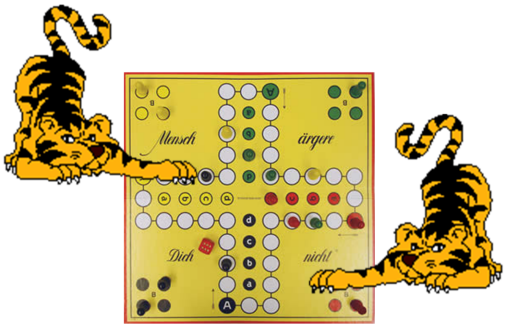
Tiger Plays Ludo

Contribution to the Tool Contest Session of AGTIVE 2007

Enrico Biermann, Claudia Ermel
Technische Universität Berlin, Germany

{enrico,lieske}@cs.tu-berlin.de



1 Overview

The TIGER project (transformation-based generation of environments) [Tig05] aims at generating visual editors as plug-ins for ECLIPSE from a visual language specification which is based on a typed graph grammar. TIGER allows a language designer to define concrete visual representations for language elements and to use this concrete syntax to define the syntax rules.

The idea of taking part in a simulation tool contest using an editor generation tool has its roots in our wish to present the visualization powers of TIGER, both in the grammar modeling phase and in the use of the defined rules as user operations in the generated editor. Since editor usage is highly interactive (i.e. each editor operation is an action evoked by the user), our tool contest contribution does not qualify for a benchmark contest. Up to now, TIGER does not provide means to control rule applications. For each rule application, the user has to select a rule from the editor palette, and, if required, to select match objects in the editor panel. This behaviour is of course sufficient for generated editors but not for simulations where often more than one rule have to be applied to perform an atomic action. In our contribution, we try to specify a minimum of available rules. Each rule represents a phase in the game, like selecting the first player, throwing the die, moving forward, or kicking out another player's token. Our aim is to make the grammar as deterministic as possible. The only choices we allow the player to make are: who will be the first player of the game, which token shall move (in case there are more than one token of the player's colour on the field), and which token shall go to the start place (if a six has been thrown). Hence, strategies are interactive user decisions modeled by different applicable rules. In the cases where there is no choice left, only one rule will be applicable to go on with the game. Due to TIGER's nature, this rule still has to be selected from the palette instead of being applied automatically.

Fig. 1 shows the start graph in the editor panel with all tokens in their initial homes. The four coloured fields around the die control the turn of the players. The editor palette to the left shows the available rules for playing.

