MODEL-DRIVEN GAME DEVELOPMENT: 2D PLATFORM GAME PROTOTYPING

Emanuel Montero Reyno and José Á. Carsí Cubel
Grupo de Ingeniería del Software y Sistemas de Información
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camí de Vera s/n, 46022, Valencia, España
E-mail: {emontero, pcarsi}@dsic.upv.es

KEYWORDS

Game Development Methodology, Game Design, Tools, Model-Driven Development.

ABSTRACT

The increasing complexity of game development highlights the need of intellectual and industrial tools to enhance productivity in terms of quality, time and cost. In this paper we propose to apply Model-Driven Development (MDD) methodology to game development, rising the level of abstraction towards game conceptual modelling. As an example, we present a game design prototype tool to prototype 2D platform games for PC, with automatic code generation from UML models.

1. INTRODUCTION

In the last decades, industrial game development has grown exponentially in complexity, highlighting the need of development methodologies to enhance productivity. Game development relies greatly on adhoc development or waterfall methodology, although other software development methodologies have been recently applied to game development: Agile Methodologies, Component-Based Development and Software Product Lines. Withouht leaving aside these previous approaches, we propose to use Model-Driven Development (MDD) methodology in order to rise the level of abstraction of game development, enhancing productivity and shifting efforts from game programming to game conceptual specification, allowing the automatization of the coding tasks in game development.

As an introductory application example of Model-Driven Game Development (MDGD) we present a prototype tool to prototype 2D platform games for PC. Prototyping and playtesting has been previously highlighted (Henderson 2006; Fullerton et. al. 2004) as fundamental engines of game design. Despite being a first approach to productivity enhance of game prototyping, interesting results can be seen working with a higher level of abstraction in game development.

This paper is organized in the following sections: section 2 presents the state of the art in game development methodologies. Section 3 describes the models and transformations that support a prototype tool for MDGD. Section 4 details the conclusions and future lines of research.

2. STATE OF THE ART

Game development is one of the industries with greater expansion of the last decades. The complexity of games has increased exponentially since its origins (Blow 2004). In the mid 80s a game could be developed in 3 months by one programmer who also did the design and art, from conception to final implementation. In 2005 a game can be developed by a team between 20 and 100 multidisciplinary specialists, including programmers, game designers, artists, writers, voice actors, musicians, etc. with a budget over 10 millions of dollars for 4 years of development. Despite the increasing size of the development team and the economic and temporal resources invested, there is a great need of development methodologies to enhance game development productivity.

Game development is a field typically characterized by adhoc low-level development. In the latest years some effort has been done to introduce development methodologies into game development. The first attempt was to use Waterfall Development to cope with the increasing technology demands of game development. Teams scaled up requiring extense game design documentation written in natural language to keep the focus of develpment. But game design documentation wasn't easy to maintain, which caused communication breakdown. Design changes were handled directly at programming level, leading to a difficult maintenance of games and low productivity.

Component-Based Development methodology has been applied to game development (Folmer 2007) in order to allow a greater reuse of recurrent functionality. The use of game development-specific middleware doesn't rise the level of abstraction in development. Games remain programmed with object-oriented languages, script languages and so on. Middlewares are successful and widely extended in game development, becoming an obligation for game development methodologies.

Agile Methodologies have been applied to game development (McGuire 2006; Miller 2008) to enhance change management during development and iterativity in game design, approaching as fast as possible to the core gameplay for the client. Although iterativity enhances game development, Agile Methodologies doesn't rise the level of abstraction in development, keeping the focus on game programming.

In academic research, Software Product Lines have been applied to game development (Furtado and Santos 2004)

using graphic domain-specific languages to represent variations in action-adventure games. Individual games are generated using code templates. The reuse of common characteristics in games of the same genre can be exploited considering this semi-automatic Software Product Line development.

3. MODEL-DRIVEN GAME DEVELOPMENT

In order to cope with the increasing complexity of game development, the level of abstraction have to be increased. With this aim we propose to apply Model-Driven Development (MDD) methodology to game development. As an introductory example a prototype tool has been implemented to prototype 2D platform games for PC from UML models. 2D platform games are a game genre characterized by a protagonist who moves and jumps into platforms, collecting prizes and destroying enemies in various ways. Classic 2D platform games include *Super Mario BROS*, *Sonic The Hedgehog* and *Bubble Bobble*.
The prototype tool uses two Platform-Independent Models (PIM) to define the structure and behaviour of games avoiding development details of the underlying technology platform. A Platform-Specific Model (PSM) describes the mapping of game actions to the hardware control devices by which the player interacts (joystick, keyboard, …). A model transformation from the previous UML models generates the code of the prototypes in C++ using a game development-specific middleware, *Haaf Game Engine*, to provide basic functionality through specialized libraries. Other programming languages and middleware could have been used as development technology platform. Finally, the generated code can be manually completed and the 2D platform game prototype can be playtested iteratively.

