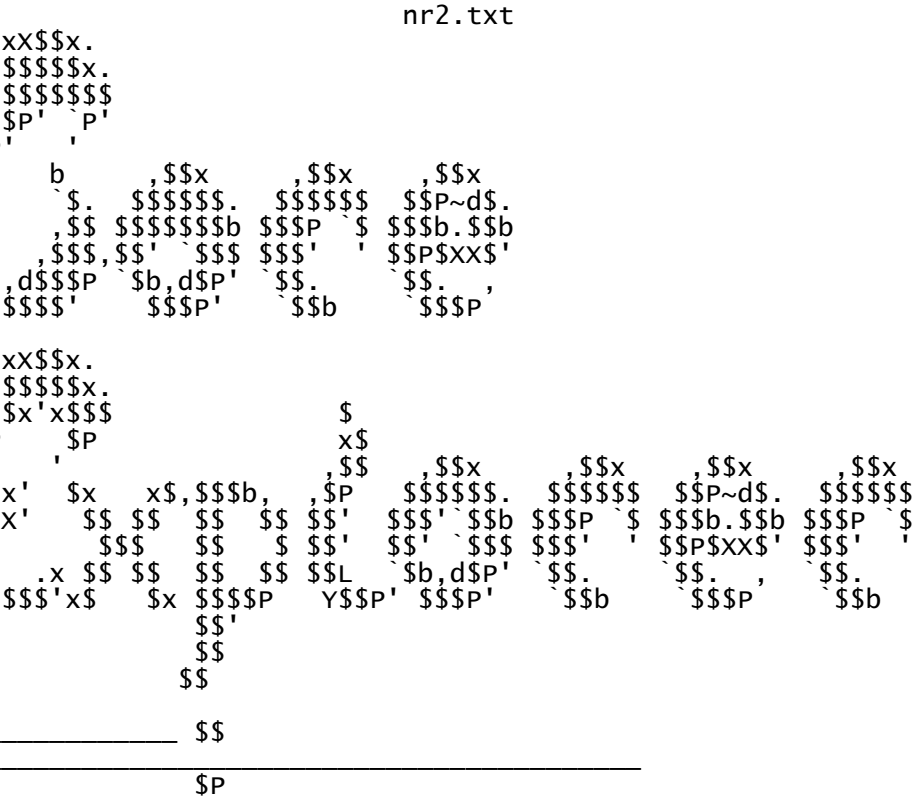

Issue 2
24/11/05

This is the second issue of the The Core Explorer. The brand new newsletter to promote the game CoreWar among new players. We will start from scratch and explain a lot of different strategies, tips and tricks on how to become more skilled in the game. We're sorry for the delay. The CoreExplorer will be posted to rec.games.corewar more frequently, but most of us had spent a lot of time coding for CSEC2005 (Corewar Single Elimination Cup 2005) organized by Christian Schmidt, so we didn't find the time to write these articles. Of course, we're looking forward to all contributions to this newsletter and some more feedback from the readers. If there is something that you want to be clarified in the next issues, or require a tip on a specific topic, you should send us an email and say what is it that you want. We hope that you'll find the topics in this issue to be interesting. Enjoy!

___ Roy's Top 10 Resources:

For new players its proven to be very hard to find good resources, here is our
top 10 list of resources and tutorials:

- The beginners guide to Redcode.....http://vyznev.net/corewar/guide.html
- Steve Bailey's Guides.....http://corewar.co.uk/sgb.zip
- Phillip Kendall.....http://www.srcf.ucam.org/~pak21/corewar/resource.html
- The Core Warrior.....http://www.corewar.info/corewarrior/corewarrior.htm
- Steve Morrell.....http://www.koth.org/info/chapter1.html
- Corewar Lexion.....http://www.corewar.info/lexicon
- Fizmo's CoreWar site.....http://www.corewar.info
- John Metcalfs Library.....http://corewar.co.uk/library.htm

- KOTH.org (Info Section).....http://www.koth.org/info.html
- rec.games.corewar (+FAQ).....http://groups.google.com/group/rec.games.corewar

Last but not least, join us on IRC!

