Differentia: Visualizing Incremental Game Design Changes

Kenneth Chang and **Adam Smith**
Design Reasoning Lab
Department of Computational Media
University of California, Santa Cruz
{kchang44,amsmith}@ucsc.edu

Abstract

Small changes in the code and assets of a videogame can have drastic impacts on the player experience. The primary way to discover these impacts is with extensive human playtesting. However, the time and effort expended for playtesting is costly enough that it cannot be conducted after every incremental build of a game. We propose an information visualization technique that approximately reifies a game's space of interactivity as a static image. Considering incremental changes to the game, our visualizations can suggest what has been effectively added, preserved, or removed from the space by those changes. These visual summary reports would ideally be used in the continuous integration (CI) software development process. We demonstrate a prototype implementation of this visualization on two commercial games, considering both minor and major design changes.

Introduction

Editing a single line of code in interactive software can have major impacts on the space of interactivity for the software. Minor edits can quickly shift a game from playable to unplayable, and the impacts of these edits are not readily understandable without thorough testing. Ideally, developers who implement incremental changes to software could directly examine the impacts on interactivity they have made from a visualization of the interactive space. As early as 1997, researchers proposed tools to visualize interactivity in Java games (Storey et al. 1997) and more recently have developed techniques that can visualize interactive stories (Partlan et al. 2019). In videogame development, there is an emphasis on verifying that the code created by a developer corresponds to the intent of what they wanted to create. There is a large gap between proper functioning of individual software modules and the overall player experience.

Because there is no oracle that can fully play and understand games, user researchers use human playtesters to encounter the significant moments of a game, reacting to them as normal players would. During testing, user research teams record these reactions, and compare them to the reactions the developers intended to elicit. Mismatches in reactions are returned back to the developers, who can apply the playtesters' feedback to the game to remedy problems in the game's playable space. While it would be nice to have a complete map of a game's significant moments, playtesting provides only a partial view of what is available in the game, bounded by time expenditure. Using recent human playtesting amplifications tools (Chang, Aytemiz, and Smith 2019), we might gather evidence of the unique gameplay experience a game supports, but the question of how to summarize this space into a compact report remains. This paper takes a step towards giving designers a tool to directly perceive the space that they are designing within.

We contribute Differentia, a technique that visualizes perceptual differences between incremental changes applied to videogames. The technique uses a (high-dimensional) gameplay map of a game's playable moments to build a two-dimensional map of differences between two successive builds of a videogame, isolating differences as visual reports that developers might quickly read and understand. This map is built by recording normal gameplay conducted by human players. We believe that this is a forward step towards improved development tools in continuous integration (CI) development environments, which emphasizes daily incremental improvements in a game's development cycle informed by automatically generated reports of software quality. We have operationalized Differentia in a software prototype and applied it to modifications of *Super Mario World* (for the Super Nintendo Entertainment System) and *Pokemon Red* (for Game Boy). We include visual representations generated by our prototype and analyze its ability to convey design changes.

Background

Overall, our work is intended to advance the practice of quality assurance testing in games. We propose to augment the continuous integration development processes increasingly being adopted in the game industry with automatically generated summary reports resulting from (semi-)automated gameplay exploration tools.

Development feedback

Incremental changes are difficult to visualize, and in game development dedicated User Research teams test the software to ensure that the experience a player has matches the intent of the developers. Traditionally, this process involves

testing the videogame on test audiences to perceive their reactions. Recently, large scale analysis of live gameplay data extracted in real time has augmented and expedited this analysis. Wherever the analysis comes from, the importance of quality user research is paramount, in short: "While no one is in danger of a collision in the world of videogames, there is the real danger of taking thousands of hours of developer efforts (not to mention perhaps millions of dollars) and releasing something substandard to the public." (Collins 1997). Because of this risk, playtesting a videogame is a key component of verifying that the game is heading in the right direction. The game must not only be enjoyable to play, but should also work as intended without bugs (Felder 2015).

