RedWire: A Novel Way to Create and Re-Mix Games

Jesse Himmelstein[1], Mikael Couzic[1], Charlene Jennett[2], Anna L. Cox[2], Raphael Goujet[1], Ariel Lindner[1], François Taddei[1]

[1]Center for Research and Interdisciplinary, 10 rue Charles V, 75004 Paris, France

[2]University College London, Gower Street, London WC1E 6BT, UK

jesse.himmelstein@parisdescartes,fr, mikaelcouzic, raphael.goujet @ gmail.com

charlene.jennett, anna.cox @ ucl.ac.uk, francois.taddei, ariel.lindner @ inserm.fr

ABSTRACT

More and more researchers want to use games as a way of engaging the general public in their research; however game development takes time and requires significant programming knowledge. The goal of RedWire is to enable researchers to create games faster without starting from scratch each time. By encouraging re-mixing and mash-ups, we hope to provide users with an easy way of sharing games and creating variations of games.

Author Keywords

Game design; Game creation; Re-mix.

ACM Classification Keywords

H.1.2. User/Machine Systems: Human Factors, D.2.2 Design Tools and Techniques: Structured programming, D.2.6 Programming Environments: Graphical environments, interactive environments, D.2.13 Reusable Software: Reuse models, D.3.2 Language Classifications: Applicative (functional) languages

INTRODUCTION

Researchers are increasingly using games - as a means of educating players [8], collecting research data [1], and encouraging behaviour change [3]. Game development can take an extensive amount of time, even for experienced game studios. In this paper we present RedWire, a new game engine for the design and creation of games. The goal of RedWire is to enable people to create games faster without starting from scratch each time. By encouraging re-mixing and mash-ups, users are provided with an easy way of creating variations of the same game. Additionally, all games will be open source and available to play and re-mix on RedWire.

RELATED WORK

The inspiration for RedWire comes from the intersection of a number of fields: remixing, browser-based game engines, visual programming, and functional programming.

Remixing is the act of adding to and modifying material from someone else in order to create something new. We draw heavily from insights of Rich Hickey, who is best known as the inventor of the Clojure programming language [4]. His analysis of the conditions that make certain programs easier to compose than others demonstrates the advantages of "untying", or decoupling, program elements from each other. The more decoupled the elements are, the "simpler" the resulting program is, and easier it will to take the program apart and recompose it [5].

Using this metric, Rich Hickey evaluates the simplicity of a number of common programming practices. Two practices that fare particularly poorly from this point of view are mutable state and object-oriented programming, which are almost universally present within game engines. Mutable state is complex because code that reads and or writes it cannot depend on it keeping its value between executions (or even during, in the case of multithreaded programs). Object-orientation, is also complex, because objects contain mutable state, and reference each other directly.

Existing visual programming environments that explore the easy access design space include Scratch [7] and Alice [2] – both allow users to create games and share their projects with others. However we argue that these environments are limited because they are object-oriented with mutable state. Another relevant example is Max/MSP [6] which offers users a dataflow architecture [9] to create interactive audio and video exhibits. This system does not include common functionality for game development, and is rarely used for that purpose.

REDWIRE INTERNALS

The goal of RedWire is to let developers remix games in a *simple* (in Rich Hickey's sense) and visual way, within the browser. We are targeting existing game and web developers who are accustomed to imperative programming languages. We decided to build RedWire as a functional *framework* that executes small blocks of imperative code in a manner that isolates, or "sandboxes", them from directly modifying state. In order to explain both the goals and operation of this novel approach to developers, we chose to compare it to a common prototyping tool used in electronics, the breadboard. Following the electronics

CHI PLAY '14, Oct 19-22 2014, Toronto, ON, Canada
ACM 978-1-4503-3014-5/14/10.
http://dx.doi.org/10.1145/2658537.2661315

metaphor, a RedWire game is made of components called *chips* that are connected to each other on a *board*. A *chip* communicates with the rest of the game via its *pins*. The chip only has access to the values provided via its pins, and cannot store its own state.