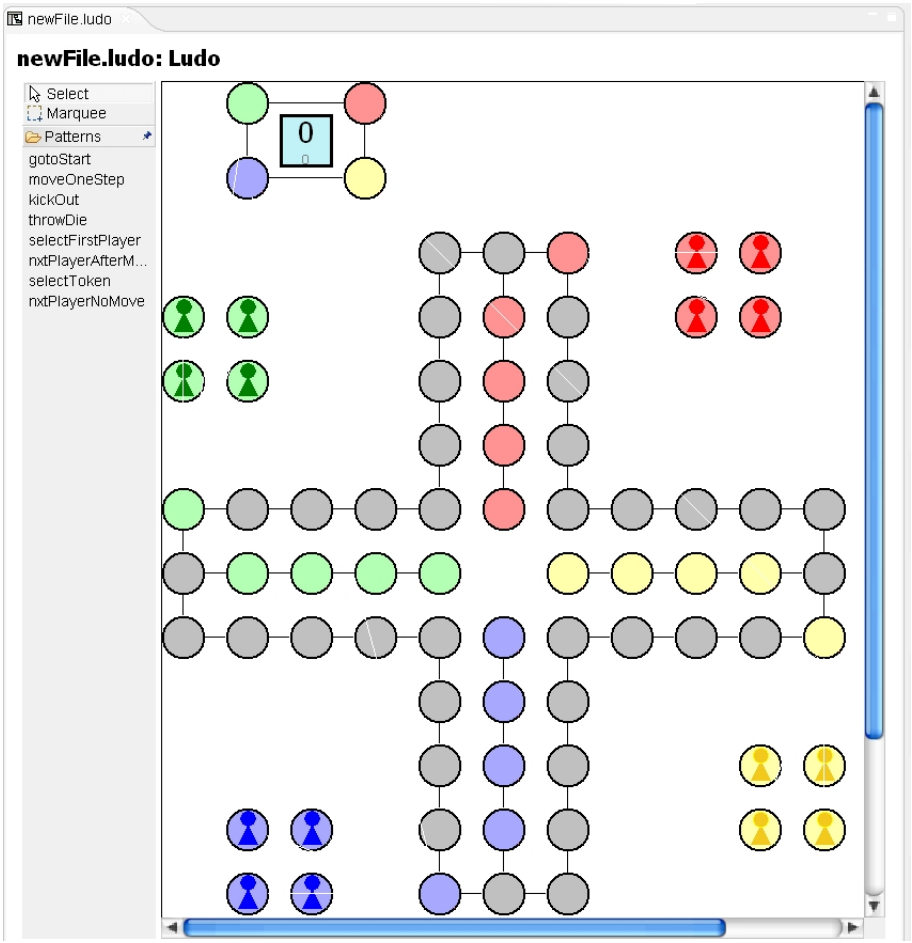


Figure 1: Startgraph of the Ludo game in the TIGER generated editor

2 Meta-Model and Concrete Syntax

The left window in Fig. 2 shows the meta-model for Ludo in TIGER, modeled as attributed type graph with inheritance.

The main symbol types are Die, Token and Field. A Token lives on a Home field, where it returns when it has been kicked out. To highlight the active player in the game control panel, we have the symbol type ActivePlayer which is a red circle drawn around the current ActiveField. Analogously, we highlight the active token by drawing a pink circle of type TokenMarker around the selected Token. Fields are connected via next-edges. Symbols of type Color are just colored overlay circles which are used to color fields of type Start (the start fields of each player), Target (the four target fields of each player) and the ActiveFields in the control panel. The normal fields where tokens move around the game, are not coloured.

In addition to the meta-model, the visual appearance of the figures and connections corresponding to the symbol and edge types can be defined in the corresponding layout editors and their property views. The figures for blueToken and Die symbols and the connection for the next edge are shown in their concrete syntax editors in the right window of Fig. 2.