This pipeline (Green 2018) of develop-test-report has become commonplace, and many avenues for speeding up this pipeline with automation are being considered. However, the complexity of analyzing human reactions in the user research component of game development does not lend itself to automation. We want to improve this process with a tool that can enhance visualizations of the game in development, not replacing user research with automation, but rather giving user researchers stronger tools.

Continuous Integration

Hand-written reports provide actionable feedback for game developers, but the involvement of another human team adds latency in the feedback cycle. In software engineering, Continuous Integration (CI) (Fowler 2006) refers to the process of automatically running build and testing tools in response to each change to the code or data of a project.

In an ACM Queue article (Coatta, Donat, and Husain 2014), the developers of *FIFA Soccer 1* (released in 2009) describe experiments with automation of the QA testing process. They created scripts that would programmatically provide inputs to the game over time to navigate between different screens of the game, verifying that they were still reachable. While the scripts could be re-run against different versions of the game, the scripts would often need to be modified as the layout of screens or the timing of interactions changed. More recently, in a GDC 2018 talk,[1] the developers of *Call of Duty* described how they integrated lightweight QA tools directly into their CI development flow. In response to each incremental code or data change, a tool directly launches the game into a set of specific scenarios in which it gathers numerical metrics (e.g. memory usage and frame rate) as well as reference screenshots for visual inspection. The strategy of jumping directly to a testing scenario is somewhat more robust than replaying a script of precise input events. However, professional QA workers still contribute to this flow, now asynchronously, by creating and maintaining the list of scenarios that will be represented in the resulting build reports. We imagine an integration of user research and CI, where some elements of traditional user research can be automated to decrease the latency of generating software quality reports.

[1] https://www.youtube.com/watch?v=8d0wzyiikXM

Automated Exploration

So far, the overlap of QA and CI has focused on spot-checking the execution of a game: gathering targeted feedback from precisely specified moments or pathways. In our vision, automated exploration techniques might be used to gather feedback from a much wider variety of moments.

In the development of the puzzle game *The Witness* (released 2016), Walk Monster[2] is the name of an automated reachability testing tool that would teleport the player to every point in the game's world grid and ask the physics engine if the player could walk to every other nearby grid point. The resulting map visualizations highlighted areas both where the player could move too much (quickly pushing into a region of the map that was supposed to only be accessible later in the game) and where the player could move too little (when leftover collision geometry blocked navigation on an apparently free path). Despite the simplicity of the exploration algorithms applied, the resulting reports (visual maps) were immediately actionable because most of the game's overall state was well captured by knowing the player's 2D position on a map.

When data from human playtesters is available, gameplay data amplification techniques might be more appropriate for automated QA purposes. The Reveal-More algorithm (Chang, Aytemiz, and Smith 2019), is based on automatically exploring branches off of a given human demonstration. In particular, this technique teleports the player to scenarios extracted from human gameplay before running less expensive, local exploration techniques. Rather than assuming the player is exploring, for example, a 2D space, some of these techniques summarize the space of play as the latent vector space implied by a gameplay moment embedding function (operationalized as a neural network). Zhan et al. (Zhan, Aytemiz, and Smith 2018) showed that fully-automated exploration can benefit from pre-training this embedding function on past recordings of human play (which might be sourced from a previous version of the game).

Given the open-ended space of automated exploration methods here, our present work is focused on only human gameplay data. To be directly applied in a CI development flow, one of the semi- or fully automated alternatives would need to replace this. Regardless of the source of the gameplay data, we still need to distill the results of that exploration into an interpretable, static report.

Visualizing Design Changes with Differentia

The goal of Differentia's visualizations is to provide an overview of differences in the experiential content present between two versions of a game, and from this overview provide specific details on demand for specific clusters of images. This zoom-in style of communication follows Schneiderman's mantra (Shneiderman 1996) "Overview first, zoom and filter, then details-on-demand," a generalized rule-of-thumb to help encapsulate and deliver information efficiently.

