UNIVERSITEIT VAN AMSTERDAM

Languages of Games and Play. Automating Game Design & Enabling Live Programming
R.A. van Rozen

Languages of Games and Play

Automating Game Design & Enabling Live Programming

In game development, the maximum number of game design iterations determines the achievable quality. This thesis explores what informs the design and construction of good games in order to help speed-up game development processes, and create better games more quickly. On the one hand, we study how Domain-Specific Languages (DSLs) can help *automate game design* by offering developers and designers abstractions and notations that raise their productivity, reduce iteration times, and improve the quality of player experiences and a game's source code. On the other hand, we explore how *generic language technology* can be leveraged and developed, in particular for constructing DSLs for *live programming* and automating game design.

The thesis begins with an extensive literature review. We study to what extent *languages, structured notations, patterns and tools*, can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play. We propose the term *'languages of games and play'* for language-centric approaches for tackling challenges and solving problems related to game design and development. Despite the growing number of publications on this topic there is currently no overview describing the state-of-the-art that relates research areas, goals and applications. As a result, efforts and successes are often one-off, lessons learned go overlooked, language reuse remains minimal, and opportunities for collaboration and synergy are lost.

Chapter 2 presents a systematic mapping study to map the state of the art in languages of games and play which contributes the following:

1. A systematic map on languages of games and play that provides an overview of research areas and publication venues.
2. A set of fourteen complementary research perspectives on languages of games and play synthesized from summaries of over 100 distinct languages we identified in over 1400 publications.
3. An analysis of general trends and success factors, and one unifying perspective on 'automated game design', which discusses challenges and opportunities for future research and development.

Our map provides a good starting point for anyone who wishes to learn more about the topic.

The next three chapters focus on *game mechanics*, one of the identified challenge areas. Many games have an *internal economy*, an abstract rule-system that determines player choices and actions that impact gameplay. Using its mechanisms, players face challenges, enact strategies, and manage trade-offs by accumulating, spending and distributing in-game resources (e.g., gems, bricks, or life essence). Unfortunately, game designers lack a common vocabulary for expressing gameplay, which hampers specification, communication and agreement.

The language Machinations has provided a conceptual framework that foregrounds structures and feedback loops associated with patterns of gameplay. We aim to speed up the game development process by improving designer productivity and design quality. We demonstrate how Micro-Machinations (MM), a DSL for game-economic mechanics that evolved from Machinations, can speed-up the game development process by improving designer productivity and game design quality.
First, we show in Chapter 3 how metaprogramming and model-checking technology can be used to formalize a DSL for game mechanics, and analyze and predict qualities of games. Next, we show in Chapter 4 how the game development process can be accelerated and feedback on game design quality can be improved by adapting and improving the game mechanics of running game software. Finally, we show in Chapter 5 how patterns can be used for constructing an interactive game design assistant (a tool) that statically analyzes and generates game mechanics. We have performed this research in continuous collaboration with industry partners in applied research projects on better languages and tools for automated game design.

Chapter 6 addresses a general software engineering challenge, which is also a key challenge for automated game design. Live programming is a style of development characterized by incremental change and immediate feedback. Instead of long edit- compile cycles, developers modify a running program by changing its source code, receiving immediate feedback as it instantly adapts in response. However, little is known about how the benefits of live programming can be obtained for DSLs. We demonstrate how generic language technology can be developed for constructing DSLs for live programming in a two-part solution. The first, leverages origin tracking and text differencing for *textual model differencing* that produces model-based deltas. The second, demonstrates how to leverage these model-based deltas in the design of DSLs that migrate an application's run-time state by *run-time model patching*. We consider these contributions first steps towards supporting the construction of *"live DSLs"* in language workbenches.

Finally, Chapter 7 reports preliminary results on improving tool support for automated game level design. Grammar-based procedural level generation raises the productivity of level designers for games such as dungeon crawl and platform games. However, the improved productivity comes at the cost of level quality assurance. Authoring, improving and maintaining grammars is difficult because it is hard to predict how each grammar rule impacts the overall level quality, and tool support is lacking. We address the lack of tool support by proposing two novel techniques. The first is a novel metric called Metric of Added Detail (MAD) that indicates if a rule adds or removes detail with respect to its phase in the transformation pipeline. The second, Specification Analysis Reporting (SAnR) enables 1) expressing level properties in a simple DSL, and 2) analyzing how qualities evolve in level generation histories. The approach opens a research area for leveraging metaprogramming techniques to address a lack of tool support. The chapter was written as an introductory example, and has the purpose of engaging students as collaborators in future applied research projects to continue this work.