Unity Heat Map Plug-in

Game Engines E2013

Jakob Melnyk, jmel@itu.dk

December 11, 2013

Contents

1 Introduction

This report is the result of the final project in the Game Engines E2013 (MGAE-E2013) course at the IT-University of Copenhagen, autumn 2013.

The aim of this project was to create a Unity[1] plug-in that allows the user to track events in their game and generate a heat map based on this event data.

On this project I collaborated with Jacob Claudius Grooss (jcgr@itu.dk) on writing the plug-in and testing the performance.

The GitHub repository can be found at https://github.com/esfdk/GameEnginesE2013/.

[1] http://unity3d.com/

2 Project Overview

In most video games, there are many things happening during gameplay. All these things can be viewed as data points (game metrics) useful to developers when it comes to understand their game and how players play the game. Examples of interesting game metrics could be, but is certainly not limited to; player positions during gameplay, where entities died (and how they died), where certain things happened and how long play sessions lasted. These game metrics can be used for finding the right balance, finding bottlenecks in the game and figure out what players find interesting about a game.

Game metrics often consist of many data points and displaying all these data points in a easily understood format is very important. Many game metrics can easily be shown in a table or diagram (such as total play time, favourite weapon or game progress rate), because these are just numbers and do not need a more detailed visual context to be understood.

However, some game metrics are not easily understood without a visual context. Tables and diagrams are good for showing how many times an event happened, but are not well suited for showing details about *where* events happened. This is where heat maps come in.

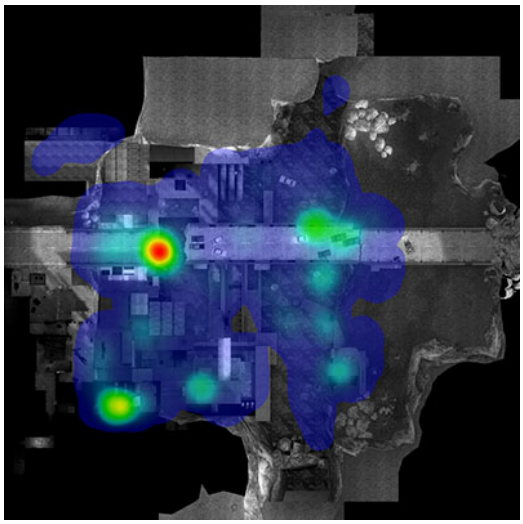

Figure 1: A heat map representing deaths in a level in Half-Life 2

Heat maps are overlays on a level that show how often an event happened at a particular spot. The overlay consists of a semi-transparent object where different parts of the object are painted with different colours. Often the colours range from either dark blue to bright yellow to dark red or from dark blue to dark purple. The further the colour is toward dark blue, the less often an event has happened there. Conversely, the further the colour is toward the dark red/purple, the more often an event has happened there.

Figure **Figure 1**[2] on page 2 displays where players died in a level in Half-Life 2: Episode 2. This

[2]Source: http://www.pentadact.com/2007-11-15-episode-two-death-maps/

can indicate more than one thing; the area might be too hard, players may have missed "the trick" or the area may just be as difficult as intended. In any of these cases, the heat map provides information useful to the developer. If the area is not supposed to be difficult, then an argument could be made for making it easier. If it was supposed to be difficult, then the developers can focus on other parts of the game.

2.1 Goal of the project

Having recently been introduced to Unity for another project (making a game called Hiraeth), Jacob Grooss and I were both interested in doing a plug-in for the engine as our final project. While we did alpha and beta testing on Hiraeth, we felt we lacked information about which part of the game world the player(s) explored. We decided to explore heat mapping plug-ins and only found one: Game Analytics[3] (GA). While the GA plug-in seemed very powerful, we had a lot of trouble actually getting it to work due to problems with their servers. Because of this, we decided to make our own simple heat mapping plug-in.

