ability to exchange messages with the referee:

- first, the player should know how to accept the referee's message informing that it's time to make a move and
- second, it should be able to tell the referee what empty cell it would like to change to X (or O) on the current move. In our model, this should also be done by sending a message.

As we already said above, Tic-Tac-Toe on the 3 by 3 board is not very challenging game. For real programmers competitions it's much more interesting to use more complicated games, like turtle races, turtle football, or if nothing else Tic-Tac-Toe on the unlimited- size board. "Game

fields” for these competitions can be created in a way, rather similar to a presented Tic-Tac-Toe realisation.

References

Papert, S. (1996) *The Connected Family*, Longstreet Press, Atlanta, Georgia, pp. 146–151.

Harvey, B. (1997) *Computer Science Logo Style*, 2nd Edition, Volume 3: *Beyond Programming*, MIT Press