Finding the User Experience Using YOGURT

Dustin Yost, *Game Engineer, Champlain College,*

***Abstract*—This paper introduces YOGURT, a game engine plugin developed in the field of Games User Research. YOGURT was conceptualized, proposed, built, tested, and iterated upon over the course of 4 months. It's purpose is to aid developers in the process of collecting analytical data representing the player experience. YOGURT is also used to then processed said data, and visually representing it in a game engine so that it is easily digestible by developers. There are few published and easy to access tools which help developers better understand the user experience, a gap YOGURT will hopefully help fill and inspire others to contribute to.**

***Index Terms*—GUR, games user research, unreal engine, tool development.**

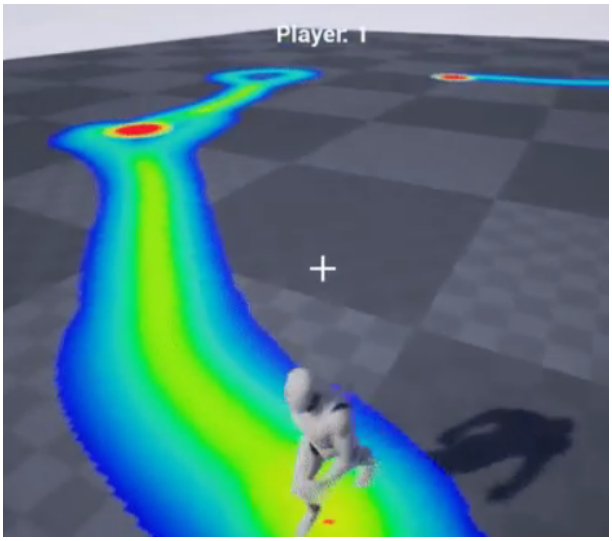

Fig. 1. An early screenshot of YOGURT as it was being tested in a live rendering environment.

I. Introduction

YOUR Own Games User Research Toolset (YOGURT) is a tool developed for the purpose of collecting player experience data, processing it, and visualizing said data in engine. At its core, it's goal is to help game developers better understand the experiences their players have in their game. This article will briefly explore the discipline of Games User Research, the field from which YOGURT was inspired. It will also touch on tool development in Unreal Engine 4, and various takeaways through the development of YOGURT. The most importantly, this article explores how programmers can support developers, within the field of Games User Research. Specifically, using YOGURT as an example, should programmers be involved in the process of better understanding the player experience, and if so, why.

dty
December 06, 2018

II. Related Works

The field of Games User Research (otherwise known as GUR) isnt exactly a new field in game development. In fact, its been around a while. GUR is the practice of analyzing user experience (UX) for the purpose of improving the user experience. That said, it has only been formally recognized as a discipline within the games industry in the last decade [17]. For some studios, user experience research may tend to fall under the umbrella category of quality assurance (QA) testing. After all, it can be more convenient to test the game for bugs and experience all at once. When comparing QA vs GUR testing, however, there is a clear distinction to their intent. Per definition, Quality Assurance testing is meant to test the quality of a game - to find bugs for the purpose of fixing them so that the game is more stable. In contrast, the purpose of Games User Research testing is to collect, analyze, and act upon on data representing the users experience, for the purpose of improving the experience. In short, QA is the practice of making a game bug free, whereas GUR must assume a game is stable and looks at how to give players the most enjoyable experience.

Games User Research can come in many forms. There are some companies, like Player Research, who have departments solely dedicated to GUR testing and analysis, and have even looked at best practices when creating a GUR lab [9]. Many studios use surveys as one way to gather knowledge on the player experience [1]. Surveys are by far the easiest to set up, but have pitfalls due to the reliance on player memory and transparency. There many are other ways to observe the player experience; recording video of players as they play the game, keeping track of player comments, the Think Aloud protocol, using other inputs like the Hopson Boxes, or utilizing biometrics. [13] [7] [11] All that said thought, one of the most useful forms of data collection involves analytics/telemetry, and requires minimal or no effort on the part of the player. [4] Analytics provide a concrete way for developers to see what players are doing in their game, or what is happening to them, so that (as developers) they can make more informed decisions. At its core, this means gathering data, performing analysis, visualizing the data, reporting on the data, and taking action according to the data. Its all about the data. [4]

Individual companies and engine developers alike have already started integrating analytics into their games and platforms. Unity presented a talk during Unite 2015 which showed off their 3-dimensional heat-mapping in engine [15], not to mention Unity has a whole section of their pipeline dedicated to tracking game analytics. There are individual developers out there who have made analytical tools before, such as Prometheus (a 2D heat-mapping tool) [6]. Prometheus was actually a major inspiration for YOGURT after the concept had

been developed, and there are likely a number of similarities in their pipelines. There are also a number of companies which provide game developers with ways to track analytics outside of specific engines. [14] [5]

