2D Game Editor On-line
A Cloud Computing Perspective on Game Edition

Bruno Miguel Cardoso
CITI-DI/FCT/UNL,
Departamento de Informática
Universidade de Évora
Rua Romão Ramalho 59
7000-671 ÉVORA, PORTUGAL
+351 266 745373
b.m.pinto.cardoso@gmail.com

Teresa Romão
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, PORTUGAL
+351 212948536
tir@di.fct.unl.pt

Abstract

The present article aims to describe the efforts taken in order to create a generic computing solution for the most recurrent problems, usually found in the production of two dimensional, sprite-based videogames, running on mobile platforms. The developed system is a web application that fits within the scope of the recent cloud-computing paradigm and, therefore, enjoys all of its advantages in terms of accessibility, application maintainability and scalability. In addition to the functional issues, the system is also explained in terms of its software architecture, since it was planned and implemented in the perspective of attaining an easy to maintain application that is both scalable and adaptable. Additionally, an algorithm is also proposed, seeking to find an optimized solution to the problem of space distribution for several rectangular areas, with no overlapping and no dimensional restrictions, neither on the final arrangement nor on the arranged areas.

Keywords

Game edition, web based solution, scalability, software engineering, collaborative work.

1. INTRODUCTION

Facing a growing activity in the last thirty years, the market of electronic entertainment – of which videogames are the most preeminent representatives – moved, just in 2007, over 41 billion dollars across the whole world. This value far surpasses the ones presented, for the same period, by the movie industry [Alves2008]. However, the production of videogames that match current market standards can be a daunting task, since it may require expertise in several artistic and technological fields, especially when the new title is intended to run on advanced consoles or personal computers.

Still, there is a particular kind of videogame whose production is much less demanding and still enjoys a wide acceptance by the consumers: the casual games. Being much smaller (in every aspect) than a traditional title, this kind of game also has much simpler rules and, thus, requires a far less dedicated gamer profile. This contributes, of course, to widening the range of its potential consumers – in fact, they have a possible user in just about anyone who wishes to spend some time entertaining themselves. Furthermore, these already widened use scenarios can still be extended if the game is targeted to run on mobile platforms. That way, a possible use context can be identified in every place where such a platform is able to operate. Summarizing this point, not only a casual game can be used by almost anyone, they can also be used almost everywhere.

However, although being simpler to create and distribute, the production of a new casual title may still require the same specialized skills that the more traditional and advanced videogames do – only perhaps not so deeply. Even though it's possible for a single professional to have all of the necessary skills, such is not, indeed, a realistic premise. It's far more common – and, perhaps, even preferable – to invest in multidisciplinarity and enlist a team of various professionals, skilled in each of the required fields of knowledge. Clearly, this will bring an extra effort to be made, in order to combine their distinct works (e.g. art, written texts, sound) into really useful, programmatically accessible resources.

Also, there is a peculiarity in general videogame development that may promote more constraints to its production. In a nutshell, the game has a concept that must not be lost during its development. This concept defines the user experience, general gameplay and provides a detailed overview on how the game should perform once implemented. Being such a comprehensive and abstract idea, the game concept may also include subjective impressions and other not so easily transmittable aspects. The responsibility of maintaining the development processes oriented to this concept

usually falls upon one professional, the game designer, who must be, before anything else, a creative and resourceful person with some technological and artistic perceptiveness. Then, in order to create a game that is true to its concept, another of the game designer's duties refers to coordinating those heterogeneous teams, where its members may have their own professionally-specific technological languages (that can even be, to some extent, quite hermetic). Since a significant part of the designer's tasks consist in transmitting the game concept's ideas to the development team, it follows that the threat posed by the occurrence of communication misunderstandings can be exceptionally dangerous to the project itself.

