Evaluation functions in Rybka 1.0 Beta and Rybka 2.3.2a

The purpose of this document is to provide a listing of differences between the evaluation functions of Rybka 1.0 Beta and Rybka 2.3.2a. It is generally assumed that the reader has a copy of the annotated ASM dumps of each.

I will progress through the evaluation functions section-by-section, giving the respective ASM dumps with Rybka 1.0 Beta on the left (and Rybka 2.3.2a on the right). The greatest structural differences are near the beginning; but once the piece loops are begun, the similarities are more evident.

Throughout I have eliminated certain instructions involved with the stack, both for the function call itself and some storing of local variables. I have also abbreviated the frequent use of a `popcnt` mechanism.

1 Function start and overview

```
0x401740: mov    %rcx,0x8(%rsp)
0x40174f: mov    (0x66e644),%edx
0x401759: mov    (0x66e640),%rbx
0x401764: and    $0x3ffff,%ebx # mat table
0x401771: mov    0x29940(%r9,%rbx,8),%rbx
0x401779: mov    %edx,%eax
0x40177f: imul   $0xd47,%eax,%eax
0x401785: mov    %rbx,%rsi
0x401788: mov    %rcx,%r8
0x40178b: shr    $0x20,%rsi
0x401799: add    %eax,%esi
0x40179b: cmp    (0x66c460),%edx # lazy
0x4017a1: mov    %esi,%edi
0x4017a3: jle    0x40293b
0x4017a9: cmp    (0x66c464),%edx # lazy
0x4017af: jge    0x40293b
0x4017b5: mov    %rbx,%rax
0x4017b8: shr    $0x18,%rax
0x4017bc: test   %al,%al  # lazy flag
0x4017be: jne    0x40293b
0x4017c4: mov    (0x66e638),%rax # pawn
0x4017d3: and    $0x7ffff,%eax # hash mask
0x4017e0: lea    (%rax,%rax,2),%rcx # 24 by
0x4017e4: mov    (0x66c2f0),%rax # phash ptr
0x4017ef: prefetchnta (%rax,%rcx,8)
0x4017f3: prefetchnta 0x20(%rax,%rcx,8)
```

```
0x46adc0: mov    %rcx,0x8(%rsp)
0x46adcf: mov    (0x7dd4d8),%r9
0x46adf9: mov    (0x7f5248),%ecx # phash mask
0x46adff: mov    %r9,%rax
0x46ae02: sar    $0x8,%rax
0x46ae0d: and    $0x3ffff,%eax # mat table
0x46ae12: movswl 0x3f6260(%r11,%rax,2),%r10d
0x46ae1b: movzbl %r9b,%eax
0x46ae1f: add    $0xffffffffffffff80,%eax
0x46ae22: imul   $0xd47,%eax,%eax
0x46ae28: add    %eax,%r10d
0x46ae2b: and    (0x7dd4c8),%rcx # pawn hash
0x46ae32: mov    (0x7f5240),%rax # phash ptr
0x46ae3e: prefetchnta (%rax,%rcx,8)
0x46ae42: mov    $0x1ff800000000,%rax
0x46ae4c: test   %r9,%rax  # check for K+P
0x46ae4f: jne    0x46b169
0x46b169: mov    -0x1c(%r15),%ecx # last pos eval
0x46b16d: mov    %r10d,%eax
0x46b170: sar    $0x5,%eax # 3399 -> centi
0x46b177: add    %ecx,%eax
0x46b179: cmp    %edx,%eax [edx is input]
0x46b17b: jl     0x46bf5c # lazy
0x46b181: cmp    %r8d,%eax [r8d is input]
0x46b184: jg     0x46bf5c # lazy
```

Both use the material index token to get a material imbalance adjustment, and then multiply the 1-3-5-10 material count by 3399. They both then check if lazy eval is operative, though Rybka 2.3.2a first uses specific pawn endgame code when applicable (omitted here). Both load the pawn hash location and prefetch it, again with an ordering difference. The specifics of how lazy eval is applied also differ between these two Rybka versions.

1.0.1 More top-level comments

Some explicit examples of PST are included in Section 3 below.

Another component that can affect evaluation is material imbalances. These changed from time-to-time over the development process from Rybka 1.0 Beta to Rybka 2.3.2a. One of the changes with Rybka 2.3.2a in particular was with Larry Kaufman making some adjustments (link).[1]

1.1 Pawns

```
0x401800: add    (0x66e628),%esi # PST op
0x401806: add    (0x66e62c),%edi # PST eg
0x4017f8: bsf wK (0x66e5e0),%rax
0x40180c: mov wO (0x66e580),%r10
0x401813: mov wN (0x66e5a0),%r8
0x40181f: mov    0x249d40(%r9,%rax,8),%rax
0x401827: xor    %r13d,%r13d
0x40182a: mov    $0xffff7f7f7f7f7f7f,%rcx
0x401839: bsf bK (0x66e5e8),%rax # bK sq
0x401841: mov    %r13d,%ebp # attackers = 0
0x401844: mov    0x249d40(%r9,%rax,8),%r15
0x401851: mov wP (0x66e590),%rax
0x401858: mov    %rax,%r14
0x40185b: not    %r10 # white occupied
0x401868: and    %rcx,%r14 # ignore h-file
0x40186b: mov    $0xfefefefefefefe,%rcx
0x40187f: and    %rcx,%rax # ignore a-file
0x401882: shl    $0x2,%r14 # half of attacks
0x40188b: or     %rax,%r14 # other half
0x40188e: mov    $0x1,%eax
0x401896: shl    $0x7,%r14 # align correctly
0x4018a4: test   %r14,%r15 # if wP AND bKatt
0x4018ac: cmovne %eax,%ebp # add 1 to Katt
```

```
0x46b18a: mov wP (0x7dd430),%rdx
0x46b191: mov bK (0x7dd488),%r14
0x46b198: mov OC (0x7dd490),%rcx
0x46b19f: mov    (0x7dd4c0),%ebx # PST
0x46b1a5: mov wO (0x7dd420),%r12
0x46b1ac: bsf bK %r14,%rax # bK sq
0x46b1b0: mov    0x18c480(%r11,%rax,8),%r13
0x46b1c2: mov    %rdx,%rbp
0x46b1c5: mov    %rdx,%rax
0x46b1c8: xor    %edi,%edi
0x46b1ca: mov    $0xffff7f7f7f7f7f7f,%r8
0x46b1d4: mov    %edi,%esi # attackers = 0
0x46b1db: and    %r8,%rbp # ignore h-file
0x46b1de: mov    $0xfefefefefefefe,%r8
0x46b1e8: not    %r12 # not Wh occupied
0x46b1eb: and    %r8,%rax # ignore a-file
0x46b1ee: shl    $0x2,%rbp # half of att
0x46b1f2: or     %rax,%rbp # other half
0x46b1f5: mov    $0x10000,%eax # (1,0)
0x46b1fa: shl    $0x7,%rbp # align correctly
0x46b1fe: test   %rbp,%r13 # if wP AND bKatt
0x46b201: cmovne %eax,%esi # (1,0) in Katt
```

The main difference here is that the White pawns code in Rybka 2.3.2a proceeds to count how many such pawns have no mobility, subtracting $(42, 339)$ for each.

[1] "[...] One of the things that I felt was wrong with Rybka (all versions thru 2.3.1) was that she undervalued pawns relative to pieces, so that she would almost always play to win a piece for three pawns, sometimes even for four, even when the pawns were rather dangerous (others have said so too). Correcting this without bad side effects was not trivial, but eventually I found the values that corrected the problem while showing an improvement in overall strength of a couple points at the same time. [...] Roughly speaking, Rybka 2.3.1 and earlier versions consider that a piece (knight or unpaired bishop) is worth about four pawns in the middlegame, even more on a full board. I think this is going too far. I would say that a fair value for a knight or unpaired bishop is about 3 1/2 pawns in the middlegame, somewhat higher in the opening, and version 2.3.2 is fairly close to my view on this."

1.2 Knights

```
0x401813: mov wN (0x66e5a0),%r8
0x4018d0: bsf    %r8,%rax # loop over wN
0x4018d4: mov    0x249b40(%rbx,%rax,8),%rdx
0x4018dc: or     %rdx,%r14 # update attacks
0x4018df: test   %rdx,%r15 # see if hits bK
0x4018e2: je     0x4018ee  # if so then
0x4018e4: add    $0x1,%ebp # add 1 to count
0x4018e7: add    $0x3ad,%r12d # and 941 to wt
0x4018ee: and    %r10,%rdx # AND the squares
0x4018f1: mov    %rdx,%rax # att with Wh occ
[...] [popcnt] [...]
0x401928: shr    $0x38,%rcx
0x40192c: mov    %ecx,%eax
0x40192e: imul   $0x79,%ecx,%ecx # 121 in op
0x401931: imul   $0xe,%eax,%eax  # 14 in eg
0x401934: add    %ecx,%edi  # add to scores
0x401936: add    %eax,%esi
0x401938: lea    -0x1(%r8),%rax # remove N
0x40193c: and    %rax,%r8    # from bit array
0x40193f: jne    0x4018d0    # and loop
```

```
0x46b238: mov wN (0x7dd440),%rdx
0x46b25c: test   %rdx,%rdx # if no WN skip
0x46b25f: je     0x46b284
0x46b261: bsf    %rdx,%rax # get first wN
0x46b265: mov    0x18c280(%r11,%rax,8),%rax
0x46b26d: or     %rax,%rbp # update attacks
0x46b270: test   %rax,%r13 # test AND with bKatt
0x46b273: je     0x46b27b      # if so, then add
0x46b275: add    $0x103ad,%esi # (1,941) to Katt
0x46b27b: lea    -0x1(%rdx),%rax # remove N
0x46b27f: and    %rax,%rdx     # from bit array
0x46b282: jne    0x46b261      # and loop
```

The Knight procedure again has a substantive difference, as Rybka 2.3.2a does not have any knight mobility bonuses. The same computation of king attacks, and even the weighting of 941 is used in each version.