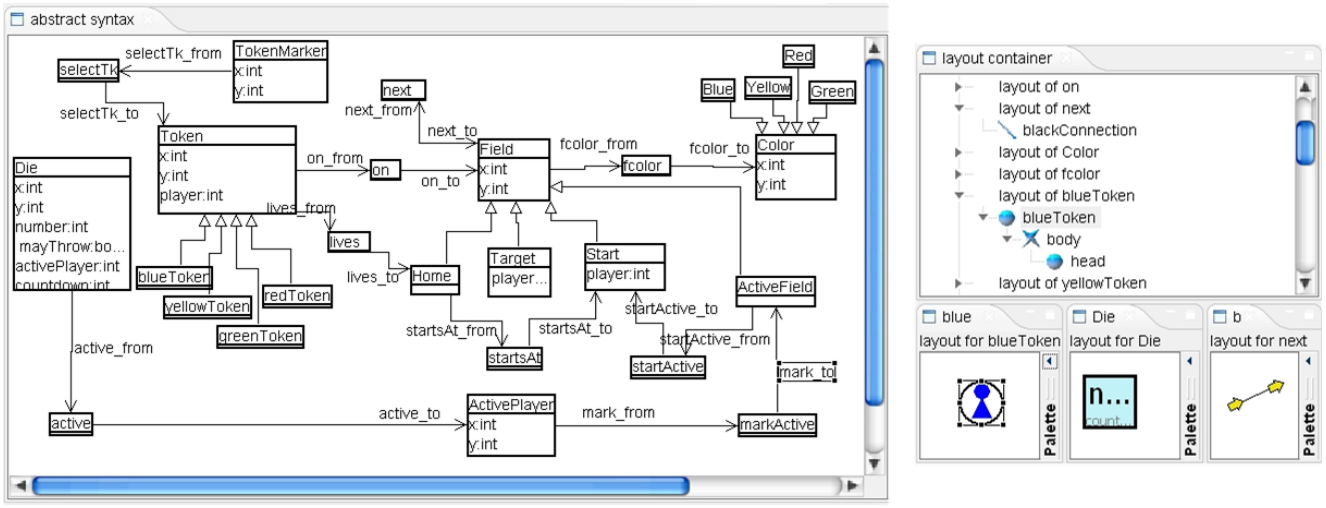


Figure 2: Meta-Model and Concrete Syntax Definition for Ludo in TIGER

3 Rules

The graph rules modelling the Ludo game are meant to be applied as follows: After selecting the first player of the game using rule selectFirstPlayer, the die is thrown applying rule throwDie. Depending on the number of the die, either the active player may go to the start place, if a six has been thrown (rule gotoStart), or he can select one of his tokens which are already on a field (rule selectToken). After a token has been set to the start field, it is automatically selected, and the player may throw the die again. When a token is selected, the token can be moved by applying rule moveOneStep. This rule has to be applied the same number of times as the number on the die. If another player's token is at the last field of the active player's move, the other player's token is kicked out using rule kickOut and returns to its home place. When the move has been completed (or the player cannot move at all), it's the next players turn which is set by rules nxtPlayerAfterMove or nxtPlayerNoMove.

In the following, we explain the rules and their application conditions in more detail. Rule selectFirstPlayer (Fig. 3) requires the user to click on one of the coloured control fields around the die. (In Fig. 3, this is the violet circle marked by in=0.) This field is highlighted by a red ring, and the die gets the number of the selected active player in its attribute activePlayer. This rule can be applied only once in the beginning, since the NAC noActivePlayer forbids its application after the red marker has been introduced to the game. From now on, this red marker is only moved to the next player, but never deleted.

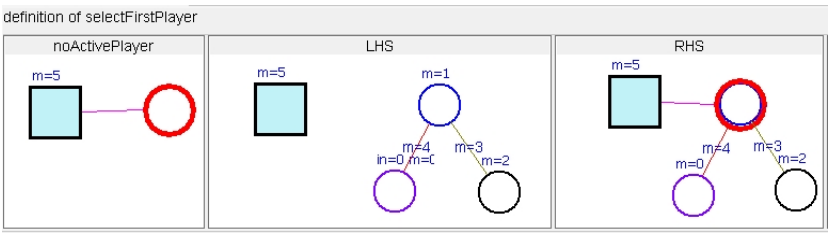


Figure 3: Rule selectFirstPlayer

Rule throwDie (Fig. 4) realizes the throwing of the die, where the resulting number is a random number between 1 and 6, realized by a call of a Java method from the Die attribute number.

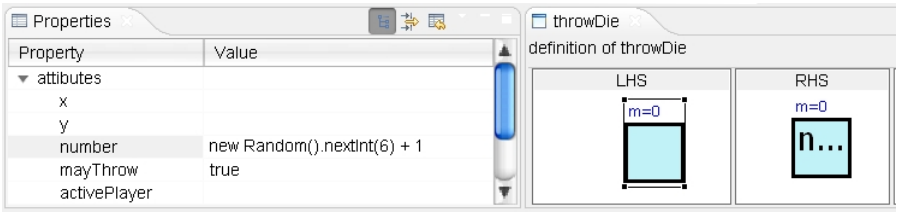


Figure 4: Rule throwDie

Rule gotoStart (Fig. 5) models how a token is moved from its home place to the corresponding start place of its colour, provided that a six has been thrown and the die has the correct activePlayer number. Note how the new position for the token in the right-hand side of the rule is taken from the position of the start place. This rule has two NACS: the first one forbids the application if this token is already out on a field, and the second one (not shown) forbids the existence of another family token on the start place.

