Automating the Implementation of Games based on Model-Driven Authoring Environments

Christos Karamanos[1], Nikitas M. Sgouros[1]

[1] Department of Digital Systems, University of Piraeus
18534, Piraeus, Greece
ckar2006@yahoo.com, sgouros@unipi.gr

Abstract. In recent years the emergence of multiple game platforms (e.g., mobile, specialized consoles, PC) along with the rising interest in game creation by users with limited technical background calls for the development of high level authoring environments that base the creation process on abstract game models and automate their implementation. This paper describes our research towards the creation of such a development and execution environment that incorporates a graphical authoring environment based on an abstract model of common action RPG games along with a 3-D rendering client that automatically translates the resulting game into an OpenGL ES implementation.

Keywords: Authoring & Content Creation, Domain Modeling & Software Generation.

1 Introduction

In recent years the emergence of multiple game platforms (e.g., mobile devices, specialized consoles, PC) has placed a considerable burden on game development thus creating the need for authoring tools that can automate multi-platform implementation. In addition, the rising interest in game creation by users with limited technical background calls for the development of high level authoring environments that focus the development efforts on describing the game idea rather than on its specific technical implementation. This can be made possible through the development of abstract game models that capture the basic characteristics of whole classes of games and allow for different instantiations of this basic model through appropriate authoring environments.

As a first step towards the development of model-driven authoring this paper describes a game creation environment that seeks to provide high level authoring capabilities for a popular class of games we refer to as *action RPGs*. In these games the player moves in 3-D space having to face a series of enemies and escape from certain traps while collecting treasure items and various devices that boost his health and physical abilities. Collecting a certain amount of treasure items allows the user to terminate successfully the game, while decreasing its health level to zero after a series of unsuccessful battles leads to unsuccessful termination of the game.

The game environment provides a graphical authoring tool that allows a user with limited technical background to specify the plan of the 3-D space that comprises the game space and the location and attributes of the rest of the game elements (enemies, traps, treasures, health points, enhancement items), In addition, the specifications created by the user are then fed to a rendering client that automatically creates and manages the 3-D game environment based on the specifications it receives via the Web from the authoring system. The authoring and rendering systems run as separate applications. In particular, the authoring environment runs as a Web application, while the rendering client executes on a variety of computing devices that will eventually run the game since it is built using OpenGL ES.

While there has been significant work on general-purpose game engines (e.g. Unity, JMonkey, Unreal) the use of these engines require significant technical background and programming expertise. There has been considerable research in game authoring environments for users with non-technical background mainly centered on the development of scripting environments [1-2] and an interpreter for executing the scripts, or on the use of UML-based models [3]. While this approach can support simple games in 2-D environments, as we move to 3-D and/or increase the complexity of the games in terms of the characters and objects involved the amount of scripting grows significantly making the addition of functionality and debugging these scripts increasingly difficult. Our approach offers a more user friendly solution to users with limited technical background based on deciding what game model to use and providing graphical ways of configuring basic features of each game model. In addition, the rendering client can be used as a stand-alone game engine API allowing for in-depth modification of the resulting game.

Game Maker [4] is a popular development tool that provides a graphical front-end to an interpreted scripting language (GML) for game creation. Our approach seeks to create a development environment on a higher level of abstraction than Game Maker. In our system users graphically configure complete game models thus significantly reducing the need for scripting. In addition, there is no need for a special-purpose scripting language since the game models can be directly mapped to OpenGL implementations thus providing more efficient and portable implementations.

In the rest of this paper, section 2 describes in detail the graphical authoring environment while section 3 presents our work-in-progress in developing the rendering client. Finally, section 4 is a discussion and future work section.

2 The Authoring Environment

The authoring environment models game space as a 2-D grid that will be subsequently mapped by the rendering client to a 3-D space. There are two main character types (see Fig. 1) that can move around in game space: players (controlled by users) and enemies (that are opponents to the players and controlled by the rendering client). Each grid cell may possess one or more of the following nine primary attributes: (i) Clear (the cell has no attributes) (ii) Wall (the cell contains a motion obstacle) (ii) Trap (the cell contains a trap which has a negative effect on some of the player's attributes (e.g. speed, agility)) (iv) Enemy Control Area (the cell

is within an area that an enemy can move into) (v) Player (the cell currently contains a player) (vi) Enemy (the cell currently contains an enemy) (vii) Treasure (the cell contains an effect that influences positively the material rewards obtained by a game character) (iix) Buff (the cell contains an effect that gives a beneficial boost to the performance of a game character) (ix) Health (the cell contains an effect that gives a beneficial boost to the health of a game character).