Figure 1 shows one of the generated reports from our visualization tool. It shows differences introduced to the game

[2] https://caseymuratori.com/blog_0005

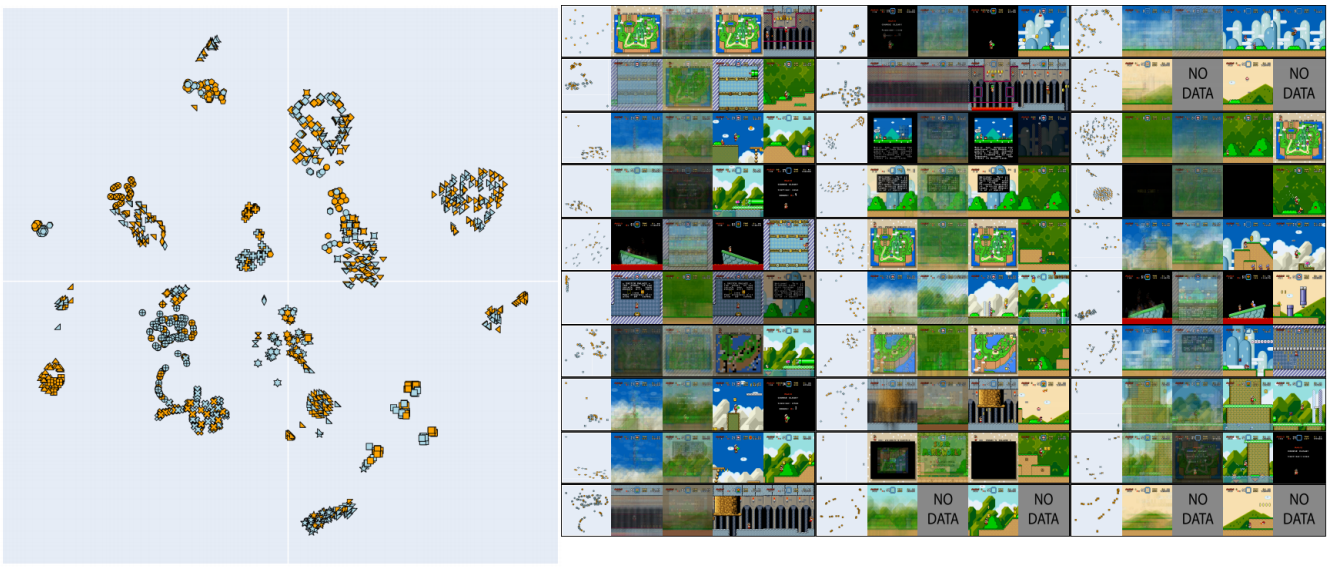

Figure 1: A static report resulting from applying Differentia to a game design change. The left side shows an overview of the moments reachable in the two versions of the game (color coded by version). The right offers filtered views of individual data clusters (zoom for details) including both average and representative sample screenshots from both versions of the game.

Super Mario World by modifying the effect of gravity on the player character (more details on this experiment are given in the next section). The scatterplot on left side of the report offers an overview of the distribution of content seen before and after the design change. Each point in the scatter plot represents a single moment sampled from the gameplay data gathered from one version of the game or the other (again, details in a later section). Clusters of points indicate collections of similar moments. When a cluster consists of data mostly from one specific version of the game, it suggests those moments of gameplay experiences were only possible in one version of the game. That is, they were either removed or freshly introduced by the recent, incremental design change. The overall arrangement of clusters in the visualization, however, is not significant. It represents only our best effort to render the high-dimensional structure of game moments into a two-dimensional chart using modern dimensionality reduction techniques.

Detail views on the right examine one data cluster at a time. Of the five small images present in each row (Figure 2 shows two such rows, enlarged), the first shows a scatterplot of the local structure within the cluster, the next two images show the average appearance of all data in the cluster (grouped by which version of the game they came from), and the next two images sample point representative point (so that details missed in the averages can be examined). Because the grouping of data into clusters is automated in our technique (and thus not perfectly aligned with developer intuitions), the per-cluster visualizations can help discriminate significant from insignificant differences.

Figure 2: The first image (left) shows the local structure of a cluster (using the corresponding glyph from the overview image). The images to the right of the scatterplot visualize (in order) a single sample screenshot from version 1, an averaged image of all version 1 screenshots, a sample screenshot from version 2, and an averaged image of all version 2 screenshots.

