The Nabla Tournament Scoring and Scheduling System for Combat Mission

Jarmo Hurri
aka Nabla
jarmo.hurri@hut.fi

Mike Meinecke
aka Treeburst155
mikeman@cablelynx.com

Version 1.0

January 20, 2003

Contents

Chapter 1

Introduction

Unlike chess, CM is not a perfectly balanced game. Due to its very nature, inequities will always exist to some degree. This is true whether playing designed scenarios or QB meeting engagements. This makes competition CM somewhat problematic.

How does one measure player performance relative to the competition when all is not equal at the beginning? To complicate matters further, we don't even know how far, or which way, the scales are tipped.

The Nabla system for CM alleviates this balance problem to a great degree. In fact, the scoring system was designed specifically for the purpose of scoring CM competitions that utilize deliberately unbalanced scenarios.

Competition scenario designers are now free to design scenarios with fun as the primary goal, rather than balance. Hours and hours of playtesting for balance are no longer necessary! Lopsided scenarios can be thrown into tournaments to keep players on their toes. No longer can players make the assumption that their forces are adequate to achieve the mission assigned if properly employed.

Scenario balance is only one of the principals addressed in the Nabla system that we feel are important in order to best assess player skill relative to the competition. The other principles concern the relative importance of different scenarios, the importance of large vs. small victories, and providing an incentive to play even when one side is losing by a large margin. All four principles are explained fully in Chapter 2.

We believe that the Nabla system takes competition CM to a new level of fun and excitement for players and scenario designers alike, while providing an excellent way to determine players' relative performance in a competition setting.

This document has been written to help tournament managers and players understand and use the scheduling and scoring system. In writing this manual we have split work to make things a bit easier for us. Roughly speaking, Jarmo (Nabla) has written Chapter 2 (the principles of the system) and the technical appendices, while Mike (Treeburst155) has written Chapters 3 (running tournaments) and 4 (using the programs). This division between the user/client of the system (Mike) and analyst/designer (Jarmo) has been a very natural way to work for us during the whole project. For us this cooperation has been very fruitful. We hope you can enjoy the results, however painful an unbalanced scenario may turn out to be...

Chapter 2

Scoring principles of the Nabla system

2.1 The four principles

A scoring system provides an incentive for the players to strive for better performance. In competitive terms this is the same as rewarding the best players. But deciding what it means to do better, or to be the best, requires some careful consideration.

- Are all victories equally important, as in typical team leagues such as soccer leagues, where the goal difference is almost always irrelevant?
- If there are victories of different sizes, how is the overall goodness of a player determined? Is a player with one big victory and two draws as good as a player with two medium-sized victories and one draw?
- Furthermore, what does it even mean to be good if the game is unbalanced – that is, what is the baseline against which performance is measured?

The Nabla system is both a set of principles stating how performance should be measured in (unbalanced) CM tournaments, and an implementation of these principles in software. The principles are described in this chapter, while the use of the software is described in later chapters.

The foundations of the scoring system can be condensed into four principles.

1. The baseline against which performance is measured is set by *average performance* in a scenario.

2. It is equally difficult to be a winner in a scenario, *regardless of victory margins*. Therefore, the level of skill needed to score well in a scenario is considered to be equal in all scenarios.

3. When determining the goodness of a player from a set of scenarios, *uniformly strong gameplay* is rewarded.

4. In a scenario, both players *should always have an incentive* to strive for more CM points. That is, gaining a few more CM points should always be rewarded.

The mathematical operations used to implement the first three objectives are, in corresponding order, *difference from median*, *normalization of mean absolute deviation*, and an *asymmetric, nonlinear scoring curve*. These will be described in detail in the next sections. The fourth principle, although it may seem trivial, is important to keep in mind. It is relatively easy to end up with a scoring system in which the other objectives are fulfilled, but in some occasions neither player can gain much by obtaining more points. This would lead to games with very little action. This objective will be covered when the scoring curve is described.

2.2 The baseline of performance

The basic idea of unbalanced scenarios is that the average result in a scenario need not be 50–50. Figure 2.1 shows the distributions of raw CM scores of two different scenarios (from a real tournament), *Economy of Force* and *Gotta Get Up*. The results suggest that *Economy of Force* is balanced in favor of the allied player, while *Gotta Get Up* in turn seems to be balanced in favor of the axis player.

To assess the goodness of performance in such games, a different baseline than the standard 50 CM points is needed. Two standard statistics used to measure the average are the *mean* and the *median*. We use the latter because it is more robust against outliers.

According to *Encyclopædia Britannica*, an outlier is "a value that appears unusually large or small and out of place when compared with the other data values". The outlier problem is real in CM because of motivational problems. A losing player, especially in a tournament, may well surrender or try weird things just for the fun, instead of trying to do his best. This may yield scores that do not represent the true skills of the players. The median, which is defined as the number below which 50% of the values of the measured variable lie, is insensitive to such extreme values. The median of the set of numbers $\{40, 40, 50, 60, 60\}$ is 50. The median of the set of

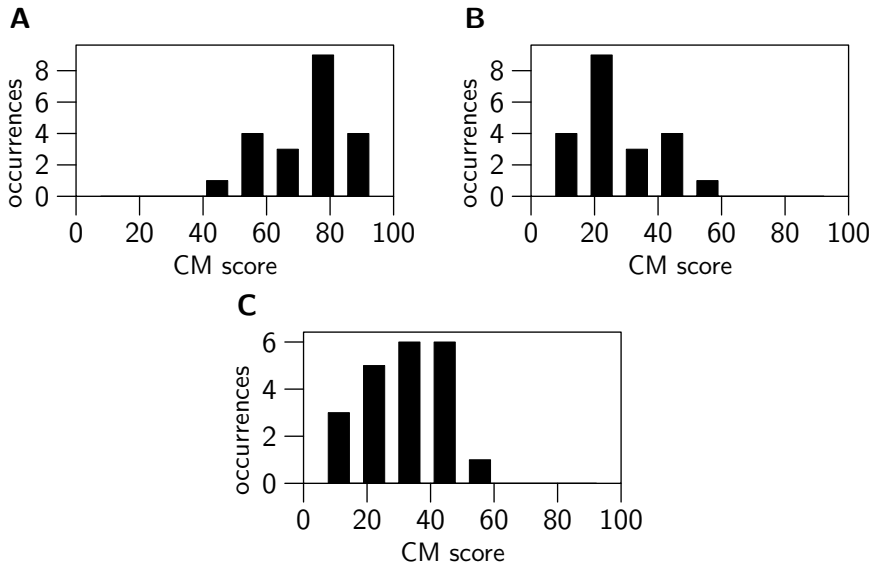

Figure 2.1: Unbalanced scenarios yield results where the average performance is not 50 CM points, but is different for allied and axis side in a scenario, and also differs between scenarios. A. Distribution of the allied scores in scenario *Economy of Force* in a tournament (median 77). B. Distribution of the axis scores in *Economy of Force* (median 23). C. Distribution of the allied scores in scenario *Gotta Get Up* in the same tournament (median 35).

