AI-based Game Design and Deployment Options with Proofdoku

Marvin Beese, mabeese@uni-potsdam.de, 786300

University of Potsdam

Abstract. Even though Answer Set Programming (ASP) has many applications in industry and research, it is not commonly used for Game Design. Proofdoku, an extension of the logic puzzle Sudoku, implements its AI-based Game Design with the use of ASP. This summary examines AI-based Game Design and Deployment Options for the use as a web service.

1 Introduction

The widely known Japanese logic puzzle Sudoku serves as a basis for the extension Proofdoku, which implements an AI-based Game Design. Meeting the traditional rules of Sudoku, the user has to solve the puzzle incrementally with explaining the reasoning of the decision by giving hints to the system. The system then checks the validity of the player's arguments and computes hints for various phases of the play. This process enables the user to learn and generalize new deductive inference patterns for Sudoku, especially if the player is not very familiar to Sudoku.
This summary is based on the article 'Answer Set Programming in Proofdoku' by Adam M. Smith [1]. In this summary, I will talk about AI-based Game Design and the decision of using Answer Set Programming (ASP) for it. After that, I will look into the deployment options for this game, more precisely in-process, out-of-process and remote service deployment.

2 AI-based Game Design with ASP

An AI-based Game Design includes an AI-system closely integrated into the core mechanics, aesthetics and the story of the Game Design.
In Proofdoku, the game is designed for letting the user manipulate sets of evidences to convince the AI-system that the reasoning for the decision is valid, whereas in other games like 'Sims', the game design revolves around understanding and manipulating AI driven character behaviors.
ASP is not commonly used for Game Design applications, whereas it has usages for the analysis of game mechanics or for modelling design spaces for procedural content generation. Those areas all share the domain of offline applications, where the interaction with the player is avoided on purpose. Yet, they can yield

design insights and generate datasets for the later impact of the player. Other literatures study systems for live interaction with single user (Butler et al. 2013), where responsiveness but not the scalability for many users is relevant, and also user interaction with ASP driven agents (Compton, Smith and Mateas 2012), where a player can interact with agents by adding and removing constraints.

3 Deployment Options

Proofdoku is engineered as a centralized ASP web service, which has to be playable inside a modern web browser without additional downloads. Responsive gameplay both on Desktop and on low-end smartphones should be possible with the main requirement, that the device needs (in the best case reliable) network access.

For the *in-process deployment*, Clingo's original code could be cross-compiled into pure JavaScript code, so that it would run in the web browser. The possible simplest implementation would be to execute Proofdoku locally in the player's browser, which would benefit in having no centralized infrastructure for solving combinatorial search and optimization problems but would lack in responsiveness, stability hazards and power consumption, especially on mobile devices. As there happens cross-compilation into JavaScript, the program would not work as efficient as native code, but would still deliver a reasonable response time in desktop browsers. Other major problems would be that during solver execution no other scripts would be allowed to work on the page, for the animations to be continued and additional player inputs to be processed. Finally yet importantly, it would be difficult for the game to recover once the browser halts.

For desktop applications, it would be possible to create a child process to execute Clingo. But this is not possible for the execution in a browser, as there do not exist access to the local file system and no possibility for native execution.

For the *out-of-process deployment* a webworker API would be taken into consideration, which could enable invisible background pages for the asynchronous execution of JavaScript code. With this, Clingo would be executed inside a webworker process and communication to the main page would still be possible by passing messages to it. Webworker solutions would put mobile devices into disadvantage, as the responsiveness of the JavaScript solver would be reduced. By choosing either in-process or out-of-process deployment, the user would have to wait for Clingo.js to be downloaded, which holds a size of 5.22 MB. Therefore remote services deployment is a solution.

By running a solver locally and neglecting scalability of the program, one can benefit by responsiveness, especially on mobile devices. Instead, with a *remote service deployment* Proofdoku runs inside virtual machines in the cloud. Therefore, web requests of the remote client invoke native compiled solvers in the virtual machines using Google Cloud Functions. This benefits in avoiding downloads by running Proofdoku native in the web browser and having faster solving speed, which is nearer to native speed. The only requirement is that the device

must make small, periodic requests to a central service, where both desktop and mobile users have the same responsiveness from the AI-system.

4 Conclusion

Proofdoku is an AI-based game, where the implementation of the Game Design consists of ASP, which is not commonly used for that. Furthermore, Proofdoku allows live interaction of the AI-system with the user.
Regarding the deployment, Proofdoku runs as a centralized web service using remote service deployment, which tackles critical problems as responsive gameplay, battery consumption and performance differences between desktop and mobile devices.

References

1. Adam M. Smith, "Answer Set Programming in Proofdoku,", In Proceedings of the Fourth Workshop on Experimental AI in Games (EXAG 4) (2017)