Server: irc.koth.org:6667
Channel: #corewars

Did we forget good resources? Mail us!

_____exploring stones - the basics_____

If you are a beginner and have read at least some of the introductory texts designed to explain the basics of the game, you've probably stumbled accross a warrior called dwarf. Dwarf is a simple warrior and not very effective by modern standards, so you won't be using it to attack the hills. However, it is an example of a larger group of warriors called stones. So, what are stones? How do we define a stone? Well, a warrior is said to be a stone if it attacks by copying some instructions through the core, hoping to hit the opponent, and NOT caring if the opponents code is at the place where the desired instructions is copied. Warriors that use the more intelligent approach to first look for the place where they think their enemy is, and then attack that place extensively are called scanners, and we will talk about them more in the next issues. The instructions that are copied by the stone to attack the enemy with are called "bombs" and we'll call them that, too, from now on. We'll also say that a stone "throws" those bombs throughout the core instead of the term "copy". We'll do so because those are terms that are more frequently used among other redcoders and also more intuitive. The bombs that are most often used in stones are DAT bombs, because they instantly kill the process that executes them (in other words, they remove that thread from the queue of execution). Some stones use other bombs, but we'll talk about that later. Let us first have a look at Dwarf, to remind ourselves how it works. Here it is (at least one version - I'm not sure how the original warrior looked like - there are many similar ways to write it):

```
step equ 4

        org dwarf
dwarf   add.ab #step, bomb
        mov.i bomb, @bomb
        jmp -2, 0
bomb    dat 0, 0
        end
```

So, how does it work? It does the following: it increases the Bfield of the DAT located at bomb label, and then it moves that DAT to where its Bfield points. After it does that, it jumps right back to the beginning and repeats the cycle. How much does it increase the Bfield of the bomb? In this case by 4. We'll call that increment a "step" of the stone. So, it does the following:

```
dwarf   add.ab #step, bomb      ;this was just executed
        mov.i bomb, @bomb               ;the next instruction for execution is
here
        jmp -2, 0
bomb    dat 0, step
```

and then:

```
dwarf   add.ab #step, bomb
        mov.i bomb, @bomb               ;the bombs has just been thrown by this
instruction
        jmp -2, 0
bomb    dat 0, step
        for (step-1)
        dat 0, 0
```

```
        rof
        dat 0, step                     ;the bomb thrown step cells away

3 cycles later...

dwarf   add.ab #step, bomb
        mov.i bomb, @bomb               ;the bombs has just been thrown by this
instruction
        jmp -2, 0
bomb    dat 0, 2*step
        for (step-1)
        dat 0, 0
        rof
        dat 0, step                     ;the bomb thrown step cells away
        for (step-1)
        dat 0, 0
        rof
        dat 0, 2*step           ;the bomb thrown 2*step cells away
```

It is quite simple, really. Now you're probably asking yourself: does dwarf ever kill itself (when CORESIZE==8000)? Well, if you try it in Corewin in melee mode, you'll see that it never does. Why? Because of its step. Because 8000 is divisable by 4. In other words, the dwarf will bomb 8000/4 places in the core, at regular intervals:

```
        dat 0, 0
        dat 0, 0
        dat 0, 0
        dat 0, n*step
        dat 0, 0
        dat 0, 0
        dat 0, 0
        dat 0, (n+1)*step
        dat 0, 0
        dat 0, 0
        dat 0, 0
        dat 0, (n+2)*step
        dat 0, 0
        dat 0, 0
        dat 0, 0
        dat 0, (n+3)*step
        ...
        ...
```