Technical Design of Differentia Prototype

To produce the aforementioned visualizations, we made a software prototype of our technique. Differentia is a collection of Python scripts that ingests a collection of screenshots generated from playing each version of the games being compared. From these screenshots, we train an autoencoder model that yields a high-dimensional vector representation of each image. The precise architecture of our autoencoder model is not significant here, and another technique that directly optimized the embedding of game moments to match their screenshots, e.g. a variational auto-decoder (Hinton and Salakhutdinov 2006), would be a fair replacement. The high-dimensional moment vectors are projected onto the 2D plane using t-SNE (Maaten and Hinton 2008), allowing us to render scatterplots. The high-dimensional vectors (not their 2D

projections) are the clustered with K-means to yield (ideally) interpretable groups for focused analysis. The specific implementation here is only one interpretation of the technique. One might consider alternative ways of representing game moments as vectors (see a comparison of alternatives by Zhan et al. (2018)), other ways of projecting high-dimensional moment vectors into 2D for scatterplot display (e.g. UMAP (McInnes, Healy, and Melville 2018)), or other methods of grouping comparable moments for inspection (e.g. spectral clustering (Ng, Jordan, and Weiss 2002)).

Visualization Design

The outputs of Differentia are: (1) an overview map of projected gameplay moments; (2) a collection of images representing the placement of gameplay moments for a specific cluster; (3) a collection of images representing the average image of each version's moments per cluster; (4) a collection of screenshots representing one frame of gameplay from a version per cluster.

Output 1 is our overview map so readers may see the entire space of mapped moments in one image. If they wish to read deeper into a cluster, outputs 2 and 3 provide information about each specific moment, showing the reader the mix of moments from each version as well as what that moment looks like on average. Output 4 gives the most specific detail, exactly one frame of gameplay, so a reader may quickly see exactly what was happening in this moment. These outputs are intended to roughly follow Schniederman's mantra: Overview first, zoom and filter, then details-on-demand (details accessed by zooming an achievable static image rather than interacting with a complex interactive visualization).

Example Applications

To demonstrate potential applications of Differentia, we applied it to modifications of commercial videogames, including changes to both static content and game mechanics. We considered two Super Nintendo Entertainment System games, *Super Mario World* and *GravHack*, and two Game Boy games, *Pokemon Red* and *Pokemon Brown*.

Super Mario World (SMW) is a platformer type game released for the Super Nintendo Entertainment System in 1990 while Pokemon Red is a role-playing game released for the Game Boy in 1996. We obtained copies of these games from Archive.org, and emulate their execution using open source hardware emulators. We use the gym-retro (Nichol et al. 2018) Python library to allow us to interface with the games, giving us the ability to run the modified games, inject controller events, screenshot gameplay, save memory snapshots, and return to gameplay moments saved in the past. This framework gives us the tools needed to capture gameplay in offline records.

While two of our games are commercial releases, two are modifications created by patching the commercial releases. GravHack is a modification of SMW where the effects of gravity upon Mario is halved, causing Mario to be able to jump higher and float down slower. The same modification was used to demonstrate automated amplification of human gameplay data in previous research (Chang, Aytemiz, and Smith 2019). The immediate effects of this change are easier jumping at the expense of being able to deftly control Mario. This modification to the game was made using Lunar Magic 3,[3] an open source ROM editor used by fans to create custom games within the SMW executable ROM. Pokemon Brown[4] is a fan-made content expansion of Pokemon Red, released in 2014. Although game mechanics remains the same, it adds new art assets to the game, as well as new regions to traverse and additional Pokemon to catch. This expansion was added to the game using ROM editing tools, and is emulated in the same manner as Pokemon Red. GravHack presents to us what a developer may implement during an incremental change in the CI workflow. Changing gravity in SMW is similar to a minor change in gameplay balancing, while Pokemon Brown can be seen as a major content release for a game in early access.

