The Effectiveness and Efficiency of Model Driven Game Design

Joris Dormans

Amsterdam University of Applied Sciences

Abstract. In order for techniques from Model Driven Engineering to be accepted at large by the game industry, it is critical that the effectiveness and efficiency of these techniques are proven for game development. There is no lack of game design models, but there is no model that has surfaced as an industry standard. Game designers are often reluctant to work with models: they argue these models do not help them design games and actually restrict their creativity. At the same time, the flexibility that model driven engineering allows seems a good fit for the fluidity of the game design process, while clearly defined, generic models can be used to develop automated design tools that increase the development's efficiency.

1 Introduction

Games are hard to design and develop. Game audiences expect a higher level of quality year after year. For contemporary triple-A console titles this means that development teams easily consists of more than a hundred designers, programmers, and artist, and a production period that spans multiple years. Even for casual games, which up until a few years ago could be developed within a couple of months by a team of under five people, we see changes. The current generation of mobile and social games is already developed by experienced studios that assign twenty to thirty developers to the task and year-long development times are no longer uncommon. In order to keep producing better quality for less money, the games industry needs to find ways to either increase revenues, or improve the efficiency and effectiveness of the development process. This paper investigates how modeling techniques can be used to do the latter, but also at the obstacles that need to be overcome to get model driven techniques accepted by the game development community.

2 Abstract Game Development Tools

For a while, within the game design community there has been a careful push for the development of abstract tools and methods for game. In a 1999 article Doug Church called for the development of 'formal abstract design tools' [1]. Since then, a number of frameworks and tools have sprung up. Many of them

are primarily design vocabularies, created to help understand and identify common structures in games, and avoiding to be prescriptive in their description of games.[1] Berndt Kreimeier suggested to apply the design pattern approach from architecture and software engineering to game design [2]. In contrast to vocabularies, design patterns are prescriptive; they describe 'good', generic solutions to common problems. However, the most prominent work on design patterns within the domain of game design to date [3], is explicitly distanced from a prescriptive approach, creating a hybrid approach that is closer to a design vocabulary than a pattern language. At the same time, Raph Koster experimented with a graphical grammar to express game mechanics [4], this approach was followed by [5–7]. In contrast to the design vocabulary approach, the focus of these grammars has not been the collections of descriptions they allow, but the design lore that they capture. For example, the Machinations framework [7] sets out to visualize the structures in game mechanics that create emergent gameplay; in other words, it departs from theoretical vision on quality in games, and constructs a tool set that allow game designers to interact with the structures that contribute to that quality more directly.

In the Machinations framework the diagrams expressing game mechanics act as a domain specific language (DSL) for a subset of game development; in this case for a game's "internal economy" [8]. Using similar DSLs for level design opens up the possibility of applying techniques and ideas from Model Driven Engineering (MDE) [9] to game design. Graph transformations might be used to transform machinations diagrams to graphs outlining interactive missions for level design or vice versa [10]. Similar ideas have been explored by Reyno and Carsí Cubel from a more technical perspective focusing on automatically generating code for games [11, 12]. The techniques typically used in a MDE approach to game development (transformational grammars, UML, Petri nets, and so on) require a considerable effort to use for game designers that do not have a background in software engineering. In fact, one of the biggest challenges to introduce any abstract game development tool, is to convince game designers of its value in the first place.

3 Design versus Engineering

Leaving aside the much longer history of board games, game development is a very young field. The current generation of prominent game developers got to the place they are today because of hard work, entrepreneurship, and bravura. For the early pioneers of the field there were no abstract tools to guide them. Nonetheless, they have created an industry. As a result there is a certain level of animosity towards abstract design tools. A fair number of game designers dismiss design tools because they do not think the tools are effective enough, they fear the tools might actually harm the creative process, or both [13, 14].

[1] Most notably among these are the Game Ontology Project (see `www.gameontology.com` and the 400 Project (see `www.theinspiracy.com/400_project`)

The lack of proven, effective design methods is a serious concern. The current DSLs and design vocabularies for games tend to have a steep learning curve, and are anything but widespread. Although many are designed for actual design work, few have found frequent use outside universities. However, effectiveness is not an argument against design tools, it is a requirement: the effectiveness of a game design tool versus the effort required to master it should be apparent. Typically, this means that using a design tool should lead to better games built in less time, and the time reduction should be more than the time required to learn the tool.

The negative effects on the creative process are trickier to deal with. Many advocates of game design tools argue that they are designed to support the creative process, while many designers experience tools as rigid and restrictive instruments. Although, it can be reasoned that this argument against abstract, theoretical methods for game design stems from a naïve perspective on art and creativity. After all, practitioners of any form of art, from painting and sculpture to cinema and performance, use abstract tools such as the theory of perspective, composition, and editing techniques, to teach and improve their form. However, it is equally important to make clear in what ways game design tools *support* rather than *restrict* game design.