The field of telemetry/analytics and biometrics is not without its case studies and research. A case study by A.Drachen looked at the player experience in platformers, focusing on comparing the difficulty of tasks with the errors made in each task. [16] There is a paper on the theoretical mathematics behind how to create heat-maps by binning data with respect to the data vs the time it was recorded. [8] There are tool suites and studies which explore the best way to interpret player emotional state, using technology such as webcams and bpm. [3] A study in the UI field explored the best way to evaluate user interfaces using different evaluation methods. [12] In an article about Gameplay Approachability Principles, Heather Desurvire and Charlotte Wiberg compared GAP with usability testing and focuses on finding the best heuristics to evaluate player experience. [2] Lennart Nacke and Craig A. Lindley explored commonly used, yet vague, terms to describe the player experience and how their definitions affect player research. [10]

III. YOGURT

A. Overview

Your Own Games User Research Toolset (YOGURT for short), is an Unreal Engine 4 plugin which helps game developers analyze player experience using telemetry from their game. It can be used with single or multi-player frameworks, but works best when there is a 2D or 3D space for the player to exist in.

As a vision, YOGURT would have support for any number of custom modules to gather data and visualize it in the editor. This ranges from 2D or 3D heat-maps, event markers, storyboards, or any other kind of visualization that a developer would find useful. As a prototype, however, the current state of YOGURT provides native support for 2 dimensional heat-maps in 3-dimensional space. It also currently provides native support for binning collected data by timestamp so it can be viewed in chunks instead of all together. In Figure 2, you can see an example of player experience represented as the area in which the player traveled through their play session.

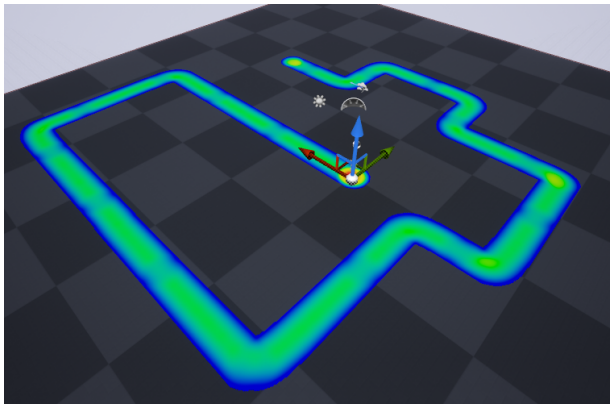

Fig. 2. A sampling of the kind of visual that YOGURT produces when visualizing a 2D heat-map.

B. Technical Overview

YOGURT is divided into two separate modules in the plugin. The first is the runtime environment (YogurtRuntime). This module gets packaged with the game and can be used like any average set of scripts in Unreal. The runtime module contains structures for data points (events or data being tracked in a game) and actors which contain implementations which save this data to disk for later usage.

The second is the editor environment (YogurtEditor). The editor module provides the support for loading data and visualizing it in the editor and it is *not* included when the game gets packaged. This helps prevent unnecessary assets from being added to the build, when players would never even need to see or touch them. It makes builds lighter, and prevents users from experiencing any potential bugs that could crop up in the editor module. The editor module contains implementations for loading data from files, as well as multithreading implementations for loading data. Data collected from games can range from a couple data points (for isolated events like deaths or comments) to thousands or millions of points (for continuous events like tracking player location or speed). It was important when implementing the system that developers weren't blocked from doing work while the data was being loaded and processed, hence the need for multithreading.

C. Implementation: First Steps

This section will assume you have basic knowledge pertaining to how to add a plugin to a project. YOGURT is open source, and as such the latest source code can be found on the github repository (https://github.com/temportalflux/YOGURT). It also assumes the implementer knows how to find the content of the plugin in the content window (any actors or textures referenced come from this directory, under /Heatmap/2D/ if not explicitly stated).

Figure 3 shows the most basic and minimal Unreal Engine level/scene/map (the term "scene" will be used to describe these assets from this point forward). After YOGURT has been successfully installed and compiled, you can create a new scene (or use an old one) and just add YOGURT to it. Figure 4 shows the same scene, this time with the Heatmap2D-DataManager and Heatmap2D-Quad actors added to it. These actors can be anywhere in the hierarchy, but they have been put in a folder labeled YOGURT for organizational and referencing purposes.

