The Formal Approach to Computer Game Rule Development Automation

Elena A. Pavlova

***Abstract*—Computer game rules development is one of the weakly automated tasks in game development. This paper gives an overview of the ongoing research project which deals with automation of rules development for turn-based strategy computer games. Rules are the basic elements of these games. This paper proposes a new approach to automation including visual formal rules model creation, model verification and model-based code generation.**

***Index Terms*—Automation, Games, Formal Languages, Software Verification and Validation**

I. Introduction

COMPUTER games are one of the most dynamic and rapidly evolving fields of information technology. Games are widely used in entertainment, education and training of personnel [1]. Still the percentage of successful projects in computer game industry is very low [2]. Low level of automation is one of the reasons of the problem. Game rule development is one of the weakly automated tasks. Game rule development includes rule design, rule-based code and data generation and results verification [3].

In this paper we define game rules as the definition of a game world entities, entity interaction rules, the main goal of the game, secondary goals, start conditions, winning conditions and a player state definition.

There is a special role of game designer in a game development team [3, 4], who is responsible for game rules design. She frequently has no technical background. In order to avoid confusion between a game designer and a software designer roles we'll use hereinafter the term "designer" for a game designer.

A game rules definition usually consists of several large text documents and a set of tables. Designers use text editing tools and spreadsheets as automation tools. The typical process of a game rules definition is shown in Figure 1

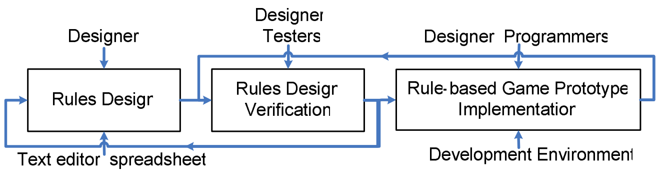

Fig.1. Computer game rule development process.

Voluminous (up to several thousand pages [3]) and ill-structured rules definitions are difficult and time-consuming to create, modify and maintain. An informal rules formulation complicates rules development automation, automatic rule-based code and data generation and rules verification, particularly rules balance checking and balancing (balance problem will be discussed further).

Some designers try to master general-purpose modeling and programming languages and corresponding modeling environments to solve the automation problem (this approach is described in detail in [3, 4]).

Special-purpose game-oriented development environments are used as an alternative (see for example [5-9], Torque Engine Advanced (www.garagegames.com/products/torque/tgea/features/), FPS Creator (www.darkgamestudio.com), NeoAxis Engine (www.neoaxisgroup.com), Offset Engine (www.projectoffset.com/game.html), Unreal Engine (www.unrealtechnology.com), C4 Engine (www.terathon.com/c4engine)). These tools frequently need a complex customization and add-in programming. The majority of tools don't allow for game genre-specific development, thus loosing long reusable experience. The rules definition is mixed with the definition of the graphics of the game, and in some cases with the definition of game artificial intelligence elements. Thus, it is difficult to modify, analyze, verify and reuse game rules independently. The rapid game rules executable prototype generation becomes almost impossible.

The tools under discussion don't allow automatic rules verification. Design-time errors could be caught at the implementation stage or later. These errors are usually treated by a manual rule definition documents review and tedious testing [3, 4].

The above demonstrates the urgency of development of the new approach to game rules automation. The goal of this paper is to develop an approach allowing automation of the whole game rules development cycle – from formal visual rule design, through rule verification, to rule-based code and data

Manuscript received March 14, 2008. E. A. Pavlova is PhD student at the Cybernetics Department, Moscow Engineering-Physics Institute (National Nuclear Research University), Moscow, Russia (phone: +7-906-723-1189; e-mail: elena.pav@gmail.com).

The research is supervised by M. V. Sergievsky, PhD, associate professor at the Cybernetics Department, Moscow Engineering-Physics Institute (National Nuclear Research University), Moscow, Russia (e-mail: maxim.sergievsky@light-site.ru).

generation. The approach should consider the domain-specific experience of the particular game genre. We took the turn-based strategy (TBS) game genre [3] for our research as game rules are the critical part for a game of this genre. The key task in TBS game development is game rules development [4].

