Unity Heat Map Plug-in

Game Engines E2013

IT-University of Copenhagen

Jacob Claudius Grooss, jcgr@itu.dk

December 11th, 2013

Contents

1 Introduction

This report is the result of a project in the Game Engines E2013 (MGAE-E2013) course on the Games line at the IT-University of Copenhagen autumn 2013.

For the project, I have worked with Jakob Melnyk (jmel) to create a plug-in for Unity that allows the user to gather data for their games and create heat maps based on this data.

2 Project Overview

When it comes to video games, there is a lot of data available from when people are playing and all of it can be tracked. Data that involves the positions of objects, where entities died (and what they died to), where certain events were performed, etc. can also be very interesting for the developers. It can be used for statistics, influencing balance, finding bottlenecks and figure out what is interesting for the players.

While the data is important, displaying it in an easy-to-understand manner is at least as important. Using an FPS[1] game as an example, showing what weapons the players prefer to use overall, their accuracy with weapons, which enemies are killed with which weapon, etc. can be visualized properly with a table. It works because the data is numbers and the numbers are not necessarily related to where in the world the even takes place.

When it comes to positioning of any kind in a game, a table is not a good choice. A table is useful for saying how many times an event happened, but it is not very helpful when it comes to stating *where* the event happened. "Event e has happened at location x, y, z so and so many times" gives some information, but where in the game is x, y, z?

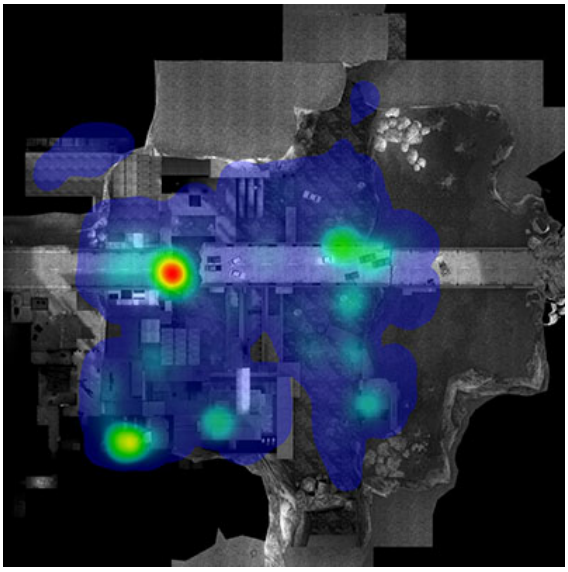

Figure 1: *A heat map representing deaths in a level in Half-Life 2*

This is where heat maps have their place. Using colors, they can show where - and how many times - an event has happened in the game. Heat maps are maps of the different places of the game with colored spots

[1] First Person Shooter

on. The colors of the spots range from an icy blue to a dark red. The darker the color is, the more times a certain event has happened there.

Looking at **Figure 1** on page 1, one place is bright red, which means that players have died a lot at that position. This can either indicate that the place is too hard, that the players have missed something that can heighten their survival, or that the place is working as an intended difficult area. Either way, the heat map gives the developers useful information, which is easy to understand and work with.

2.1 Goal of the Project

As both Jakob and I are rather interested in game balance and mechanics, we decided we wanted to make a tool that could assist in generating heat maps. After talking it through, we decided to create the tool for Unity, for a couple of reasons.

Unity works on a set of rules. Every object in the game world has a position and every object has access to methods that are the same for all objects (such as the Update() method, which is called every frame on every object). This means we can assume that every object can do certain things, and base our tool on that.

Another nice things is that plugins in Unity are actually just collections of scripts and prefabricated objects. As such, it is easy to export and import in another game. Copy the necessary scripts and objects, and you have a working heat map tool for another game.

Unity is also fairly easy to work with, when it comes to creating new tools for it. The API used in Unity is solid, and gives access to basically anything one will need. The documentation and support for Unity is also extensive, making it easy to figure out what one can do and how to do it.