3.1. Structure Diagram

2D platform game structure can be specified using class diagrams extended with stereotypes. Stereotypes are UML extensions that allow the creation of model elements suited to the problem domain, which in this case is 2D platform games for PC. The *PlayerCharacter* stereotype describes a game entity controlled by the player. The *Enemy* stereotype defines a game entity controlled by the game system opposing the main character of the player. The *Entity* stereotype describes the other passive game entities such as prizes and platforms. Each stereotype is abstractly represented by a coloured shape in the 2D platform game prototype: the main player character is rendered as a green circle, the enemies as red circles, the prizes as yellow circles and the platforms as orange squares.
In the example *Bubble Bobble* prototype, the UML structure diagram extended with stereotypes specifies all game entities and its relationships. The game level is affected by gravity (attraction towards the ground) and friction (resistence to movement). The main game entities are the player character *Bub*, the bubbles, the *Benzo* enemies, the prizes and the platforms. The player character has a number of lives and a score that increases destroying enemies or collecting prizes. Bubbles can contain a trapped enemy inside.

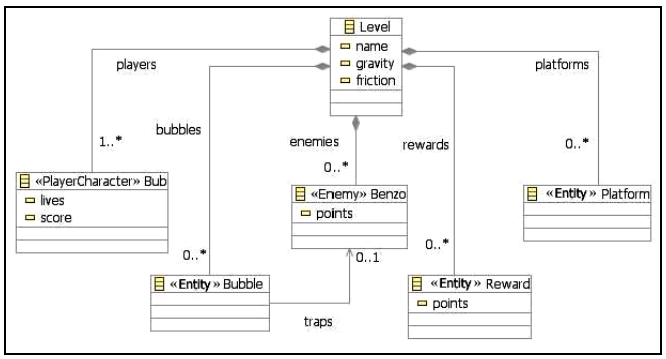

Figure 1: Structure Diagram of *Bubble Bobble*

3.2. Behaviour Diagram

UML state transition diagrams can be used to specify the basic behaviour of each 2D platform game entity. State transitions are triggered by game events which change the internal game entity state. The final game entity state represents the destruction or death of the game entity.
In the *Bubble Bobble* example prototype the behaviour diagram of the player character describes that *Bub* can be alive either on the ground or in the air. In both cases he can blow bubbles but he can only move and jump from the ground. If *Bub* collides with a bubble, he will pop it. If *Bub* collides with a prize he will gain its points. If *Bub* collides with an enemy he will lose a life. Losing all his lives will destroy *Bub*.

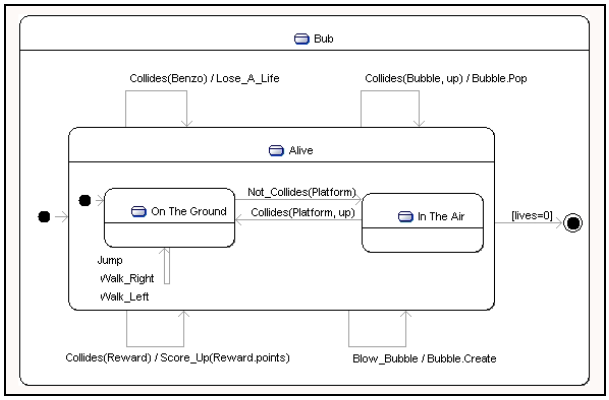

Figure 2: Behaviour Diagram of the Player Character of *Bubble Bobble*

3.3. Control Diagram

Game control can be understood as a hardware device configuration allowing the player to interact with the game. Game control mappings specify which controls are associated to each game action. Game actions are game events triggered by the players. Figure 3 describes a metamodel to capture control diagrams for 2D platform games for PC.