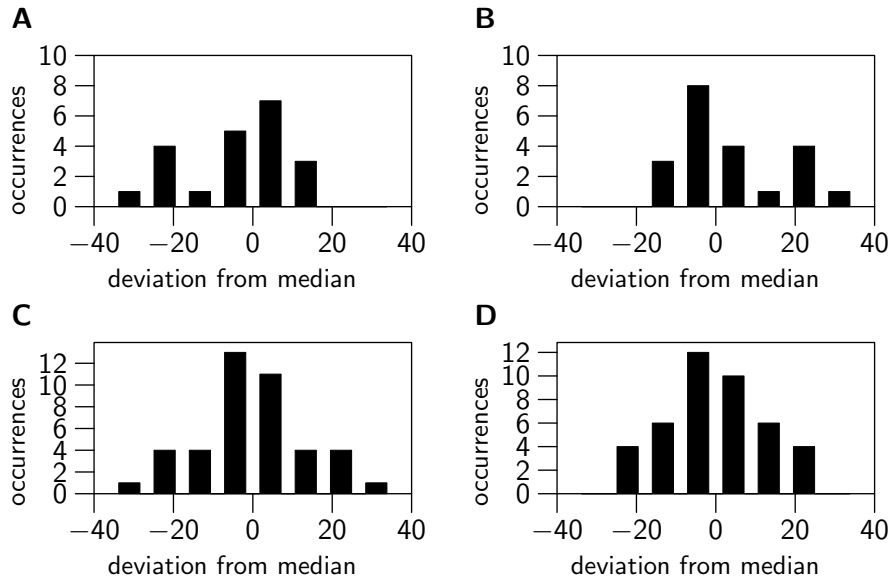

Figure 2.2: Subtracting the median from the raw CM scores takes care of the unbalance problem for the two different sides of the same scenario, and also for two different scenarios (see Figure 2.1). A. Distribution of the allied deviations from median in scenario *Economy of Force*. B. Distribution of the axis deviations from median in *Economy of Force*. C. Distribution of all (both axis and allied) deviations from median in *Economy of Force*. D. Distribution of all deviations from median in scenario *Gotta Get Up*. The histograms are not completely symmetric because some of the deviations are exactly zero, and assigning them to the symmetrically placed bins is ambiguous. As a solution, all zero deviations are assigned to the the bin left of zero.

numbers $\{40, 40, 50, 60, 100\}$ is the same. The means for these two sets are 50 and 58, respectively.

Subtracting the median from the raw CM scores takes care of the unbalance problem for the two different sides of the same scenario, and also for two different scenarios. Referring to the two scenarios in Figure 2.1, the medians of the scores of the allied player in *Economy of Force* is 77, while the median for the axis players is 23 (these always sum to 100). The median for the allied players in *Gotta Get Up* is 35. Figure 2.2 shows how after the subtraction of median, the distributions of the scores are centered around a common baseline, zero difference from the median.

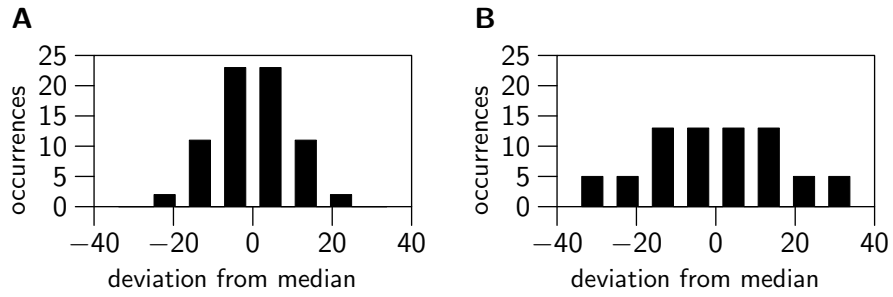

Figure 2.3: While in general it is just as difficult to play well in any scenario in a tournament, the spread of differences from median can be very different for different scenarios. If these values were used directly as scores in a tournament, the mean reward for winning (being above the median) in different scenarios would be very different. This would lead to emphasizing the outcomes of the scenarios more than the skills. A. Distribution of deviations from median in scenario *Fire on the Mountain* (mean absolute deviation from median 8.3). B. Distribution of deviations from median in scenario *Another Day* (mean absolute deviation from median 13.8).

2.3 The importance of different scenarios

Consider the deviations from median for two different scenarios, *Fire on the Mountain* and *Another Day*, shown in Figure 2.3. Imagine a hypothetical scoring system in which the deviations from median of different scenarios were just summed up to give the final tournament score. If you were a winner in *Fire on the Mountain* (above median), how big would your reward for this be compared with being a winner in *Another Day*? Or, if you were a loser, how severely would you be punished?

To answer these question, let us consider the *average* reward or punishment from these scenarios. As a measure of this we compute the *mean absolute deviation* from the median. Because the distributions are symmetric (positive and negative sides are mirror images), this single number describes both the average reward and the average punishment. For the two scenarios, the mean absolute deviations are 8.3 and 13.8, respectively. In *Another Day*, both players had to make big decisions (e.g., whether to try to take another flag) under great uncertainty, and these big decisions lead to more varying results.[1] As a result of this spread, it would be much more important to be a winner in *Another Day* than in *Fire on the Mountain*. In our opinion this should not be the case.