Incremental Change: SMW vs GravHack

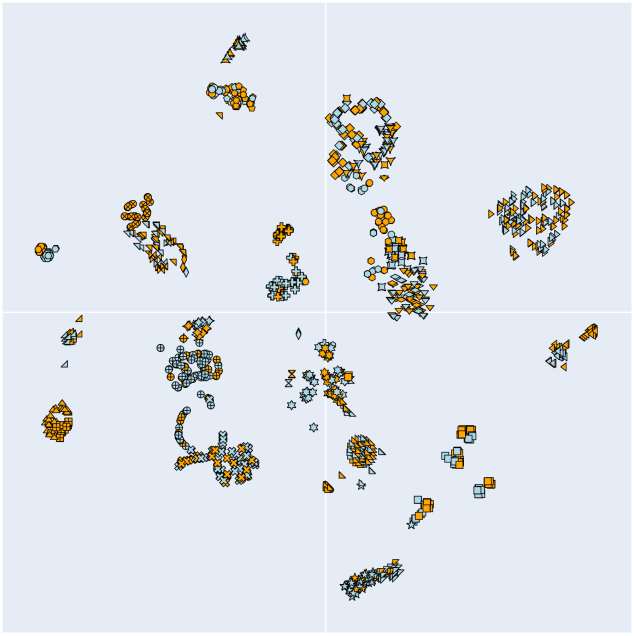

Figure 3: SMW vs. GravHack: Game moments are divided into discrete clusters. Some clusters have strong biases towards one game version or the other, however the clusters with solely moments from one version suggests that this particular moment was present in only one version of this particular playthrough.

In the overview image Figure 3, we observe that distinct modes of gameplay have clustered into distinct groups. While many clusters are populated with moments from both game versions, some clusters contain data from just one version or the other: this is evidence that a numerical change to the game's mechanics has resulted in additions and removals of experiential game content.

[3] https://fusoya.eludevisibility.org/lm/program.html

[4] https://www.romhacking.net/hacks/134/

Figure 4: SMW deletion: In this cluster, a moment was present in SMW, but not GravHack. In GravHack, Mario flies off screen during this cutscene, causing the game to become unresponsive.

A major standout is the cluster seen in Figure 4. This cluster detected that this end card scene was present in the unmodified version of SMW (right) and not in GravHack. During the gameplay of GravHack, this cutscene did not function correctly. Normally, Mario is supposed to jump onto a plunger to demolish the castle, however because the gravity was lowered, Mario jumps off screen and crashes the game. Hence, Differentia did not find any moments in GravHack that matched this moment in SMW.

Figure 5: Most of the gameplay between SMW and GravHack remained the same. Visually, most levels did not change, however the changed user perception of progressing the level was not well captured.

For most other parts of the game, Differentia does not suggest obvious additions or removals. Because the way we embed game moments into high-dimensional vectors in this prototype operates at the time between frames of animation (details in a later section), the most obvious change to the human player (floaty avatar controls) is not represented in the summary report because the player can navigate (floating or not) through almost all of the same game modes and levels. An alternate embedding strategy that was sensitive object velocities could separate floating moments into distinct clusters.

Extensive change: Pokemon Red vs. Brown

In a major content change, Differentia notices a very stark difference in the game's viewable content. In the space of the game itself, Pokemon Brown overhauls the entire game's art assets, including backgrounds, while leaving the layout and most of gameplay the same. We see a formation where many of the clusters from one version are very far away from the clusters of the second. In the center, there is one big cluster of similar game moments. In the game itself, there are many gameplay moments that are the same, such as the