In this implementation, only 3 variables have been tweaked from the default blueprints. In Figure 5, the scale of the Quad actor has been set to 50x50x1, and the Size Ratio field has been set to 1x1. This means that in world space, the heat-map takes up 500x500 units on the XY-plane (the actor at unit scale is 100x100 units). The Size Ratio field tells the inner system that the ratio of width to height of the quad is 1:1. If this was a rectangle, the Size Ratio could be 2:1, 1:2, 50:3, or whatever the scale requires to have an accurately scaled rect (not stretched). The last field is labeled "Unit to Uv Scale" in the DataManager actor (Figure 6). This field indicates how

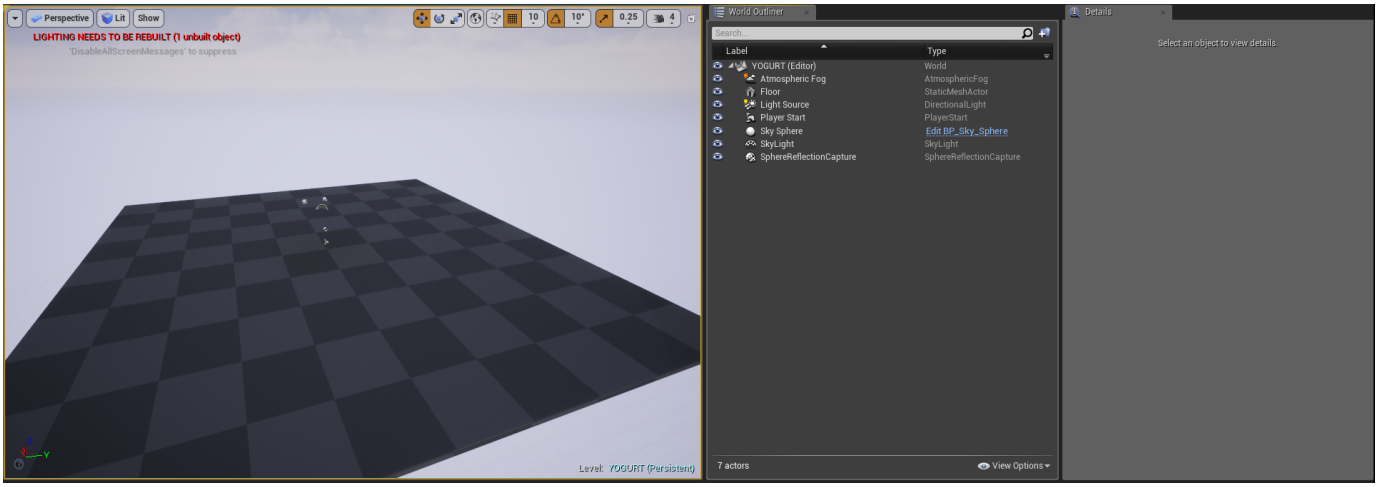

Fig. 3. The default scene created by UE4, without YOGURT.

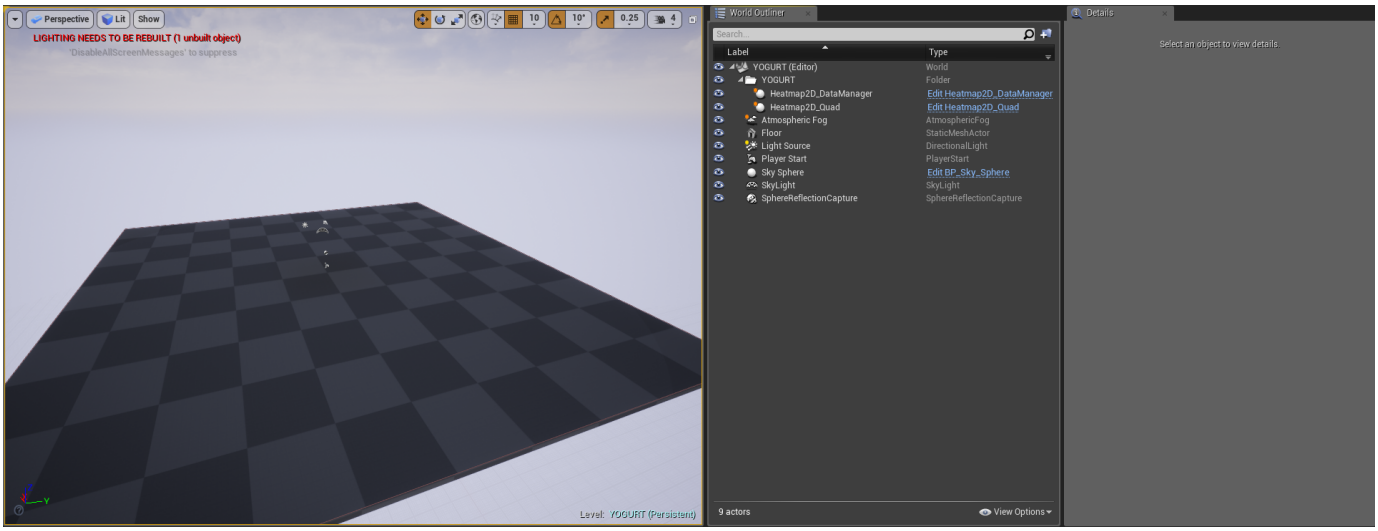

Fig. 4. The default scene, but with the minimally required 2 YOGURT actors, the data manager and a single quad.