To reduce the possibility of human error emerging from the natural language's intrinsic subjectivity, a recurring solution is frequently adopted: the usage of game editing applications. These systems offer, through interactive and friendly user interfaces, the possibility to parameterize certain game properties that further tune the game player's experience. Such kind of solution can be found both in amateur and professional game productions, since they all have the same basic requirements. However, mainly due to resource availability, there are notable differences between the two. While industrial editors use to be larger, internally developed, desktop applications that provide support to a wide range of tasks, the editors used in amateur productions tend to be smaller, single-tasked applications that are developed by other amateur producers, who, then, distribute them freely over the web. Being larger applications, the industrial editors also have more effort applied to their development and, as so, they are not lightly replaced or discarded. Instead, every time a new and sufficiently valid production requirement arises, they are likely to be subject of maintenance interventions, with the purpose of adding, altering or removing existing functionalities. Such maintenance procedures can be achieved by creating software patches or, more often, by tampering directly with the editor's source code. Either way, the needs that motivate these "ad-hoc" interventions can have some degree of urgency which may force the maintainers to disregard the (eventually) existing code architecture and methodologies. This will end up in posing difficulties to future maintenance tasks.

Besides from programmatic disadvantages that this "ad-hoc" development methodology is likely to impart, the application's user interface also gets prejudiced. In fact, specific widgets that once have been assigned to operate on specific areas of an interface, may now find themselves competing with lately developed controls that offer whole new functionalities. Such occurrences are hardly positive for already experienced users, even in the unlikely case that the developers had the time to carefully plan the alteration, and may ultimately lead to software abandon [Cardoso08].

On the other hand, because the free editors used in amateur productions are independent applications, the production lines that use them will have to include efforts to merge their different outputs and to make sure they integrate, as intended, into the developing game.

There are several such applications on the web, each one being specially oriented to assist in a specific task. Particularly for sprite based videogames, a most useful functionality relates to sprite packing. This means aggregating the images that compose a sprite's animations, the frames, into new, bigger images, the atlases, in an optimized – but not necessarily optimal – way. In fact, the frames usually come from the artists who create them arranged in larger images, known as strips. These strips are rather raw resources and, seeing as they have a lot of unused space between the frames, they are too much "unwieldy" to be used in the final product. Indeed, since these large images would have to be loaded into memory, using them directly would inevitably result in increasing a game's memory requirements. So, a sprite packer typically operates by receiving the already cut frames, making an arrangement out of them and exporting both the produced atlas and the necessary metadata references that allow external agents (e.g. the game engine) to univocally refer to a specific frame within the atlases larger scope.

However, these free sprite packers fall short of being truly supportive applications, because they do just that - pack the frames passed in as input - while neglecting that there are other frame related tasks to accomplish (e.g. cut them out of the strips or parameterize the sequences that are, in fact, the animations themselves). Some examples of these free editors, programmed in various languages, would be the Packer[1], from the 2D Slick Game Library, and the Image Packer[2].Even though these applications are freely distributed over the Internet, they are not true web applications; instead, they are small programs that must be downloaded and locally executed on the client machine. As previously stated, none of them allows for anything else than packing the frames and some do not even have anything but rather rudimentary atlas composing algorithms.

With the intention of simplifying the implementation of such algorithms, the developers often place on the users the responsibility of making some very important decisions, like the final dimensions of the produced atlas. This may have significant impacts on the application's real utility and even contribute to degrade its performance since not only direct user intervention will be required, but also it makes way for poor dimensional choices that may imply large amounts of unused area.

2. THE 2D GAME EDITOR ON-LINE

The system proposed in this paper, named 2D Game Editor On-line (2DGE-O), consists of a web application intended to provide support for the most recurrent tasks found in two dimensional, sprite based videogame production. These tasks include: strip loading; selective cutting of the frames embedded on a loaded strip; aligning the sequences of frames that compose a sprite's

[1] http://slick.cokeandcode.com/
[2] http://www.pascalgamedevelopment.com/forum/index.php

animations and exporting the manipulated information to standard formats, such as XML and *.png* atlases. To further enhance the exporting process, an algorithm has been perfected to automatically create optimized atlases. One of the system's key features lies on its strong end user orientation and, to this end, much attention was devoted to its user interface. The system is implemented with a mix of technologies, like PHP and MySQL on the server side and, on the client's end, DHTML, Java Applets and AJAX for asynchronous communications with the server. Fig. 1 shows the frame cutting Applet (on the right, below the "Strip blaze" label), embedded on the system's general web interface.