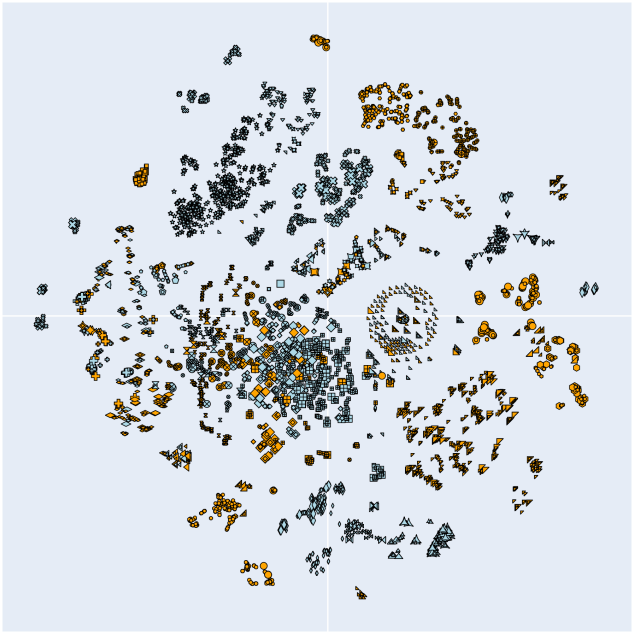

Figure 6: Red vs. Brown: Major content changes show a major shift in moments. Strongly different visual content produces clusters that are placed far away from the other version. Whatever remains similar congregates in the middle.

menu layout, the RPG style top down perspective, and the battle screen. However, many of the art assets have changed, and because of that unique moments present in one game are visualized as absent in the other.

Figure 7: New content: Some moments remained the same, such as the starting house for both Pokemon Red and Brown. Differing moments are illustrated in the bottom row, such as a new Charmander sprite for Pokemon Brown.

In the zoomed in cluster seen in Figure 7, we observe one cluster in which a moment is present in both versions of the game (top row), and one moment present in only Pokemon Brown (bottom row). Overall, many clusters out of the 30 created with K-means were detected to only have moments from either Red or Brown, indicating that the games have very different gameplay moments. In reality, the content of Pokemon Brown is extremely different: art assets have changed, backgrounds have changed, yet gameplay has not been modified. The bottom row of Figure 7 shows a specific moment in Pokemon Brown where this particular bat-

tle scene does not happen in Pokemon Red, demonstrating Differentia's ability to find visual differences. Despite major changes, it is significant that Differentia does not simply report that the entire space of interactivity has been added and removed as a whole—many structures can still be matched up before and after the change.

No change: SMW vs. SMW

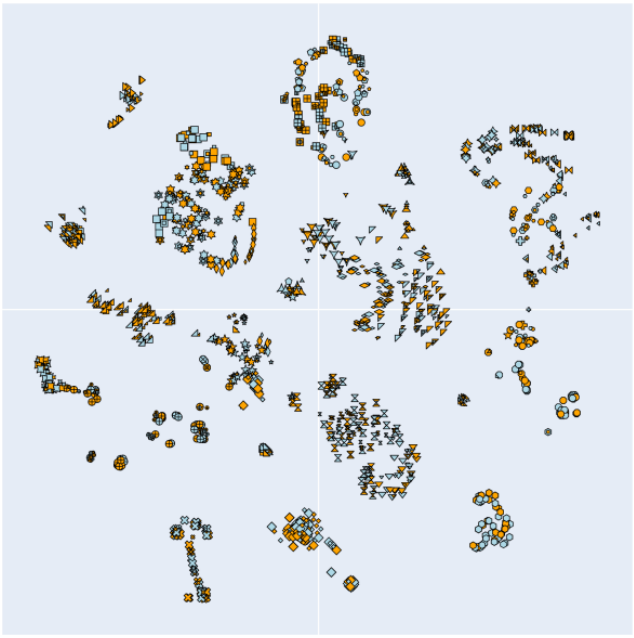

Figure 8: SMW vs. SMW (no changes): All clusters contain moments from both versions, and there are no clusters that contain moments from only one version.

In this experiment, we probe for false-positives in our change-visualization system prototype: we see if it highlights additions/removals when none have actually been made. Extracting two distinct sets of human gameplay data from a single version of the game, and then running the components of our Differentia prototype with different random seeds, it seems plausible that the system might identify and report some differences as an artifact of data collection and analysis despite their being no underlying game design change. At the same time, it would be encouraging to see the system abstract over these incidental differences and report the consistent structure of the unchanging game.

In Figure 8, we see no clusters that predominantly contain moments from one (identical) version of the game or the other. The substructure within clusters is not identical (because, e.g. the human player played through levels in slightly different ways each time), but the system has nevertheless decided to place the corresponding moments together as desired.