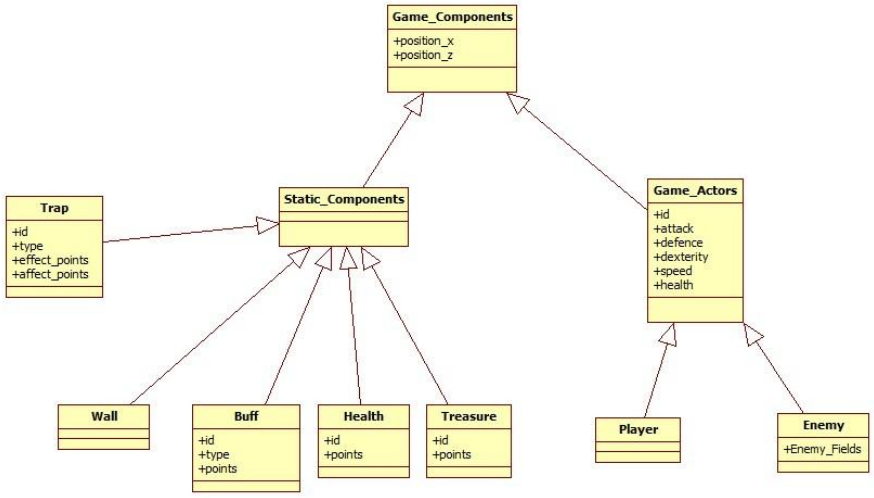

Fig. 1: Class diagram of the available game components.

When combined these primary attributes give rise to fourteen composite attributes for each cell (e.g., Buff in Enemy Control Area, Health in Enemy Control Area, Player in Enemy Control Area etc). In addition, the author now has to set attribute values for the game characters (players & enemies) and the rest of the game features (health items, treasure items, enhancement items, traps). For the player type these include setting points in a numerical scale for: (i) Speed (how fast the player moves in game space) (ii) Health (how fit is the game character) (iii) Attack (the potential for winning a battle with an enemy as an attacker) (iv) Defense (the potential for winning a battle with an enemy as a defender) (v) Agility (the ability to escape from a trap).

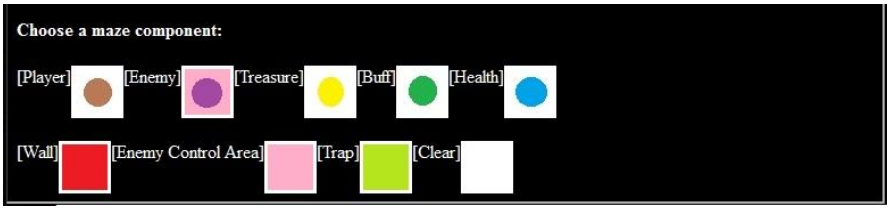

Fig. 2: Graphical representation of the available game components.

For the enemy type the game author can set the same attribute values as the ones for the player type. In addition the author can set the value of the Enemy Areas attribute which describes the set of grid cells in which the specific enemy can move.

For health items, the game author can set the amount of health points the item would have in a numerical scale. The player's total health points will increase according to these points when player retrieves the health item.

For treasure items, the game author can set the amount of treasure points the item would have in a numerical scale. The player's total treasure points will increase according to these points when player retrieves the treasure item.

For enhancement items (buffs), the game author can set values for: (i) Buff Type (the player's attribute that will be enhanced if player retrieves the enhancement item – attack, defense, speed, agility) (ii) Buff Points (the amount of enhancement points).

For trap items, game author can set values for (i) Trap Type (the player's attribute that will be affected/decreased if player activates the trap) (ii) Trap Effect Points (the trap's activation threshold) (iii)Trap Affect Points (the amount of decreased points after trap activation)

A player and an enemy battle each other whenever their distance becomes less than an amount defined by the rendering client. In this case the rendering client randomly assigns to one of the characters the role of the attacker while the other one becomes the defender (the roles of the characters change consecutively while the battle progresses). A random value is added to the attack/defense attribute of the attacker/defender and the winner of the battle is the character with the larger value of his corresponding attribute which is then subtracted from the health points of the losing character. The encounter continues until the player escapes the battle by increasing the distance against the enemy more than the defined threshold or if one of the characters loses all its health points.