scaled up the texture UV coordinate should be for the meshes. Because there is only one quad actor in the hierarchy, and the default texture on which the player position is rendered is 1024x1024 pixels, this parameter has been set to 1024. For our quad, whose Size Ratio is 1:1, the pixel size of the quad will be 1024x1024, thereby taking up the whole texture. As seen in Figure 7, the quad takes up the entire texture space (this actor can be found in the YOGURT content directory, labeled Heatmap2D-GeneratedUVs).

With this implementation, the player can run around in the environment. The default implementation gathers data on the first player by tracking their position every tick. This data can then be loaded into the editor. After running the game, either in standalone, packaged, or in editor, there will be log messages from LogYogurtRuntime stating where the data was written to. In this example, data was written to the default path and timestamp of ”C:/Program Files/Epic Games/UE_4.20/Yogurt/2018.12.08-13.34.39_Heatmap.dat”. The filename was plugged into the ”File Paths” array field, as seen in Figure 8. After inputting this filename, there is a button exposed labelled ”Load Data” in the DataManager (Figure 9). Pressing that button yields the image shown in Figures 2 and 10.

D. Implementation: Deep Dive

YOGURT was created with a modular thought process in mind, and because of this, it already yields much control to developers while still providing some pre-setup systems so that developers can just import and use if desired. The following sections will perform a deep dive into the underlying systems which form the core and native modules for YOGURT.

1) Setup: At its inception, YOGURT was planned to have native support for heat-maps. Therefore, many of the variables in the Heatmap2D-DataManager are framed specifically around 2-dimensional heat-maps of a 3-dimensional world. Each of the following fields can be found in Figure 8. The ”Module Id’ field allows the system to keep track of data files, even if the modules were started at the same time and all have the same file name format. Think of it as a unique identifier that differentiates between different modules. If you

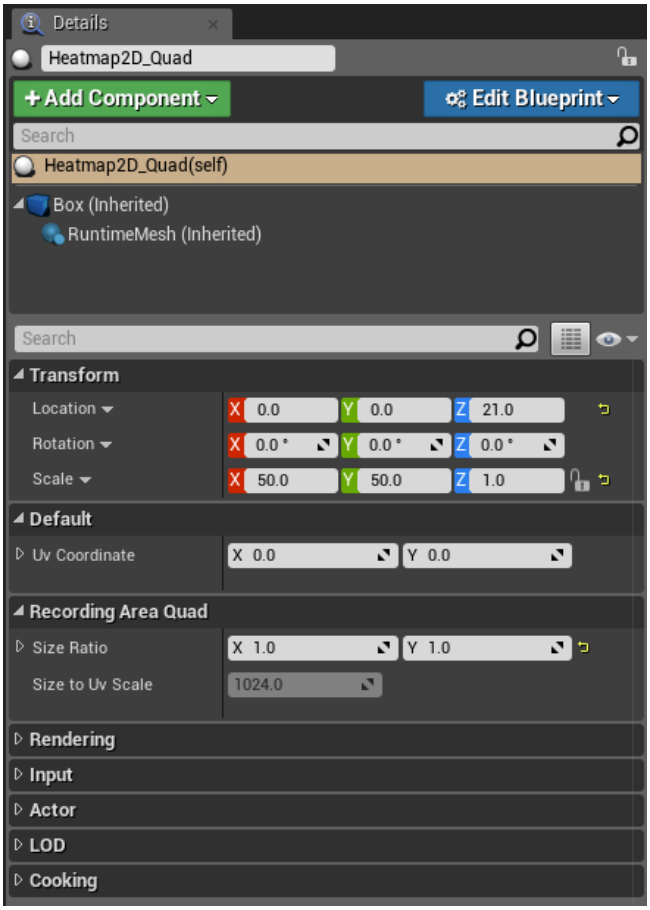

Fig. 5. The details panel for the Heatmap2D-Quad actor.