1.3 Bishops

```
0x401949: mov wB (0x66e5b0),%r9
0x401950: mov    (0x66e600),%r11 # rot
0x401957: test   %r9,%r9 # if no wB skip
0x40195a: je     0x401a48
0x401971: mov    (0x66e608),%rdx # rot
0x401978: bsf wB %r9,%rax
0x40197c: mov    %eax,%r8d
0x40197f: lea    0x0(,%rax,4),%rax
0x401987: movzbl 0x29640(%rax,%rbx,1),%ecx
0x40198f: shl    $0x6,%r8
0x401993: shr    %cl,%rdx
0x401996: movzbl 0x29540(%rax,%rbx,1),%ecx
0x40199e: mov    %r11,%rax
0x4019a1: shr    %cl,%rax
0x4019a4: and    $0x3f,%edx
0x4019a7: add    %r8,%rdx
0x4019aa: and    $0x3f,%eax # rcx has sq att
0x4019ad: mov    0x231940(%rbx,%rdx,8),%rcx
0x4019b5: add    %r8,%rax
0x4019b8: or     0x229940(%rbx,%rax,8),%rcx
0x4019c0: or     %rcx,%r14 # r14 is running
0x4019c3: test   %rcx,%r15 # see if bK att
0x4019c6: je     0x4019d2
0x4019c8: add    $0x1,%ebp # if so add 1
0x4019cb: add    $0x1a2,%r12d # and 418 wt
0x4019d2: and    %r10,%rcx # AND w NOT(wOcc)
[...] [popcnt] [...]
0x401a20: shr    $0x38,%rcx
0x401a24: mov    %ecx,%eax
0x401a26: imul   $0x74,%ecx,%ecx # 116 in op
0x401a29: imul   $0x95,%eax,%eax # 149 in eg
0x401a2f: add    %ecx,%edi # add to score
0x401a31: add    %eax,%esi
0x401a33: lea    -0x1(%r9),%rax # remove B
0x401a37: and    %rax,%r9   # from bit array
0x401a3a: jne    0x401971   # and loop
```

```
0x46b284: mov wB (0x7dd450),%r9
0x46b28b: mov    (0x7dd4a0),%r11 # rot
0x46b292: test   %r9,%r9 # if no wB skip
0x46b295: je     0x46b38b
0x46b2c0: mov    (0x7dd4a8),%rdx # rot
0x46b2c7: bsf wB %r9,%rax
0x46b2cb: movzbl 0x16bf80(%r14,%rax,4),%ecx
0x46b2d4: mov    %eax,%r8d
0x46b2d7: mov    %eax,%r10d
0x46b2da: shr    %cl,%rdx
0x46b2dd: movzbl 0x16be80(%r14,%rax,4),%ecx
0x46b2e6: shl    $0x6,%r8
0x46b2ea: and    $0x3f,%edx
0x46b2ed: mov    %r11,%rax
0x46b2f0: shr    %cl,%rax
0x46b2f3: add    %r8,%rdx
0x46b2f6: mov    0x174280(%r14,%rdx,8),%rcx
0x46b2fe: and    $0x3f,%eax # rcx has sq att
0x46b301: add    %r8,%rax
0x46b304: or     0x16c280(%r14,%rax,8),%rcx
0x46b30c: or     %rcx,%rbp # rbp is running
0x46b30f: test   %rcx,%r13 # see if bK att
0x46b312: je     0x46b31a  # if so then add
0x46b314: add    $0x101a2,%esi # (1,418)
0x46b31a: and    %r12,%rcx #  # AND w/~wOcc
[...] [popcnt] [...]
0x46b347: shr    $0x3c,%rax
0x46b34b: imul   $0x790079,%eax,%eax
0x46b351: add    %eax,%ebx # (121,121) each
0x46b353: movzbl 0x3353f0(%r10,%r14,1),%eax
0x46b35c: cmpl   $0x3,0x3dd320(%r14,%rax,4)
0x46b365: jne    0x46b36d # if trapped
0x46b367: add    $0xfa40fa41,%ebx # (1471,1471)
0x46b36d: lea    -0x1(%r9),%rax # remove B
0x46b371: and    %rax,%r9   # from bitboard
0x46b374: jne    0x46b2c0   # and loop
```

Note that Rybka 2.3.2a puts the trapped bishop check in the bishop loop (in particular, more than one bishop can now be trapped). The king attack weighting of 418 is used in each version. The mobility and trapped adjustments differ slightly; mobility is $(116, 149)$ in Rybka 1.0 Beta and $(121, 121)$ in Rybka 2.3.2a, and trapped as 1802 in Rybka 1.0 Beta and 1471 in Rybka 2.3.2a. The definitions of "mobility" and "trapped" remained the same, though the computation of trapped was changed.

1.3.1 Rybka 1.0 Beta trapped bishop code

For comparison, here is the trapped bishop code for Rybka 1.0 Beta (it appears later). Note that if multiple bishops are trapped, then only one penalty is applied (this is of course quite minor).

```
0x4026d4: mov    $0x4020200000000,%rdx     Mask: B5, B6, C7
0x4026de: shr    $0x7,%rax
0x4026e2: and    %r10,%rax                 Mask with black pawns
0x4026e5: test   %rax,%rdx                 test with wh bishops
0x4026e8: jne    0x402703
0x4026ea: mov    %rcx,%rax
0x4026ed: mov    $0x20404000000000,%rdx    Mask: G5, G6, F7
0x4026f7: shr    $0x9,%rax
0x4026fb: and    %r10,%rax
0x4026fe: test   %rax,%rdx                 if trapped bishop
0x402701: je     0x40270f
0x402703: sub    $0x70a,%esi                 subtract 1802 for op
0x402709: sub    $0x70a,%edi                                and eg
```

1.4 Rooks

```
0x401a48: mov wR (0x66e5c0),%r10
0x401a4f: mov    (0x66e5f8),%rax # rot
0x401a56: mov bP (0x66e598),%r15
0x401a5d: test   %r10,%r10 # if no wR
0x401a60: je     0x401c06 # then skip
0x401a70: mov    %rax,%rdx
0x401a73: bsf    %r10,%r9 # wR loop
0x401a7e: movzbl 0x29840(%rcx,%r9,4),%ecx
0x401a8e: mov    %r9d,%r8d
0x401a91: shr    %cl,%rdx
0x401a94: movzbl 0x29740(%rax,%r9,4),%ecx
0x401a9d: mov OC (0x66e5f0),%rax
0x401aa4: shr    %cl,%rax
0x401aa7: shl    $0x6,%r8
0x401aab: and    $0x3f,%edx
0x401aae: add    %r8,%rdx
0x401ab1: and    $0x3f,%eax
0x401ab4: add    %r8,%rax # rcx has attsq
0x401abe: mov    0x241940(%r8,%rdx,8),%rcx
0x401ac6: or     0x239940(%r8,%rax,8),%rcx
0x401ace: or     %rcx,%r14 # r14 is running
0x401ad1: test   %rcx,%r13 # test near bK
0x401ad4: je     0x401ae0
0x401ad6: add    $0x1,%ebp # if so, add 1
0x401ad9: add    $0x29a,%r12d # and wt 666
0x401ae0: and    %rbx,%rcx # AND ~WhOcc
[...] [popcnt] [...]
0x401b38: shr    $0x38,%rcx # mob count
0x401b3c: mov    %ecx,%eax
0x401b3e: imul   $0x4f,%ecx,%ecx # 79 op
0x401b41: imul   $0x54,%eax,%eax # 84 eg
0x401b44: add    %ecx,%edi
0x401b46: add    %eax,%esi
```

```
0x46b382: mov bK (0x7dd488),%r14
0x46b389: xor    %edi,%edi
0x46b38b: mov wR (0x7dd460),%r10
0x46b392: mov    (0x7dd498),%rax # rot
0x46b399: mov bP (0x7dd438),%r13
0x46b3a0: test   %r10,%r10 # if no wR
0x46b3a3: je     0x46b4d4 # then skip
0x46b3c0: mov    %rax,%rdx
0x46b3c3: mov OC (0x7dd490),%rax
0x46b3ca: bsf    %r10,%r9  # wR loop
0x46b3ce: movzbl 0x16c180(%rdi,%r9,4),%ecx
0x46b3d7: mov    %r9d,%r8d
0x46b3da: shr    %cl,%rdx
0x46b3dd: movzbl 0x16c080(%rdi,%r9,4),%ecx
0x46b3e6: shl    $0x6,%r8
0x46b3ea: shr    %cl,%rax
0x46b3ed: and    $0x3f,%edx
0x46b3f0: add    %r8,%rdx
0x46b3f3: and    $0x3f,%eax # rcx has attsq
0x46b3f6: mov    0x184280(%rdi,%rdx,8),%rcx
0x46b3fe: add    %r8,%rax
0x46b401: or     0x17c280(%rdi,%rax,8),%rcx
0x46b409: or     %rcx,%rbp # rbp is running
0x46b40c: test   %rcx,%r15 # test near bK
0x46b40f: je     0x46b417 # if so then
0x46b411: add    $0x1029a,%esi # add (1,666)
0x46b417: and    %r12,%rcx # AND ~WhOcc
[...] [popcnt] [...]
0x46b458: shr    $0x3c,%rax # mob count
0x46b45c: imul   $0x440052,%eax,%eax
0x46b462: add    %eax,%ebx # (68,82) each
```

Here is the first part of the Rook evaluation. Both give a weighting of 666 to a Rook that attacks a square around the enemy King. The mobility bonus is $(79, 84)$ per square in Rybka 1.0 Beta and $(68, 82)$ in Rybka 2.3.2a. The definition of mobility remains the same.