Limitations and Future Work

Our current implementation of the technique has two major limitations: it is not fully automated enough for CI workflows (it requires human gameplay for each game version analyzed), and human interpretation is needed to decide if a cluster that has been apparently added or removed represents a design problem. While the goal should not be to eliminate human intervention and interpretation, future work should allow for human effort to be contributed asychronously. Future work should examine how a recording of gameplay in one version of a game can be automatically translated or adapted into an incrementally-changed version of a game. The ability to enhance automated exploration efficiency (of a fixed game) by pre-training on past records of human gameplay data has been previously demonstrated (Zhan, Aytemiz, and Smith 2018), but transfer has not be attempted across incremental design changes to the game. Meanwhile, if a dataset of the gameplay implications of past design changes can be classified as breaking or non-breaking changes, it may be possible to train a system to automatically interpret the results of clustering produced by Differentia. For example, such a system might easily learn that any change that has the effect of removing reachablilty of the credits screen is a breaking change that requires immediate human attention. Finally, we should also consider the body of games with procedural content generation and/or non-deterministic gameplay, since our current approach may focus too much on the differences in normal content generation and not on perceivable gameplay changes.

Figure 9: SMW unchanged: In two human gameplay traces from the early game, we do not see any difference in how the moments were grouped, indicating that Differentia considers these moments present in both (identical) versions.

Conclusion

We have illustrated the Differentia technique and demonstrated its ability to visualize player-perceivable changes in commercial videogames. Differentia represents an effort to make changes to the medium of interactive design directly observable. In our current visualization prototype, we can visually report shifts in gameplay due to incremental changes in game code as well as larger content changes. Our technique has some drawbacks regarding automation and interpretation, however we open up new avenues of research to find ways to better interpret and report the clusters.

References

Chang, K.; Aytemiz, B.; and Smith, A. M. 2019. Reveal-more: Amplifying human effort in quality assurance testing

using automated exploration. In *2019 IEEE Conference on Games (CoG)*. IEEE.

Coatta, T.; Donat, M.; and Husain, J. 2014. Automated QA testing at EA: Driven by events. *Queue* 12(5):20–10.

Collins, J. 1997. Conducting in-house play testing. *Gamasutra*. https://www.gamasutra.com/view/feature/3211/conducting_inhouse_play_testing.php.

Felder, D. 2015. Design 101: Playtesting. *Gamasutra*. https://www.gamasutra.com/blogs/DanFelder/20151105/258034/Design_101_Playtesting.php.

Fowler, M. 2006. Continuous integration. https://martinfowler.com/articles/continuousIntegration.html.

Green, J. 2018. Introducing the quality assurance pipeline. *Gamasutra*. https://www.gamasutra.com/blogs/JoshGreen/20180418/316677/Introducing_the_Quality_Assurance_Pipeline.php.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *science* 313(5786):504–507.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9(Nov):2579–2605.

McInnes, L.; Healy, J.; and Melville, J. 2018. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Ng, A. Y.; Jordan, M. I.; and Weiss, Y. 2002. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, 849–856.

Nichol, A.; Pfau, V.; Hesse, C.; Klimov, O.; and Schulman, J. 2018. Gotta Learn Fast: A New Benchmark for Generalization in RL. *arXiv preprint arXiv:1804.03720*.

Partlan, N.; Carstensdottir, E.; Kleinman, E.; Snodgrass, S.; Harteveld, C.; Smith, G.; Matuk, C.; Sutherland, S. C.; and El-Nasr, M. S. 2019. Evaluation of an automatically-constructed graph-based representation for interactive narrative. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1–9.

Shneiderman, B. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, 336–343. IEEE.

Storey, M.-A.; Wong, K.; Fracchia, F. D.; and Muller, H. A. 1997. On integrating visualization techniques for effective software exploration. In *Proceedings of VIZ'97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*, 38–45. IEEE.

Zhan, Z., and Smith, A. M. 2018. Retrieving game states with moment vectors. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhan, Z.; Aytemiz, B.; and Smith, A. M. 2018. Taking the scenic route: Automatic exploration for videogames. *Proc. of the 2nd Workshop on Knowledge Extraction from Games co-located with 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*.