Fig. 6. The details panel for the Heatmap2D-DataManager actor.

had multiple telemetry modules running at the same time, this is what would differentiate them so they didn't overwrite each others data files. The "Root Data Path" is the folder where the data files will be written to. It defaults to the root directory of the executable, suffixed by a directory labeled "Yogurt". If a command line value named "GurFolder" is supplied, the "Root Data Path" field will be overwritten with that value. The boolean field "Should Record Data" indicates if the system should be recording data or not. By default, this is true, but it can be controlled by providing the command line value named "GurShouldRecord". This is useful in the event that you want to package a product to not record data by default, and either launch specific builds with it enabled or prompt the user to allow data to be recorded on them. The next field labeled "Record Filename Format" is how the system knows how to name data files. By default, it uses a specific time format macro, and a moduleId macro. The time macro is string replaced using the time that the level was opened (specifically when the BeginPlay function is called on this script). The moduleId macro is replaced using the 'Module Id" field mentioned earlier. If the command line value named "GurFilenameFormat" is provided, then the default value is overwritten with the value provided. The field labeled "Render Target" is the texture used to render data points to. This is referred to in the code as the splat-map, as it contains a splat of every data point processed. By default, it is a RenderTarget texture stored in the YOGURT content directory. YOGURT only uses the red channel of this texture. "File paths" is an array of data files to load in. During playtime, these file paths are outputted to the log. The name is actually a misnomer, as these filepaths are relative to the "Root Data Path", and are commonly just file names, unless "Record Filename Format" contains a forward slash ("/") indicating a directory. The "Texture Size" field are the dimensions YOGURT uses to process data that will be outputted to the "Render Target" splat-map, and should be the width and height of the splat-map texture. "Unit to Uv Scale" was mentioned earlier, and it is basically the resolution of the quad splatmap renders. Assuming all quads have "Size Ratio"s relative to one another, this field will scale up their sizes uniformly. In the future, it would be wise to use multiple mip layers or other channels of the splat-map, but currently the system is bound to using an algorithm to map the UVs of quads into one texture atlas. "Amount of Actors to Pack" is a field best left to the value -1. It is mainly a debug variable use to step through the uv packing algorithm. It tells a portion of the C++ code how many quads it should pack into the splat-map. Any remaining quads

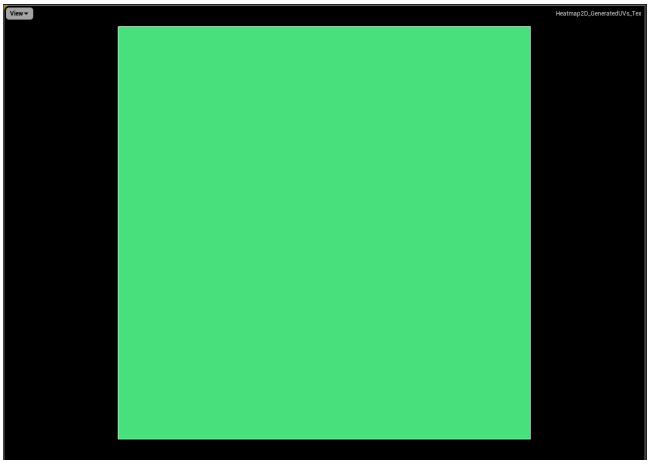

Fig. 7. The Heatmap2D-GeneratedUVs RenderTarget. Take note that the texture is 1024x1024 pixels, and a solid color fills it up. This means that the quad in Figure 4 references the entire texture.

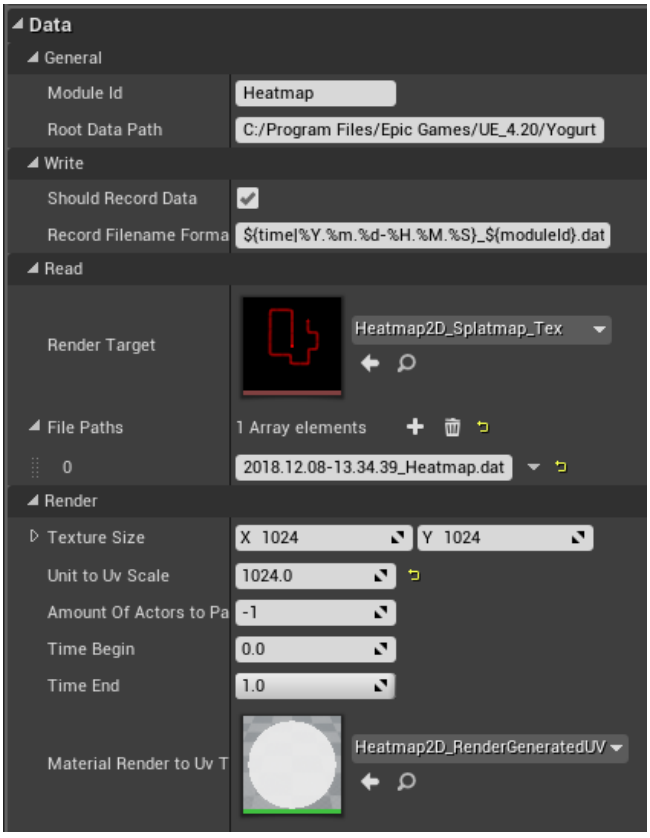

Fig. 8. More details of the DataManager referenced in Figure 6, this time taken after data has been saved and the filename inputted.