In Unity all game objects can contain scripts. Scripts must either inherit from MonoBehaviour (most commonly used) or Editor. Any script inheriting from MonoBehaviour can override different event functions (such as OnDestroy, Start, OnMouseUp, etc.) and the Update function. The Update function is called once for every frame. This makes for a quite consistent API in Unity.

The goal of the project was to create a simple plug-in capable of tracking position data at certain intervals (breadcrumbs) and the positions of events when they happen using the consistent API of Unity.

2.1.1 Plug-in features

Because of the elements described in the goal of the project, we decided on four required features for the plug-in:

- **Track anything** - Any kind of event or object should be trackable through the plug-in.
- **Track heat map on per event basis** - The plug-in should be able to generate heat maps based on specific objects and/or events (not just all objects and events).
- **Track anything** - The plug-in should not require too much time and effort to set-up for use.
- **Stand-alone** - The plug-in should not require an internet connection and/or other software/systems besides Unity.

[3] http://www.gameanalytics.com/

3 How To Use

The heat mapping plug-in consists of two parts: tracking objects and generating the heat map.

To access the plug-in, the custom package must be imported. The package consists of 4 folders. *Scripts/HM_Tracker* and *Prefabs/HM_HeatMap* are the elements used by the developer to track events and visualise heat maps. The rest of the files are used internally in the plug-in.

3.1 Tracking game objects

To track a game object, the *HM_Tracker* script must be attached to the object. Attaching the *HM_Tracker* to an object allows the user to change the tracking interval, save interval and toggle which of the default events that are tracked.
By default, the tracker can record the position of the game object when the following events happen: Breadcrumb (object position at intervals), Awake, onDestroy, OnMouseUp and OnTriggerEnter.

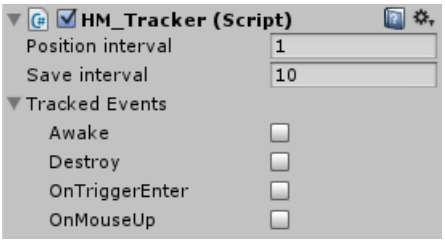

Figure 2: The tracker options.

The OnMouseUp event is only tracked on every call to Update on the game object. Assuming 30 calls to update per second (30 fps), the tracker will register a maximum of 30 mouse up events per second.

The tracking interval determines the minimum interval between every breadcrumb in seconds. This value cannot be less than one second[4].

The save interval is the minimum time between the positions being saved to file[5]. If a game uses a lot of memory, a lower save interval would mean less memory is used, but would also mean that the hard drive is accessed more often.

The tracked data is saved in a file named "HeatMapData DATE TIME.xml" in a folder called "HeatMapData". Tracked data for individual game objects are saved with the file name "HeatMapDataNameOfGameObject.xml" in folder "HeatMapData/DATE TIME/".

Because we track individual game objects in this manner, the names of tracked game objects must be unique. While this issue can be dealt with by the user of the plug-in, this could certainly be an element to improve in a future version of the plug-in.

[4] If the value was lower, it could become a memory hog and take too long to save event info.
[5] Like the tracking interval, this value cannot be less than 1. We felt that saving event information more often than breadcrumbs can be tracked did not make sense.

3.1.1 Custom events

There are two ways to track custom events.

The first way is to create a separate script that calls the AddEvent method of the *HM_Tracker* component on the game object whenever the event happens. This is how we designed custom events to be tracked.

The second way is to modify the *HM_tracker* script, so that the event is tracked from inside the script itself.

3.2 Generating the heat map

A copy of the *Heatmapping/Prefab/HM_HeatMap* prefab should be part of the unity scene hierarchy. The prefab has a script attached that allows for generating heat map visuals. The *Heat Marker* variable of the script should not be changed.

