Advanced Project Work

Report

Game Metrics

Supervisors
Mr. Jayson Mackie
Mr. Simon McCallum

Yann Prik
Erasmus Student
Master 2nd Year
Media Technology & Information Security

Summary

ABSTRACT

Metrics in video games are representing every kind of parameter, which interfere or not with what the players are seeing in a game. These metrics can be position of a characters in a level, amount of ammunition available, color of a weapon etc... Or it could also be data that are used while someone is playing but which do not represent something special in the game. The second kind of metrics are very useful to debug game while it's running, while the first kind of metrics are more useful to improve the quality of a game or to get direct statistic coming from the players behavior or structure of any parameter.

To be more concrete, and to speak about the purpose of this project, I had to create a heatmap. A heatmap is directly representing the metrics in function of their spatial repartition on a map, and also in function of their frequency and/or intensity. Hence, to visualize these data it is obvious that we first need to create a database. It should contain a large enough amount of data so as their visualization represent a common player behavior. We are proposing tools which are able to make the link between game metrics and their visualization on a heatmap. One would implement our code in his source code to connect to a website dedicated for heatmap. He would have the choice to display his metrics, and also to apply filters if he only wishes to display a few of them. Moreover, the system is totally dynamic which means he can watch his heatmap while playing to the corresponding game. This solution can be very helpful to improve the quality of maps, or to fix unbalanced teams problems for instance.

This paper is based on *Quake III* metrics, which source code is available for many years on the net, and still possessing a huge fan community.

Introduction:

The video game industry became the two last decades a very important business, currently generating more money than the cinema industry. Most part of games were developed to be available on game boxes or PC, and by companies which size is becoming bigger and bigger. Indeed, nowadays to develop a video game many years are needed, even when there are no financial problems freezing its continuation for a while. It means, in this kind of project, companies can afford paying people to design very specific features. Furthermore, by doing that they can control the quality of the work which normally has to work very efficiently on the global product because especially dedicated for it.

But nowadays, thanks to the great expansion of powerful smartphones and social networks, many small companies were born and are proposing everyone to play their games. They are very different from the games I was describing before for many reasons. These new development teams do not have the same funds and can't afford having too much employees. And even though they have, they are not sure about the future of their games since they are not distributing their products on common platforms and experiencing hard competition with other developers.

We are proposing in this project to integrate into small companies work our game metrics solution. These metrics can be position of a character in a level, the amount of ammunition available, color of a weapon etc... Or it could also be data that are used while someone is playing but which do not represent something special in the game. The second kind of metrics are very useful to debug game while it's running, while the first kind of metrics are more useful to improve the quality of a game or to get direct statistic coming from the players behavior or structure of any parameter. Since the game metrics management can be painful and time consuming to develop, it would be easier for small firms to adopt our system. By integrating our code in their source code, they would allow their application to be easily improved. They can use our project for themselves, or for the players who would like to have some visual statistics and information about the metrics.

In the next parts of this document, we'll describe what were the steps we followed to create the code to extract the metrics, how we populated the database with them, and then how we created a dynamic heatmap. We were using the source code of *Quake III Arena* in order to extract interesting metrics, and a JavaScript canvas of heatmap available on the net to display the metrics. My explanation guideline will follow the direction of the date, from the time they are created – while playing game – to the time when they are stored and then displayed. But it doesn't correspond to the exact methodology we employed to get these results, we will include some extra information about the intern test processed to guaranty each subsystem was working fine.

I/ Quake III metrics

In this part we are concerned about extracting data from the game Quake III Arena, mainly the position since they constitute the elementary metrics to get a heatmap. Indeed, if we have no position it is impossible to represent any data on our heatmap. Then when we have them, we are able to represent couple of data, one couple containing a position and a desired metric. It is very important to notice that we are intervening in this part only one time in our project. When the task is realized, it is un-necessary for us to do it once more – except for debugging – since this is companies' role which would adopt our solution. Indeed, they don't have to communicate their source code to apply the connection, they need to keep a trace over their metrics and use them to be read on our servers. We will see later what kind of data form we are expecting to receive.

a) Test Compiling

The Quake III source code, written in C and some files in assembly language, is available for free on the net since 2005, allowing one to create his own mod and modify each component at will. It can be changing settings like amount of life or ammunition when spawning – appearing – on a map, or totally changing everything like adding some textures or creating a new story.

