Performance Evaluation of User-Created Open-Web Games

Navid Ahmadi
Faculty of Informatics
University of Lugano
Lugano, Switzerland
navid.ahmadi@usi.ch

Mehdi Jazayeri
Faculty of Informatics
University of Lugano
Lugano, Switzerland
mehdi.jazayeri@usi.ch

Alexander Repenning
Computer Science
Department
University of Colorado
ralex@cs.colorado.edu

ABSTRACT

The rise of HTML5 and Web browsers' execution performance has led to the emergence of several open-Web games developed by professional developers but not by end users. To create their games, end users require higher level development environments and domain-specific languages which impose execution performance overhead. This overhead becomes a critical factor in determining whether the Web can be used as a hosting platform for end-user programming of computer games. In this article we present the performance evaluation results of user-created games developed using AgentWeb, an open-Web game design environment for non-programmers. Our findings show that Web is a hospitable environment for executing games built using high-level game design environments .

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; D.2.8 [**Software Engineering**]: Metrics

General Terms

Languages, Measurement, Performance

Keywords

Performance Evaluation, End-User Programming, High-Level Languages, HTML5

1. INTRODUCTION

With the continuous improvement in Web browser technologies, the Web is becoming a gaming platform. Web is interesting for game developers due to its inherent properties: cross-platform execution, world-wide availability, and cloud-based deployment. However, serious gaming was not possible on the Web due to the lack of essential gaming requirements of graphics and interactivity. As a result, games were delivered to the Web using proprietary rich interactive applications (RIA) frameworks such as Flash [1].

SAC'12 March 25-29, 2012, Riva del Garda, Italy.

Lately, with the emergence of HTML5 standards, including Canvas as a native graphics element in the browser, and improvement in the JavaScript execution engines, professional Web developers have started developing open-Web games. However, creating open-Web games has remained inaccessible to typical Web users with no programming skills. End users require design environments and programming languages with higher levels of abstraction that usually cause execution overhead at run time. Among several challenges to support online end-user game design, the critical one is whether end-user designed games can be executed as open-Web applications despite the introduced overhead. Therefore, the introduced overhead becomes critical when deciding whether the Web can be a platform for end-user game design.

In this paper, we present our findings regarding the described challenge by evaluating the performance of user-created games using an open-Web game design environment called AgentWeb, which enables end users to create games with simple logic, often categorized as casual arcade games such as Tetris, Frogger, PacMan, Space Invaders and Sokoban. Our findings show that the Web is hospitable for executing a certain range of complexity, which is usually good enough for running the type of interactive games that are built by end users.

2. AGENTWEB: ONLINE GAME DESIGN

AgentWeb is an online game design environment targeted for end users. It offers end users a development environment, i.e., an IDE, to create the game objects, draw their depictions, program them, and put them together in the game scene, all from inside the Web browser. AgentWeb's game design model is based on AgentSheets, a desktop-based end-user game authoring tool [5]. AgentWeb IDE consists of four main panes:

- *Agent Gallery* maintains the game objects as a list of agents. Each agent consists of one or more depictions that are used when drawing the game scene.
- *Image Editor* lets users draw depictions for the agents.
- *Programming Environment* lets users program the agents using a visual programming language.
- *Scene Editor*, also known as *worksheet* in the IDE, lets users create the game scene by instantiating the game objects and executing them. The worksheet is organized as a rectangular grid of cells. Each cell can contain zero or more agent instances.

Game	Number of Agents	Scene Size (cells)	Depiction Size (pixels)	Screen Size (pixels)
Sokoban	86	13x9	32x32	416x288
Frogger	∼290	20x13	32x32	640x416
Space Invaders	∼325	15x17	32x32	480x544
Game of Life	6144	96x64	8x8	768x512

Table 1: Benchmarked Games

To support the end users, AgentWeb employs known techniques from the fields of human-computer interaction and end-user programming [2] such as direct manipulation, visual programming, and interactive programming.

AgentWeb is targeted at end-users, rather than professional programmers. It, therefore, provides a high-level visual programming language that lets users define the behavior of agents using a drag and drop interface. The game model is agent-based and the agent-behavior language is rule-based. The behavior of each agent is defined by a set of methods. Each method consists of a set of rules. Each rule is a composition of instructions that are either conditions or actions. When a method is called at runtime, the rules are tested sequentially. The first rule with all of its conditions being satisfied will execute all its actions and the method returns immediately.