1.4.1 Rooks continued

```
0x401b48: mov    0x24a340(%r8,%r9,8),%rax
0x401b50:test wP %rax,(0x66e590) # half-open
0x401b57: jne    0x401b96 # if on half-open
0x401b59: add    $0x40,%esi # add 64 op
0x401b5c: add    $0x100,%edi # and 256 eg
0x401b62: test   %r15,%rax # full open test
0x401b65: jne    0x401b73 # if so
0x401b67: add    $0x3cb,%esi # add 971 op
0x401b6d: add    $0xac,%edi # and 172 eg
0x401b73: testb  $0x80,0x31(%rsp) # flag
0x401b78: je     0x401ba8 # for king safety
0x401b7a: test   %rax,%r13 # and file is near
0x401b7d: je     0x401b9f   # opposing king
0x401b86: add    $0x79,%esi # add 121 op
0x401b89: test   %rdx,%rax # file same as bK
0x401b8c: je     0x401baf    # then
0x401b8e: add    $0x355,%esi # add 853 op
0x401ba8: mov bK (0x66e5e8),%rdx
0x401baf: and    $0xfffffffffffffff8,%r9d
0x401bb3: cmp    $0x30,%r9d # check 7th
0x401bb7: jne    0x401bea # if so then
0x401bb9: mov    $0xff000000000000,%rax
0x401bc3: mov bP %r15,%rcx # see if bP
0x401bc6: and    %rax,%rcx # on 7th
0x401bc9: mov    %rdx,%rax # or bK
0x401bcc: mov    $0xff00000000000000,%rdx
0x401bd6: and    %rdx,%rax # on 8th
0x401bd9: or     %rax,%rcx
0x401bdc: je     0x401bea  # if either
0x401bde: add    $0xf6,%esi # add 246 op
0x401be4: add    $0x402,%edi # 1026 eg
0x401bea: lea    -0x1(%r10),%rax
0x401bee: and    %rax,%r10 # clear bit
0x401bf8: jne    0x401a70 # and loop
```

```
0x46b464: mov    0x18ca80(%rdi,%r9,8),%rax
0x46b46c:test wP %rax,(0x7dd430)
0x46b473: jne    0x46b491 # if half-open
0x46b475: add    $0x68010c,%ebx # (104,268)
0x46b47b: test   %r13,%rax # r13 is bP
0x46b47e: jne    0x46b486 # if open
0x46b480: add    $0x27300b4,%ebx # (627,180)
0x46b486: test   %r14,%rax # r14 is bK
0x46b489: je     0x46b491 # if bK on open
0x46b48b: add    $0x31b0000,%ebx # (795,0)
0x46b491: and    $0x38,%r9d # r9 is wR sq
0x46b495: cmp    $0x30,%r9b # check if 7th
0x46b499: jne    0x46b4b6
0x46b49b: mov bP %r13,%rax
0x46b49e: mov    $0xffff000000000000,%rcx
0x46b4a8: or  bK %r14,%rax # see if bK/bP
0x46b4ab: test   %rax,%rcx # is on 7th/8th
0x46b4ae: je     0x46b4b6 # if so then add
0x46b4b0: add    $0xc80430,%ebx # (200,1072)
0x46b4b6: lea    -0x1(%r10),%rax
0x46b4ba: and    %rax,%r10 # remove wR
0x46b4c4: jne    0x46b3c0 # and loop
```

For the second part of Rook evaluation, there are some differences beyond just numerology. Rybka 1.0 Beta gives a bonus for a Rook on a half-open file adjacent/on the opposing King file when king safety is operable (a material-based flag). The bonus is 121 (opening only) if adjacent and 853 if on. Rybka 2.3.2a has no material restrictions and only adds a bonus of $(795, 0)$ when the half-open rook is on the same file as the enemy King,

The condition for a 7th rank bonus also differs slightly, in that Rybka 1.0 Beta looks for a pawn on the 7th or a King on the 8th, while Rybka 2.3.2a looks for a pawn or King on the 7th or 8th (so a King on the 7th yields the bonus in Rybka 2.3.2a, but not in Rybka 1.0 Beta).

1.5 Queens

```
0x401c30: bsf    %r11,%r15
0x401c3b: lea    0x0(,%r15,4),%r8
0x401c43: mov    %r15d,%r9d
0x401c46: movzbl 0x29840(%r8,%rdx,1),%ecx
0x401c4f: mov    %rax,%rdx
0x401c52: mov OC (0x66e5f0),%rax
0x401c59: shr    %cl,%rdx
0x401c5c: movzbl 0x29740(%r8,%r10,1),%ecx
0x401c65: shl    $0x6,%r9
0x401c69: shr    %cl,%rax
0x401c6c: and*   $0x3f,%edx + $0x3f,%eax
0x401c72: add    %r9,%rdx
0x401c75: mov    0x241940(%r10,%rdx,8),%r10
0x401c7d: add    %r9,%rax
0x401c87: or     0x239940(%rdx,%rax,8),%r10
0x401c8f: movzbl 0x29640(%r8,%rdx,1),%ecx
0x401c98: mov    (0x66e608),%rax # rot
0x401c9f: shr    %cl,%rax
0x401ca2: movzbl 0x29540(%r8,%rdx,1),%ecx
0x401cab: and    $0x3f,%eax
0x401cae: add    %r9,%rax
0x401cb1: or     0x231940(%rdx,%rax,8),%r10
0x401cb9: mov    (0x66e600),%rax # rot
0x401cc0: shr    %cl,%rax
0x401cc3: and    $0x3f,%eax
0x401cc6: add    %r9,%rax # r10 has sq att
0x401cc9: or     0x229940(%rdx,%rax,8),%r10
0x401cd1: or     %r10,%r14 # r14 is running
0x401cd4: test   %r10,%r13 # see if hits bK
0x401cd7: je     0x401ce3
0x401cd9: add    $0x1,%ebp # if so inc att
0x401cdc: add    $0x214,%r12d # with 532 wt
0x401ce3: and    %rbx,%r10 # AND ~WhOcc
[...] [popcnt] [...]
0x401d3f: shr    $0x38,%rcx
0x401d43: mov    %ecx,%eax # for each sq
0x401d45: imul   $0x25,%ecx,%ecx # 37 in op
0x401d48: imul   $0x36,%eax,%eax # 54 in eg
0x401d4b: add*   %ecx,%edi + %eax,%esi
0x401d4f: cmp    $0x30,%r15d # check 7th rk
0x401d53: jne    0x401d88 # if so then
0x401d55: mov bP (0x66e598),%rcx
0x401d5c: mov    $0xff000000000000,%rax
0x401d66: mov    $0xff00000000000000,%r8
0x401d70: and    %rax,%rcx # see if bP/7th
0x401d73: mov bK (0x66e5e8),%rax
0x401d7a: and    %r8,%rax # or bK on 8th
0x401d7d: or     %rax,%rcx
0x401d80: je     0x401d88 # if either
0x401d82: add    $0x58c,%edi # 1420 in eg
0x401d88: lea    -0x1(%r11),%rax
0x401d8c: and    %rax,%r11 # clear bit
0x401d96: jne    0x401c30 # and loop
```

```
0x46b4f3: bsf    %r11,%r12
0x46b4fe: lea    0x0(,%r12,4),%r8
0x46b506: mov    %r12d,%r9d
0x46b509: movzbl 0x16c180(%r8,%rdx,1),%ecx
0x46b512: mov    %rax,%rdx
0x46b515: mov    (0x7dd4a8),%rax # rot
0x46b51c: shr    %cl,%rdx
0x46b51f: movzbl 0x16bf80(%r8,%r10,1),%ecx
0x46b528: shl    $0x6,%r9
0x46b52c: shr    %cl,%rax
0x46b52f: and    $0x3f,%edx
0x46b532: and    $0x3f,%eax
0x46b535: add    %r9,%rdx
0x46b538: mov    0x184280(%r10,%rdx,8),%r10
0x46b540: add    %r9,%rax
0x46b54a: or     0x174280(%rdx,%rax,8),%r10
0x46b552: movzbl 0x16c080(%r8,%rdx,1),%ecx
0x46b55b: mov OC (0x7dd490),%rax
0x46b562: shr    %cl,%rax
0x46b565: movzbl 0x16be80(%r8,%rdx,1),%ecx
0x46b56e: and    $0x3f,%eax
0x46b571: add    %r9,%rax
0x46b574: or     0x17c280(%rdx,%rax,8),%r10
0x46b57c: mov    (0x7dd4a0),%rax # rot
0x46b583: shr    %cl,%rax
0x46b586: and    $0x3f,%eax
0x46b589: add    %r9,%rax # r10 has sq att
0x46b58c: or     0x16c280(%rdx,%rax,8),%r10
0x46b594: or     %r10,%rbp # rbp is running
0x46b597: test   %r10,%rdi # see if bK att
0x46b59a: je     0x46b5a2 # if so add
0x46b59c: add    $0x10214,%esi # (1,532)
0x46b5a2: and    %r15,%r10 # AND ~WhOcc
[...] [popcnt] [...]
0x46b5fe: shr    $0x38,%rcx
0x46b602: imul   $0x2c0026,%ecx,%ecx
0x46b608: add    %ecx,%ebx # (44,38) each
0x46b5af: and    $0x38,%r12d # check 7th
0x46b60a: cmp    $0x30,%r12b
0x46b60e: jne    0x46b62b
0x46b610: mov bP  %r13,%rax # r13 is bP
0x46b613: mov    $0xffff000000000000,%rcx
0x46b61d: or  bK %r14,%rax # r14 is bK
0x46b620: test   %rax,%rcx # see if bK/bP
0x46b623: je     0x46b62b # is on 7th/8th
0x46b625: add    $0x5ce,%ebx # (0, 1486)
0x46b62b: lea    -0x1(%r11),%rax
0x46b62f: and    %rax,%r11 # clear bit
0x46b639: jne    0x46b4f3 # and loop
```