But the first ineluctable step to overwhelm is compiling the game. Before thinking about working on Quake III, we want to test the system with a simpler game: *Tux Racer*. Indeed, the structure of the game is lighter than *Quake III* one, and it is easier for the next steps to identify and extract every desired metric. It first seems to be very easy to compile a game because new compilers such as *Visual Studio 2010* are quite efficient and easy to use, but in fact it is the most painful and time consuming step of the project. It was the first time we've had to compile a video game, so were thinking to load the source files and let the compiler do. In fact, there were many incompatibilities because many files were missing on the computer. The source code execution is relying on some libraries and function definitions. We were following tutorials explaining step by step how to do so, but apparently errors are always appearing on unexpected situations. Because of the extreme number of errors while compiling *Tux Racer*, and the lack of explanation on the official webpage, we shifted on *Quake III* which possesses a much larger fan community giving tips to unlock delicate situations.

After many days solving problems one by one we finally succeeded to compile the game. It's pretty sure if we have to compile another game, it will be much more quicker, or even very fast since now we know what kind of libraries are required and how to link them correctly to the project. Normally a successful build is creating the necessary files that have to be executed to run the application. With *Quake III*, even though the source code is available for free, it is still necessary to possess an original version of the game, and install it to have the totality of the files required to play. Especially the installation provides a file containing the textures, sounds and many more elements. Hence, the files we can get by compiling the sources have to replace some files provided by the installation, mainly a new executable. Files *qagemex86.dll* and *ioquake3.x86.exe* were 2 out of the 4 files to replace.

```
1>------ Build started: Project: quake3, Configuration: Release Win32 ------
2>------ Build started: Project: game, Configuration: Release Win32 ------
2>  game.vcxproj -> C:\Users\110093.HIG\Courses\ioq3-vs2008\misc\msvc10\..\..\build\game_release\qagamex86.dll
1>  quake3.vcxproj -> C:\Users\110093.HIG\Courses\ioq3-vs2008\misc\msvc10\..\..\build\quake3_release\ioquake3.x86.exe
3>------ Skipped Build: Project: ui, Configuration: Release Win32 ------
3>Project not selected to build for this solution configuration
========== Build: 2 succeeded, 0 failed, 2 up-to-date, 1 skipped ==========
```

Figure 1 : Successful build

The interesting code is located into “game” project, so after each build we have to replace the two underlined files and execute the game to run it. We created a simple windows batch to execute this 3 operations, it is really saving time!
Unhappily, it is not enough to work. We took a special version of the source code available on a SVN server, which is helping people to create new mods. So we have to set up one of their program *ioquake3* in the *Quake III* install folder, as well as the last game patch. Moreover with this program we need to use a special executable and unlock the developer mod by adding some parameters to the target of its shortcut. It was not easy to understand all the previous steps, but such a relief when at last, it is running perfectly!
We can now start to find and extract all the metrics we want.

b) Search and extract

Here comes the time to dig into the gigantesque source code representing Quake III. Ironically I thought it would be the hardest part of the project, because the files are very large and include quite optimized code, not necessarily easily understandable. The first reflex is to start at the main() function, and see which files are called. The aim is to identify the one which is executed while you are playing. Indeed, by including inline code in this specific file, we are sure that our new functions are read while playing. We’ve located a file where some basic elements are defined such as the number of ammunition when first taking a weapon. As shown in the example (fig2) bellow we can choose the value of ammunitions, from 10 to 25 for instance.

```
/*QUAKED weapon_rocketlauncher (.3 .3 1) (-16 -16 -16) (16 16 16) suspended
*/
    {
        "weapon_rocketlauncher",
        "sound/misc/w_pkup.wav",
        { "models/weapons2/rocketl/rocketl.md3",
        NULL, NULL, NULL},
/* icon */      "icons/iconw_rocket",
/* pickup */    "Rocket Launcher",
        10,
        IT_WEAPON,
        WP_ROCKET_LAUNCHER,
```

F **Figure 2 : Change ammunition number**

Changing one parameter is quite easy, but now we have to find a way to store positions of players somewhere. The first idea is to store them into a .txt file since the rest of the structure – connection to database and heatmap – is not realized at this time. As we said before, the biggest problems are occurring on unexpected time. I also identified quite quickly which element could provide me the current position of players, but when I was checking the data in my .txt file, same numbers were returned. It turned out I was writing the address and not the data…a very classic beginner trap. I had a second big mistake using %d instead of %f in my fprintf() function. The results were just signed integers, and not the expected float indicating the positions. It was then very easy to integrate all the desired parameters to my file such as amount of ammunitions, weapon in use, health and armor points…
Now came the most entertaining part: playing the game. Obviously we need to fill the file – and later the database – with a lot a data which can be enough representative of a special behavior, and interesting to analyze later. As I told before, the connection between the game and the database is one of the last steps I made, I have to be sure each step is executing correctly. Let's now see what we are doing with the game metrics.