As far as the author of this paper is concerned the creation of the rule development environment supporting visual game rule representation as a single formal model, automatic formal verification of this model and model-based automatic code and data generation is a new approach to game rules development automation. Formal domain-specific languages weren't used for TBS game rules definition before. Static analysis and formal property monitoring weren't used for computer game rules verification.

II. Formal Background

A. The Turn-Based Strategy Game Rule Description Language

The developed language is domain-specific (DSL) [10], i.e. it considers a specificity of turn-based strategy computer games. The language allows entity definition for the problem domain of the concrete game (notably entity data and behavior); behavior constraints; entity relationships and reactions of entities to the other entity behavior.

Entities are represented as objects in the language. Entity data correspond to object properties, and behaviors correspond to methods. Object orientation allowed more flexibility and simplicity compared to class orientation. The language type system was developed. The language type system includes simple types for evaluation of constraints and game genre-specific types (types for the turn-based strategy domain). The type system provided the necessary level of abstraction and allowed to separate the object specification from the implementation. Types are also used for model error detection.

The language is prototype-based [12]. The new (clone) object maintains the independent copy of properties, methods and the link to the initial (prototype) object. An object may have only one prototype object. The modification of prototype object doesn't influence the clone object and vice versa. The main object modification method is property and method update. The specified prototyping mechanism allowed object elements reuse and seems to be the natural object-creation mechanism for the turn-based strategy games domain.

The language syntax was formally defined using a context-free LL(1) grammar [13]. Both textual and graphical notations for the language are available. We defined the formal denotational semantics [13] for the language, using the $\lambda\varsigma$ – calculus [12] for denotats and elements of Hoare logic [13] for precondition and postcondition behavior constraints. Conditions described in Hoare logic are used for model consistency checking and correct method invocation planning in game scenarios not for verification by deduction analysis. Entities react to the behavior (i.e. method invocation) of other entities by means of the special methods called reactors. Several reactors may be attached to one method. Reactor execution changes the state of the game. Reactors' preconditions and postconditions depend on the game state. Thus the reactors invocation order is important. Reactors' invocation algorithm considers reactors' preconditions and postconditions to execute as much reactors as possible. The developed language is described in detail in [11].

B. Turn-Based Strategy Game Rule Verification

Design-time game rule verification allows incorrect rule detection and following correction prior to implementation. This paper defines a correct rules model as consistent and balanced. The rules model consistency is defined as syntactic and semantic interface consistency of objects constituting the model. Model consistency guarantees the correct interaction of objects. The applied consistency checking method is fully described in [14].

Balance is a game domain-specific concept. In general it means rules fairness [3]. The rules of a particular game are fair if the player success depends only on his abilities. Concrete definitions of balance are given in [3, 4]. The example of a misbalanced game is a game having unequal start conditions for players, giving one player the advantage allowing winning no matter what other players do. Rules balance verification is a key task in TBS game development [3]. In this paper rules are considered to be balanced if none of the competing sides defined by the rules has an advantage; there are no invincible troops and the result of the game is independent of who moved first. In this paper rules balance is verified by means of formal properties monitoring [15].

III. The Prototype Rules Development Environment

The architecture of the designed rule development environment is considered in this section. The architecture model is illustrated in Figure 2. The arrows connecting model elements represent dataflows. The rules development environment consists of the graphical user interface allowing visual game rules model creation, the rules verification tool, the rules translator and the data (rules models) storage.