1.5.1 Queens continued

A few instructions in the above left are condensed so as to fit on the page. As can be seen, the 7th rank condition in Rybka 2.3.2a is modified (as with the Rooks). The weighting of 532 for King threats is retained, while the mobility bonus changes from $(37, 54)$ to $(44, 38)$. Beyond the change of "7th rank" definition as with Rooks, the bonus here changes from 1420 in 1486 in the endgame.

1.6 Calling pawn evaluation and king safety

```
0x401db1: callq  0x4087b0 # pawneval
0x401db6: mov    (0x66e618),%r9d # oo flags
0x401dbd: mov    %rax,%r8 # pawn_info ptr
0x401dc5: movswl 0x4(%rax),%eax # op score
0x401dc9: add    %eax,%esi # from pawneval
0x401dcb: movswl 0x6(%r8),%eax # eg score
0x401dd0: add    %eax,%edi # from pawneval
0x401dd2: testb  $0x80,0x31(%rsp) # flag
0x401dd7: je     0x401e36 # for bK safety
0x401de5: mov    0x262e80(%rcx,%rbp,4),%ecx
0x401dec: imul   %r12d,%ecx # r12 is wt
0x401df7: sar    $0x5,%ecx # div by 32
0x401e03: add    %ecx,%esi # add to op
0x401dfa: movzbl 0x260b50(%rax,%r12,1),%eax
0x401e09: movzwl 0xe(%r8,%rax,2),%edx
0x401e0f: mov    %edx,%ecx # Pen(FileWing)
0x401e05: test   $0x4,%r9b # test for Bl oo
0x401e11: je     0x401e1d
0x401e13: movzwl 0x12(%r8),%eax # Pen(g8)
0x401e18: cmp    %edx,%eax # see if better
0x401e1a: cmovb  %eax,%ecx # if so replace
0x401e1d: test   $0x8,%r9b # test Bl ooo
0x401e21: je     0x401e2d
0x401e23: movzwl 0xe(%r8),%eax # Pen(c8)
0x401e28: cmp    %ecx,%eax
0x401e2a: cmovb  %eax,%ecx
0x401e2d: lea    (%rcx,%rdx,1),%eax # add
0x401e30: shr    %eax # and div by 2, avg
0x401e32: add    %eax,%esi # add to op
```

```
0x46b674: callq  0x471030 # pawn eval
0x46b679: add    0x18(%rax),%ebx # add score
0x46b67c: mov    %rax,%r9 # pawn_info ptr
0x46b684: mov    $0x180000000000,%rax # bK safe
0x46b68e: test   %rax,(0x7dd4d8) # test mattoken
0x46b695: je     0x46b6d8 # if !flag then skip
0x46b6a3: movzwl %si,%ecx # %si = packed (att,wt)
0x46b6a6: movzbl 0x34a360(%rax,%r10,1),%eax
0x46b6af: movslq %esi,%rdx
0x46b6b2: sar    $0x10,%rdx # get att count
0x46b6b6: mov    0x34a7b0(%r10,%rdx,4),%r8d
0x46b6be: imul   %ecx,%r8d # mult by weight
0x46b6c8: sar    $0x5,%r8d # divide by 32
0x46b6c2: movswl 0x10(%r9,%rax,2),%ecx # Pen(FW)
0x46b6cc: add    %ecx,%r8d # add shelter/storm
0x46b6cf: shl    $0x10,%r8d # redo bit packing
0x46b6d3: add    %r8d,%ebx # add to op
```

```
It seems the condition for king safety ends
up being the same as in Rybka 1.0 Beta.
Namely, a queen and another piece.
```

The king safety is the same, except that Rybka 2.3.2a does not consider castling rights when looking at the shelter/storm penalty. The pawn evaluation is explored more completely in Section 2 below. The bit-packing in Rybka 2.3.2a for the (`attackers`, `weight`) construction induces some methodological differences.

1.7 Parallel code for Black

These procedures are then repeated for Black.

1.8 Passed Pawns

```
0x4023cb: movzbl 0x14(%r8),%r9d # files
0x4023d0: mov w0 (0x66e580),%r12
0x402410: mov    0x260b90(%r10,%r9,4),%eax
0x402418: mov    0x260b10(%r10,%rax,8),%rax
0x402420: and    %rbx,%rax # get file
0x402423: bsr    %rax,%r10 # get sq of pawn
0x40242e: mov    0x24a340(%rax,%r10,8),%rcx
0x402436: mov    %r10d,%r8d
0x402439: shr    $0x3,%r8 # rank of pawn
0x40243d: add    0x260f90(%rax,%r8,4),%esi
0x402445: add    0x260fb0(%rax,%r8,4),%edi
0x40244d: test   %r11,%r11 # add base score
0x402450: jne    0x40248a # if no Bl piece
0x402452: test   %rcx,%r12 # and P is
0x402455: jne    0x402478 # not blocked
0x402457: mov    (0x66e610),%eax # wtm test
0x40245d: lea    (%rax,%r10,2),%rcx
0x402468: test   %r13,0x262280(%rax,%rcx,8)
0x402470: je     0x402478 # and bK not in SQ
0x402472: add    $0x6400,%edi # 25600 bonus
0x402478: test   %r15,0x262a80(%rax,%r10,8)
0x402480: je     0x4024d1 # SQUARE for btm
0x402482: add    $0x6400,%edi # 25600 bonus
0x402488: jmp    0x4024d1 # else
0x40248a: test   %rcx,%r12 # if Wh no block
0x40248d: jne    0x402497 # rank-based bonus
0x40248f: add    0x260fd0(%rax,%r8,4),%edi
0x402497: mov b0 (0x66e588),%rax
0x40249e: test   %rcx,%rax # and Bl no block
0x4024a1: jne    0x4024b4 # rank-based bonus
0x4024aa: add    0x260ff0(%rdx,%r8,4),%edi
0x4024bb: mov    %r14,%rax # Bl squares att
0x4024be: not    %rax # NOT these
0x4024c1: and    %rcx,%rax # in path
0x4024c4: test   %rax,%rbp # Wh squares att
0x4024c7: jne    0x4024d1 # if OK bonus(rank)
0x4024c9: add    0x261010(%rdx,%r8,4),%edi
0x4024d6: lea    0x8(%r10),%ecx # (sq + 8)
0x4024e1: shl    $0x6,%rcx # lookup dist
0x4024e5: add    %rcx,%rax # bK to sq+8
0x4024e8: movzbl 0x261280(%rax,%r10,1),%edx
0x4024f6: add    %rcx,%rax
0x4024f9: imul   0x261050(%r10,%r8,4),%edx
0x402502: movzbl 0x261280(%rax,%r10,1),%ecx
0x40250f: imul   0x261030(%r10,%r8,4),%ecx
0x402518: sub    %ecx,%edx # sub wK/wP dist
0x40251a: add    %edx,%edi # add bK/wP dist
0x40250b: lea    -0x1(%r9),%eax # clr file
0x40251c: and    %eax,%r9d
0x40251f: jne    0x402410 # and loop
```

```
0x46bbdb: movzbl 0x1c(%r8),%r9d # files
0x46bbea: test   %r9d,%r9d # if none skip
0x46bbf4: je     0x46bcca
0x46bbed: mov w0 (0x7dd420),%rdi
0x46bbfa: mov    %rbp,%r10
0x46bbfd: mov b0 (0x7dd428),%rbp
0x46bc0b: not    %r10
0x46bc10: movzbl 0x34a3a0(%r9,%r15,1),%eax
0x46bc19: mov    0x335c30(%r15,%rax,8),%rax
0x46bc21: and    %r12,%rax
0x46bc24: bsr    %rax,%rdx # loop over files
0x46bc28: mov    %edx,%r8d
0x46bc2b: mov    0x18ca80(%r15,%rdx,8),%rcx
0x46bc33: test   %rcx,%rdi # if Wh no block
0x46bc36: jne    0x46bc47
0x46bc38: mov    %r8,%rax
0x46bc3b: shr    $0x3,%rax # add Bonus(rk)
0x46bc3f: add    0x34a500(%r15,%rax,4),%ebx
0x46bc47: test   %rcx,%rbp # if Bl no block
0x46bc4a: jne    0x46bc5b
0x46bc4c: mov    %r8,%rax
0x46bc4f: shr    $0x3,%rax # add Bonus(rk)
0x46bc53: add    0x34a520(%r15,%rax,4),%ebx
0x46bc5b: mov    %r10,%rax
0x46bc5e: and    %rcx,%rax
0x46bc61: test   %rax,%r11 # if path is OK
0x46bc64: jne    0x46bc75
0x46bc66: mov    %r8,%rax
0x46bc69: shr    $0x3,%rax # add Bonus(rk)
0x46bc6d: add    0x34a540(%r15,%rax,4),%ebx
0x46bc75: lea    0x8(%rdx),%ecx # (sq + 8)
0x46bc78: shr    $0x3,%r8
0x46bc7c: shl    $0x6,%rcx
0x46bc80: lea    (%rcx,%rsi,1),%rax
0x46bc84: movzbl 0x352f00(%rax,%r15,1),%edx
0x46bc8d: lea    (%rcx,%r13,1),%rax
0x46bc91: movzbl 0x352f00(%rax,%r15,1),%ecx
0x46bc9e: imul   0x34a580(%r15,%r8,4),%edx
0x46bca7: imul   0x34a560(%r15,%r8,4),%ecx
0x46bcb0: sub    %ecx,%edx # sub wK/wP dist
0x46bcb2: add    %edx,%ebx # add bK/wP dist
0x46bc9a: lea    -0x1(%r9),%eax
0x46bcb4: and    %eax,%r9d # clear file
0x46bcb7: jne    0x46bc10 # loop
```