are discarded and left without a UV coordinate - which is not good when it comes time for data to be rendered to the splat-map. Finally, the ”Material Render to Uv Texture” field is for rendering the packed UVs to the Heatmap2D-GeneratedUVs texture. This texture is used specifically for debugging the uv packer. You can ignore the fields labeled ”Time Begin” and ”Time End”.

2) Runtime: When a YOGURT DataManager is put into a scene, it listens for the BeginPlay event. For heat-mapping,

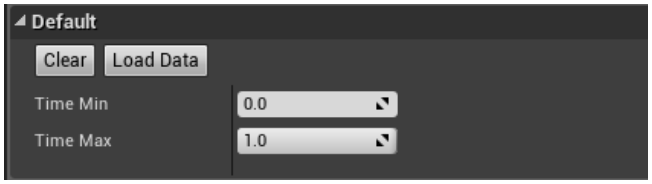

Fig. 9. The section in the details panel DataManager (referenced in Figure 6) for clearing, loading, and selecting data timeframes.

this triggers the system to look for any Quad actors in the scene, and creates UV coordinates for them using a binary-tree packing algorithm. This process is also done every time data is loaded in editor. A Heatmap2D-DataManager will then listen on every tick, check to see if it should record data, and then proceed to grab the first player character from GameplayStatics and use the actors current position to record the relative UV coordinate to the data file. The data is saved using a thread via C++ code. Only one thread is allowed to run at a time, which helps prevent multi-file access. You can see this process happen in the log as the game is run.

3) Visualizing: As seen in Figure 9, there are buttons for ”Clear” and ”Load Data”, as well as fields labeled ”Time Min” and ”Time Max”. These fields all have to do with how data is loaded in the editor. The ”Clear” button will wipe all local data loaded, and clear the render targets for the generated UVs and the splat-map. ”Load Data” will look at the ”File Paths” array, and load every available file specified. Each file is expected to be a binary file with data stored in it from the Runtime phase. The loading of these files is done on a separate thread, similar to how data is written in the Runtime phase. For heat-maps in 2D (the only native module), the thread will then process this data. The process of this data entails looking at each recorded UV coordinate, and drawing a circle around it in the splat-map. Once all the data points have been processed, the thread queues a command to the render thread to copy over pixel data into the splat-map render target. This way, developers only experience a momentary jump in performance when the data is copied, as opposed to rendering data to the texture for every data point, as was done when YOGURT was in early prototyping phases. The ”Time Min” and ”Time Max” fields indicate what time chunks the thread should process from. If the timestamp of a given loaded data point, divided by the last data point’s timestamp, is outside of the range formed from [Min, Max], then the data point is not loaded. Therefore, it is the given assumption that:

$$Min < Max \tag{1}$$

and

$$0 \leq Min, Max \leq 1 \tag{2}$$

IV. Case Study: Capital Vice

YOGURT was built alongside a game in active development called Capital Vice so that the teams could determine what kind of GUR tool would be most useful. As such, YOGURT’s development was influenced by communication with the development team of Capital Vice. During planning, this resulted in the goal to use a heat-map to track player position throughout