While it is true that in real life big decisions carry more serious consequences,

[1] See `http://www.battlefront.com/cgi-bin/bbs/ultimatebb.cgi?ubb=get_topic;f=1;t=026534`.

there is one major argument that speaks against such scoring in gaming tournaments. In general it is *just as hard* to obtain a victory (in terms of scoring above median) in a scenario with a small mean absolute deviation as in one with a large deviation, and it is just as difficult to be within the best 5 players in both scenarios. Winning is difficult even if the margins are small. So, in terms of the *skills* of the players, not the sizes of the *outcomes* of actions based on these skills, good play in either of the scenarios should be rewarded similarly.

This principle would not be reflected in a scoring system in which the mean final scores of the winners would be much larger for a scenario with large absolute deviation. Such a system would reward victory in one scenario much more than victory in another, thereby measuring outcomes more than skills.

The performance of a player relative to the other players is an important measure of skills. In some bridge scoring systems (where the games are also unbalanced) this idea is implemented by giving n points to the best player, $n-1$ to the second best etc. Note that such a system does not take into account the sizes of the victories in any way. While the Nabla system does not go this far, the scheme it uses can be considered as a variant of such scoring with a finer scale.

In the Nabla system, the deviations from median are normalized so that in each scenario the average reward for a winner is 1. Mathematically this is done by dividing each deviation from the median (like the numbers in Figure 2.3) with the mean absolute deviation (8.3 in *Fire on the Mountain* and 13.8 in *Another Day*).

The resulting normalized scores for the two scenarios in Figure 2.3 are shown in Figure 2.4. The spread in the two sets of scores is now much more uniform, reflecting the fact that performance in the two scenarios is now roughly equally important for doing well in the tournament. If normalized scores from the scenarios would be summed up to yield the final score in the tournament, the scenarios would be *exactly* equally important (if measured by mean reward or punishment). But the scoring curve, introduced in the next section, changes things a bit.

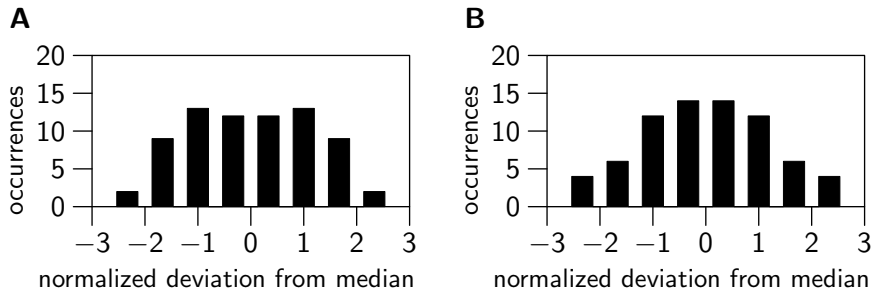

Figure 2.4: Normalization of mean absolute deviation from median makes performance in two scenarios equally important for doing well in the tournament (see Figure 2.3). After normalization, the average reward for being above the median is 1 in all scenarios. A. Distribution of normalized deviations from median in scenario *Fire on the Mountain*. B. Distribution of normalized deviations from median in scenario *Another Day*.

2.4 Victories and losses of different sizes and incentive to play

2.4.1 Rewarding victories of different sizes

Consider the normalized deviations of two different players from three different scenarios: player I has results $\{1.5, 0, 0\}$, while player II has results $\{0.5, 0.5, 0.5\}$. Which one of these players did better in the tournament?

While we acknowledge that obtaining a big victory over your enemy can be really difficult if your enemy is serious, there are two reasons why we think that player II did better than player I.

- There are three factors which can contribute to a good result in a scenario: good skills relative to your opponent, good luck, and an opponent which didn't really try. We want to measure the overall CM skills of the player. It is the thing that stays most stable over different scenarios, while luck and the attitude of the opponent may vary. Especially in tournaments, an opponent who has decided not to try seriously any more may be the real cause behind a single very good result. A set of good results in a number of scenarios is difficult to explain in terms of luck or moody opponents.
- Different scenarios tend to test different skills. The results of player II suggest that his range of overall CM skills is wider than the range of skills of player I.

Because of this, the Nabla scoring system emphasizes *uniformly strong gameplay* over single large victories.

How is this idea implemented mathematically? If we just take the sum of the scores of players I and II, the result is a tie. However, if we apply a mapping like the one shown in Figure 2.5A before taking the sum, the final score of player II will be bigger than the score of player I (see figure caption for details).

This is the approach used in the Nabla scoring system. It uses a *nonlinear* scoring curve – as opposed by the linear scoring curve, shown as the dashed line in Figure 2.5A. The use of the nonlinear curve emphasizes the importance of many small victories over a few large ones, thereby rewarding uniformly strong gameplay.

It is also possible to plot the slope (derivative) of the nonlinear curve of Figure 2.5A. This plot is shown in Figure 2.5B. If you think of a single scenario, the slope of the curve is a measure of how much the final scoring changes if the player obtains an additional CM point. (Comparisons between scenarios are complicated by the normalization procedure described in Section 2.3.) An additional CM point near the median (zero) yields a much bigger increase in final score than an additional CM point yields if you are already winning big time (for example, if you are at point 2).

The scoring curve shown in Figure 2.5 is an example of a curve that can be used in the actual scoring program. In the scoring program (see Section 4.2, page 24) the user can select between two different mathematical forms for the curve, and must also define a parameter which controls the rate at which the curve flattens. The details of these two functions and the parameter are given in Appendix A.

2.4.2 Penalizing losses of different sizes

The scoring curve in Figure 2.5A is a mapping from positive normalized deviations to the final score. But what about the negative side of the scoring curve? How big is the punishment for losing? Until this point in this manual, the different scoring schemes we have seen have all described *zero sum games*. That is, if one player has increased his score by an amount, the other player's score has decreased by the same amount. If the negative side of the scoring curve would be as in Figure 2.6A (compare this to Figure 2.5A), this would be the case. But this need not be so.