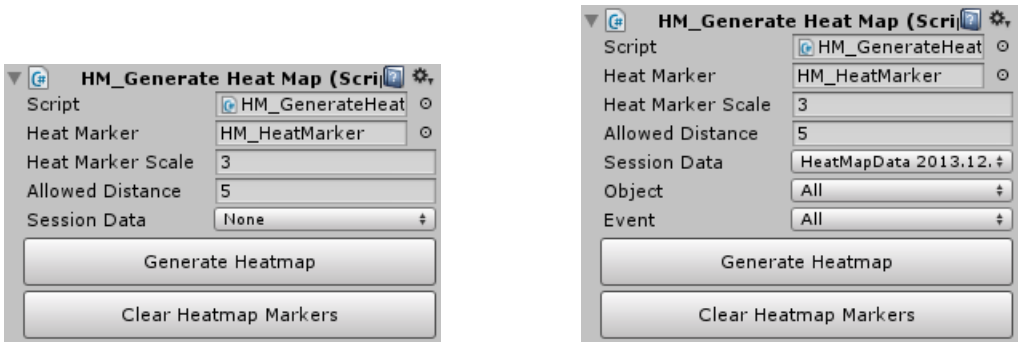

Figure 3: The options for generating heat map. Left picture is without any data selected, the right is with data selected.

Heat Marker Scale determines the scale of the sphere surrounding a tracked event. *Allowed Distance* determines how far away two positions can be from each other before they count as increased density.

Session Data lets the user choose from different datasets in the "HeatMapData" folder. Currently, it is only possible to load data files from the "HeatMapData" folder.
Once a data file has been loaded, lists of the objects tracked in the file and their events appear in the editor. Selecting a specific object and/or event type will only instantiate heat markers for those objects/events in the scene.

Clicking the *Generate Heatmap* button instantiates heat markers based on the selected data file and the game object / event choices. After the objects are instantiated, the density of the objects are calculated and the colours of the heat markers are set based on their density relative to the highest density.

The *Clear Heatmap Markers* button destroys all objects in the scene with the tag "HeatMarker".

3.2.1 Loading data from more files

Currently the only way to load data from more than one file is to load (and generate) the heat map data one file at a time. This is highly inconvenient, if there are large data sets (as there often is with heat map data. I discuss possible solutions to this problem in section 6.2 on page 12).

4 Overview of the Plug-in

The plug-in consists of eight scripts (six behaviour scripts, two editor scripts), two prefabs and one external plug-in (consisting of two scripts).

4.1 Folder structure

The plug-in contains four folders:

- **Editor** contains custom editor menus for the two of the scripts in the plug-in.
- **Plugins** contains two scripts from an external plug-in.
- **Prefabs** contains two prefab objects used in the plug-in.
- **Scripts** contains the scripts for the main logic of the plug-in.

4.2 Prefabs

The plug-in uses two prefabs to render heat maps. The *HM_HeatMarker* is used for marking event positions and the density/heat of these positions in the heat map. The *HM_HeatMap* is essentially an empty game object with the *HM_GenerateHeatMap* script attached.

4.3 HM_Tracker

This script is attached to objects to track (see section 3.1). By default, it tracks the position of the object at different intervals (breadcrumbs) and can be set up to track other events.
The tracker script keeps track of both how often to track breadcrumbs and how often to save tracked events to file. Between every save, the tracker keeps a list of all occured events. This list is reset on every save to reduce the risk of out of memory problems.
Every tracker script also contains its own individual writer used to generate the XML files used for the data sets.

4.4 HM_HeatMarkerScript

HM_HeatMarkerScript contains the logic for calculating the density of a heat marker based on the heat markers around it. Calculating the density of a heat marker is O(n) where n is equal to the amount of heat markers in the scene.

4.5 HM_GenerateHeatMap

HM_GenerateHeatMap is in charge of generating the heat markers for the heat map. It does this in four steps:

4.5.1 Load Session Data

When the Generate Heat Map button in the editor is clicked, the script loads the specified XML data file.

4.5.2 Instantiate Heat Markers

After the data set has been loaded, the script scans through the data set and instantiates heat markers based on the options selected in the editor (game object(s) and event type(s)) and sets their parent to the object containing the *HM_GenerateHeatMap* script (usually the *HM_HeatMap* prefab).