1.9 Passed pawns continued

Again there are a few minor details about register-loading elided, so as to fit the page. The "standard" bonus for a passed pawn was moved to pawn evaluation. The "unstoppable passer" code in Rybka 1.0 Beta seems to have been moved to the separate pawn endgame evaluator in Rybka 2.3.2a. There is parallel code for Black passers. The values, with Rybka 1.0 Beta on the left:

```
UnblockedOwn  {0,0,0, 26, 78,257, 262, 262}   {0,0,0, 27, 82,164, 274, 274}
UnblockedOpp  {0,0,0,133,394,788,1311,1311}   {0,0,0,139,412,825,1372,1372}
PassedFree    {0,0,0,101,300,601,1000,1000}   {0,0,0,106,314,629,1046,1046}
AttDistance   {0,0,0, 66,195,391, 650, 650}   {0,0,0, 69,204,409, 680, 680}
DefDistance   {0,0,0,131,389,779,1295,1295}   {0,0,0,137,407,815,1355,1355}
```

1.10 Patterns

The trapped Bishop code for Rybka 1.0 Beta was indicated in §1.3.1 above, though here would be its proper place in the code. Rybka 1.0 Beta then does blocked Bishops, which are omitted in Rybka 2.3.2a. Both then do blocked Rooks. Here is an example of the blocked Bishop code in Rybka 1.0 Beta (it appears 4 times, with the obvious modifications for white/black and king/queenside).

```
0x40277d: test   $0x20,%cl   # check if F1 sq of wB is et
0x402780: je     0x402796
0x402782: bt     $0xc,%rdx   # check if E2 sq of wP is set
0x402787: jae    0x402796
0x402789: bt     $0x14,%rax  # check if E3 sq is nonempty
0x40278e: jae    0x402796
0x402790: sub    $0x780,%esi # if all true, sub 1920 in op
```

Here is first the blocked White Rook code in Rybka 1.0 Beta, and then the same for Rybka 2.3.2a

```
0x4027fc: test   $0x103,%r8   # Mask A1,A2,B1 to wR
0x402803: je     0x402811
0x402805: test   $0x6,%r15b   # Mask B1,C1 to wK
0x402809: je     0x402811
0x40280b: sub    $0x780,%esi  # if both true, sub 1920 in op
0x402811: test   $0x80c0,%r8  # Mask G1,H1,H2 to wR
0x402818: je     0x402826
0x40281a: test   $0x60,%r15b  # Mask F1,G1 to wK
0x40281e: je     0x402826
0x402820: sub    $0x780,%esi  # if both true, sub 1920 in op

0x46bde1: mov    0x3351f0(%rdi,%rax,8),%rax # K-based mask
0x46bde9:test wR %rax,(0x7dd460)
0x46bdf0: je     0x46bdf8
0x46bdf2: add    $0xf9e10000,%ebx # penalty (1561,0)
```

Rybka 2.3.2a uses a mask based on the square of the King (rather than generic masks), but the result is the same. The scores changes from 1920 to 1561.

1.11 Interpolation, drawishness, and finale

```
0x402885: test   %bl,%bl # Bend test
0x402887: je     0x4028bc
0x402889: or     %rcx,%r9 # OR wB|BB
0x40288c: mov    $0x55aa55aa55aa55aa,%rax
0x402896: test   %r9,%rax
0x402899: je     0x4028bc
0x40289b: mov    $0xaa55aa55aa55aa55,%rax
0x4028a5: test   %r9,%rax
0x4028a8: je     0x4028bc # if opp col
0x4028aa: mov    %esi,%eax
0x4028ac: cltd
0x4028ad: sub    %edx,%eax
0x4028af: sar    %eax     # div op by 2
0x4028b1: mov    %eax,%esi
0x4028b3: mov    %edi,%eax
0x4028b5: cltd
0x4028b6: sub    %edx,%eax
0x4028b8: sar    %eax     # div eg by 2
0x4028ba: mov    %eax,%edi
0x4028bc: movzbl 0x32(%rsp),%eax # phase
0x4028c1: mov    0x261074(%r11,%rax,8),%ecx
0x4028c9: mov    0x261070(%r11,%rax,8),%eax
0x4028d1: imul   %edi,%ecx # interpolate
0x4028d4: imul   %esi,%eax
0x4028d7: add    %eax,%ecx
0x4028d9: sar    $0xd,%ecx # final result
```

```
0x46be1d: lea    -0x8000(%rax,%rcx,1),%r9d
0x46be25: movzwl %bx,%eax # unpack (16+16)
0x46be28: lea    -0x8000(%rax,%rcx,1),%r8d
0x46be30: mov    %r14,%rax # mat token
0x46be33: shr    $0x1a,%rax # phase part
0x46be37: movzbl %al,%ecx # out of 64
0x46be3a: mov    $0x55555556,%eax
0x46be3f: imul   %ecx
0x46be41: mov    $0x3f000000000000,%rcx
0x46be4b* mov    %edx,%eax # (rounding)
0x46be52: mov    $0x40,%eax # bounds
0x46be57: cmp    %eax,%edx  # for phase
0x46be59: cmovg  %eax,%edx
0x46be5c: movslq %edx,%rax
0x46be5f: mov    0x34a5a4(%rdi,%rax,8),%r10d
0x46be67: mov    0x34a5a0(%rdi,%rax,8),%eax
0x46be6e: imul   %r9d,%eax
0x46be72: imul   %r8d,%r10d # interpolate
0x46be76: add    %eax,%r10d
0x46be79: mov    %r14,%rax
0x46be7c: sar    $0xd,%r10d # final result
0x46be80: and    %rcx,%rax
0x46be83: mov    $0x30000000000000,%rcx
0x46be8d: cmp    %rcx,%rax # check for Bend
0x46be90: jne    0x46bed7
0x46be92: shr    $0x2d,%r14 # pawn imbalance
0x46be96: and    $0x7,%r14b # starts at 2
0x46be9a: cmp    $0x4,%r14b # +1 wP, -1 bP
0x46be9e: jg     0x46bed7 # 5,6,7 are bad
0x46bea0: mov bB (0x7dd458),%rax
0x46bea7: mov    $0x55aa55aa55aa55aa,%rcx
0x46beb1: or  wB (0x7dd450),%rax
0x46beb8: test   %rax,%rcx
0x46bebb: je     0x46bed7
0x46bebd: mov    $0xaa55aa55aa55aa55,%rcx
0x46bec7: test   %rax,%rcx
0x46beca: je     0x46bed7
0x46becc: mov    %r10d,%eax
0x46bed2* sar    %eax  # half score if oppB
```

Rybka 1.0 Beta next checks if an opposite bishop endgame is extant, and divides both opening/endgame scores by 2 if so, followed by interpolation. Rybka 2.3.2a reverses these two steps, and has a slightly different condition (pawn imbalance is either 2 or less [as with Rybka 1.0 Beta], or 6 or more) for applying the halving. I omitted some instructions (like `cltd`) that ensure proper rounding.

1.11.1 Interpolation continued

Both Rybkas use a table of 64 entries involving "Fruit phases" of which only 25 entries can ever be used. By looking at the respective table entries it turns out that the interpolation in Rybka 1.0 Beta is not exactly linear, but the one used by Rybka 2.3.2a is.