The fairly fast decreasing slope of the scoring curve on the positive side (Figure 2.5) implements the idea of rewarding uniformly strong gameplay. As was noted above (see page 11), there are two motivations for this. First,

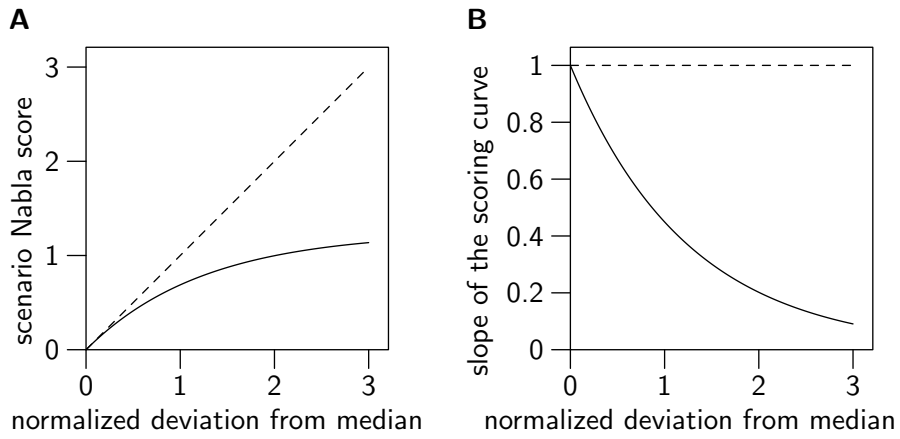

Figure 2.5: The positive side of the nonlinear scoring curve emphasizes the relative importance of many victories, even if they are smaller, over a single very large victory. A. The nonlinear scoring curve (solid line) rewards additional points near the median (zero) much more than far away from the median. When the normalized deviations from median are mapped to final scores using this function, three victories with normalized deviation 0.5 are more valuable than one victory with normalized deviation 1.5 (final scores $3 \times 0.41 = 1.23$ and 0.87, respectively). For comparison, if a direct sum of the normalized deviations (linear curve) would be used (dashed line), the single large victory would be as valuable as the three smaller victories. The equation of the scoring curve shown here is $\frac{5}{4}\left(1 - e^{-\frac{4d_\mathrm{n}}{5}}\right)$, where d_n denotes the normalized deviation from the median. B. The slope of the scoring curve is a measure of how much the final scoring changes if the player obtains an additional CM point. Comparing the slope of the scoring curve (solid line) at different points can be used to assess the importance of winning an additional CM point at different victory levels. For example, the ratio of the slope at zero and at 2 is $\frac{1}{0.2} = 5$. The point zero is the median, and point 2 is already quite a large victory. So winning an additional CM point at the median yields a fivefold increase in the final score when compared with winning the point when you are already at value 2. If scoring would be linear (dashed line), the additional score would be identical at all victory levels.

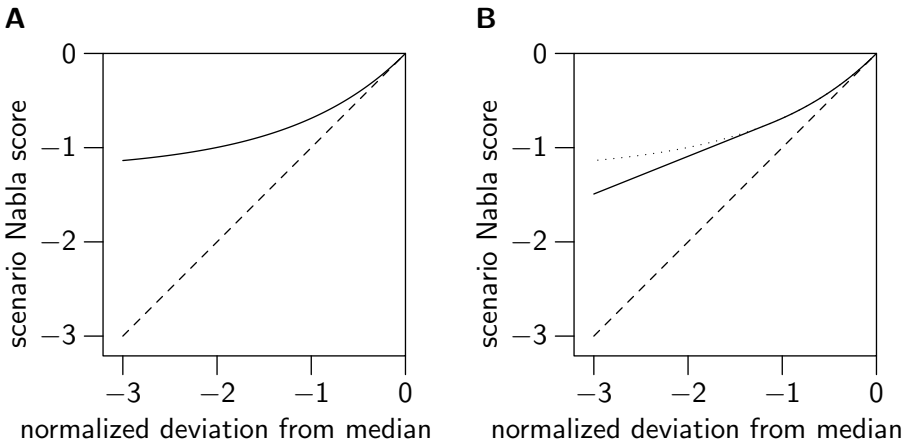

Figure 2.6: In case of a large victory, the negative side of the scoring curve punishes losers harder than winners are rewarded, but not too excessively so that the losers still have a chance to do fairly well in the whole tournament. A. If the loser would be penalized with a negative value as large as the reward of the winner, the negative side of the scoring curve (solid line) would be a mirror image of the positive side (see Figure 2.5A). The whole scoring curve would then be called symmetric. The linear scoring curve (dashed line) is shown here for comparison. B. The asymmetric scoring curve (solid line) punishes large losses more heavily than a symmetric scoring curve (dotted line) would, because it deviates from the symmetric scoring curve at large negative values. The linear scoring curve (dashed line) is shown here for comparison.

two of the three reasons for obtaining a very good score – your opponent's attitude problem and luck – are not related to your skills and are not under your control, and in tournaments the attitude problem is a serious one. Second, we want to reward players who master a wide variety of CM skills. These arguments turn around if you think about a very bad score.

First, there are also three reasons for obtaining a bad score: your opponents skills are better, you are unlucky, or you have an attitude problem. Now two of these three factors *are* under your control, and if *you* have an attitude problem, you should rightly be punished. Second, if you lose royally in a game because you do not have the skills it shows that there are some CM skills which you do not master. (But you still shouldn't be punished harder for a single large loss than for many small ones, because, in our opinion, having deficiencies in many skills is worse than a deficiency in a single skill.)

Because of these arguments the loser is penalized somewhat more heavily than in a zero sum game setting, although not *too* heavily so that a single loss will not destroy his chances in the tournament completely. The difference between the Nabla scoring curve on the negative side and a symmetric, zero sum game curve is illustrated in Figure 2.6B. As can be seen, the slope of the negative curve stays constant below some point, whereas the slope of a symmetric, zero sum game curve would continue to decrease. Because of this change, in case of a large victory the loser will be penalized slightly more than the winner will be rewarded. Remember that here a "large" victory is defined in terms of *normalized* deviation from median. As can be seen in Figure 2.4, in both scenarios *Fire on the Mountain* and *Another Day* there were players whose normalized deviation was below -2. The asymmetry of the scoring curve would have affected the scores of these players.

The minimum slope of the negative side of the scoring curve is given as a parameter to the scoring program (see Section 4.2, page 24). If a minimum slope of zero is given to the program, the curve will be completely symmetric. So it is possible for a tournament manager to bypass this feature.

2.4.3 Incentive to play

Consider a hypothetical symmetric scoring curve, shown in Figure 2.7A. As was discussed above, the slope of the scoring curve describes the size of the increased final score in case the player wins another CM point. The slope of the hypothetical symmetric scoring curve of Figure 2.7A, shown in Figure 2.7B, indicates that if one player is already winning by a large margin, then the incentive of both players to try to score an extra CM point is greatly reduced.