4.5.3 Calculate Density

After instantiating the heat markers, the script makes every heat marker calculate their density and figures out the highest density. This runs in $O(n^2)$ because every heat marker must check every other heat marker to calculate the density. Due to this complexity, generation of heat markers may take a very long time on large data sets.

4.5.4 Set Color

Once the density of all heat markers has been determined, the script calculates the relative density of every heat marker and generates a colour based on this relative density. This operation runs in O(n) time.

4.6 HM_ControlObjectScript

The *HM_ControlObjectScript* makes sure that only one data folder is created for each tracked game session. Additionally, it ensures that the seperate data files for each tracked game object is collected into one data file at the end of the game session. At the start of the game session, a dummy object containing this script is automatically instantiated.

4.7 HM_Event & HM_EventTypes

HM_Event represents an event in the game and contains the logic for how to print itself in an XML-document.*HM_EventTypes* is a helper class that contains the names of the standard event types used in the plug-in.

4.8 Editor scripts

The two editor scripts change the appearance of the *HM_GenerateHeatMap* and *HM_Tracker* by adding drop-down lists to select data files, toggle options for tracked events and buttons for generating and clearing the heat map.

4.9 External plug-in

The external plug-in is only used for a smooth colour transition on based on the density. The plug-in is called ColorX [6].

[6]Source: ColourX package http://wiki.unity3d.com/index.php/Colorx

5 Test

We tested our plug-in on three different games. An example top-down shooter game called Angry Bots[7], an example 2D-platformer game[8] and a first person exploratory game called Hiraeth[9] we made for a different course. We tested these three types of games to see how our heat map plug-in worked on different types of games.

In addition to testing the three different game types, we also the tested the performance of our plug-in. The test was performed on a laptop with 8GB memory, Nvidia Geforce GTX 765M graphics card and Intel Core i7 processor (2.4 GHz) on the Hiraeth game.

5.1 Tested games

5.1.1 Hiraeth

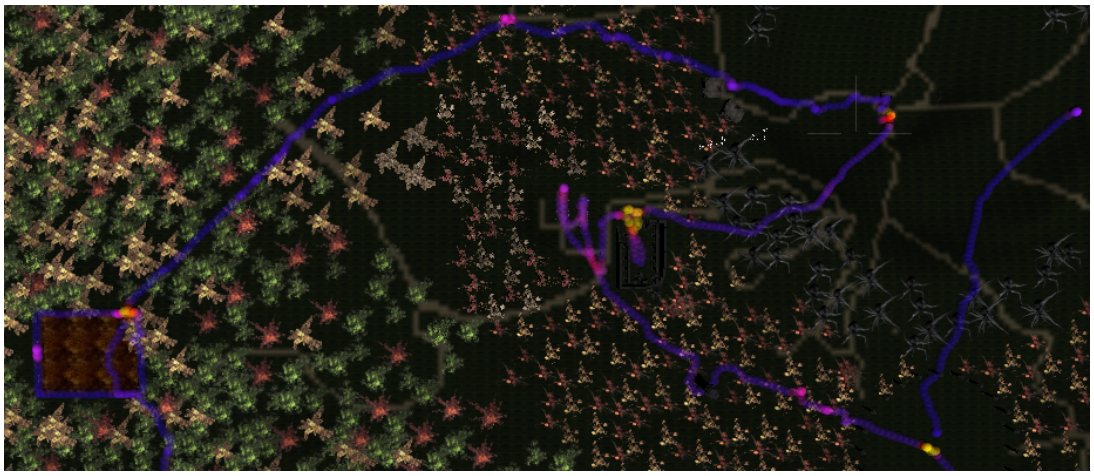

Figure 4: A heat map generated in Hiraeth.

Hiraeth features a relatively large square terrain with different height levels. The game was made for the Game Design E2013 course at IT-University of Copenhagen. The plug-in worked as intended. **Figure 4** shows an example heat map generated in the game.