Since the bomb is located just below the jmp -2, 0 instruction, the jmp, mov, and add instructions won't be hit if the dwarf uses this step. what about other steps? Well, every other step that produces the same pattern can be safely used. It is even better to use some other step here. Why? Well, event though, let's say,
3044 produces the same pattern (try it!)(because 4 = gcd(8000, 3044)) - using it instead of 4 is better in general. At this point you're probably not familiar with
optima program for evaluating stone steps or any other algorithm to do that, right?
We won't be doing some tedious analysis here, just try to explain it in simple terms. Our goal is to defeat our opponent. We don't know where it is. Our warrior simply throws the bombs around and hopes for a lucky hit. However, the step is very
significant. Not every step has the same probability of hitting the target in the desired time. Let us think about this a little. 4 and 3044 eventually produce the

same pattern - so, everything that can be killed by first step should be killed by the second one, and vice versa. Also, they are used in the same warrior and thrown at same speed. So, where is the difference? Before we continue, we'll have to learn some more expressions:

        - a space between two consecutive places in the core that were hit by bombs
        is called a gap.

- if gcd(STEP, CORESIZE) = k, then we say that value STEP is mod-k in core
of size CORESIZE.

You have probably guessed how we're going to evaluate our steps. If the greatest gap in the coreat a time t is m, then no warrior with main body of size greater than m could've survived until then. So, the smaller the gap, better chances for eliminating our opponent. We'll also observe the change of average gap in time and the standard deviation on the set of all gaps in the core, etc.
If you like, you can design an algorithm of your own to calculate these evaluations for stone steps. You can use your own formula. But let's get back to the point. The greatest gap when using 4 is MUCH larger than when using 3044. So, 3044 is better by that measure. We've probably said enough about optimizing steps for now. We should focus on making some warriors first. :)
Let's look at some better stones now.

First, we need to think a little. What should we change in a dwarf? What is it that we want to accomplish? We want to increase the speed (if the size isn't drastically increased by this, it is usually a good idea). We want to give some extra processes to the stone, to make it more resistant to paper attacks (more about it in the following part of this text). We also want to make it enter what is called a "coreclear phase" - to copy DATs over the entire core, striking a final blow to the opponent. In the next Core Explorer issues, we'll also discuss other bombs, and some advanced tricks. But this is enough for now.

So, why making a stone with more than 1 process in the bombing loop? Why will it have some sort of resistance to paper attacks. Well, we're going to explain some strategic ideas later on, but let's just say for now, without explanation (if you're that eager to hear it, you can always skip some lines and find it later in the text) that ost stones get easily beaten by papers (replicators). So, by giving the stone some sort of resistance to papers, we want to make it lose less when it's confronted by papers, which is necessary if we want to have any success with it. Now, remember what we were talking about in the previous issue. How do silk-style papers work? If you don't remember, look at it once again. It's important to know that. Observe what happens when a paper copy overwrites a stone (or some other enemy). When will the enemy survive such an event? The answer is very simple: if it has enough processes in it to use that paper copy for its further attack. So, if a 6 process silk-style paper is used, the stone has to have at least 6 processes at some place if it wants to be able to survive overwriting by a paper copy. We'll see later on that multiprocessing is important in some other cases, as well and not only when fighting papers. For now, just memorize that it's generally a good idea to have many processes in the loop, if they cooperate well - not to disrupt the functioning of the stone.

Enough talking. Time for our next example!

```
;redcode
;name CE_example_stone_1
;assert CORESIZE == 8000

step equ 3039
val1 equ step
val2 equ val1+step

        org stone
stone   spl #2*step, >2*step
        mov }val1, val2
        add.f stone, -1
dstr    djn.f -2, <-2200
        end
```

This stone is faster. Why. Well, it also has 3 working instructions in the bombing loop (we don't count spl, because it doesn't affect the speed much), and we attack two places at once, instead of just one. We attack one by using "}" Afield postincrement mode, and hit the other place with whatever the Afield of the first place is pointing to. How do we know that it's a DAT field that has been thrown. We don't. But in most cases, it will be,

at least in the early phase, when the core isn't filled with enemy code and various decoys. So, if we denote by "c" the speed where in n cycles, we attack n places (like the speed of light in corewar), then dwarf operates in 0,33c and this stone in 0,66c and even a little faster. Why faster? Well, it also has what is called a djn stream. Look at what dstr instruction does when it's executed. Try it. It decrements the fields of neighbour instructions and continues doing that until this stone is stopped when it hits itself. It doesn't die, however, because it has the SPL instruction at the top. If you're wondering how that SPL works, here is the explanation:

- the immediate addressing mode is interpreted as location '0' in execution. So, spl #x, y works the same way as spl 0, 0. It constantly creates new processes and keeps one process on itself for further process creation.