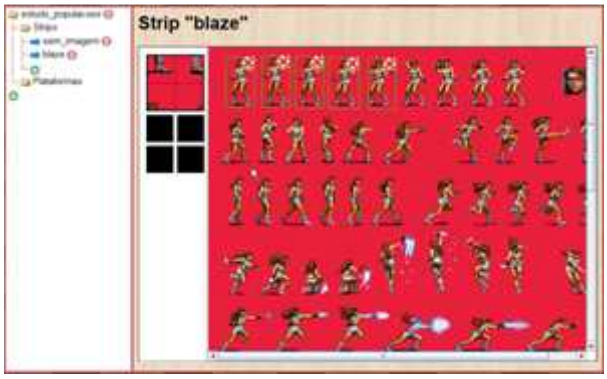

Fig. 1- 2D Game Editor On-line interface overview.

An essential difference that conceptually distinguishes this solution from the ones described in section 1 lies in its way of integrating the multitude of tasks, necessary to produce a videogame, into a single concept, the project. This integrative perspective closely resembles the one that is traditionally found on industrial game editors.

2.1 Game Edition in Cloud Computing Paradigm

The 2DGE-O is a web application that, according to the NIST's definition of Cloud Computing [Mell09], adheres to the Software as a Service Model. This means that the user does not control or manage the underlying cloud infrastructure, and it offers all of its functionalities with the sole requisition of a thin client interface - in this case, a regular browser and some of its common resources, like the JVM[3], cookies and JavaScript enabled. This allows for a game development team to be geographically dispersed and still be able to collaborate on the same game projects. In another perspective, that of the developer's, the main advantage that emanate from basing such a system on the web, is the possibility of focusing the maintenance efforts on just one copy of the application – precisely the one that is installed on the production server. Any modifications performed on this server's deployed 2DGE-O instance will instantly be made available for all of its users.

2.2 The System's Internal Architecture

Noting the enormous product variety on modern videogame's market, one can infer that is impracticable to create an editor that attends to all of this industry's requirements. Then, to cope with this impossibility, another objective of the 2DGE-O is to provide the basis for future functional module development. In this sense, there was a necessity for programming methodologies that contributed to maintaining the code as human readable as possible, while defining clearly the procedures to follow when creating new components. Such sort of problem is not new in the scope of software engineering. In fact, these requirements are so commonly encountered, that they have given origin to more than one solution meant to solve them, generically referred to as "architectural patterns". This concept is defined as an "*idea that was useful in a specific practical context and that is likely to prove useful in others*" [Fowler96].

There are several architectural patterns that enforce code methodologies and, at the same time, keep a close attention to the user interface and its respective mechanics. The most representative would be the Model-View-Controller (MVC). In spite of having been first proposed to be applied on *SmallTalk* environments [Krasner88], this pattern was the most adequate to the 2DGE-O characteristics. The MVC imposes, as stated in [Buschmann96], the separation of an application's code in triads made of three functional components, according to the responsibilities they assume in the system's functionalities: the Model is responsible for the data and provides the methods to operate on it; the View renders the corresponding model's information to the user and, lastly, the Controller reacts to and makes decisions based on user input. The user interface (or parts of it) is formed by View-Controller pairs. Such pairs can be connected directly to another triad's Model, if the necessary methods are already implemented in it. This way, in order to create a new functional module and expect it to integrate smoothly into the already existing interface, the programmer must implement only – in the best case – a new View-Controller pair and associate it with an already existing Model. Of course, if there is no Model that already performs the actions required by the new pair, then a new one will have to be implemented. Such well defined procedures contribute greatly to maintain the 2DGE-O's code legible and organized after such alterations are completed.

2.3 The Atlas Composing Algorithm

The usage of texture atlases in games can bring many advantages to their performance and this is especially true when the game is intended to run on low resourced platforms, such as mobile phones. They allow saving memory that would, otherwise, have to be allocated to very consuming tasks, like image swapping. Since a game might have several sprites rendered simultaneously on screen, these operations can turn into considerable nuisances.

An algorithm, proposed by [Scott], configures an interesting option to solve the problem of packing a number of rectangular areas (the original proposal considers these as being lightmaps) into a fairly optimized image. To this end, it represents a two-dimensional empty area (the atlas) as a binary tree that registers growth whenever new images are added and

[3] Java Virtual Machine.

arranged. Nonetheless, there are some issues with the described approach that drastically detracts from it, namely, it assumes the final atlas dimensions to be known *a priori* or, at least, an upper bound for them. This is not a very practical solution, since the users will end up searching for a satisfactory result through a *generate and test* (or *trial and error*) methodology. To avoid this, it's necessary to find a way to provide more autonomy to the algorithm.