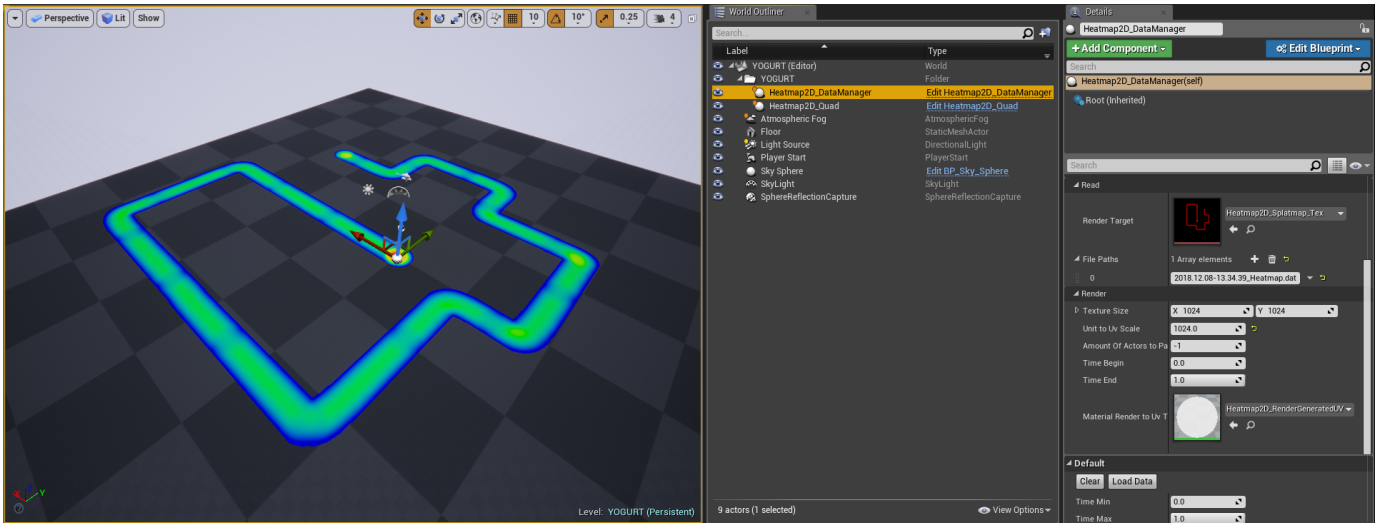

Fig. 10. Another version of Figure 2, this time with the full inspector view

gameplay. Capital Vice was a multi-player networked game, with a playtime of approximately 15 minutes. Once Yogurt was stable enough to start recording data, it was able to gather player telemetry from each play session. Data from these sessions was then used to not only relay reports back to the team, but fine tune how YOGURT worked and its underlying systems. It is valuable to note that the studio in which YOGURT was developed did not have a GUR lab. Instead, data was gathered in the QA lab, which functioned as a hybrid of QA and GUR, for the purpose of finding bugs and getting player feedback on games. This meant that data gathered during testing was not always valuable, and could have be tainted by bugs.

After Yogurt was stable and was able to start gathering data, various bugs were found in how the system processed and rendered data. Over the course of the remaining weeks, YOGURT was iterated upon, bugs patched, and visualization improved. Namely these tweaks varied from moving logic from blueprints to C++, adding multithreading to keep the editor from freezing, reworking the algorithms for how data is processed and rendered, and refactoring the system so new modules would be easier to implement.

V. Discussion

Working on tools for the Games User Research discipline can be time consuming and difficult. Had this tool been developed in isolation, it would not have been as easy to iterate and find bugs. The reality was though, that YOGURT was developed in tandem with a game in the prototyping phase. As such, it allowed development to be informed by lenses of both data science and game development. This style of development also allowed for easy access to be able to internally adjust the tool as the game team needed to.

Starting this process of iteration as early as possible was also critical to YOGURT's success. Gathering the perspectives and goals of the developer team to get a proof of concept working early can make or break a tool. Actually working with a game team allowed both teams to focus in and cater specific data modules depending on what kind of data would be useful for game iteration. Scope and feature creep exist in this realm, just like any other software or development realm. Some features of YOGURT were cut due to time constraints and bugs. As a data scientist, having control of the tool to design, create, and iterate on has been extremely useful. Data science is not the end all be all though. YOGURT would not have been as successful without the skills and insight that come from programmers in the game field. YOGURT would not have been able to rapidly iterate as it did without having programmers embedded on both teams, thus it was critical to have these skills and resources shared.

All that said, the most critical portion to creating any good tool comes down to scalability. When it comes to constructing a GUR tool, it is especially important to be sure that the core of what your working on is modular. Modularity can enable rapid iteration and testing when done right, or cause weeks of delays and refactoring if done wrong. At one point or another, the development team will ask for something the tool didn't account for already. So as a programmer participating and innovating in the GUR field, it is incredibly valuable to have a tool which you can easily add another module to and it still work basically the same.

VI. Future Works

Looking to the future of Your Own Games User Research Toolset, there are a number of things that can be fine tuned and added that would make the tool more useful. Unfortunately, the code base was not always modular. There were portions of the semester in which crunched crept in, and it made the code behind the tool less malleable. It would be great to expose the system and make it easier to work with, so that new modules and ideas are easier to implement (this is a given for any project though). YOGURT would be a great place for other kinds of modules like static event trackers or storyboards. There is even a place for eye-tracking, to see how players interact with UI in games. For the timelines portion of YOGURT, the playback option for developers to see what

players are doing could be huge. Finally, making this all easier to access in Unreals SlateUI system, perhaps a new editor window, would do wonders for developer ease of access - especially for small teams and studios.

