Zenet: Repair of Game Logic using Dynamic Monitoring of Invariant Violations

Chris Lewis
University of California, Santa Cruz
1156 High St
Santa Cruz, California, USA
cflewis@soe.ucsc.edu
http://cflewis.com

Jim Whitehead
University of California, Santa Cruz
1156 High St
Santa Cruz, California, USA
ejw@soe.ucsc.edu

ABSTRACT

In 2008, the cost of developing video games broke the $100 million mark, underlining the scope and complexity of developing modern games. This complexity makes games difficult to test, as there are far too many possible game states to explore. Using a real-time fault monitoring tool to classify and repair unwanted game states reduces the number and severity of user-visible bugs. If game developers used such a tool, the integrity of released games would be improved, ensuring a consistent, high-quality gaming experience.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.1 [**Software Engineering**]: Requirements/Specifications

Keywords

Runtime software-fault monitoring, rule engine, video games

1. INTRODUCTION

With the release of Grand Theft Auto IV in 2008, the cost of developing video games reached $100 million [1], underlining the scope and complexity of modern game development. From many perspectives, this complexity makes games some of the most technically sophisticated software engineering projects today, with systems combining game engines, physics engines and artificial intelligence. However, the testing processes employed in the games industry remain traditionally low-tech, employing human testers, sometimes numbering in the hundreds, to manually play through the game, attempting to expose and reproduce bugs by exploring as many game states as possible [6, 5]. We believe such QA processes are unscalable, eventually requiring legions of testers in order to achieve any useful coverage of the number of game states.

We propose a different solution: checking the game state at run-time for undesirable characteristics, and performing repairs if necessary. To perform such a task requires a *run-time software-fault monitoring tool.* Our project, Zenet, aims to build a robust, scalable, monitoring tool using a rule engine to specify violations of game design invariants. Whereas game rules buried within a game implementation are inside lengthy, complex code, we can instead specify concise, human-readable rules about what constitutes a fail state, along with repairs. We believe that such a system can ensure the integrity of a game by enforcing a specification of the game design, reducing the time and expense necessary to test games while improving the quality of released games.

This abstract will outline the technical problems associated with video game bugs, an explanation of why related systems do not address these problems, as well as a description of our intended research approach and current progress. We end with our intended evaluation methods and conclusion.

2. TECHNICAL PROBLEM

While games have the same bugs as other software domains , two particular characteristics of common video game bugs make debugging games difficult: bugs are often *stateful* and *cross-cutting.*

Stateful bugs require monitoring a game state over time in order to ascertain whether a bug has occurred. Analyzing the game state at any one point would not reveal the bug. For example, in *Super Mario World* [**?**], when Mario jumps, it is expected he must come down. A possible bug may allow Mario to hover in the air. Any single inspection of the game state would reveal Mario at a valid position, but only by monitoring the state over time can we ascertain that Mario hasn't returned to the ground as expected.

Cross-cutting bugs are issues that only appear across various subsystems. The most important aspect of cross-cutting bugs is that they can occur even if the subsystems themselves are semantically correct. For example, given a bug-free physics engine, an explosion could block the only doorway out of the room. If the player has no means with which to clear the doorway and leave the room, this is a game design bug. Although the physics engine functioned correctly, the game has reached an undesirable game state.

Such bugs are difficult to detect using automated methods or code inspection, which is why the games industry relies on manual testing. A secondary effect of manual testing is that there is no verifiable specification of how the game operates, leaving game designers, who are often not programmers, unable to ensure that the game operates as they expect.

3. RELATED WORK

To our knowledge, there is only one other application of runtime software-fault monitors to games, implemented by DeLap et al. [2]. They investigated using a monitor as a form of cheat detection, verifying the integrity of a player-to-player transaction in an online game, with the monitor running on the game server. This research, while an important first step, does not address the entire spectrum of possible video game bugs, nor does it investigate the scalability or user-friendliness of their system, which we believe are important to the video game domain.