While it is true that in unbalanced games it can be very difficult to know

in the middle of a game what your position is on the curve, sometimes it is obvious that one player has succeeded and the other player has lost. Therefore, with a scoring curve like that shown in Figure 2.7A, situations would come up where both players would no longer be greatly interested in what is happening on the battlefield.

While the original motivation behind the asymmetric curve was related to the reasons for big victories and losses, it also has the nice side effect of alleviating this incentive problem. This can be seen in Figure 2.7D, which shows the slope of the asymmetric curve of Figure 2.7C. The slope on the negative side of the curve never falls below 0.4. This provides a direct incentive for the losing player to try to score more CM points, because it allows him to improve his score to a reasonable degree. Furthermore, it also provides an *indirect* incentive for the winning player: while his own score increases quite slowly if he scores more CM points, it does lower the possibility of his opponent in winning the tournament, thereby increasing his own chances to win.

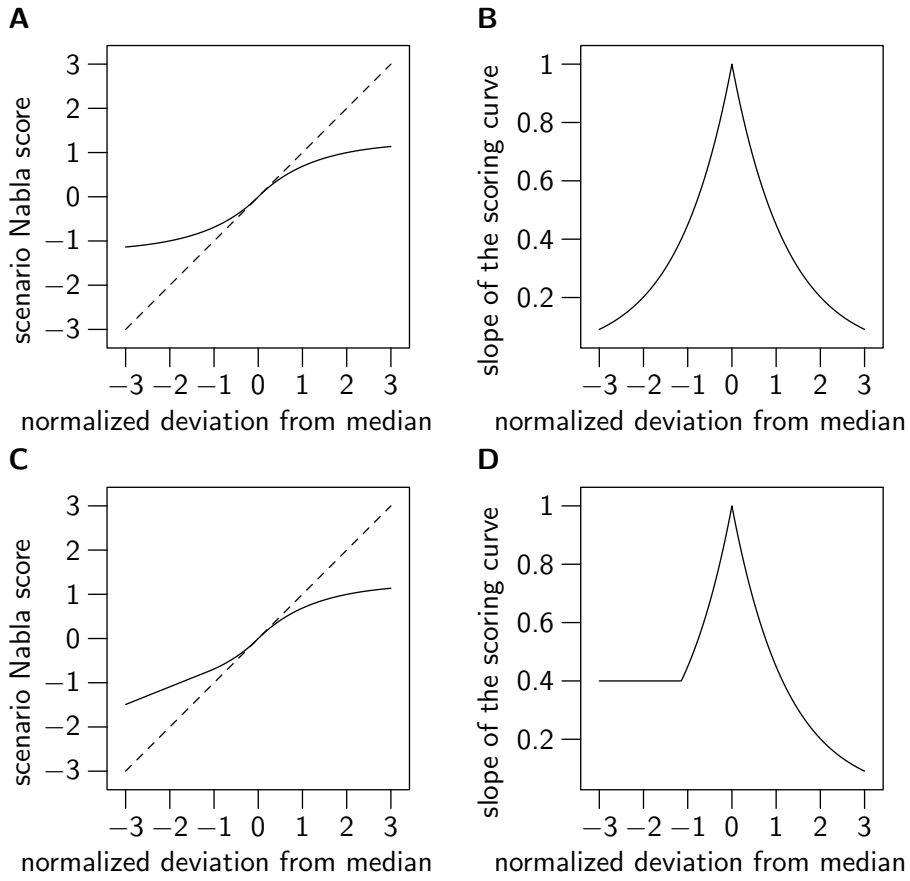

Figure 2.7: The asymmetric scoring curve tries to retain better (than a symmetric curve) the incentive of both players to strive for more CM points. This is done by keeping the slope on the negative side moderately steep, thereby providing a direct incentive (opportunity to score more points) for the losing player, and an indirect incentive (opportunity to increase chances to win the tournament) for the winning player. A. A hypothetical symmetric scoring curve (solid line). The linear scoring curve (dashed line) is shown here for comparison. B. The slope of the hypothetical symmetric curve. Notice that far away from the median the slope is very small when compared to the slope at zero. This indicates that the incentive to score more CM points is greatly reduced when one side is winning by a large margin. C. The asymmetric scoring curve (solid line). D. The slope of the asymmetric scoring curve. Notice how the slope on the negative side never falls below 0.4. This provides a better possibility for *both* sides to affect the score of the losing player, thus providing an incentive to play.

Chapter 3

How to run tournaments using the Nabla system

3.1 Tournament organization

To use the scoring system you must hold a round-robin tournament with a fairly large even number of players. A large number of players is desirable because the more times a scenario is played, the better the scoring program can determine scenario balance. Experience has taught that the best way to organize the Nabla scored tourney is to use 24 players and divide them into 4 equal sections of 6 players. This arrangement means that each scenario will be played 12 times.

You will need one less scenario than the number of players in each section. In the example above, you would need five scenarios. Section size can be any even number of players you desire, but you must keep in mind the number of scenarios you will need. Also, all players will need to play each scenario one time. How long do you want your tourney to run?

3.2 Section playoffs (if desired)

If you want to hold a Nabla scored playoff between the section winners, you will need to have an even number of sections. The section winners will then become the Finalist Section. In the case of my 4 sections of 6 players (highly recommended) the Finalist section would have 4 players, and require 3 scenarios.

Because there will only be a small number of players in the playoff round, you will either need to use an adaptation of the scoring system, or allow

non-section winners to compete amongst themselves in sections the same size as the Finalist Section. As mentioned above, the scoring system works best if the scenarios are played many times. The more the better, in fact. Your non-section winner players will help to establish the balance of the playoff scenarios for more accurate scoring.

The adapted system for scoring small tourneys (playoffs) is based on the same fundamental concept, comes in a few different variations, is simple to implement, and will be explained later. Suffice it to say at this point, that the full Nabla system is a better way to determine relative performance among players. Besides, most of your tourney losers are quite happy to compete amongst themselves in a mini-tourney while the finalists do battle. In effect, you are putting them to work so you can score the finalists more accurately.