By nature game design is a flexible process. It is impossible to plan and design a game on paper and expect good results from simply building the game as per specification. Too often, what looks good on paper does not work as expected, or simply lacks fun. Gameplay is the emergent result of players interacting with a dynamic system. Until that system exists in some prototype form, it is impossible to say whether or not it works. This means that an important task of the game designer is to spot opportunities as they arise during the development of a game. A famous illustration of this effect is the development of *SimCity* from the editor for another game (*Raid on Bungeling Bay*). While working with the editor, game designer Will Wright discovered that it was much more fun to build the urban environment than it was to fly around it and shoot enemies as the original game intended.

MDE is a suitable approach to deal with the dynamic process of designing games. By capturing different aspects of games in different models and describing how one model can be transformed into a different model allows for a sufficient level of agility: for one game mechanics might be the natural starting point, while for another game interactive missions might be. At the same time, using models forces designers to think about their game at an abstract level. In addition, different models foreground different structures in the game's design; in a way, each model acts as a different lens. Machinations diagrams foreground feedback loops that operate in a game's economy, by building such a diagram, the designer will have to deal with these structures consciously. Daniel Cook's skill chains [6] visualize the dependencies between the skills players must master to successfully play the game: it is a useful lens that should be used to investigate tutorial stages and learning curves in any game.

However, model driven engineering has not been applied to game design extensively. As discussed above, within the game design community there are no widely accepted abstract models for game development, not even for specialized tasks such as level design or interactive storytelling. Without clearly defined, widely accepted models that have a proven positive impact on the development process, it will be hard to convince the games industry to start applying model driven engineering techniques on a larger scale. Yet, it is the argument of this paper that the game development community would do good to investigate the opportunities presented by MDE, as these opportunities could lead to a more efficiently designed games which in turn leaves more room to elevate creative game design to the next level.

4 Applying Models Effectively

Looking at the common problems encountered during game development, problems regarding design and managing game features are the most prominent [15]. Traditionally, the content and scope of a game's design is recorded using a game design document. However, these documents have a poor track record: they are considered to be a burden to create while they are hardly consulted by the development team.

Using generic, abstract models can increase the effectiveness of game development far beyond the current practice of writing game design documents. Clearly defined, generic models are less open to different interpretations by different team members. In the most ideal case, the model's syntax is completely unambiguous. When done right it is even possible to use formal models such as Petri nets [16, 17] to identify structural strengths and weaknesses of a design in an early stage. Although similar effects can be reach with relative simple and informal models such as the skill chain diagrams (see above). These diagrams expose a number of structural characteristics such as the number of connections between individual skill nodes and the relative width and depth of the diagram, which are indications of the relative steepness and length of the learning curve, respectively.

The Machinations framework is a good example a generic, abstract model that foregrounds structural qualities of a game's design. The number of nodes, connections, but also the number of feedback loops in a Machination diagram, are all important indications of the complexity of the game and the quality of the gameplay that emerges from it. The pattern library that is part of the framework catalogues common structures that are indicative of particular dynamic effects. The use of these patterns helps designers to understand and improve the dynamic gameplay. At the same time the patterns are flexible enough to enhance creativity. There are many ways to implement each single pattern, and the number of ways to combine patterns is infinite.

The wide range of levels of abstraction the Machinations diagrams are able to express can be leveraged to create a design strategy. A game designer can start out with a fairly simple model of the game and elaborate on it by slowly

increasing the complexity by replacing simple constructions with more complex ones. The design patterns can be used as a guide for this process. Relative simple patterns can be replaced by more complex ones. Within the Machinations framework this process is referred to as elaboration, and each pattern in the library indicates what patterns it elaborates and by what patterns it is elaborated.

5 Applying Models Efficiently

The application of model driven engineering techniques also creates opportunities to increase the efficiency of the game development process. Generic, abstract models can act as quick, early prototypes that would normally require more time and effort to build. They can be used to simulate games and collect gameplay data at a very early stage. In addition, these models can be used to develop automated design tools that can speed up the process and freeing designers from manual work to focus on those aspects of the development process that benefit most from their creative labor.

These days, all game developers are convinced that prototyping is a critical aspect of developing games. Because gameplay is an emergent result of the game as a dynamic system, the best way to find out whether or not a game works is to build the system and set it in motion. Visual representations such as Machinations diagrams can help identify important structural features that create the emergent gameplay and help designers to shape the gameplay towards a target form, they can never fully replace a play test session using a playable prototype. Fortunately, Machinations diagrams can be executed and allow user interaction when running. This means that they can actually serve as abstract, low fidelity prototypes. At the same time, the diagrams allows the designer to define artificial players and quickly run many simulated play sessions to collect data. This helps designers to balance games in a very early stage. Although, there remains a gap between the model and the real game. Data collected in this way primes designers for gameplay effects that might occur, and already allows them to find out what measures can be taken to counter-act certain unwanted effects.