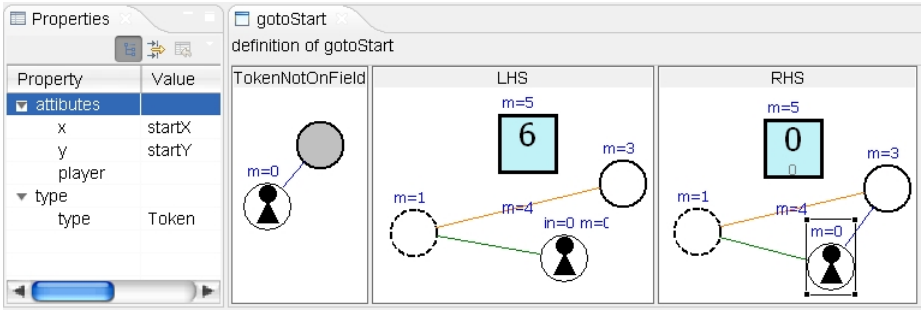


Figure 5: Rule gotoStart

Rule selectToken (Fig. 6) models how a token is selected for moving, if it belongs to the current active player (Token attribute player) and if there is not already a selected token. Again, the position of the selection marker is taken from the x and y coordinates of the token (tkX and tkY).

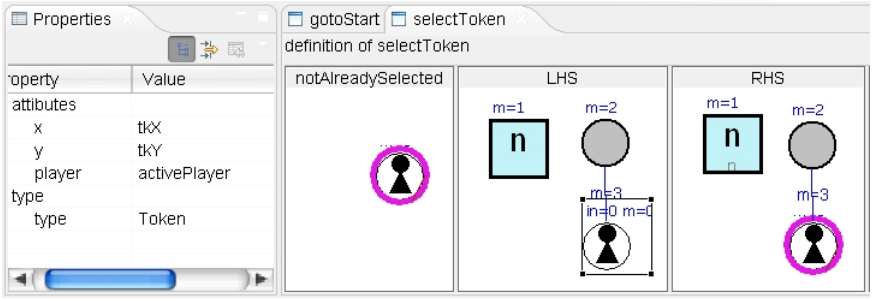


Figure 6: Rule selectToken

The most important rule is rule moveOneStep (Fig. 7). We decided to model token movements stepwise because it's nicer to see a token stepping forward instead of doing one big step. Hence, the rule has a countdown variable which decrements the die number in each step. Moreover, rule moveOneStep has six NACs, forbidding that other tokens of the same colour are on the last field of the move. Another NAC forbids that the next field is the start field of the active player, and yet another one forbids that the next field is the Target field of another player.

Moreover, one NAC stops this rule application when the countdown has reached zero. The boolean Die attribute mayThrow is set to true if the die shows a six, and to false, otherwise. So, after a six has been thrown, the active player may throw again.

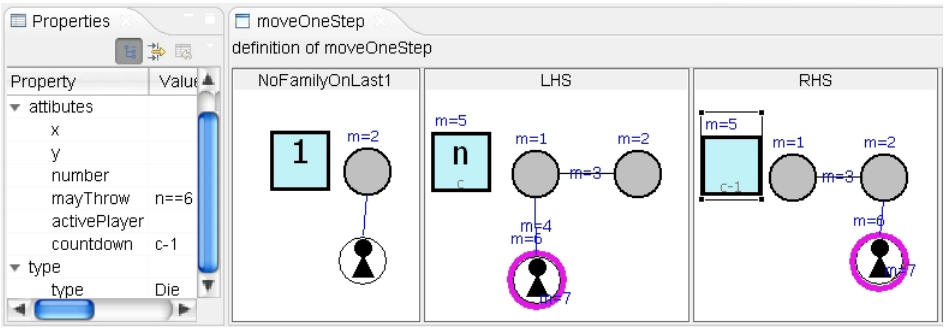


Figure 7: Rule moveOneStep

After a move has been completed, it may be the case that there is a token from another player at the same field as the token of the active player. In this case, the token from the other player may be kicked out by the active player, using rule kickOut in Fig. 8. The kicked-out token gets its as new position the position of its home field.