3.3 Setting up the matches

Players will only play others in their section. They will play every person in their section one time. They will play each scenario one time. As far as is mathematically possible, players will find that all others in their section are playing the same side of a scenario as them an equal amount of times. For example, over the course of all the scenarios, you would find that Fred, Mary, and Joe all played the same side as you three times. This does not work out perfectly except with 8 player sections!

How does one set up matches with all these requirements? You must use the Nabla scheduling program. With it you generate a schedule for each section of your tourney. I will explain how to use this program later, but rest assured it is not difficult. For now, it is enough to know that there is a program that puts together a schedule of matches for each section that meets all the requirements listed above, and even a bit more. Below is what the scheduling program attempts to do in order of priority.

1. All players play each scenario one time.
2. All players play every other player one time.
3. Same side, same scenario instances (see first paragraph of section for explanation).
4. Equal number of games from each side.

It is important to realize that only objectives 1 and 2 above can be achieved perfectly every time. Objective 3 works out perfect for 8 player sections.

Objective 4, being low priority, is quite often way off. Also, since you will always be using an odd number of scenarios, how can players play each side the same number of times? The point is that all these things are optimized as much as possible starting with the things that are really important.

Note that attack/defend duties are not considered. This is entirely scenario dependent, and also unimportant for purposes of comparing player performance. Players should be able to attack or defend from either side. If they are weak when defending as the Allies, for example, and they draw a schedule with several Allied defenses, they're just going to have a rough tourney.

3.4 Scoring the tournament

Once all the game results are in your hands you must create the input file for the scoring program. This is simple, and will be explained later. The input file is just the final scores of the games. Upon execution of the program, these raw CM game scores will be converted to the Nabla score. The higher the Nabla score, the better a player performed over the course of the tourney in relation to his competition.

This is what the scoring program does:

1. It first looks for scores that do not add up to 100 due to contested/unoccupied VLs, and split the difference between the players equally (this is for CMBO, note that in CMBB CM points *always* total 100). For example, a final score of 70-20 would be converted to 75-25. Scores will always add up to 100 after this adjustment. There is a *very* good reason for doing this involving agreements between players designed to maximize their scores. Perhaps you can figure it out?
2. The median score for each side of all the scenarios is determined.
3. The difference between a player's score and the median score for the side he played will be determined for all scenarios.
4. The standard deviations from the median scores are determined for all scenarios. This value will always be the same for both sides of a given scenario due to step 1 above.
5. The difference from the median will then be divided by the standard deviation resulting in the "normalized difference from the median".
6. The normalized difference from the median is then assigned a Nabla score for the scenario. This is done with a formula created by Nabla that is at work inside the scoring program.

7. The average of all a player's Nabla scores (one for each scenario) is then determined, resulting in the player's final tourney score. The high score in each section is the winner of the section.

Chapter 4

Using the scoring and scheduling programs

The scoring program and the scheduling program are console (non-graphical) programs, running under DOS or Linux. Here we will concentrate on how to run the programs under DOS. It is beyond the scope of this manual to explain the basic DOS skills necessary to work with the programs. If you are unfamiliar with DOS, do not be dismayed. Mike (Treeburst155) will be more than happy to generate schedules for your tourney and crunch the scores. It does not take much time at all.

4.1 The scheduling program

The scheduling program is run for each section of your tourney separately. It requires two input files in simple text format. One file is simply a list of the scenarios, the other a list of the players in one section. You can name these text files anything you desire. These two input files must be placed in the same folder as the scheduling program.

A typical scenario file would look like this:

```
Scenario_A
Scenario_B
Scenario_C
Scenario_D
Scenario_E
```

That's all there is to it. Note there are no spaces. A typical player file would look like this:

```
Redwolf
Wreck
ciks
White_4
Nabla
Treeburst155
```

Again very simple, just make sure there are no spaces. That is, there can be *no spaces within scenario names or player names*. Note there are five scenarios and six players. You will always have an odd number of scenarios and an even number of players (scenarios +1). Once you have created these two files for a section you must place them into the folder where the scheduling program is located.

The next step is to open a DOS window and navigate to the folder containing the scheduling program and your two input files. Once within the folder you are ready to type in the command line. Let's call our scenario file `scenarios.txt`, our player file `players.txt`, and have the schedule (the output) go to `schedule.txt`. Assuming the Nabla Scheduling Program is called `nsched.exe` (you can name it anything you want) our command line would then look like this exactly:

```
nsched.exe -o scenarios.txt players.txt schedule.txt
```

Note that the scenario file must come first, followed by the player file, and finally the output file. Once executed you should soon see a new text file called `schedule.txt` (the name you chose in the command line) appear in the folder. This is your schedule of matches for the section for all the scenarios.

The Allied player is always listed on the left in the resulting schedule. You cannot pick and choose which player plays which side. You could choose to have all the German players be on the left instead; but, to keep things consistent between several Nabla system users, the convention is that the Allies are always on the left.

The `-o` (not zero) you see in the command line allows the program to do a brute force optimization of the schedule with regard to the number of times players play the same side of a scenario as any other player. It is best if players are compared to all others an equal amount of times, but it is not possible to get this perfect in most cases. I know it comes out perfect for eight players. The `-o` also tries to make it so players play each side an equal amount of times. Again this goal cannot be achieved perfectly. In fact, sides can be quite lopsided.

When working with 8 players or more the program will work for a long time

on its brute force crunch if you don't have a fairly fast computer. The `-o` can be omitted if you want; but you should use it unless you are crunching a schedule for eight or more with a slow computer.

The scheduling program will see to it that every player plays every scenario one time, and plays every other player one time. It then optimizes the number of comparisons, one player to another, as best it can. By "comparisons" I mean the same players playing the same side of a given scenario.

4.2 The scoring program

To use the scoring program you will need one input file containing the final game scores for the players. This is again a simple text file. In fact, it is nothing more than all your section schedule files combined, with the game scores added.

The best way to put together the scoring input file is to record the scores for your tourney, as they come in, on the schedules. You simply type in the scores after the names, being sure to leave a space. Make sure there is a space between the Allied player's score and the Axis player's name too. For example:

schedule file: `Treeburst Nabla`

schedule file with inserted scores: `Treeburst 95 Nabla 5`

After you have inserted all the scores into all the section schedules, you will need to create one big schedule file by copying and pasting. You would take all the names and scores for Scenario_A from all the section schedules and move them to one file under Scenario_A. You would then copy all the Scenario_B results to the same file and place them under Scenario_B, and so forth.