Zenet shares similarities with Java-MaC [4] (the same system used by DeLap et al.) and DB Rover [3]. All three systems use dynamic analysis with temporal logic to allow programmers to specify rules that analyze state over time.

[1] The developers of MaC have performed much interesting work on how to monitor and instrument programs efficiently, using various filters on automatically-inserted monitoring points to reduce overhead on the monitor itself. Zenet will not attempt to deal with efficiency at such a low-level. We instead plan to investigate how to make Zenet scalable by looking at how different rule files could be swapped dynamically, evaluating high priority rules more often than those of a lower priority.

DBRover operates on a separate validation server, monitoring databases for data integrity. DBRover can capture data for analysis at a later date. We intend to build upon these ideas in our scaling of Zenet, investigating how data logs of in-game events in an online game could be processed overnight by a separate server, generating warnings or fixes during scheduled maintenance periods.

4. RESEARCH APPROACH

We propose creating a runtime software-fault monitor using a rule engine. Rule engines, such as Jess [**?**] and Drools [**?**] are mature technologies deployed in enterprise environments. Rule engines are optimized to evaluate conditions quickly, as well as offering tools such as Complex Event Processing, which provides temporal logic. We intend to use Drools to create Zenet, as Drools has a strong focus on writing rules that code-illiterate experts can read, which will allow game designers to verify the rule sets.

Our first step is to define a taxonomy of common video game bugs. Such a taxonomy will guide the development of Zenet to ensure it can cover the range of possible game bugs. We then plan to build a monitor for a popular, computationally-intensive open-source game in order to evaluate Zenet's effect on game frame-rates, and analyze its flexibility as an embedded system. Finally, we will implement Zenet as a distributed server process to be used for online games, illustrating the scalability of the approach. We anticipate needing to design a dynamic rule analysis policy, evaluating high priority rules more often, in order to function quickly enough on the large data sets that such games create.

5. CURRENT PROGRESS

Our current progress has been focused on preliminary studies using Drools as a monitor, with encouraging results. We have instrumented an open-source Java clone of *Super Mario World* called *Infinite Mario World* [**?**], and are able to successfully monitor game events and repair bugs with no human-observable performance loss.

[1]Cut these paragraphs?

6. INTENDED EVALUATION

Evaluating Zenet will take three forms:

- An investigation into the scalability of Zenet as an embedded system by monitoring frame rate changes.
- Using Zenet to run on game logs from an online game. The game logs can be simulated or real.
- User studies requesting game designers to verify rules written by programmers, then analyze how effectively game designers can write their own rules.

We believe this provides coverage of the various important facets of Zenet.

7. REFERENCES

[1] Bowditch, G. Grand theft auto producer is godfather of gaming. April 2008.

[2] DeLap, M., Knutsson, B., Lu, H., Sokolsky, O., Sammapun, U., Lee, I., and Tsarouchis, C. Is runtime verification applicable to cheat detection? In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2004), ACM, pp. 134–138. Available from World Wide Web: http://dx.doi.org/10.1145/1016540.1016553.

[3] Drusinky, D., and Fobes, J. L. Real-time, on-line, low impact, temporal pattern matching. In *7th World Multiconference on Systemics, Cybernetics and Informatics* (2003), pp. 345–348.

[4] Kim, M., Viswanathan, M., Kannan, S., Lee, I., and Sokolsky, O. Java-mac: A run-time assurance approach for java programs. *Formal Methods in System Design 24*, 2 (March 2004), 129–155. Available from World Wide Web: http://dx.doi.org/10.1023/B:FORM.0000017719.43755.7c.

[5] Sloper, T. Testers – the unsung heroes of games, October 2009. Available from World Wide Web: http://www.sloperama.com/advice/lesson5.htm.

[6] Starr, K. Testing video games can't possibly be harder than an afternoon with xbox, right? July 2007.