You may ask now why did we use ">" in the SPL Bfield. We didn't have to. But it can damage the opponent, and we lose nothing by incorporating it in our stone.

3039 is a mod-1 step. And not only that. It is THE best mod-1 step according to optima score. So, if you are searching for good steps, here is one that is very good. It doesn't mean, however, that it will work best in your warrior. You need to test several before you make the final version to submit to the hill.

The only true disadvantage of the previous example is that it doesn't coreclear. We're going to have a look a stone with a coreclear phase. Stay tuned!

```
;redcode
;name CE_example_stone_2
;assert CORESIZE==8000


step equ 3351
ini equ (2*step+1)
ds equ 3800

        org stone
stone   spl #2*step, <2*step
        mov ini, ini+step
        mov sbomb, *-1
        add.f stone, @-1                ;will be hit with its bomb
        djn.f @-2, <ds          ;later to begin coreclear
sbomb dat >-1, >1
        end
```

Before we begin the analysis of this warrior, first trt to notice something important. Try this warrior in Corewin. Set the speed to some of the lower values. Observe carefully. What is the order in which the instructions are executed. The opposite of what you expected, right? bottom -> up. That's what happens when you have a SPL at the top. I won't attempt to reason why it is so, but I'm sure that once you try it out and see for yourself, you'll understand how it works.

Now, we're going to analyze this instruction by instruction.

```
        spl #2*step, <2*step
```

We've already seen how this works, so no need to waste time here. Let's continue. Now we're going to go through the rest of the warrior from bottom to top.

```
        djn.f @-2, <ds
```

The bottom loop instruction. It makes a djn stream. If you look at the stone closely, you'll see where it jumps. It jumps at the instruction

just below the SPL. Why didn't we write djn.f -3, <ds? Well, we could have :). Try it like that and see if there's any difference.

```
add.f stone, @-1
```

Not much to explain here. increases the attack instructions values by a desired number.

```
mov sbomb, *-1
```

First attack. This way, both places are attacked by a real DAT bomb, unlike the previous example where we've only damaged the first place with a postincrement. It also means that we'll definitely throw a DAT at the next place.

```
mov ini, ini+step
```

Another attack instruction. At where its Afield points to is already a copy of our sbomb. So, we throw it again STEP places away from the first place. That's why we later increment by 2*STEP.

Now, have a look at the bomb. It looks a little strange, doesn't it? Well, we haven't been discussing coreclears yet, so I can't explain it completely. For now, just memorize that it is a good bomb to use and that it's able to inflict great damage to coreclears - most likely kill them.

And now we're going to look at how this stone coreclears. Once again, I advise you to try this warrior in CoreWin and look at the display, use a single step mode after you see that it started coreclearing, and you'll be able to grasp the concept in no time. If you don't use CoreWin or simply don't want to do this, we'll also try to explain it here. After some time, the stone hits itself with its bomb. It looks like this at that time:

```
t = t0

spl #6702, <6702
mov 4651, 2
mov 3, *-1
dat >-1, >1
djn.f @-2, <3800
dat >-1, >1

1 cycle after t0 :

spl #6702, <6702
mov 4651, 2
mov 3, *-1
dat >-1, >1
djn.f @-2, <3800
dat >-1, >1

the djn was executed, and decremented one place far away
in the code in front of us, there were no changes

2 cycles after t0 :

spl #6702, <6702
mov 4651, 2
mov 3, *0
dat >-1, >1
djn.f @-2, <3801
dat >-1, >1

3, 4, 5 cycles after t0, nothing changes - mov 3, *0
doesn't really do anything, and mov 4651, 2 either.
The same goes for djn instruction. After that, it's time
```

for another execution of dat >-1, >1.