1.11.2 Move bonus

Rybka 1.0 Beta concludes by giving a bonus of 3 centipawns (by now everything has been changed from the 3399ths scale) to the side on move. I omit the saving of positional scores and mobility in the pointer structure.

```
0x4028dc: cmpl   $0x0,(0x66e610)   # if wtm
0x4028e3: jne    0x402902
0x4028e5: lea    0x3(%rcx),%eax    # add 3 to score
0x402900: jmp    0x402921
0x402902: mov    $0x3,%eax         # else
0x402907: sub    %ecx,%eax         # subtract 3
0x402939: jmp    0x40295f          # goto return
```

1.11.3 Drawishness from pawn structure

Rybka 2.3.2a does not give such a "tempo" bonus (at least not directly), but instead has a final adjustment based on the drawishness of the pawn structure.

```
0x46bed7: test   %r10d,%r10d       # split based on who is winning
0x46beda: jle    0x46bf12
0x46bedc: movzbl 0x1e(%r12),%ecx   # white drawishness from pawninfo
0x46bee2: mov    $0x88888889,%eax
0x46bee7: imul   $0x9c,%ecx,%ecx   # check if close enough to use...
0x46beed: imul   %ecx
0x46beef: add    %ecx,%edx
0x46bef1: sar    $0x7,%edx         # (rounding assurance omitted)
0x46bef4* mov    %edx,%eax
0x46befb: mov    $0x64,%eax
0x46bf00: cmp    %eax,%r10d
0x46bf03: cmovl  %r10d,%eax
0x46bf07: imul   %eax,%edx
0x46bf0a: sar    $0x8,%edx
0x46bf0d: sub    %edx,%r10d
0x46bf10: jmp    0x46bf46
0x46bf12: movzbl 0x1f(%r12),%ecx   # black drawishness from pawninfo
[...]
```

2 Pawn evaluation

2.1 Preliminaries

```
0x4087bb: mov    (0x66e638),%r14 # P hash
0x4087c6: mov    %r14,%rax
0x4087c9: shr    $0x20,%r14
0x4087cd: and    $0x7ffff,%eax
0x4087d7: lea    (%rax,%rax,2),%rcx
0x4087db: mov    (0x66c2f0),%rax
0x4087e2: cmp    %r14d,(%rax,%rcx,8)
0x4087e6: lea    (%rax,%rcx,8),%r15
0x4087ef: je     0x408f74 # if hashhit, ret
0x4087f9: mov bP (0x66e598),%rbx
0x40880f: mov wP (0x66e590),%rdi
0x40881e: test   %rdi,%rdi # if no wP
0x40883d: je     0x4089f9 # then skip
```

```
0x047103f: mov    (0x7dd4d0),%r14 # P hash
0x0471046: movzwl %r14w,%r12d
0x047104a: mov    %r14,0x8(%rsp)
0x047104f: shl    $0x5,%r12
0x0471053: add    (07xf1518),%r12
0x047105a: cmp    %r14,(%r12) # if hashhit
0x047105e: je     0x471803 # then return
0x0471064: mov wP (0x7dd430),%r11
0x04710b2* mov bP (0x7dd438),%rdi
0x0471076: test   %r11,%r11 # if no wP
0x04710ac: je     0x471224 # then skip
```

Both start by looking for a pawn-hash hit. The pawn-hash entries are 24 bytes in Rybka 1.0 Beta and 32 bytes in Rybka 2.3.2a.

2.2 Main Loop

The main loop (replicated for Black pawns of course) is almost the same in both Rybka 1.0 Beta and Rybka 2.3.2a. Other than the varying of the scores, the only component that differs is that Rybka 2.3.2a adds the "fixed" bonus for a passed pawn at this stage rather than in the passed pawn loop, and only does so for the frontmost such pawn on a given file.

2.2.1 Main loop continued

```
0x408870: bsf    %rsi,%rax
0x408874: mov    %eax,%r10d
0x408877: test   %rdi,0x22720(%r14,%rax,8)
0x40887f: je     0x408888 # doubled
0x408881: sub    $0x9e,%r8d # 158 eg pen
0x408888: test   %r13,0x24a340(%r14,%rax,8)
0x408890: sete   %al # open file
0x408893: test   %rdi,0x22920(%r14,%r10,8)
0x40889b: jne    0x4088be # isolated
0x40889d: test   %al,%al  # if also open
0x40889f: je     0x4088b1
0x4088a1: sub    $0x39b,%r9d # 923 op pen
0x4088a8: sub    $0x144,%r8d # 324 eg pen
0x4088af: jmp    0x408908  # else
0x4088b1: sub    $0x4f,%r9d  # 79 op pen
0x4088b5: sub    $0x144,%r8d # 324 eg pen
0x4088bc: jmp    0x408908  # if not isol
0x4088be: test   %rdi,0x22b20(%r14,%r10,8)
0x4088c6: jne    0x408908  # check bkwd
0x4088c8: test   %rbx,0x22f20(%r14,%r10,8)
0x4088d0: jne    0x4088e6
0x4088d2: test   %rbx,0x23320(%r14,%r10,8)
0x4088da: je     0x408908
0x4088dc: test   %rdi,0x249f40(%r14,%r10,8)
0x4088e4: jne    0x408908
0x4088e6: test   %al,%al    # if also open
0x4088e8: je     0x4088fa
0x4088ea: sub    $0x368,%r9d  # 872 op pen
0x4088f1: sub    $0x129,%r8d  # 297 eg pen
0x4088f8: jmp    0x408908   # else
0x4088fa: sub    $0x196,%r9d  # 406 op pen
0x408901: sub    $0x129,%r8d  # 297 eg pen
0x408908: mov    0x23720(%r14,%r10,8),%r11
0x408910: and    %rbx,%r11  # check passed
0x408913: jne    0x408928   # if so,
0x408915: mov    %r10b,%al  # then
0x408918: and    $0x7,%eax  # store file
0x40891b: or     0x23b20(%r14,%rax,4),%ebp
0x408923: jmpq   0x4089d3   # else check
0x408928: test   %al,%al    # for candidate
0x40892a: je     0x4089d3 # (via open file)
0x408930: mov    0x22b20(%r14,%r10,8),%rcx
0x408938: and    %rdi,%rcx # white pawn mask
[...] [popcnt] [...] [white/black prot/att]
0x4089af: shr    $0x38,%rcx
0x4089b7: shr    $0x38,%rax
0x4089bb: cmp    %ecx,%eax # if guarded
0x4089bd: jb     0x4089d3  # enough then
0x4089bf: shr    $0x3,%r10 # add Bonus(rank)
0x4089c3: add    0x23b40(%r14,%r10,4),%r9d
0x4089cb: add    0x23b60(%r14,%r10,4),%r8d
0x4089d3: lea    -0x1(%rsi),%rax
0x4089d7: and    %rax,%rsi # clear bit
0x4089da: jne    0x408870  # and loop
```

```
0x4710e0: bsf    %rbx,%rax
0x4710e4: mov    %eax,%r9d
0x4710e7: test   %r11,0x332df0(%r15,%rax,8)
0x4710ef: je     0x4710f8 # doubled penalty
0x4710f1: sub    $0x4e00a5,%r8d # (78,165)
0x4710f8: mov    0x18ca80(%r15,%rax,8),%rcx
0x471100: test   %rcx,%r13
0x471103: sete   %al # open file
0x471106: test   %r11,0x332ff0(%r15,%r9,8)
0x47110e: jne    0x471126 # isolated
0x471110: test   %al,%al  # if also open
0x471112: je     0x47111d # penalty is
0x471114: sub    $0x20a029e,%r8d # (522,670)
0x47111b: jmp    0x471162 # else penalty is
0x47111d: sub    $0x9c00c9,%r8d  # (156,201)
0x471124: jmp    0x471162 # if not isol
0x471126: test   %r11,0x3331f0(%r15,%r9,8)
0x47112e: jne    0x471162 # check bkwd
0x471130: test   %rdi,0x3345f0(%r15,%r9,8)
0x471138: jne    0x47114e
0x47113a: test   %rdi,0x3349f0(%r15,%r9,8)
0x471142: je     0x471162
0x471144: test   %r11,0x18c680(%r15,%r9,8)
0x47114c: jne    0x471162
0x47114e: test   %al,%al  # if also open
0x471150: je     0x47115b # penalty is
0x471152: sub    $0x1a20218,%r8d # (418,536)
0x471159: jmp    0x471162 # else penalty is
0x47115b: sub    $0x8200a7,%r8d  # (130,167)
0x471162: mov    0x334df0(%r15,%r9,8),%r10
0x47116a: and    %rdi,%r10 # check passed
0x47116d: jne    0x471190  # if so, then
0x47116f: mov    %r9b,%al  # store file
0x471172: and    $0x7,%eax
0x471175: or     0x34a4a0(%r15,%rax,4),%esi
0x47117d: test   %rcx,%r11 # if passed pawn
0x471180: jne    0x471201  # is frontmost
0x471182: shr    $0x3,%r9  # add Bonus(rk)
0x471186: add    0x34a4c0(%r15,%r9,4),%r8d
0x47118e: jmp    0x471201
0x471190: test   %al,%al   # else check
0x471192: je     0x471201  # for candidate
0x471194: mov    0x3331f0(%r15,%r9,8),%rdx
0x47119c: and    %r11,%rdx # (via openfile)
[...] [popcnt] [...]
0x4711e9: shr    $0x3c,%rcx
0x4711ed: shr    $0x3c,%rax
0x4711f1: cmp    %cl,%al # if well-protect
0x4711f3: jl     0x471201
0x4711f5: shr    $0x3,%r9  # add Bonus(rk)
0x4711f9: add    0x34a4e0(%r15,%r9,4),%r8d
0x471201: lea    -0x1(%rbx),%rax
0x471205: and    %rax,%rbx # clear bit
0x471208: jne    0x4710e0  # and loop
```