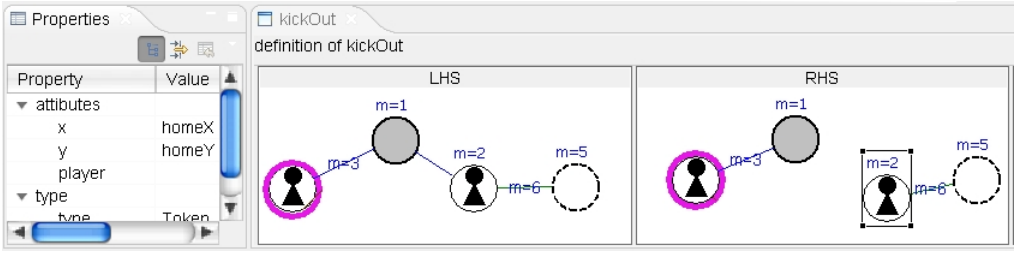


Figure 8: Rule kickOut

Last but not least, two further rules give the control of the game to the next player. Rule nxtPlayerAfterMove (Fig. 9) deals with the situation that a token has been moved previously, and is still selected.

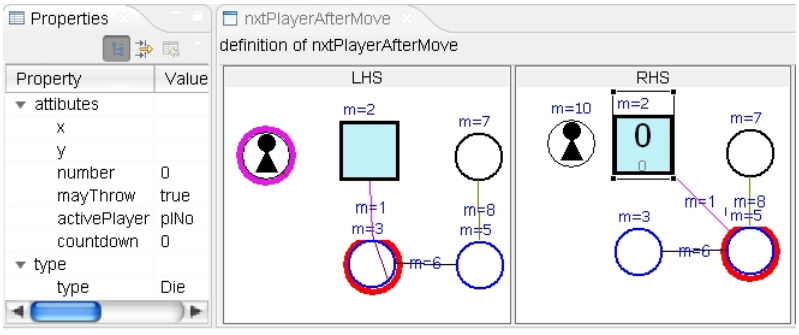


Figure 9: Rule nxtPlayerAfterMove

The other rule nxtPlayerNoMove (not depicted) is very similar, except that in its left-hand side there is no selected token. Instead, a NAC forbids its application if there is a selected token somewhere on the field. This rule

is applicable in the case that the previous active player could not use the thrown number to make a move or to go to his start place (e.g. all his four tokens are in their homes and the active player throws a three).

4 Conclusion: Strong and Weak Points

The strong point of our approach to model Ludo using TIGER is the straightforward visual way to define a concrete visualization for model elements in the TIGER designer. Here, no code has to be written by hand, and yet, a nice ECLIPSE plug-in is obtained, where model visualization is realized in a timely fashion using the ECLIPSE Graphical Editor Framework GEF [GEF06].

As mentioned before, TIGER is an editor generator and hence provides no means to control rule application, and to apply more than one rule without user interaction. Certainly, for a simulation case study as Ludo, this is not adequate. As future work we plan to extend the TIGER designer in a way that the modeler can also define structured units of transformation rules. Moreover, strategies like "*apply all rules from a particular set of rules as long as possible* or "*..as long as a specified condition holds*" should also be an option. This would offer an alternative to manual rule selection which is the basis to generate not only visual editors but more sophisticated environments for editing, simulation and model transformation.

From a gaming point of view, the palette then should offer only user actions like "take a new card", but the deterministic actions of the player would be encapsulated, thus also allowing the computer to take the role of one or more players.

References

[GEF06] Eclipse Consortium. *Eclipse Graphical Editing Framework (GEF) – Version 3.2*, 2006. http://www.eclipse.org/gef.

[Tig05] Tiger Project Team, Technical University of Berlin. *Tiger: Generating Visual Environments in Eclipse*, 2005. http://www.tfs.cs.tu-berlin.de/tigerprj.