5.1.2 Angry Bots

The Angry Bots game is a top-down shooter where the player moves around and shoots robots. When we tested, we successfully registered breadcrumb events and custom death events for the player game object. When we generated the heat map, the heat markers were very hard to see due to the transparency (see figure **Figure 5** on page 10). The screenshot on the left has some purple orbs that are very hard to see - those are the heat markers. Giving the the developer control over the transparency of the heat marker material could be an improvement for cases like this.

[7]Source: https://www.assetstore.unity3d.com/#/content/12175
[8]Source: https://www.assetstore.unity3d.com/#/content/11228
[9]https://github.com/esfdk/DarkForest

Figure 5: Angry Bots heat map with different levels of transparency.

5.1.3 Example 2D-platformer

Tracking the player object worked as expected in the 2D-platformer game, but the rendering of the heat map did not. When we render the heat map, we use a transparent shader to render the 3D-spheres. The 2D-platformer does not support the use of the standard transparent shader of Unity, which means that the objects are not rendered.

5.2 Performance test

We tested the performance of our plug-in on our own Hiraeth game. This game is the one we had collected most data on, so we decided we could better evaluate the results when testing with this game.

5.2.1 Tracking/saving event data

We tested how the much the tracking of events impacted the performance of our game by playing with no objects being tracking and with different amounts of objects being tracked.

Results

Test number	Elements tracked	Impact
1	0	N/A
2	10	No noticeable impact
3	20	No noticeable impact
4	70	No noticeable impact
5	150	No noticeable impact

As shown by the table, even with 150 objects tracked at a time (far more than what we would assume is necessary for most games), we could see no impact on the performance of the game. I conclude from these results that the saving/tracking of event data does not need to improved from a performance perspective.

5.2.2 Generating the heat map

To test the performance of the heat map generation, we timed the process using .NET DateTime. We tested this by loading the data set generated by tracking 70 objects in the scene. This data set contained 3224 event entries.

Results

Test number	Events loaded	Time in seconds
1	3224	37.081
2	3224	35.106
3	6448	61.844
4	6448	86.323
5	9672	169.625
6	9672	129.206
7	12896	238.759
8	12896	218.894
9	16120	311.298
10	16120	325.748
11	19344	443.752
12	22568	581.286

As suspected (and discussed briefly in section 4.5.3) large amounts of data slow down the heat map generation process significantly. 22568 events correspond to 6.26 hours of gameplay if breadcrumbs are tracked every second. This is a lot of a single tester, but if 30 people are tested, 6.26 hours of gameplay does not seem like a lot of time.

6 Issues

In this section I describe the various issues with the implementation of our plug-in.

6.1 Heat map generation performance

Because of the $O(n^2)$ operational complexity of the density calculation, the performance of the heat map generation is very poor on large data sets. While this does not make the plug-in non-functional, it makes it very cumbersome to use as intended.
This could possibly be solved by doing large bins (bins being areas in the game world) and then only checking the heat markers in the bins that are within the allowed range of the positon that is currently having its density calculated.

6.2 Loading data from more files

Currently the plug-in is only able to load one data set at a time. This causes a problem combined with the previous point. In our tests, loading data from just the single session of 3324 data points took 35 seconds. If data had to be loaded from many different files, such as from playtest sessions, the process would take an extremely long time.
A possible solution could be to make a seperate file combiner, so that loading would only need to happen once per generation of heat mapping.

6.3 Unique game object names

As I mention in section 3.1, the names of tracked game objects must be unique. If they are not, an input/output exception occurs and data is not properly recorded.
To fix this, I would suggest adding a check to see if there is are other objects with the same name being tracked. If there is, create a new file with the same name except with a number at the end. Because the render check for all events in a node with a specific name, it does not matter if the objects with the same name end up in different XML nodes when combined on game close.

6.4 Lack of Unity 2D-tool support

With Unity 4.3 came highly improved support for 2D-platformer games[10]. As mentioned in section 5.1.3, our plug-in is not very compatible with the 2D tools provided by Unity. Although the positions were tracked, we were not able to render the semi-transparent heat markers used to render the heat map. A possible solution to this problem could be to make the heat markers completely solid.