II/ Data Storage and visualization

Before directly speaking about the future of the previous metrics, it's necessary to know in which place they can be hosted, and what format we are expecting to get. Actually, because the whole solution would be dedicated to host heatmap and data, we need to add some security measures, and system to identify users. Then, in addition to the metrics we need to include unique identifiers, time of connection/disconnection from the server and other parameters. And since users do not have direct access to the entire database, we need a system to pass information. I'm now going to explain which method we used to stock information and get the heatmap.

a) Create Database

One of the easiest solutions to interact with a database is to use *WAMPSERVER*. This software is providing MySQL connection to local or non-local database, together with Apache security system. It is very easy to use and ergonomic. An embedded interface of phpMyAdmin is available to manage databases and tables. I created a database named quake3, containing 3 different tables, metricession, playsession and eventdata. The two first tables are used to manage connection to the database and accept or reject users, identifying them thanks to their IDs. We are just interested in the table evendata, which will host all our metrics as well as security identifiers. Basically we need to create columns containing the 3 axis positions of the players and a magnitude which corresponds to the intensity of heatmap spot.
To test the database Mr. McCallum gave me metrics from one of his game called *Ball Bouncer*[1].
People in an amphitheater are watching themselves on a huge screen, and are interacting with an embedded virtual ball. The ball is "physically" reacting with the players' moves, so are recorded place of impact between players and the ball as well as intensity corresponding as the number of people smashing the ball.

rowID	metricMD5	playMD5	eventTime	gameTime	eventType	eventSubtype	x	y	z	magnitude
0	MD5	MD5	2011-12-17 23:57:18	1.2345	1	2	155	214	0	1
1	MD5	MD5	2011-12-14 19:58:34	1.2345	1	2	146	218	0	0
2	MD5	MD5	2011-12-14 19:58:34	1.2345	1	2	169	207	0	0
3	MD5	MD5	2011-12-14 19:58:34	1.2345	1	2	476	449	0	1

Figure 3 : example of data from BallGame

On the screenshot above (fig3), is represented the database as we can see thanks to phpMyAdmin. When the database possesses a huge amount of these data, we can use them into the future heatmap by sampling processing SQL requests on a webpage.

b) Write database

The main components to connect to the database were written by Mr. Mackie in C++. He provided me files to authenticate to a database, but I still needed to make the link between the *Quake III* metrics and his code, as well as parsing correctly the data. The first test was done with the *Ball Bouncer* metrics. One vector – kind of linked list – is containing in each of its component another vector. These second vectors contain the parsed values, it means all the metrics coming from the game. The data in the second vectors will fill a specific URL, it will add php parameters. For each second vector we obtain an URL of this form:

localhost/hig/?metricID=MD5&playID=MD5&gameTime=1.2345&eventType=1&eventSubtype=2&x=1&y=2&z=3&magnitude=4.0