2.2 Features

Having decided to write the tool for Unity, we decided on the features there should be in the tool:

- **Track anything** - The tool should be able to track any object, no matter how simple or complex the object is.
- **Generate heat map on event basis** - The tool should be able to generate a heat map based only on certain events. If all the tracked events are shown at the same time, the heat map will practically be useless.
- **Easy to use** - Finding a tool that seems to cover all your needs, just to figure out that it takes ages to set up properly is never fun.
- **Stand-alone** - Gathering and processing of data should not depend on other systems, or even connection to the internet. It should be able to work as long as the game is running.

These features were the must-have of the tool, as without them it would not be able to properly gather and visualize data.

3 How To Use

As one of our features was that the plug-in should be easy to use, we had a lot of focus on that. We have therefore boiled it down to two parts: Tracking events on objects and generating the heat map.

To use the plug-in, the package (which consists of four folders) needs to be imported. The elements the developer need for for tracking and visualizing are `Scripts/HM_Tracker` and `Prefabs/HM_HeatMap`. The rest of the files are used internally to handle events, inspector UI, marker colors/transparency, etc.

3.1 Tracking game objects

To track an object, the `HM_HeatMap` script must be attached to the object. With the script attached, the developer can change the position tracker interval, save interval and which of the default events that are tracked through the object inspector UI.

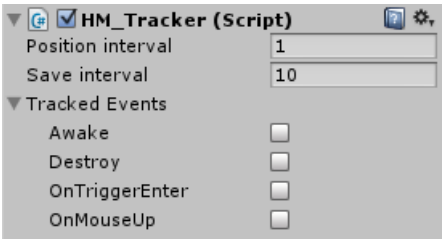

Figure 2: *The tracker options.*

The *Position interval* option determines the minimum amount of seconds between BreadCrumb (see below) events. It cannot be set to less than one second, as setting it lower could possibly become a memory hog and rarely result in any real gains compared to tracking every second.

The *Save interval* option is the minimum amount of seconds between the tracked events being saved to an XML file. It cannot be set to less than one second, as we felt that being able to save more often than tracking the position of an object did not make sense. It is possible, however, if the *Position interval* is set higher than one, but we chose not to limit it more than it is.

The default events the tracker supports are BreadCrumb, Awake, Destroy, OnTriggerEnter and OnMouseUp.

BreadCrumb is the name of the event that tracks the object's position, and is always activated. It is tracked in the Update method, and will be tracked at the chosen position interval.

Awake, Destroy and OnTriggerEnter are tracked every time the method is called on the object. Calling these methods is something Unity handles internally, and we can assume that without lag, these events will be tracked successfully every time they happen.

OnMouseUp happens as part of the Update method, like the BreadCrumb event. The default amount of calls to Update per second is thirty, so this event cannot be tracked more than thirty times per second.

The tracker will save all events that it is set to track, and every time the *Save interval* is reached, these events will be saved to a file named "HeatMapData/YYYY.MM.DD hh.mm.ss/HeatMapDataNameOfGameObject.xml". When the game finishes, all the individual files will be combined to a single file named "HeatMapData/HeatMapData YYYY.MM.DD HH.MM.SS.xml", which is the one that is used for generating the heat map.

Custom Events

While the tracker can track certain events by default, the developer can also use it to track custom events. There are two ways to do this.

The first way is to call the `AddEvent` method on the attached `HM_Tracker` script from another script and pass the event name and position as parameters. This is how we intend for it to work.

The second way is to modify the `HM_Tracker` script itself, to allow tracking of the custom event from inside the script.

3.2 Generating the heat map

To generate the heat map, the `HM_HeatMap` prefab should be placed in the scene. The prefab has the `HM_GenerateHeatMap` script attached, which is what generates the heat map. In the object's inspector, the developer has access to the settings of the heat map script.