In order to avoid the "visual spaghetti" that plagues large graphs, the user doesn't wire chips directly to one another, but to shared *buffers*. There are two kinds of buffers. The *memory buffer* stores "plain-old data", which allows the user to store state and hook up chips to one another. The *IO buffers* enable the purely functional system of chips to interact with the outside world. For example, there is a *keyboard IO buffer* that provides the list of keyboard keys that are currently pressed down, and a *mouse IO buffer* that gives the current mouse position. There are also output IO buffers such as the *canvas* buffer that draws the list of provided shapes to the screen.

USER JOURNEY

When a user finds a game that she would like to remix, she "forks" the game to create her own identical copy. Similar to other live coding environments, the user can keep the game playing while they make changes. Inspired by Brett Victor's work [10], RedWire supports "recording". When the user toggles the recording button and then presses play, RedWire memorizes all the data generated by the input IO services. This includes all input from the player, such as mouse movements, clicks, and keyboard presses. After stopping the recording, any changes the user makes can be retroactively applied to the recording, so that the user can directly see the results of the changes without having to play the game again.

Another feature that allows the user to experiment with the game is *muting*. When a user mutes a chip, it is temporarily deactivated. This draws from the strengths of the RedWire architecture, which ensures that muting the chip does not prevent the rest of the game from functioning.

HACKDAY EXAMPLE

Earlier this year at the Citizen Cyberscience Summit 2014 we included Redwire as a hack day challenge. The person that completed the challenge created a game that he called "Shoot sleeping computers and earn professor hats."

The creator of the re-mixed game had little programming experience. He created this game by re-mixing an existing game called "Stupendous Side Scrolling Space Shooter". Instead of a space ship shooting asteroids, he changed the images so that the player is shooting un-used computer cycles. He added a point system consisting of "professor hat" badges that are won by destroying the spare computer cycles. He explained that his idea was to illustrate the idea of volunteer computing, where volunteers donate their spare computer processing power to be used in scientific research. This example illustrates the potential of Redwire for the design and creation of games.

CONCLUSION

In this paper we have presented RedWire, a new game engine for the design and creation of games. RedWire is the first game engine to adopt an architecture specifically designed for re-mixing and mash-ups. The engine provides researchers with an easy way of sharing games and creating variations of games for comparative study. With further development work, we predict that RedWire will be a useful tool for creating research games. In future work, we will add a number of features to help designers share code across games. We will also integrate advanced metrics, remote game recording, and multiplayer capabilities.

ACKNOWLEDGMENTS

This research was funded by the EU project Citizen Cyberlab (Grant No 317705).

REFERENCES

1. Von Ahn, L. Games with a Purpose. *Computer 39*, 6 (2006), 92–94.
2. Cooper, S., Dann, W., and Pausch, R. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges 15*, (2000), 107–116.
3. Fogg, B.J. *Persuasive Technology: Using Computers to Change What We Think and Do*. Morgan Kaufmann, 2003.
4. Hickey, R. The Clojure Programming Language. *Proceedings of the 2008 symposium on Dynamic languages*, ACM New York, NY, USA (2008), 1.
5. Hickey, R. Simple Made Easy. *InfoQ*, 2011. http://www.infoq.com/presentations/Simple-Made-Easy.
6. Place, T. and Lossius, T. Jamoma: A Modular Standard for Structuring Patches in Max. *Proceedings of the 2006 International Computer Music Conference*, (2006), 143–146.
7. RESNICK, M., MALONEY, J., MONROY-HERNÁNDEZ, A., et al. Scratch: Programming for All. *Communications of the ACM 52*, (2009), 60–67.
8. Ritterfield, U., Cody, M., and VORDERER, P. Serious games: Explication of an oxymoron - introduction. In *Serious Games - Mechanisms and Effects*. 2009, 4–9.
9. Sousa, T. Dataflow Programming Concept, Languages and Applications. *Doctoral Symposium on Informatics Engineering*, (2012).
10. Victor, B. Inventing on Principle. 2012. https://vimeo.com/36579366.