2.3 Drawishness from pawn files

Rybka 2.3.2a computes which non-edge files have pawns of each colour, and determines a value by table lookup from the 2^6 possibilities. The White code:

```
0x471224: movzbl (0x7dd432),%eax
0x47122b: movzbl (0x7dd431),%ecx
0x471232: or     %rax,%rcx
0x471235: movzbl (0x7dd433),%eax
0x47123c: or     %rax,%rcx
0x47123f: movzbl (0x7dd434),%eax
0x471246: or     %rax,%rcx
0x471249: movzbl (0x7dd435),%eax
0x471250: or     %rax,%rcx
0x471253: movzbl (0x7dd436),%eax
0x47125a: or     %rax,%rcx                  # b-g files
0x47125d: movzbl 0x351440(%rcx,%r15,1),%eax  # table lookup
0x471266: mov    %al,0x1e(%r12)             # store
```

2.4 Computing Shelter/Storm

```
0x408baa: mov    %r8w,0x6(%r15) # save
0x408baf: mov    %r9w,0x4(%r15) # scores
0x408ba6: movzbl (%rsp),%eax
0x408bb4: mov    %al,0x14(%r15) # passed
0x408bb8: movzbl 0x4(%rsp),%eax
0x408bc7: mov    %al,0x15(%r15) # files
0x408bbd: mov    $0x1c0000000,%r9
0x408bcb: mov bP (0x66e598),%rcx
0x408bdd: and    $0x7000000,%eax
0x408bd7: mov    %rcx,%rax
0x408bda: mov    %rcx,%r8
0x408bdd: and    $0x7000000,%eax
0x408be2: shr    $0x5,%r8
0x408be6: and    $0x38000000,%r8d
0x408bed: or     %rax,%r8
0x408bf0: mov    %rcx,%rax
0x408bf3: and    $0x700,%ecx
0x408bf9: and    $0x70000,%eax
0x408bfe: shr    $0x5,%r8
0x408c02: or     %rax,%r8
0x408c05: shr    $0x5,%r8
0x408c09: or     %rcx,%r8
[...]
0x408c44: movzwl 0x25b80(%r13,%r8,2),%eax
0x408c58: add    0x23b80(%r13,%rdx,2),%ax
```

```
0x471436: mov    %r8d,0x18(%r12) # score
0x471440: movzbl (%rsp),%eax
0x471444: mov    %al,0x1c(%r12) # passed
0x471449: movzbl 0x4(%rsp),%eax
0x47144e: mov    %al,0x1d(%r12) # files
0x4713f8* mov    $0x1c0000000,%r9
0x471453: mov bP (0x7dd438),%rcx
0x47145a: mov    %rcx,%rax
0x47145d: mov    %rcx,%r8
0x471460: and    $0x7000000,%eax
0x471465: shr    $0x5,%r8
0x471469: and    $0x38000000,%r8d
0x471470: or     %rax,%r8
0x471473: mov    %rcx,%rax
0x471476: and    $0x700,%ecx
0x47147c: and    $0x70000,%eax
0x471481: shr    $0x5,%r8
0x471485: or     %rax,%r8
0x471488: shr    $0x5,%r8
0x47148c: or     %rcx,%r8
[...]
0x4714c7: movzwl 0x356b00(%r15,%r8,2),%eax
0x4714db: add    0x354b00(%r15,%rdx,2),%ax
```

After saving scores and files of passed pawns in a structure, both Rybka versions compute shelter/storm via masks (only the first is shown here).

3 PST

The Rybka 2.3.2a PST tables start at `0x750840` in the 64-bit single-cpu version and are packed as two 16-bit scores. The Rybka 1.0 Beta PST tables (appearing on the left here) start at `0x64b4f0` in the 64-bit version, with opening/endgame values alternatively as 16-bit entries.

3.1 White Pawns

Table 1: Opening values

-543	-181	0	181	181	0	-181	-543
-543	-181	0	181	181	0	-181	-543
-543	-181	0	181	181	0	-181	-543
-543	-181	0	255	255	0	-181	-543
-543	-181	0	181	181	0	-181	-543
-543	-181	0	181	181	0	-181	-543
-543	-181	0	181	181	0	-181	-543
-543	-181	0	181	181	0	-181	-543

0	0	0	0	0	0	0	0
-522	-117	104	326	326	104	-117	-522
-522	-117	104	326	326	104	-117	-522
-548	-156	65	287	287	65	-156	-548
-574	-182	26	235	235	26	-182	-574
-600	-209	-13	182	182	-13	-209	-600
-613	-222	-39	143	143	-39	-222	-613
0	0	0	0	0	0	0	0

Table 2: Endgame values

291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291
291	97	0	-97	-97	0	97	291

0	0	0	0	0	0	0	0
258	168	78	-11	-11	78	168	258
257	168	78	-11	-11	78	168	257
218	128	38	-50	-50	38	128	218
180	90	0	-90	-90	0	90	180
167	76	-12	-102	-102	-12	76	167
167	76	-12	-102	-102	-12	76	167
0	0	0	0	0	0	0	0

3.2 White Knights

Table 3: Opening values

-5618	-1724	-1030	-683	-683	-1030	-1724	-5618
-1366	-672	22	369	369	22	-672	-1366
-314	380	1074	1421	1421	1074	380	-314
-325	369	1063	1410	1410	1063	369	-325
-683	11	705	1052	1052	705	11	-683
-1388	-694	0	347	347	0	-694	-1388
-2440	-1746	-1052	-705	-705	-1052	-1746	-2440
-3492	-2798	-2104	-1757	-1757	-2104	-2798	-3492

-4582	-1404	-839	-555	-555	-839	-1404	-4582
-1111	-546	19	302	302	19	-546	-1111
-254	311	876	1159	1159	876	311	-254
-264	302	867	1149	1149	867	302	-264
-555	9	574	858	858	574	9	-555
-1130	-565	0	283	283	0	-565	-1130
-1988	-1422	-858	-574	-574	-858	-1422	-1988
-2845	-2280	-1715	-1432	-1432	-1715	-2280	-2845

Table 4: Endgame values

-448	-336	-224	-168	-168	-224	-336	-448
-336	-224	-112	-56	-56	-112	-224	-336
-224	-112	0	56	56	0	-112	-224
-168	-56	56	112	112	56	-56	-168
-168	-56	56	112	112	56	-56	-168
-224	-112	0	56	56	0	-112	-224
-336	-224	-112	-56	-56	-112	-224	-336
-448	-336	-224	-168	-168	-224	-336	-448

-810	-627	-417	-365	-365	-417	-627	-810
-600	-417	-157	-104	-104	-157	-417	-600
-443	-183	104	157	157	104	-183	-443
-391	-157	130	183	183	130	-157	-391
-417	-183	78	130	130	78	-183	-417
-496	-261	0	52	52	0	-261	-496
-653	-496	-261	-209	-209	-261	-496	-653
-836	-679	-522	-470	-470	-522	-679	-836

3.3 White Bishops

Table 5: Opening values

-504	-588	-441	-294	-294	-441	-588	-504
-588	84	-147	0	0	-147	84	-588
-441	-147	378	147	147	378	-147	-441
-294	0	147	672	672	147	0	-294
-294	0	147	672	672	147	0	-294
-441	-147	378	147	147	378	-147	-441
-588	84	-147	0	0	-147	84	-588
-755	-839	-692	-545	-545	-692	-839	-755

-862	-914	-862	-862	-862	-862	-914	-862
-914	-444	-600	-600	-600	-600	-444	-914
-862	-600	-235	-391	-391	-235	-600	-862
-862	-600	-391	-52	-52	-391	-600	-862
-862	-600	-391	-52	-52	-391	-600	-862
-862	-600	-235	-391	-391	-235	-600	-862
-862	-444	-600	-600	-600	-600	-444	-862
-966	-1045	-1045	-1097	-1097	-1045	-1045	-966

Table 6: Endgame values

-294	-196	-147	-98	-98	-147	-196	-294
-196	-98	-49	0	0	-49	-98	-196
-147	-49	0	49	49	0	-49	-147
-98	0	49	98	98	49	0	-98
-98	0	49	98	98	49	0	-98
-147	-49	0	49	49	0	-49	-147
-196	-98	-49	0	0	-49	-98	-196
-294	-196	-147	-98	-98	-147	-196	-294

-653	-665	-679	-691	-691	-679	-665	-653
-665	-542	-555	-568	-568	-555	-542	-665
-679	-555	-475	-496	-496	-475	-555	-679
-691	-568	-496	-431	-431	-496	-568	-691
-691	-568	-496	-431	-431	-496	-568	-691
-679	-555	-475	-496	-496	-475	-555	-679
-665	-542	-555	-568	-568	-555	-542	-665
-653	-665	-679	-691	-691	-679	-665	-653

3.4 White Rooks

Table 7: Opening values

-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208
-208	-104	0	104	104	0	-104	-208

-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692
-692	-607	-522	-437	-437	-522	-607	-692