The visual program of agents is translated to JavaScript in order to execute natively inside the browser. The code translation is performed dynamically and incrementally. The compilation process is completely transparent to the game designer, i.e., in the user interface there is no button to save/apply the changes made in the program. The compiler is written in JavaScript. Therefore, there is no communication with the server during compilation. By leveraging dynamic client-side incremental compilation with no server communication involved, we achieve low compilation times that are imperceptible to the game designer.

Client-side dynamic compilation blurs the line between programming and execution by enabling *interactive programming*, an end-user programming technique that facilitates game programming. Accordingly, end users are able to modify the agent program while the game is running and the changes are immediately applied into the execution. Interactive programming provides immediate feedback on agent behavior at run time. A typical game design use-case of interactive programming is in tuning the speed of moving objects in the game. In the absence of interactive programming, a game designer would have to recompile the program manually after a modification and re-run the game to explore if the agent behavior is satisfactory.

AgentWeb contains a runtime system that executes agent instances and renders them in the game scene. It contains agent instances which contain the compiled JavaScript code, a game engine which handles the execution of agents, a user input handler suitable for games, and a graphics engine which renders the agents in an HTML5 canvas element. The game engine maintains the list of agent instances that are being executed, executes the agents, receives the user input, e.g., mouse clicks and keystrokes, and applies them to the corresponding agent.

After each execution cycle, the graphics engine renders the game scene according to the updated position and depiction of agents. The agents are rendered in a HTML5 <canvas> element that provides native graphics drawing in the Web browser. The graphics engine employs a differential rendering algorithm that updates only those parts of the scene that have been updated during the last execution cycle. The differential drawing algorithm avoids rendering the whole scene, thus reducing canvas drawing costs significantly.

Compared to a one-off game developed by professional programmers, each of these features require data structures and algorithms that impose execution cost at run time. The major sources of overhead are:

- Direct manipulation [7] is implemented using an MVC architecture [4]; the architectural layers of the MVC impose extra execution time that may cause a delay in interactive response time. For example, when the model of the agents or the scene changes, there is a delay until the change is displayed to the user in the application.
- The visual program is interpreted or compiled to JavaScript in order to be executed in the browser. Even though AgentWeb compiles the code rather than interpreting it, which is several times faster in execution, the compiler unfolds each construct of high-level language to several lines of JavaScript code.
- Interactive programming requires dynamic compilation of the code (at run time), i.e., the compilation process runs as the user is programming. We compile the code incrementally and hot-swap the changed code. In this case, the runtime overhead is limited only to the times that the user is programming the game, not when the game is being played.
- Runtime system—consisting of the game engine and graphics engine—is responsible for executing high-level instructions that the end user uses to define the behavior of agents and rendering them in the browser. Each high-level instruction, such as "move this game object one cell to the north", is composed of several JavaScript function calls and possibly causes a game scene update.

3. PERFORMANCE EVALUATION

We implemented three computer games and one mathematical simulation, game of life, and measured their performance. The benchmarked applications and their properties are listed in Table 1. The implemented games are instances of typical arcade games often created using existing end-user programming environments such as AgentSheets, Scratch [6] and GameMaker [3], while we use the game of life as an instance of an application with higher processing and graphics rendering demands compared to games.

The evaluation has been conducted across the latest versions of Web browsers that are capable of running Agent-