For each trap the game author specifies its activation and affect values. Each time a player/enemy enters a trap a random value is added to the value of his agility and their total is compared with the activation value for the trap. If this total is larger, the player/enemy escapes from the trap unharmed. Otherwise, the value of one of his attributes (attack, defense, speed, agility) defined by the author for this trap is reduced by the affect value of the trap.

While a trap reduces the value of a character attribute, a buff increases the value of a chosen attribute by an amount set by the game author. Finally, a treasure increases the value of a treasure item that is defined by the author. For each treasure item the author defines a specific amount for its successful collection. Successfully collecting all treasure items provides a winning terminating condition for a player in the game.

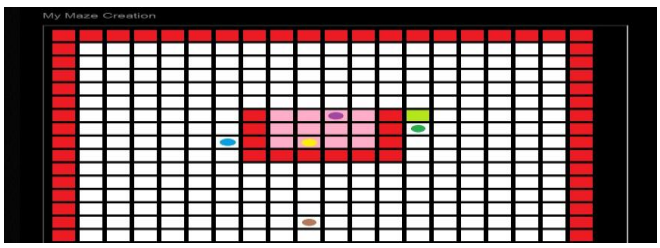

Fig. 3: A snapshot of the 2-D game grid with some of the components defined**.**

Figures 2 and 3 provide snapshots of the graphical environment used during authoring. The technologies used for the development of the application are: Spring

Framework MVC, Java, JSP, JSTL, CSS, Javascript – JQuery, AJAX. Execution of the editor is implemented using an Apache Tomcat web server. The Spring Framework MVC (Model View Controller) was used to make the application parts easier to recognize and manage. As a framework that is based on the MVC model, it consists of three distinct entities (model, view and controller) and provides a clear division between them. Business logic and user interface are treated as totally two different layers that are easy to manage. The use of Spring annotations makes request handling easy and straightforward. Furthermore, the ability to use features such as security and role management for future expansion of the editor, makes Spring fit for use as the base of the Web client. The operational logic of the application was implemented in Java while the client side (View) consists of a set of JSP pages formatted with CSS and using Javascript – JQuery. AJAX technology was used for the direct communication between the client and the server so that the request-response to materialize without being visible to the user during the construction of the game space.

3 The Rendering Client

The rendering client is responsible for creating and executing the game based on the specifications received by the authoring system. In addition it can be used as a stand-alone game management API as long as it receives game specifications that are compatible with the ones created by the authoring system. It is implemented in Java based on the OpenGL ES API. Figure 4 provides a snapshot of the game environment created by the client on a mobile device.

4 Discussion & Future Work

We describe our research towards the creation of authoring environments based on abstract game models that describe the basic characteristics for whole classes of games and allow their automatic instantiation in a variety of computing platforms. These game models provide configurable implementations for all the mechanics in each game type including player management, terrain generation, navigation and interaction among the game objects. Our approach seeks to eliminate the need for special-purpose scripting languages by providing direct mapping of the game models to OpenGL implementations. There are a lot of open problems that need to be addressed in order for this approach to become feasible. In particular, there is the need for the development of automated synthesis capabilities so that parts of different game models could be combined in order to automatically create implementations for new types of games not envisaged by the system. Furthermore, the complexity of the mapping between game specification and its implementation is expected to increase as the game models are extended to describe higher level behavior between game objects (e.g., conflicting goal-driven behavior between characters) and more complex plot structures (e.g. aristotelian plot conceptions). More immediate future work in this area is focused on running user trials of the authoring and rendering systems along

with the development of abstract models for other classes of games (e.g., sports, business or social games) and the provision of multiplayer capabilities in the rendering client.

Fig. 4: Snapshot of the rendering client depicting the player (the blue & yellow cube) close to a trap (the green floor) facing an enemy (the black & purple cube) and a health item (the blue cube).

Acknowledgments. Presentation of this work was partially supported by the University of Piraeus Research Center.

References

1. McNaughton, M., Cutimisu, M., Szafron, D., Schaeffer, J., Redford, J., Parker, D., Scriptease, Generative design patterns for computer role-playing games, 19th IEEE Int. Conf. On Automated Software Engineering, pp. 88-99 (2004)
2. MIT, Scratch. Website (2009) http://scratch.mit.edu
3. Reyno, E. M., Carsi Cubel, J. A., Automated Prototyping in Model-Driven Game Development, Computers in Entertainment (CIE), vol. 7, no. 2 (2009)
4. Habgood, J., Overmars, M., The Game Maker's Apprentice, APress, 2006