With sufficient sophisticated models and tools, MDE for game design can be taken much further. Recent experiments with mixed initiative procedural content generation (PCG) [18, 19] sketch the possibility of creating design tools that automate certain tasks to speed up the process of design. Procedural content generation is already being used in commercial games, in most cases to generate levels either during design time, or every time the player starts a new session. The mixed initiative approach to PCG has the computer algorithm collaborate with a human designer in order to get better results. However, PCG techniques are typically tailored towards a specific game. Using generic and abstract models, the same techniques can be applied to multiple games, and hopefully will lead to the development of intelligent, generic design tools. MDE and model transformations seem to be a perfect match for this development as they offer enough flexibility to deal with the agility of the game development process while they are defined clearly enough to be automated [10]. Ideally a game designer will be able to go

back and forth through and make changes to different, connected models of the same game, although it is a considerable technical challenge to create a system that allows a designer to step back to an early model of a game, make changes to it, and automatically reapply all transformations that generated the current state of the game.

6 Discussion

However promising the prospect of applying MDE to game design is, the current state of the art of MDE in game design still faces many challenges. A lack of widely accepted models to represent different aspects of games means that currently no one can expect game designers understand and apply these models. Successful models need to be expressive enough to be able to deal with an infinite variety of games, while still be intuitive to game designers. In addition, they should not require too much effort to master. At this moment is remains unclear whether or not models like this are going to surface soon. No one should expect one single model for the entire process of game development, or every aspect of a game, to emerge. It is better to focus on different models for particular aspects of games and game design. Using multiple models creates a more flexible frame work that and allows designers to adopt MDE techniques one step at a time. However, multiple models also present a problem. It is not always clear how transformations can be defined that allow a model of a certain type to be transformed into a model of another type. For example, a graph based model might be used to express mechanics or missions, but there is no transformational grammar that specifies how a graph can be transformed into a spatial map representing a game level.

MDE can be used to develop generic, automated design tools. These should be build to support designers in their task. Ideally a design tool manages trivial, repetitive task and foregrounds potential structural strengths and weakness of a design. No tool should require trivial, repetitive task from the designer. This allows the designer to focus on the creative tasks. It is critical that the designer stays in control of the creative aspect of the design process at all times. Model transformations are a good way of codifying design strategies for a particular game or a game genre. Using automated design tools, development teams should expect a considerable effort to get the tool up and running. For this reason it is unlikely that the entire process can be automated at once. Small steps should be taken for each new development cycle, starting with steps are more generic and can be easily reused in subsequent projects.

Finally, we have very little experience of applying these techniques to actual games. Currently, many game development studios have settled into particular ways to develop games that are not necessarily suited to MDE. They have set up content pipelines and follow milestone plans. Further research on how to integrate MDE techniques and automated design tools into this process is needed.

References

1. Church, D.: Formal abstract design tools. Gamasutra (1999)
2. Kreimeier, B.: The case for game design patterns. Gamasutra (2002)
3. Björk, S., Holopainen, J.: Patterns in Game Design. Charles River Media, Boston, MA (2005)
4. Koster, R.: A grammar of gameplay: game atoms: can games be diagrammed? Presentation at the Game Developers Conference (2005)
5. Bura, S.: A game grammar. (2006)
6. Cook, D.: The chemistry of game design. Gamasutra (2007)
7. Dormans, J.: Machinations: Elemental feedback structures for game design. In: Proceedings of the GAMEON-NA Conference. (2009)
8. Adams, E., Rollings, A.: Fundamentals of Game Design. Pearson Education, Inc., Upper Saddle River, NJ (2007)
9. Brown, A.: An introduction to model driven architecture. (2004)
10. Dormans, J.: Level design as model transformation: A strategy for automated content generation. In: Proceedings of the Foundations of Digital Games Conference 2011, Bordeaux, France. (2011)
11. Reyno, E.M., Carsí Cubel, J.A.: Model-driven game development: 2d platform game prototyping. In: Proceedings of the GAME ON Conference, 2008. (2008)
12. Reyno, E.M., Carsí Cubel, J.A.: Automatic prototyping in model-driven game development. ACM Computers in Entertainment **7**(2) (2009)
13. Guttemberg, D.: An academic approach to game design: Is it worth it? Gamasutra (2006)
14. Sheffield, B.: Defining games: Raph koster's game grammar. Gamasutra (2007)
15. Petrillo, F., Pimenta, M., Trindade, F., Dietrich, C.: What went wrong? a survey of problems in game development. ACM Computers in Entertainment **7**(1) (2009)
16. Brom, C., Abonyi, A.: Petri nets for game plot. In: Proceedings of AISB. (2006)
17. Araújo, M., Roque, L.: Modeling games with petri nets. In: Proceedings of DiGRA 2009. (2009)
18. Smith, G., Whitehead, J., Mateas, M.: Tanagra: A mixed-initiative level design tool. In: Proceedings of the Foundations of Digital Games Conference 2010, Monterey, CA. (2010) 209–216
19. Smelik, R., Turenel, T., de Kraker, K.J., Bidarra, R.: Inegrating procedural generation and manual editing of virtual worlds. In: Proceedings of the Foundations of Digital Games Conference 2010, Monterey, CA. (2010)