We found a solution by supplying an empty rectangular area of unlimited dimensions and define it as the base upon which the arrangement will be made. Inside it, another rectangular boundary will be kept in memory, first with dimensions set to 0 and then, as more frames are added, it will be iteratively enlarged to accommodate them. This area is referred to as the "frontier". The key feature of the algorithm lies in an heuristic approach that provides guidance every time this frontier needs to expand. Many approaches were attempted, but the most satisfactory results have been achieved when using a special scalar measure, R, that, for any rectangular area, $R = \frac{shorter\ side}{longer\ side}$. It's trivial to deduce that, for any given rectangle, $0 < R \leq 1$, and that $R = 1$ only when the area is a perfect square. The algorithm starts by calculating the average R for the whole population of frames that will be arranged. Then, it proceeds to successively inserting the frames and, whenever one is encountered that cannot be arranged inside the current frontier, the algorithm will expand its borders, choosing the resize that grants the closer proximity between the frontier's own R and the frame population´s average R.

3. USER TESTS

The user tests were performed in two sessions, with the same testing group composed of five users, whose average age was about 35 years and all have worked with IT for a period of 2 to 14 years. One of them was an amateur game designer. These sessions followed a previously planned script that included the main tasks an editor is required to perform, such as uploading strips, cut its frames and building a few animations with them.

In the appointed tasks, the user's feedback was mostly positive, consistent and very focused on the functionalities offered by the application. The user interface revealed itself both as easy to use and easy to learn. Apart from the positive comments about the system's functionalities and on how they were implemented, there was a significant interest on the space allocation algorithm and its potential application to problems outside the context of videogame production (a possible example would be the reduction of website images download time).

There were, however, some invaluable remarks made by the users that contributed to further improve the system's interaction with the end-users. As an example, one of the registered observations suggested further refinement of the included zooming tool, so that it may provide more in-depth support to the tasks of frame selection and cutting.

4. CONCLUSIONS AND FUTURE WORK

The 2D Game Editor On-line results of an innovative approach to a recurrent problem, i.e. the cloud-computing paradigm's answer to the game edition's requirements. Put simply, the system is a web application, accessed through common browsers, which allows for remote collaboration between geographically dispersed teams. Also, because the source code is solely located on the server, the client-server model eases the efforts associated to application maintenance. This way, when necessary to alter the current version, the maintainers can intervene directly on the server, while keeping the clients unaware, or unaffected, by the operations.

Because of the MVC pattern applied to the system's architecture, the system ends up being highly scalable, functionally flexible and maintainable.

The user's feedback during the tests was very positive. However, despite the fact that the 2DGE-O gives full support to sprite manipulation tasks, there are a number of improvements to be done on the already implemented functionalities, which will greatly add to the system's usability and performance. As an example, the atlas composition algorithm could be further optimized if allowed to operate with rotation. This way, before deciding to expand the current atlas's frontiers, the algorithm should attempt to reinsert the same frame, after a 90º rotation. As future work, there will also be implemented a undo/redo functionality that allows users to revert their work to previous states, as well as the user suggested refinements to the zooming tool (referenced on section 3). Additional user test to evaluate the system's usability and performance will also be performed.

5. REFERENCES

[Alves2008] Alves, L. *Estado da Arte dos games no Brasil: trilhando caminhos*. Proceedings of ZON Digital Games 2008, 6-7 Nov, 2008, Porto, Portugal, 9-18.

[Buschmann96] Buschmann, F. et al., *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons Ltd, West Sussex, England, 1996.

[Cardoso08] Cardoso, B. and Romão, T. *Um Caso Prático de Reabilitação de um Editor de Jogos Móveis Utilizando o Paradigma Model-View-Controller*. Proceedings of Interacção 2008, Évora, Portugal, 109-144.

[Fowler96] Fowler, M. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Boston, EUA, 1996.

[Krasner88] Krasner, G. E. and Pope, T., *A cookbook for using the model-view-controller user interface paradigm in SmallTalk*. JOOP, 1(3), Aug, 1988, 26-49.

[Mell09] Mell, P. Grance, T. 2009. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology Website.

http://csrc.nist.gov/groups/SNS/cloud-computing

[Scott] Scott, J. *Packing Lightmaps*. Website accessed on May 10, 2010.
http://www.blackpawn.com/texts/lightmaps/default.html