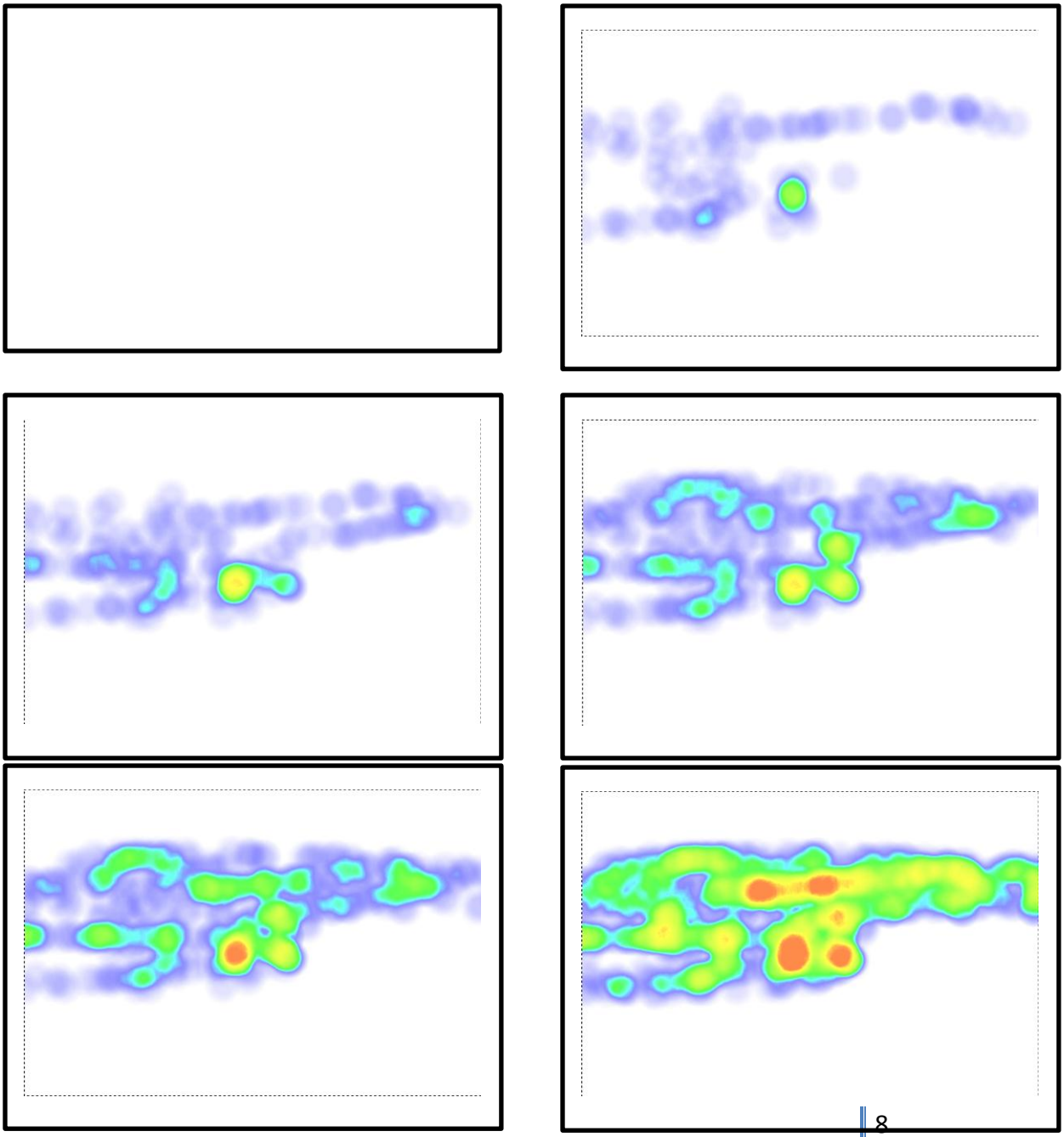
Of course, “localhost” will be replaced the website name hosting the heatmap.

Each time there is a connection to the previous link, a php code on this page is extracting all the parameters and adds them to the database thanks to a simple SQL request. The php code is included into the file *functions.php* which is called by *index.php*. When conducting many tests, the database is quickly filled with non-relevant data, that’s why we included 2 more functions, one to clean the database, and one to sort the rowID so as we still can know how many data are present just by reading this figure.
With *Ball Bouncer* metrics all the process is static since the data are coming from a .txt file. But when implemented into the *Quake III* source code, all the metrics a directly created while playing, parsed and added in the URL and then added to the database.
So this is the way companies could populate their own database to visualize their metrics. We don’t need more than an URL containing every kind of data they wish to set up. After this operation we are taking control over their data, maybe encrypt them is they need to stand secret. We are managing all the possible tools they would like to integer to the heatmap.
The next and final step is to create the heatmap.

c) Heatmap visualization

Now, we've described all the mechanisms which have conducted us to populate the database with the metrics directly coming from the game, while playing it. This last step is relying on a heatmap code. It is written in JavaScript. The original version[2] does not provide any access to database, it is reading static data, or users inputs via a web interface. So my work was to make a link between the php code to connect to the database, and the elements in JavaScript included in the HTML code.

There are certainly many ways to do so, but writing HTML code with php is the easiest one. We can create elements containing database information, and then retrieve them thanks to their HTML id through the JavaScript code. Even better, by creating HTML forms, we can provide users to choose between different HTML codes to select. It means the Script can adapt to users' choice and display different information on the heatmap.