6 cycles after t0 :

```
spl #6702, <6702
mov 4651, 2
mov 3, *1
dat >-1, >1
djn.f @-2, <3801
dat >-1, >1
```

the mov 3, *1 will overwrite itself with a dat >-1, >1 now:

7 cycles after t0 :

```
spl #6702, <6702
mov 4651, 2
dat >-1, >1
dat >-1, >1
djn.f @-2, <3801
dat >-1, >1
```

Now everything is set for the coreclear to begin. We only need to look at the first 3 instructions now:

```
spl #6702, <6702
mov 4651, 2
dat >-1, >1
```

The top SPL constantly produces new processes, so our warrior won't die yet. After a process executes the MOV, it falls through to the DAT, and increments the Bfield of MOV instruction. Therefore, the MOV instruction will throw a bomb one cell further away each time, until it overwrites the entire core with it. Of course, it will kill itself eventually, but we're hoping that it will kill the opponent before that happens. If the opponent is also a stone or a scanner, that will surely happen. Papers are a problem.

Of course, we could've been wrong in some places in this text. You shouldn't trust us. :) You need to test everything by yourself. You won't be able to learn corewar well simply by reading texts and tutorials. You have to try making your own warriors. So, start Corewin and experiment. The hill is waiting! Before we end this article, I'd like to give you some test results of the example warriors in this issue - obtained through 400 rounds fights and also by sending them to 94b for test. Look at this:

```
Example_stone_1 vs Dwarf3044    W% 61,5  L% 5,8  T% 32,8
Example_stone_2 vs Dwarf3044    W% 60,5  L% 10,0  T% 29,5
Example_stone_2 vs Example_stone_1 W% 44,0  L% 23,3  T% 32,8
```

If this hasn't convinced you that we were right, then nothing will :).

Scores at 94b hill:

```
Dwarf4  51,8            W% 12,9  L% 74,0  T% 13,1
Dwarf3044       72,7            W% 19,7  L% 66,7  T% 13,6
Example1        102,0           W% 23,2  L% 44,4  T% 32,4
Example2        104,6           W% 31,8  L% 58,9  T% 9,4
```

It looks like we've made a gigantic leap forward. We started with that weak little dwarf, and now we have those good stones. We'll use one of them for out hint warrior in this issue.

_____some strategic advices_____

You've probably heard about stone/paper/scissor analogy in corewar. I'll try to explain it a little here. At least when it comes to paper vs stone fight, since

those are the warriors that we explained a little so far. Like the original game says, paper beats the stone, stone beats scissors, and scissors bear the paper. So, papers (replicators) beat stones. Why? Well, it's not that complicated, actually. Papers make many copies of themselves quickly. Stones that use DAT ammunition simply can't kill all those paper copies fast enough. That's why stones are seldomly used alone, more often they are only used as a component in a bigger and more complex warrior. If they are used alone, then they have to use some smarter bombs, and implement some advanced tricks + be very well optimized, etc. You probably won't get good scores as a beginner if you try to use the stone without any protection to increase its survival rate against papers and warriors that use papers as components. One way to achieve that is to make a stone + paper. We'll do that as a hint warrior in this issue. Stone/papers score well on the hills lately. The stones help papers against scissor-type warriors, and papers help stones survive and score good against other stones.

_____Core Explorer Warrior #2_____

As mentioned above, we'll make a simple stone/paper here. Good stone papers also use some advanced warrior parts, such as qscans, but we won't use them since we're still far away from explaining them here. For all those advanced beginners - you can find the qscan tutorial at Jens Gutzeit's webpage.

We'll use the paper from the previous issue and add a stone from this issue. It should work better now :).