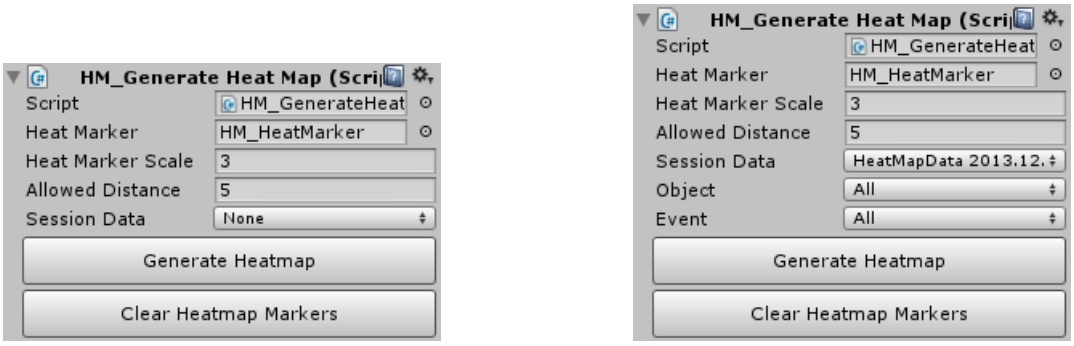

Figure 3: *The options for generating heat map. Left picture is without any data selected, the right is with data selected.*

Heat Marker is a field that contains the element that is used for visualizing the positions where events happened. The field should contain the `HM_HeatMarker` prefab.

Heat Marker Scale determines the scale of the objects that surrounds a tracked event (the *Heat Marker* object). *Allowed Distance* determines how far two heat markers can be from each other and still count as being "near" each other.

Session Data loads a list of the all XML files in the "HeatMapData/" folder and displays them in a dropdown menu. Upon choosing a file, the *Object* and *Event* options are loaded. *Object* contains a list of all the objects that have been found in the *Session Data* file. *Event* shows a list of all the event types that have been tracked for the chosen object.

The *Generate Heatmap* button will use the chosen parameters to generate markers for each tracked event from the chosen data. It will position the markers in the scene, and color them according to how many other markers that are nearby. Multiple combinations can be visualized at the same time, simply by pressing the button with different data selected.

The *Clear Heatmap Markers* button is used to remove the heat map again by destroying all instances of the heat marker object in the scene.