III/ Miscellaneous

a) Sum up with graph

1

Extract metrics
In memory

```
x 212.000000 y 2360.000000 z 56.125000
x 249.897980 y 2336.664307 z 56.125000
x 397.457672 y 2245.521240 z 24.125000
x 429.310425 y 2225.845703 z 28.125000
x 447.278625 y 2214.746582 z 28.125000
x 578.317444 y 2133.803223 z 24.128126
x 683.675354 y 2068.721924 z 24.128126
x 787.399841 y 2004.648926 z 24.128126
```

Parse metrics
&
Add to URL

2

localhost/hig/?metricID=MD5&playID=MD5&gameTime=1.2345&eventType=1&eventSubtype=2&x=212&y=2360&z=56&magnitude=10

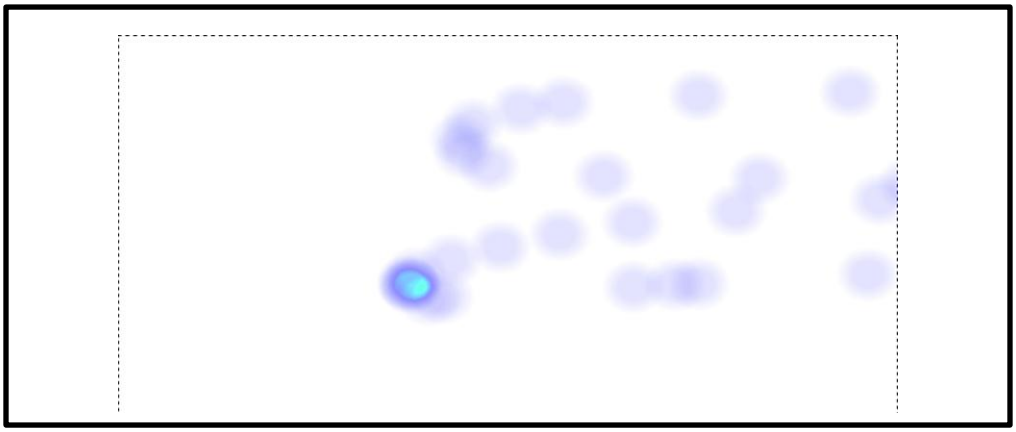
3

Connect &
write database

rowID	metricMD5	playMD5	eventTime	gameTime	eventType	eventSubtype	x	y	z	magnitude
383	MD5	MD5	2011-12-18 01:07:17	1.2345	1	2	250	233	0	10
384	MD5	MD5	2011-12-18 01:07:17	1.2345	1	2	397	224	0	10
385	MD5	MD5	2011-12-18 01:07:17	1.2345	1	2	429	222	0	10
386	MD5	MD5	2011-12-18 01:07:17	1.2345	1	2	447	221	0	10

4

Create heatmap (will add a nicer one later !!!)



b) Improvements

As the *Quake III* code source is quite huge, we didn't use the best location to implement the code which extracts the metrics. As a consequence, the game is not very playable with this configuration. Normally you should play this game without lag, but unfortunately the time to connect the database is quite huge compare to the answer time of the game. For our global solution this is not a big issue since this will be other developers' task to integrate the code. But for our own pleasure, it would be great to improve the both the location of the file in the source code, and to have a very responsive test server.

We could also improve the heatmap quality. Firstly, it is just including very simple filters such as selecting death location or repartition of health on the map. So it would be very interesting to add as much as parameters as we want, like use of weapons, number of ammunitions remaining, best position to kill other player, safest places - contrary of death places – etc… Secondly, we do not provide any background on the map, there are just colored spots on a white background. We can guess how the map looks like with the position filter, but the details are missing. Actually this is not that simple to make the connection between the position metrics and the heatmap. If one event appears to be outside the heatmap rectangle, we absolutely need to take it into account. Normally it is automatically including a dynamic resize of the map, but this part of JavaScript code is not functional. Tries to fix it were not successful. Furthermore, we also need to apply data on correct localization on the heatmap, to be very ergonomics it should be totally centered, and when we have a dynamic system it's quite complicated to implement. Hence, all our heatmap are not yet very nice to watch, but at least are based on real data.

c) Get the code

You can find all the code we made, and explanations to run it from A to Z. Normally, there should be no problem to get a nice heatmap after playing a while. I even included a method to play fluently the game and read later the data, but in this case the heatmap is not dynamic anymore.

The code is available on this address:

https://docs.google.com/open?id=0Bw4odICvlZY8MWI0Y2ZmOTQtZGFmNS00ZmFmLTlmOTMtOGQxYTllZDU2Yjlm

It should remain available quite a long time, since it is hosted on my Gmail documents account. If not it is still possible to contact me by my HiG mail address to get t.

REFERENCES

[1] http://www.ballbouncer.org/

[2] Patrick Wied (2011), http://www.patrick-wied.at/static/heatmapjs/