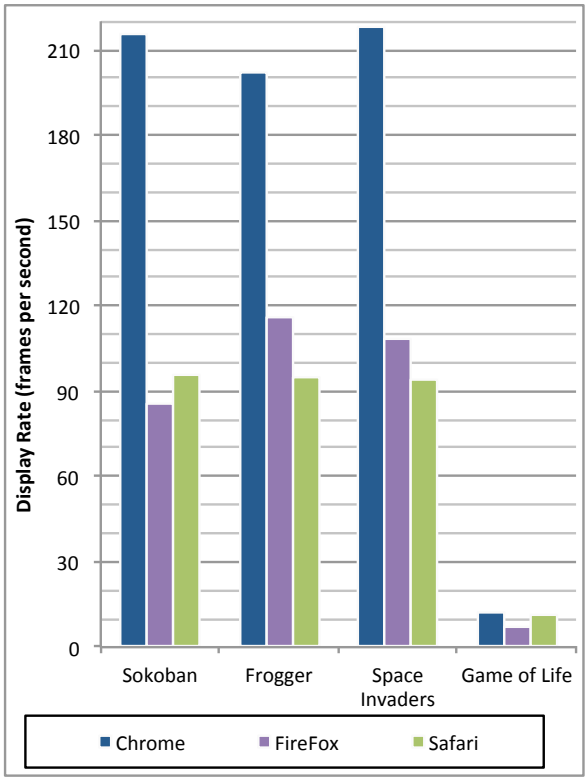

Figure 1: Display rate measured from execution of benchmarks in a number of browsers. The results confirm the feasibility of executing sophisticated interactive applications such as games and simulations as open-Web applications.

Web, i.e., Chrome 13, FireFox 6, and Safari 5. As end users often use only one favorite browser for Web browsing, benchmarking across the browsers is required to compare the state of the art capabilities of browsers in running user-created interactive games. All the browsers were freshly started and only one tab was opened during the performance evaluation.

Each application was executed for at least 30 seconds or 100 execution cycles, to achieve statistically valid results. The mean and standard error values were computed for all the measured parameters to validate the precision of computed values. The measured errors are not significant and may be neglected. All the evaluations were performed using a MacBook Pro with 2.66 GHz Core 2 Duo processor, 4GB of RAM.

To find out whether games created in AgentWeb meet the interactivity required by computer games, we measured the *display rate*, which is the common metric to measure the responsiveness of interactive applications including computer games [8]. By today's standards, 30 frames per second (fps) is the acceptability threshold for playing almost any game in order to satisfy the player. The required display rate depends on the genre of the game. 30fps is required by action games in which the player tracks animated objects. Less interactive games such as card and board games do not have such a strict requirement. In AgentWeb, the display rate is equivalent to the number of times, i.e., *cycles*, that the runtime system executes the agents, as at every cycle a complete interaction with user take place, including applying the user input and rendering the scene in the browser.

Figure 1 shows the display rate for the benchmarked games. Despite the diversity of achieved performance across different browsers, the results confirm that all the benchmarked browsers are capable of executing end-user developed open-Web interactive games beyond the required speed. Game of life runs slower obviously due to its large number of agents as shown in Table 1. However, as a simulation application, game of life does not impose the interactivity demands that games do.

4. CONCLUSIONS

Today's World Wide Web is a hospitable open platform for executing end-user developed computer games. Although the performance varies across different Web browsers, the browsers benchmarked in this paper, i.e., Google Chrome, Mozilla FireFox and Apple Safari, meet the interactivity requirements of casual computer games. Although end users need the assistance of a high-level domain-specific language for creating games, the overhead introduced by language compilation and underlying runtime system are low enough to execute typical casual games using open-Web standards. As the browsers continue to improve, we can expect lower browser overhead, faster JavaScript engines, and faster canvas rendering engines, all of which combine to lead to higher display rates for open-Web games.

5. ACKNOWLEDGMENTS

This research has been funded in part by Swiss National Science Foundation and supported in part by the National Science Foundation. Opinions expressed are those of the authors and not necessarily those of the mentioned organizations.

6. REFERENCES

[1] J. Allaire. Macromedia Flash MX—A next-generation rich client. *White paper, Macromedia*, page 10, 2002.

[2] B. Nardi. A Small Matter of Programming: Perspectives on End User Computing. *MIT Press*, Dec. 1992.

[3] M. Overmars. Teaching computer science through game design. *Computer*, 37(4):81–83, 2004.

[4] T. Reenskaug. Models - views - controllers. *Technical note, Xerox PARC*, 1979.

[5] A. Repenning, A. Ioannidou, and J. Zola. AgentSheets: End-User Programmable Simulations. *J. Artificial Societies and Social Simulation*, 3(3), June 2000.

[6] M. Resnick, B. Silverman, Y. Kafai, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, and J. Silver. Scratch: Programming for All. *Communications of the ACM*, 52(11):60–67, Nov. 2009.

[7] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, Aug. 1983.

[8] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys*, 16(3):265–285, Sept. 1984.