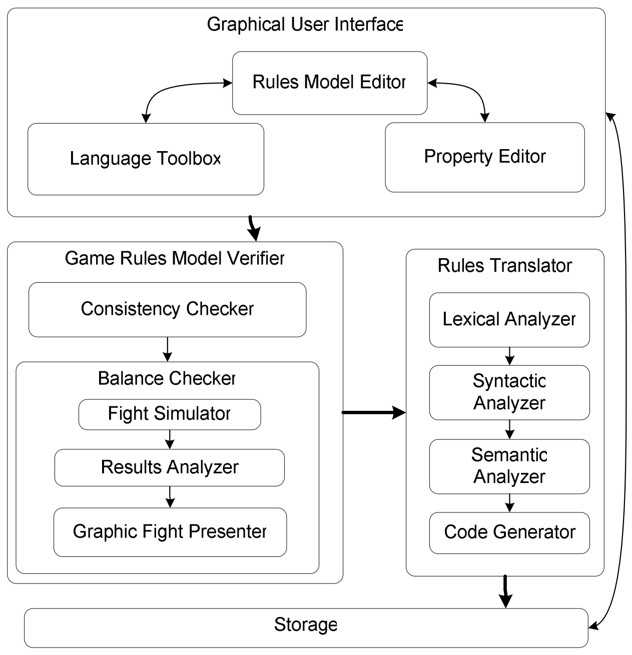

Fig. 2. The rules development environment architecture.

The graphical user interface enables visual rules definition in the rules model editor. Rules are defined using the graphical language notation. The example screenshot of the prototype rules development environment is shown in Figure 3.

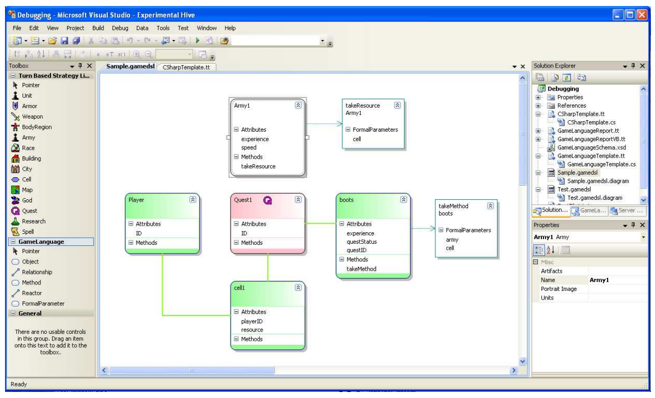

Fig. 3. The rules development environment and the example rule model.

The toolboxes contain all necessary language graphical primitives. Model element properties could be customized in the property editor. The model description is stored in the textual language notation.

The verification tool checks for rules model consistency and balance using methods defined in the section I.B. The translator transforms the verified textual rules representation into the C++ code. This programming language is considered to be the most popular for game development.

We developed the prototype rules development environment for turn-based strategy game rule development support. The prototype was developed using Microsoft Visual Studio 2008 SDK that enabled using managed code and rapid language and visual modeling tool creation.

IV. Related Work

There are several approaches reported in literature for dealing with the game rules development automation problem. Moreno-Ger *et al.* [5] consider adventure games creation for educational purposes. The textual adventure game-specific language and the corresponding interpreter are created. One will need to master the textual notation of the language to use the proposed environment. That might complicate a game designer’s job. Rule verification is not considered. Moreno-Ger *et al.* [6] extend the environment proposed in [5] by new reusable adventure game-specific entities.

Hu [7] also considers adventure games for education. The proposed education model describes education roles (a teacher, a student, a course-book, etc.). The education model-based TorqueEngine script extension is developed. The rules are stored in several files using basic TorqueEngine principles. Rule verification is not considered. The educational adventure game development is significantly restricted by the education model proposed.

Furtado *et al.* [8] suggest an approach and a general framework for game development. The informal visual modeling language is defined and the corresponding tool is created. The language is intended to describe game rules, game graphics and sound. The language operates the notions of “game state”, “program”, “audio component”, etc.. Two layers of abstraction are mixed in a single game model (the game components layer and the specific component object model layer). That may complicate game designer’s job. The proposed framework supports the model inspection for states reachability, states existence and constraints existence. Balance checking is not supported.

Amory [9] deals with educational quest and adventure games development automation. The approach includes the development of interface library encapsulating concepts from the corresponding game genre domains. An educational quest or an adventure game could be created implementing the interfaces of this library. Rules verification is not considered.

Thus, the key advantages of our approach is separation of the rules and the graphics definitions, the design-time rules verification possibility (including balance verification) and turn-based strategy game genre specificity consideration.