Any one of the section schedules can serve as the base for this combined file. You're simply transferring all the results to one file, making sure to place the results you are transferring under the correct scenario name. In other words, you will be copying the results for one scenario at a time from each section schedule. When this transfer is complete you have created your input file for the scoring program. Not only that, you have done it *without typos*. This is very important because the program will not work if player names are spelled differently from scenario to scenario.

If you do find yourself creating a scoring input file from scratch there are some things to know.

- The scenario name is always preceded by `#`.
- No spaces within scenario names (You can have a space after the `#`)
- No spaces within player names
- There must be a space between all scores and player names
- There can be no misspellings
- Everything is case sensitive

In light of the above, it is *much* easier just to copy the results from the section schedules, thereby removing the possibility of typos. If the program doesn't work, you've probably just left out a space between a name and a score.

Below is a sample scoring program input file. I just took the first two scenarios from an old tourney. There were actually seven scenarios, so you can see how big the scoring input file can be. This file was created by the method described above. Note I have several blank lines separating sections and scenario names. The program does not care how many blank lines you have, or where you put them. It *does* care about spaces.

```
# We_Cant_Wait

Bertram 19 Miron 75
Johnson 45 Jukka 55
Pixelmaster 37 Moore 63
Redeker 67 Sowden 32

Holien 54 Kingfish 41
Georges_Mick 30 Tom 65
Svensson 37 CapDog 63
vonLucke 37 Kettler 54

TabPub 35 Zalewski 60
Gaspari 56 Travisano 44
Juha 40 Rohde 51
Enoch 26 Dickens 74

# Duel_At_Dompaire

Miron 69 Moore 18
Bertram 28 Jukka 71
Johnson 37 Sowden 63
```

```
Pixelmaster 44 Redeker 52

Kingfish 47 CapDog 53
Holien 72 Tom 28
Georges_Mick 46 Kettler 44
Svensson 63 vonLucke 27

Zalewski 45 Rohde 51
TabPub 31 Travisano 64
Gaspari 29 Dickens 71
Juha 41 Enoch 59
```

Now that you have created you input file for the scoring program, you must place it in the same folder as the scoring program. You then open your DOS window, navigate to the folder, and type in the command line.

Before we begin talking about the command line, let me issue a word of caution. The command line has three parameters that must be entered. These relate to Nabla's scoring formula. It is not easy to explain what they do. I will simply present you with your choices for each parameter, and give a *basic* explanation for each. I will then follow up with several examples of valid command lines, along with the command line I recommend.

You do not have to understand the command line to be able to use it. You just have to make sure everything necessary is *in* the command line. If the "usage" section below throws you, don't fret. The "Command Line Examples" section, immediately following, will clear things up for you. You can simply use my recommended command line and be done with it, if you so choose.

The command line usage:

```
nscore.exe -d >debug.txt <asinh or exp> <coefficient> <minslope> input.txt output.txt
```

Explanation:

nscore.exe The name of the scoring program is typed in first.

-d > This is optional, but very interesting. It creates a file which has all the statistics in it. It contains scenario medians, standard deviations, players' normalized differences from the median, and Nabla scores for each individual scenario. We use it to check the details, and to debug the program.

debug.txt This is simply the chosen name for the file described above. It can be anything you want as long as you use the `.txt`-extension.

asinh or exp One or the other must be typed in. They each represent a scoring curve with a different shape. You can choose whichever curve you like best.

<coefficient> A number must be typed in here. You can use decimal points. This number determines how quickly the scoring curve falls off. The lower the number, the steeper the curve remains. A typical value here, and the one I recommend is .8 for an "exp" curve, and "2.3" for an "asinh" curve.

<minslope> You must enter a number between 0 and .999 here. This parameter determines the point at which the scoring curve, on the losing side, ceases to flatten. This was put in to punish extremely poor play due to voluntary surrender or outright wrecklessness. The higher the number, the more punishing your curve will be on the losing side.

input.txt You type in the name of your scoring program input file. For example, `scores.txt`. It must be a text file.

output.txt Here you enter the name you want for the final Nabla score file. Again it must be a text file. For example, `final.txt`.

Note that the input filename must always come before the output filename in the command line.

4.3 Command line examples

Let's call the scoring program `nscore.exe`, our input file,`scores.txt`, our program output file, `final.txt`, and the debug file, `stats.txt`. All these files can be called anything you want as long as you don't change the extension. Typical valid command lines would be:

```
nscore.exe -d >stats.txt asinh 2.5 .7 scores.txt final.txt

nscore.exe asinh 2.1 .2 scores.txt final.txt
(note: no stats file here)

nscore.exe -d >stats.txt exp .040 .7 scores.txt final.txt

nscore.exe -d >stats.txt exp .8 .42 scores.txt final.txt
(my recommendation)
```

I *highly recommend* `exp .8 .42` for the three parameters. If you want to use the Nabla system, but don't want to explore the effects of different parameters, mine are the way to go. I've spent a great deal of time analyzing different scoring curves. I may change my parameters as the database of Nabla scored scenarios increases. If so, the manual will be updated to reflect my new recommendations.

If you wish to play with the curve by trying different parameters, Nabla has given you the option to do so. Your tourneys can be scored with your own customized Nabla scoring system. Just be sure you've taken a good look at the effects of your parameters on final scores before you score a tourney with them.

4.4 The output file

The output file will list every player in the entire tournament along with their final Nabla score for the tournament. The players will be listed from highest score to lowest without regard for sections. You will have to separate the players into sections manually to determine who had the high score in each section. This doesn't take long with 24 players.

Below this main output, each scenario will have its own list of players with their Nabla score for that single scenario. This list is currently in alphabetical order, but will likely be changed to display the players from high to low score for the scenario. These Nabla scores for individual scenarios can also be viewed in the stats (debug) file. The info is just easier to find in the output file.

Once you've separated your output back into sections, thus determining the section winners, you can go online and announce your section winners. You have successfully scored your tournament with the Nabla scoring system.

4.5 Playoff scoring

As stated earlier, the best way to score playoffs is to let all the players play the scenarios in sections the same size as your finalist section. This way you can use the full Nabla scoring system. If you do want the playoffs to be just among the section winners there are alternatives. The one I like best is as follows.