Since you already know how that paper works and how this stone works, we won't be explaining it here once again. There is no need to. We'll just give you the code of what we'll send to 94b hill with some short explanations.

```
;redcode-94b
;name CoreExplorerWarrior #2
;author Core Explorer staff
;assert CORESIZE==8000
;strategy an example of a stone/paper
;strategy boot, no qscan
;strategy paper from the previous issue


pStep1 equ 2341
pStep2 equ 2421
pStep3 equ 3241
bStep equ 100

step equ 3351
ini equ (2*step+1)
ds equ 3800

sbd equ 2831
pbd equ 873

        org boot
        for 56
        dat 0, 0
        rof
boot    spl 1, {-522
        spl 1, {-811
        spl 1, {-938
        mov <sgo, {sgo
sgo     spl sbomb+1+sbd, sbomb+1
        spl *sgo, {-1133
        mov <pgo, {pgo
pgo     jmp bomb+1+pbd, bomb+1
        for 10
        dat 0, 0
        rof
fsilk1  spl @0, }pStep1
```

```
                mov.i }fsilk1, >fsilk1
fsilk2  spl @0, >pStep2
                mov.i }fsilk2, >fsilk2
                mov.i bomb, <bStep
                mov.i {fsilk2, {bsilk1
bsilk1  jmp pStep3, 0
bomb            dat.f < 1, {1
        for 10
        dat 0, 0
        rof
stone   spl #2*step, <2*step
        mov ini, ini+step
        mov sbomb, *-1
        add.f stone, @-1
        djn.f @-2, <ds
        dat 0, 0
        dat 0, 0
sbomb   dat >-1, >1
        end
```

Notice that the paper is slightly changed. We've added some extra attack by changing SPL's in the paper. How does this work? Well, only one thing is different now. Before the paper copies itself over a location where it sends its processes to wait, it damages that place by decrements/increments. This will increase the score a little.

We've also moved the sbomb a little away from the stone itself. Why? Well, if someone is looking for us, we are less visible this way. This won't help the score much, because we've moved the sbomb only 2 places away. However, you should note that this is usually done in good warriors.

One thing is important. Why do we split twice to the stone before going to the paper? Well, if we didn't, paper would've slowed down the stone too much, and we would achieve less wins. If your goal is durability rather than W%, then you could split only once. But it will probably have a negative impact on the score. The alternative is to use a stone that has two SPL's at the top to increase the speed in which the stone creates new processes. Both tricks were used in good warriors, so it's hard to say which is better. The first one is more frequently used these days.

It is always tricky to do a good balancing of stone/paper's processes. This warrior is not optimized. It could probably score even better if it were.

Why did we copy our code away from the starting place (called: "booting")? Well, just in case that enemy is looking for our code, we leave the original code as a decoy and use a booted copy. It is also a bad idea to have a paper close to the stone in the beginning. Why? If some warrior coreclears or attacks close locations in the core, it could kill both of our components that way, and we don't want that to happen. So, we separate them to be far apart, and if we lose one of them, the other will probably survive. :)

Now look at the boot part of the warrior once again. Why do we have those {x Bfields. It's simple. We could damage the opponent, and it doesn't cost us anything. We could've used 0 Bfields instead, but what's the point of that? This way, we can hope for an occasional lucky hit and maybe some extra points.

One of the main problems for beginners is that they have the ideas, but lack the experience to write good warriors, because they don't know these tricks, which are commonly used in warriors on the hills. In the very beginning everyone wishes only for their warriors to work. Later on you'll become greedy and want to score well on the hill, as well. It's only natural. This section of Core Explorer is designed to introduce to you all these tricks. It is not as introductory as the above part. Here you may stumble accross things that won't be perfectly clear right away. But you can always come back later on and have another peek at this code. It's not so complicated. It probably is, if you are an absolute beginner. But, if you have some experience, and have tried some warriors at 94b already - you'll probably find these sections useful.

If you have any suggestions, you can always mail us. :) And should, if you want something to be done differently.