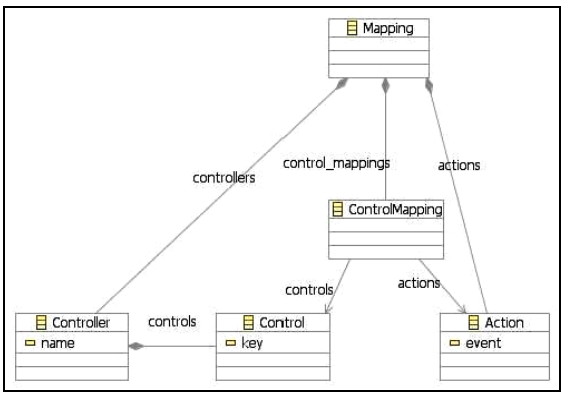

Figure 3: Control Metamodel for 2D Platform Games

The control diagram is a Platform-Specific Model (PSM). Two kinds of techology platforms can be distinguished in game development: development technology and target technology. Development technology platforms include programming languages, SDKs, *middleware* and other technology used in the construction of the game. Target technology platforms include the hardware devices in which the game will be played. In present days there is a great variety of target technology platforms: PC, flatbed and handheld consoles, arcades, mobile phones, etc. Development technology aspects can be automated through the use of Platform-Specific Models (PSM) such as the control diagram.

4. CONCLUSIONS AND FUTURE WORK

Despite the efforts invested in the construction of a prototype tool for 2D platform game prototyping it is fair to declare its shortages and limitations. The UML models used to specify 2D platform games are closer to software engineers than to game developers. It would be of great interest to develop a conceptual model close to game developers terminology, deriving through model transformations the structure and behaviour models.

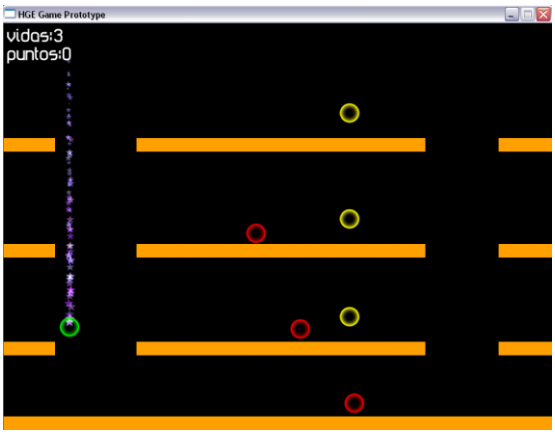

Figure 4: *Bubble Bobble* Prototype

Focusing on the productivity results, a week was invested to manually implement the *Bubble Bobble* example game prototype. Using the model-driven approach, another prototype was automatically built in a day, with 93% of code generation from the UML models. Model to code transformations in *MOFscript* derive automatically most of the game prototype code from the models, notably rising the productivity of the development process. These automated transformations assign each game element semantics, explicitly embedding 2D platform game domain knowledge and good practices in the code generation. In order to informally validate these results, another game (*Super Mario Bros*) was prototyped reusing *Bubble Bobble* models. 94% of the game code was automatically generated. The model-driven approach enhances productivity in game development and reusability of the software artifacts. The programmer only has to manually code aspects not specified in the models (such as the artificial intelligence of the enemies).

Table 1: Automatic Code Generation of 2D Platform Game Prototypes

Proto-type	Structure Code	Behaviour Code	Total %
Bubble Bobble	485 / 495 lines	232 / 274 lines	93%
Super Mario Bros	393 / 393 lines	184 / 215 lines	94%

As future work remains the application of Microsoft's XNA game specific middleware to model-driven game development. The first step is to define a game specific modelling language to precisely describe games using concepts closer to game developers. The next step is to build PSM models for the XNA development techonology platform, and transformations between platform independent and specific models. Finally, code will be generated to each target technology platform: PC and XBOX 360.

REFERENCES

Blow, J. 2004. "Game Development: Harder Than You Think". In *ACM Queue*, 1 (10).

Folmer, E. 2007. "Component Based Game Development: A Solution to Escalating Costs and Expanding Deadlines?". In CBSE.

Fullerton, T. and Swain, C. and Hoffman, S. 2004. "Game Design Workshop: Designing, Prototyping, and Playtesting Games". CMP Books, 157.

Furtado, A. W. B. and Santos, A. L. M. 2006. "Using Domain-Specific Modeling towards Computer Games Development Industrialization". In *DSM Forum*.

Henderson, J. 2006. "The Paper Chase: Saving Money via Paper Prototyping". In *Gamasutra*.

McGuire, R. 2006. "Paper Burns: Game Design with Agile Methodologies". In *Gamasutra*.

Miller, P. 2008. "Top 10 Pitfalls Using Scrum Methodology for Video Game Development". In *Gamasutra.*