Table 8: Endgame values

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522
-522	-522	-522	-522	-522	-522	-522	-522

3.5 White Queens

Table 9: Opening values

-588	-392	-294	-196	-196	-294	-392	-588
-392	-196	-98	0	0	-98	-196	-392
-294	-98	0	98	98	0	-98	-294
-196	0	98	196	196	98	0	-196
-196	0	98	196	196	98	0	-196
-294	-98	0	98	98	0	-98	-294
-392	-196	-98	0	0	-98	-196	-392
-789	-593	-495	-397	-397	-495	-593	-789

-1784	-1625	-1545	-1466	-1466	-1545	-1625	-1784
-1625	-1466	-1386	-1306	-1306	-1386	-1466	-1625
-1545	-1386	-1306	-1226	-1226	-1306	-1386	-1545
-1466	-1306	-1226	-1146	-1146	-1226	-1306	-1466
-1466	-1306	-1226	-1146	-1146	-1226	-1306	-1466
-1545	-1386	-1306	-1226	-1226	-1306	-1386	-1545
-1625	-1466	-1386	-1306	-1306	-1386	-1466	-1625
-1947	-1788	-1709	-1629	-1629	-1709	-1788	-1947

Table 10: Endgame values

-648	-432	-324	-216	-216	-324	-432	-648
-432	-216	-108	0	0	-108	-216	-432
-324	-108	0	108	108	0	-108	-324
-216	0	108	216	216	108	0	-216
-216	0	108	216	216	108	0	-216
-324	-108	0	108	108	0	-108	-324
-432	-216	-108	0	0	-108	-216	-432
-648	-432	-324	-216	-216	-324	-432	-648

-931	-732	-633	-533	-533	-633	-732	-931
-732	-533	-434	-335	-335	-434	-533	-732
-633	-434	-335	-235	-235	-335	-434	-633
-533	-335	-235	-136	-136	-235	-335	-533
-533	-335	-235	-136	-136	-235	-335	-533
-633	-434	-335	-235	-235	-335	-434	-633
-732	-533	-434	-335	-335	-434	-533	-732
-931	-732	-633	-533	-533	-633	-732	-931

3.6 White Kings

Table 11: Opening values

1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407
1407	1876	938	0	0	938	1876	1407

835	1218	452	-313	-313	452	1218	835
861	1244	478	-287	-287	478	1244	861
913	1296	530	-235	-235	530	1296	913
965	1348	582	-182	-182	582	1348	965
1018	1400	635	-130	-130	635	1400	1018
1070	1453	687	-78	-78	687	1453	1070
1122	1505	739	-26	-26	739	1505	1122
1148	1531	765	0	0	765	1531	1148

Table 12: Endgame values

-2406	-1604	-1203	-802	-802	-1203	-1604	-2406
-1604	-802	-401	0	0	-401	-802	-1604
-1203	-401	0	401	401	0	-401	-1203
-802	0	401	802	802	401	0	-802
-802	0	401	802	802	401	0	-802
-1203	-401	0	401	401	0	-401	-1203
-1604	-802	-401	0	0	-401	-802	-1604
-2406	-1604	-1203	-802	-802	-1203	-1604	-2406

-2220	-1480	-1109	-740	-740	-1109	-1480	-2220
-1480	-740	-370	0	0	-370	-740	-1480
-1109	-370	0	370	370	0	-370	-1109
-740	0	370	740	740	370	0	-740
-740	0	370	740	740	370	0	-740
-1109	-370	0	370	370	0	-370	-1109
-1480	-740	-370	0	0	-370	-740	-1480
-2220	-1480	-1109	-740	-740	-1109	-1480	-2220

3.7 Brief comments

Some of these PST patterns are still "Fruit-like" in their origin. For instance, upon adding 522 to the Rook values, the prior file weighting is reconstructed. Similarly, up to questions of rounding, upon adding 335 to Queens in the ending and 1306 in the opening, the Rybka 2.3.2a array again has the same weightings as Rybka 1.0 Beta.

However, some of the others are more just based on the "idea" of Fruit PST (centralisation for instance), and extra elements such as rank-weighting for Knights in the ending are added.

4 Recapitulation in summary form

Here is a list of the similarities and differences between Rybka 1.0 Beta and Rybka 2.3.2a. The bullets indicate those that I perceive to the be same.

- 1. The king attacks criterion for pawns (and lack of weighting) remained the same.

2. Knight mobility was removed.

- 3. King attacks criterion and weighting (941) for knights remained the same.
- 4. The bishop mobility computation remained the same, with the linear bonus changing from $(116, 149)$ to $(121, 121)$.
- 5. King attacks criterion and weighting (418) for bishops remained the same.
- 6. Trapped bishops are computed differently, but with the same pattern ending up being used. The penalty dropped from 1802 to 1471.

7. Blocked bishops were removed.

- 8. The opposite bishop condition is essentially the same (except for the almost irrelevant case when one side is up by 6 or more pawns), though the method of computation is different. The score is still halved, though it now occurs after the interpolation rather than before.
- 9. The rook mobility computation remained the same, with the linear bonus changing from $(79, 84)$ to $(68, 82)$.
- 10. King attacks criterion and weighting (666) for rooks remained the same.
- 11. The half-open file condition remained the same, with the bonus changing from $(64, 256)$ to $(104, 268)$.
- 12. The open file condition remained the same, with the bonus changing from $(971, 172)$ to $(627, 180)$.

13. The (semi-)open rook-attacks-king criterion changed in two ways (ignoring the material-based condition, and ignoring the "adjacent" file bonus), with the bonus changing from 853 to 795.

14. The 7th rank condition changed slightly, allowing a king on the 7th to trigger it, and the bonus changed from $(246, 1026)$ to $(200, 1072)$.

- 15. The blocked rook computation was changed, but with the same end result. The penalty dropped from 1920 to 1561.
- 16. The queen mobility computation remained the same, with the linear bonus changing from $(37, 54)$ to $(44, 38)$.
- 17. King attacks criterion and weighting (532) for queens remained the same.

18. The 7th rank condition changed slightly, allowing a king on the 7th to trigger it, and the bonus changed from 1420 to 1486.

- 19. The computation of doubled pawns remained the same (with no open/closed difference), the penalty changing from $(0, 158)$ to $(78, 165)$.

- 20. The computation of isolated pawns remained the same (with an open/closed dichotomy). The penalty for half-open files changed from $(923, 324)$ to $(79, 324)$, and for closed files went from $(522, 670)$ to $(156, 201)$.

- 21. The computation of backward pawns remained the same (with an open/closed dichotomy). The penalty for half-open files changed from $(872, 297)$ to $(406, 297)$, and for closed files went from $(418, 536)$ to $(130, 167)$.

- 22. The king danger code remained the same, with a change in the weighting from the number of attackers.

```
Rybka 1.0 Beta: {0, -1, 37, 71, 100, 100, 100, ...}
Rybka 2.3.2a:   {0,  0  30, 57,  81,  81,  81, ...}
```

- 23. King shelter/storm code remained the same, the values changing a bit.

- 24. The candidate pawn criterion remained the same. Here are the comparative values, with Rybka 1.0 Beta on the left.

```
CandOp { 0, 0, 0, 382,1131,2263,3763,3763} {0,8,8,224,653,1306,0,0}
CandEg {18,18,18, 181, 501, 985,1626,1626} {0,8,8,224,653,1306,0,0}
```

25. The passed pawn code remained much the same, with three bonuses being applied. Rybka 2.3.2a moves the "base" score to the pawn evaluation. The "unstoppable passer" bonus has been removed from the general code. Special code for evaluating pawn endgames was added.

```
UnblockedOwn  {0,0,0, 26, 78,257, 262, 262}   {0,0,0, 27, 82,164, 274, 274}
UnblockedOpp  {0,0,0,133,394,788,1311,1311}   {0,0,0,139,412,825,1372,1372}
PassedFree    {0,0,0,101,300,601,1000,1000}   {0,0,0,106,314,629,1046,1046}
AttDistance   {0,0,0, 66,195,391, 650, 650}   {0,0,0, 69,204,409, 680, 680}
DefDistance   {0,0,0,131,389,779,1295,1295}   {0,0,0,137,407,815,1355,1355}
```

Combining the arrays from different places, the "standard" bonuses are:

```
PassedOp {  0,  0,  0,489,1450,2900,4821,4821} {0,60,60,318,822,1593,2613,0}
PassedEg {146,146,146,336, 709,1273,2020,2020} {0,60,60,318,822,1592,2612,0}
```

- 26. The material imbalance table stayed the same, modifying some values.

27. The interpolation was changed to linear.

28. The tempo bonus was removed.

29. The usage details of lazy evaluation were modified.

4.1 New features in Rybka 2.3.2a

1. Pawn anti-mobility.
2. Drawishness adjustment derived from files occupied by pawns.
3. Special pawn endgame evaluator.

4.2 Brief comments

On a raw basis, I counted 20 of 29 evaluation features that stayed the same, though some of those 20 were changed in implementation details (like blocked bishops/rooks), and to offset this, others that I counted as "changed" might be argued to have rather minor differences (like majors on the 7th). Of these 20 elements that remained essentially the same, all but the material imbalance table appeared in Fruit 2.1, though for a few (such as candidate/backward pawns and shelter/storm) there were already some differences in details in Rybka 1.0 Beta.