Generate the schedule for your playoffs as normal. Record the game scores on the schedule as normal, but be sure to split the difference for contested/unoccupied VLs. You aren't using the scoring program so you must

remember to do this manually. For each side of each scenario make the highest score achieved equal to 100. Increase the other scores to the original percentage of these high scores. Add the scores for each player, and the highest total is the champion.

An example set of results could look like this:

```
# Scenario_A

Jim 80 Jack 20
Nabla 60 Treeburst 40
Steve 40 Charles 60
```

Jim's score would become 100 because he has the high Allied score, Nabla's score would become 75% of Jim's 100 (60 is 75% of 80), or 75, and Steve's score would be raised to 50 (maintaining his 50% relationship to the high score). In this way, the scores retain their original relationship to each other.

You would do the same thing for the Axis scores. Charles' score becomes 100 since he did best. Note that Charles has done as well as Jim now. They both gained 100 points for the scenario because they both achieved the high score for their respective sides. Treeburst's score moves up to 66.67, and Jack goes to 33.33.

4.6 Final words

You now have enough information to successfully hold a tournament using the Nabla scoring system. Feel free to ask any questions you may have. We can be reached at `mikeman@cablelynx.com` or `jarmo.hurri@hut.fi`.

Exploring the scoring curve and the benefits/drawbacks of different parameters, not to mention the math involved, is another subject. You can run hypothetical tourneys through the scoring program to see the effects of different parameters. If you have a good scientific calculator you can also work with the formulas directly.

Appendix A

Details of available scoring functions and parameters

The two different types of scoring functions supported by the system are

$$f_{\exp}(d_{\mathrm{n}}, a) = \operatorname{sign}(d_{\mathrm{n}}) \frac{1}{a} \left(1 - e^{-a|d_{\mathrm{n}}|}\right) \quad \text{(A.1)}$$

$$f_{\operatorname{asinh}}(d_{\mathrm{n}}, a) = \operatorname{sign}(d_{\mathrm{n}}) \frac{1}{a} \operatorname{asinh}(a\,|d_{\mathrm{n}}|), \quad \text{(A.2)}$$

where d_{n} denotes the normalized deviation from the median, and a is a parameter controlling the rate at which the steepness of the curve changes.

The difference between the two functions and the role of the parameter is illustrated in Figure A.1. The difference is not great: tournament manager can mainly control the rate at which the slope falls near zero, and the slope of the curve near extreme values.

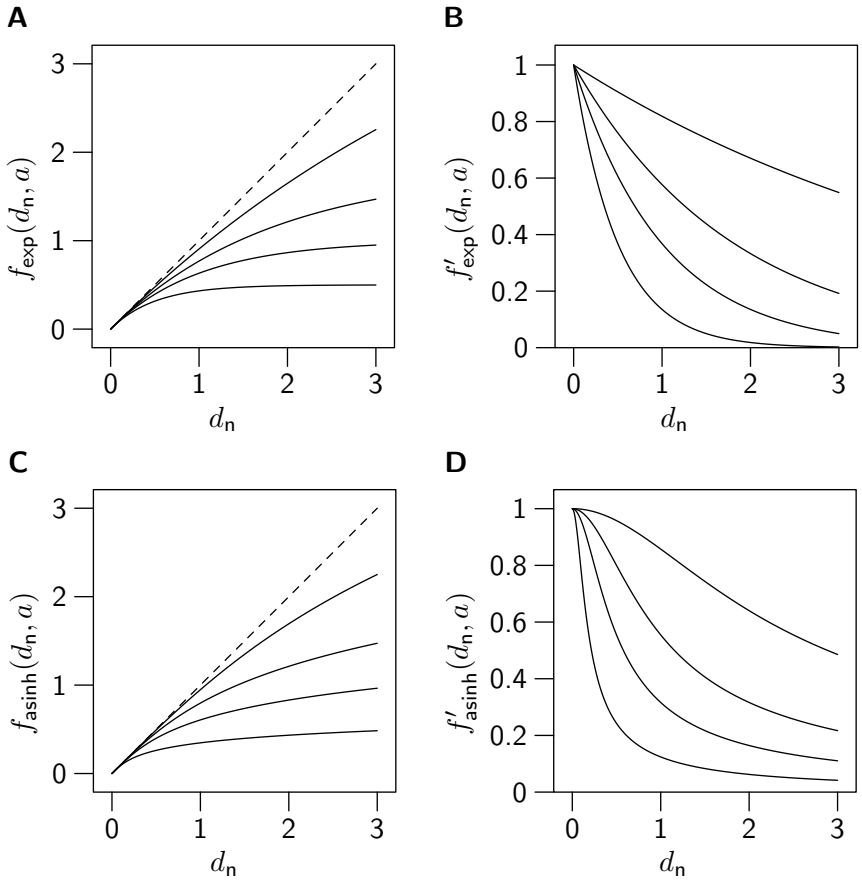

Figure A.1: Illustration of the two scoring function families. A. Function $f_{\exp}(d_n, a)$ for four different parameter values (solid lines, from top curve to bottom curve, $a = 0.2$, $a = 0.55$, $a = 1$ and $a = 2$). The linear scoring function (dashed line) is shown here for comparison. B. Slopes $f'_{\exp}(d_n, a)$ for four different parameter values (from top curve to bottom curve, $a = 0.2$, $a = 0.55$, $a = 1$ and $a = 2$). C. Function $f_{\text{asinh}}(d_n, a)$ for four different parameter values (solid lines, from top curve to bottom curve, $a = 0.6$, $a = 1.5$, $a = 3$ and $a = 8$). The linear scoring function (dashed line) is shown here for comparison. D. Slopes $f'_{\text{asinh}}(d_n, a)$ for four different parameter values (from top curve to bottom curve, $a = 0.6$, $a = 1.5$, $a = 3$ and $a = 8$).

Appendix B

A few words about the code

The source code is written in standard C++, and should compile with any good compiler. The source utilizes the Standard Template Library (STL) quite heavily.

The source code of the programs is not available at the moment. This is just because the code needs to be packaged better before it can be made public. When the source code is packaged, it will be made available via web page `http://www.cis.hut.fi/ jarmo/nabla-system`.