V. Conclusions and Future Work

In the previous sections we presented a detailed description of the new approach to game rule development automation for the turn-based strategy game genre. The approach includes the development of a visual formal rule description language, a formal rule verification method and a rules development environment supporting single formal rule model creation, verification and rule-based executable prototype generation. Following this approach we developed the necessary formal basis and the prototype tool for game rules development. The prototype considers turn-based strategy game-specific experience, allows rule balance verification, rapid rules prototype development and rule reuse.

The developed language simplifies rules development. The application of domain-specific languages to TBS rules definition is a new approach to TBS development. The application of formal verification at design-time allowed error detection prior to implementation.

The main advantages of the proposed approach are as follows: rules definition and game graphics definition separation, rules verification automation and TBS genre-specific knowledge consideration.

Additional approach improvements include further development of the balance verification method. We are going to broaden the definition of balance and check for the so-called dynamic balance [3], i.e. the balance at every turn of the game. So players could be guaranteed positive experience throughout the game. We plan the extension of the language type system and the object cloning mechanism revision. Finally, it is planned to extend the approach to the real-time strategy game genre which is very similar to the turn-based strategy genre. This will introduce a concept of mission (missions do not exist in turn-based strategies) and will lead to complex time-constraints consideration. It is expected that our new results will facilitate the development of a development environment for strategy games creation.

References

[1] M. J. Taylor, M. Baskett, G. D. Hughes, S. J. Wade, “Using soft systems methodology for computer game design,” *Systems Research and Behavioral Science,* #24., pp. 359-368, 2007.

[2] M. Brydon, A. Gemino, “Classification trees and decision-analytic feedforward control: a case study from the video game industry,” *Data Mining and Knowledge Discovery,* vol. 17 , Issue 2, pp. 317–342, 2008

[3] A. Rollings, D. Morris, *Game Architecture and Design. A New Edition*. Indianapolis: New Riders Publishing, 2004.

[4] G. Wihlidal, *Game Engine Toolset Development.* Boston, MA: Thomson Course Technology PTR, 2006.

[5] P. Moreno-Ger, I. Martinez-Ortiz, J. L. Sierra, B. Fernandez-Manjon, “Language-driven development of videogames: the <e-Game> experience,” *Entertainment Computing - ICEC 2006*, pp. 153-164, 2006.

[6] P. Moreno-Ger, J. L Sierra., I. Martinez-Ortiz, B. Fernandez-Manjon, “A documental approach to adventure game development,” *Science of Computer Programming*, vol. 67 , Issue 1, pp. 3-31, Jun. 2007.

[7] W. Hu, “A reusable eduventure game framework,” *Transactions on Edutainment I* , pp. 74-85, 2008.

[8] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho, “A computer games software factory and edutainment platform for Microsoft .NET,” *IET Software*, vol. 1, Issue 6, pp. 280 – 293, Dec. 2007.

[9] A. Amory, “Game object model version II: a theoretical framework for educational game development,” *Educational Technology Research and Development,* vol. 55, Number 1, pp. 51 – 77, Feb. 2007.

[10] J. Greenfield, K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools,* 2004.

[11] E. A. Pavlova, “Design of formal domain-specific language for computer game rules development for turn-based strategy game genre,” *Computer and Information Technology Reporter*, Feb. 2009. (in Russian).

[12] M. Abadi, L. Cardelli, *A Theory of Objects,* 1996.

[13] P. D. Mosses (2002). Fundamental Concepts and Formal Semantics of Programming Languages – An Introductory Course. [Online]. Available: http://wiki.daimi.au.dk/dSprogSem-01, 2002.

[14] A. V. Gavrilov, E. A. Pavlova, “Functional interface analysis for formalization of complex information system design,” *Information Technologies*, #9, pp. 9-15, 2008. (in Russian).

[15] V. V. Kulyamin, “Software verification methods,” *Russian State Analytical-Review Articles Contest for Priority Concept “Information-Telecommunication Systems*, 2008. (in Russian.).