Now, let's submit this thing to 94b... And the score is... :

26.CoreExplorerWarrior #2 125,0 W% 22,2 L% 19,4 T% 58,3

That is not so bad. The 25th warrior has a score of 126,1. So, you may take this warrior and play with the boot distances and paper steps, try to improve it. If you manage to improve it just a little you'll be on the hill. :)

Notice that it scored much better than the paper did last time, and much better than the stone did this time. stone/papers really are a nice combination. The reason for it not entering the hill is a large number of ties against other papers.

We still lose some possible points because we lack a qscan. This warrior could easily score +8 points on the hill. So, there is even more potential in this piece of code :).

_____
_____94b hill report_____

The hill hasn't aged much sine the last time - only 8 successfull challenges. And it is more than a month since the last issue now. You need to try harder! So, the situation hasn't changed much and most of what was said the last time is still true and can be applied to the current hill, as well. Now that you've seen how to make a stone/paper, you could try making those. Stone/imps can do well, if you know how to make them. The hill is perhaps a little more diverse than the last time, so I guess that most strategies could do well, if carefully implemented. Try anything that has your attention now. Experiment. Look at the code of some of the warriors currently on the hill. Some of them are published and you can find them at Koenigstuhl infinite hill. Look at Svarog. Look at Yatima. Look at Blur2. In the end, it is your decision what you want to make. If you're discouraged by the strength of the current hill, don't be. It'll change soon. By the end of this year, age limit will be introduced to 94b hill. So, all of the old warriors will automatically be removed from the hill, making space for new submissions. However, I think that you should take your chances now, too. It is more challenging :).

Current hill status:

%W/ %L/ %T Name Author Score Age
1 33.5/ 24.8/ 41.7 Yatima v2.0.5 Jens Gutzeit 142.2 80
2 41.4/ 41.7/ 16.9 Svarog Nenad Tomasev 141.1 112
3 40.7/ 43.1/ 16.2 test Andreas Scholta 138.4 11
4 28.8/ 19.2/ 52.0 D'n'B [v0.3] inversed 138.4 16
5 40.0/ 42.1/ 17.9 Dragonfly S.Fernandes 138.0 9
6 39.5/ 41.0/ 19.5 QSCAN!!! Daniel Rivas 138.0 2
7 29.4/ 21.4/ 49.2 Kompaktor inversed 137.3 3
8 26.0/ 15.4/ 58.6 Snare_Rush_v0.7 Inversed 136.6 17
9 39.3/ 42.7/ 18.0 Xenocitrum_v0.8 inversed 136.0 24
10 41.3/ 47.4/ 11.3 Star Sascha Zapf 135.1 243
11 28.8/ 23.4/ 47.8 Blotter inversed 134.2 13
12 39.1/ 44.1/ 16.8 Unknown Neo 134.2 198
13 27.4/ 21.9/ 50.7 stealthbomb Fizmo and Neutrino 132.8 111
14 30.6/ 28.6/ 40.8 3[sm]md v0.2 inversed 132.7 10
15 30.3/ 28.5/ 41.1 The Silent Death Anonymous 132.1 94
16 37.9/ 44.7/ 17.4 Blur 2 Anton Marsden 131.0 8
17 36.0/ 41.5/ 22.5 Planter datagram 130.5 1
18 25.1/ 19.9/ 55.0 Biomass_v0.8 inversed 130.2 27
19 25.8/ 22.5/ 51.8 Barkosta[v0.3] inversed 129.0 26
20 29.1/ 29.3/ 41.6 ptest Sascha Zapf 128.8 4
21 27.6/ 27.9/ 44.5 Tom David Moore 127.2 226
22 36.7/ 46.5/ 16.9 dx42e inversed 126.9 20
23 36.8/ 47.1/ 16.0 think twice v.2 (optiMAXe el kauka 126.6 53
24 21.8/ 17.3/ 60.9 Everybody must get STONED madjester 126.3 148
25 36.8/ 47.5/ 15.7 Blindfolded Miz 126.1 22

Questions? Articles? Suggestions? Compliments? Mail us!
Authors: Nenad <tomasev at nspoint.net>, Roy <roy_van_rijn at gmail>