VII. Conclusion

Throughout this article, youve just barely scratched the surface of GUR. There are thousands of games user researchers out there, some of whom are published and have written articles or books on what GUR is and how its used. You have read through a journey into GUR tool development, starting with the inspirations to create YOGURT, the development process and hiccups along the way, and finally the tool itself and conclusions found from development. YOGURT is just one way to approach Games User Research. It is one approach to supporting developers and data scientists in the GUR field. As programmers, it is important to remain aware of fields tangential to game development, GUR especially. Programmers have the knowledge to be able to make great tools for other developers and data scientists alike, so utilize that knowledge and make some amazing stuff.

References

[1] F. Bruhlmann, E. Mekler, "Surveys in Games User Research," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 141-162, Dec. 2018.

[2] Heather Desurvire and Charlotte Wiberg. 2008. Master of the game: assessing approachability in future game design. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '08). ACM, New York, NY, USA, 3177-3182. DOI: https://doi.org/10.1145/1358628.1358827

[3] A. Dingli, A. Giordimaina and H. P. Martinez, "Experience Surveillance Suite for Unity 3D," *2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)(VS-GAMES)*, Skvde, Sweden, 2015, pp. 1-6. doi:10.1109/VS-GAMES.2015.7295774 keywords:, url:doi.ieeecomputersociety.org/10.1109/VS-GAMES.2015.7295774

[4] A. Drachen, S. Connor, "Game Analytics for Games User Research," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 333-353, Dec. 2018.

[5] Drachen, A. (2018). *What Is Game Telemetry? - GameAnalytics.* [online] GameAnalytics. Available at: https://gameanalytics.com/blog/what-is-game-telemetry.html [Accessed 7 Dec. 2018].

[6] Justin Gibbs. 2018. *Prometheus*: Games User Research Tool Development Using Best Practices. In *Proceedings of Guildhall Thesis Defense 2018 (Guildhall18)*. ACM, New York, NY, USA, Article X, 10 pages. https://doi.org/10.475/123_4

[7] T. Knoll, "The think-aloud protocol," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 189-202, Dec. 2018.

[8] S. Kumatani, T. Itoh, Y. Motohashi, K. Umezu and M. Takatsuka, "Time-Varying Data Visualization Using Clustered Heatmap and Dual Scatterplots," *2016 20th International Conference Information Visualisation (IV)*, Lisbon, Portugal, 2016, pp. 63-68. doi:10.1109/IV.2016.50 keywords:Data visualization;Space heating;Image color analysis;Clustering algorithms;Correlation;Process control, url:doi.ieeecomputersociety.org/10.1109/IV.2016.50

[9] S. Long, "Designing a Games User Research lab from scratch," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 81-96, Dec. 2018.

[10] Lennart Nacke and Craig A. Lindley. 2008. Flow and immersion in first-person shooters: measuring the player's gameplay experience. In Proceedings of the 2008 Conference on Future Play: Research, Play, Share (Future Play '08). ACM, New York, NY, USA, 81-88. DOI=http://dx.doi.org/10.1145/1496984.1496998

[11] L. E. Nacke, "Introduction to Biometric Measures for Games User Research," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 281-299, Dec. 2018.

[12] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90), Jane Carrasco Chew and John Whiteside (Eds.). ACM, New York, NY, USA, 249-256. DOI=http://dx.doi.org/10.1145/97243.97281

[13] M. Sangin, "Observing the player experience," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 175-188, Dec. 2018.

[14] Usability Testing and Market Research by SimpleUsability Behavioural Research Consultancy. (2018). *What we do - Usability Testing and Market Research by SimpleUsability Behavioural Research Consultancy.* [online] Available at: http://www.simpleusability.com/our-services/games-testing/ [Accessed 7 Dec. 2018].

[15] YouTube. (2018). *Unite 2015 - Custom Events Best Practices and Introduction to Heatmaps.* [online] Available at: https://youtu.be/aWZiB7jX7C0?t=28m [Accessed 7 Dec. 2018].

[16] Rina R. Wehbe, Elisa D. Mekler, Mike Schaekermann, Edward Lank, and Lennart E. Nacke. 2017. Testing Incremental Difficulty Design in Platformer Games. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5109-5113. DOI: https://doi.org/10.1145/3025453.3025697

[17] V. Zammitto, "Games User Research as part of the development process in the game industry," *Games user research. 1st ed. New York, NY: Oxford University Press*, vol. ED-1, pp. 15-30, Dec. 2018.