[10]Source: http://blogs.unity3d.com/2013/08/28/unity-native-2d-tools/

7 Conclusion

The goal of the project was to create a Unity plug-in that was capable of tracking events and rendering a heat map. Based on the results of our testing, I conclude that the plug-in lives up to the requirements we set for the project.

The plug-in is capable of tracking any kind of event on chosen game objects by either using the default events or creating custom events based on the needs of the developer. When generating a heat map, the user is able to select which game object and which event types to generate heat markers for.

As seen in the screenshots from the Hiraeth and Angry Bots games in section 5, the quality of the heat map visuals are not stellar, but I believe they are of an acceptable quality. As mentioned in the Test section, an improvement on this could be to give the user more control over the colours, the transparency and the object shape.

There are issues with the plug-in, as described in section 6 and the focus should be on remedying these problems if more work were to be done on the project.

7.1 Future Work

In addition to the points I describe in section 6, I would consider implementing an option to generate a binned heat map instead of the 3D spherical version we used for this project. Additionally I would implement a method to reduce the amount of elements in the scene when generating the heat map.

Binned heatmap A binned heat map is when the heat markers in the world are predetermined. Every tracked position that falls inside an area (e.g. a 1x1 square) increases the density of that square by 1.

A binned heat map would take less time to generate than our 3D-spherical version. The 3D-spherical version runs at $O(n^2)$ and the binned heat map would be O(n+m), where *n=number of tracked events* and *n = number of bins*.

Binned heat maps are generally viewed in a top-down 2D-perspective. This makes them quite good for games / levels that do not have multiple floors. Figure **Figure 6**[11] on page 14 is an example of a binned heat map.

Reducing number of objects in the scene The current implementation of our heat map plug-in generates one *HeatMarker* per tracked event that is loaded. This means that if there is a lot of data, there will be very many game objects in the scene. This can slow down the Unity engine (if not cause Out of Memory exceptions), which can be problematic.

I would suggest implementing the following method for combining tracked events: If two or more event positions are within allowed range, combine them to one game object and calculate the middle of their positions and use this for the new position.

This would not work in all cases, but it could potentially lead to a very high reduction of the amount of unnecessary objects in the scene.

[11] Source: Game Engines E2013 course Power Point slides: 13_metrics_final.ppt

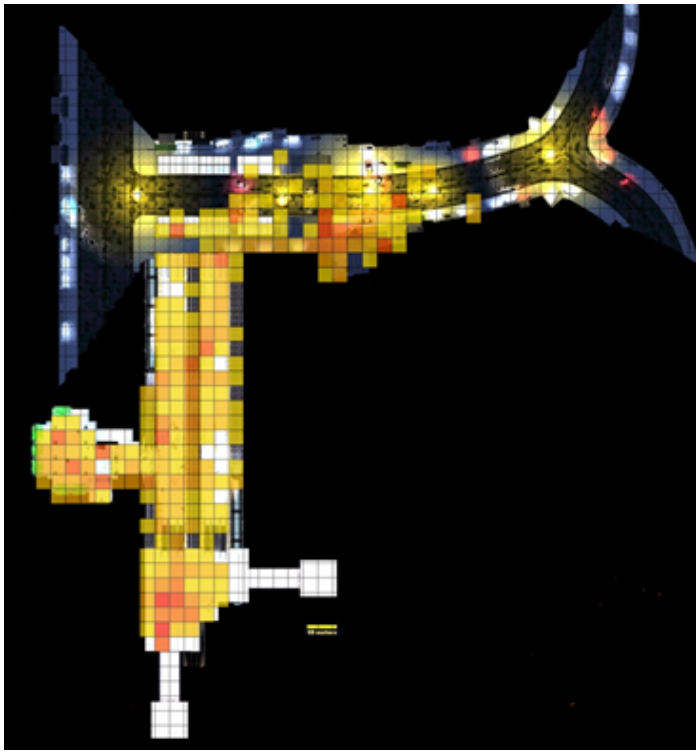

Figure 6: An example of a binned heat map.