4 Overview of the Plug-in

PHPHPHPHPHPHPH

4.1 Placeholder

PHPHPHPHPHPHPH

5 Testing

5.1 Games

5.1.1 Hiraeth

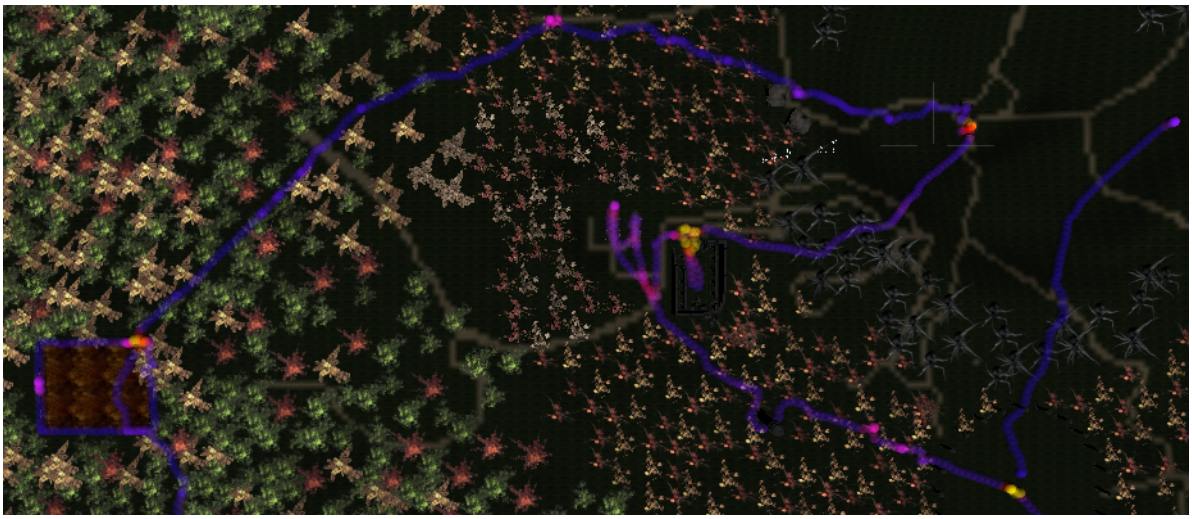

Figure 4: *A heat map generated in Hiraeth.*

5.1.2 AngryBots

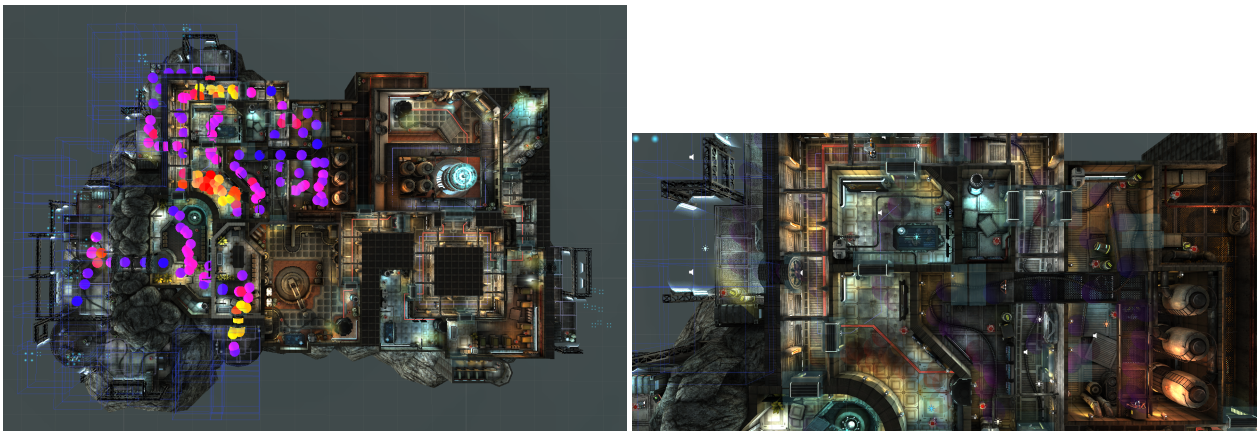

Figure 5: *Derp*

5.1.3 2D Platformer

No pic

5.2 Performance

5.2.1 Saving data

5.2.2 Generating the heat map

6 Issues

While the plug-in works, there are some issues with it.

6.1 Data from multiple files

As the plug-in lets the user choose which file to load data from, it is possible to load data from multiple different files. It is cumbersome, however, as the the user will have to choose a file, load the data, choose the next file, load data, etc. Loading data from tens of files or more will end up becoming an annoyance, rather than a helpful tool.

The solution to this, is to allow the user to select multiple files to load data from. Instead of a drop down menu, a menu that allows the user to select which files to load from would be a lot more user friendly.

6.2 Performance of heat map generation

As described in section 5.2.2, generating the heat map becomes slower the more points that are being loaded and processed.

It becomes slower because the script has iterate over every marker multiple times. The first iteration instantiates each marker and places it in the world, according to the tracked position. The second iteration iterates over all the markers, and, for every marker, counts how many other markers that are near it by checking the distance to every other marker. Lastly, it sets the color of each marker according to the density it had.

Binned heat map

Process data separately.

6.3 Object names when tracking

For each object that is being tracked, the object will create a file named "HeatMapData¡objectName¿.xml" and open an XmlTextWriter to that file. This causes a problem when multiple objects have the same name, as they will overwrite previous files. For example, if ten *Enemy* objects are being tracked, only the last one to be loaded will have a file to write to, and the other nine will lack the file their writer is linked to. Thus only one object will actually be tracked.

The simple way to solve this problem, is to check if the file already exits, and in that case append a number to the end of the new file name. This would leave a lot of "HeatMapData¡objectName¿¡number¿.xml" files in the session folder, but as the user will never have to interact with these files, it should not be an issue.

6.4 Unity 2D tool support

In version 4.3 of Unity, better 2D tools were. As we were testing the plug-in on other games than the project we created it in, we decided to test it on the 2D Platformer demo, as described in section 5.1.3.

We found that the plug-in works when it comes to gathering data and generating a heat map from this data. However, while the heat markers were added to the scene, they were not rendered. We believe it is due to

the material being transparent. Creating a default sphere works, but as soon as its material is changed to transparent, it is also not shown.

One way to solve this issue, would be to create a custom texture that retains some sort of transparency, while still being possible to render.

7 Conclusion