resend

Questions

1) After removing the dashed separator lines, what do I do next?

After removing separator lines and blank lines, keep only the **actual data rows** and organize them into **two lines per player**:

- **Line 1 (player line):** player number, name, total points, and round results (`W/L/D` plus opponent number, or `B/H/U`)
- **Line 2 (info line):** state and rating text (used to extract **Pre-Rating**)

Conceptually:

1. Pair the two lines for each player
2. Parse line 1 into `player_num / name / total_points / round1~round7`
3. Parse line 2 into `state / pre_rating`
4. Combine into one player table
5. Extract opponent numbers from round cells (only rounds that contain an opponent number)
6. Look up opponents' pre-ratings and compute the average (exclude `B/H/U` because they have no opponent)
7. Export the required 5-column CSV

2) What does the "remove dashed lines" regex mean?

The pattern `^\\s*-+\\s*$` matches a line that is **only dashes** (with optional whitespace):

- `^` start of line
- `\\s*` optional whitespace
- `-+` one or more `-`
- `\\s*` optional whitespace
- `$` end of line

Using `!str_detect(...)` keeps lines that **do not** match that separator format.

Step 1

```R
library(tidyverse)
library(stringr)

raw <- readLines("tournamentinfo.txt", warn = FALSE)

raw <- raw[!str_detect(raw, "^\\s*-+\\s*$")]  # drop dashed separator lines
raw <- raw[!str_detect(raw, "^\\s*$")]        # drop blank lines
raw <- str_squish(raw)                        # optional: normalize whitespace

idx <- which(str_detect(raw, "^\\s*\\d+\\s*\\|"))  # player line starts with numb
p1 <- raw[idx]        # line 1 per player
p2 <- raw[idx + 1]    # line 2 per player
```

R

Explanation

- `readLines()` reads the file line-by-line into a character vector.
- The two filters remove separator lines and blank lines so indexing becomes stable.
- `idx` locates each player's "line 1"; `p2` uses `idx + 1` to pair the following "line 2".

Step 2

```R
parse_line1 <- function(x) {
  parts <- str_split(x, "\\|", simplify = TRUE) |> as.character()
  parts <- str_trim(parts)
  parts <- parts[parts != ""]

  tibble(
    player_num   = as.integer(parts[1]),
    name         = parts[2],
    total_points = as.numeric(parts[3]),
    round1       = parts[4],
    round2       = parts[5],
    round3       = parts[6],
    round4       = parts[7],
    round5       = parts[8],
```

```
    round6       = parts[9],
    round7       = parts[10]
  )
}

line1_df <- map_dfr(p1, parse_line1)
```

R

- Splits each player line by `|`, trims whitespace, drops empty fields.
- Extracts fixed-position fields into columns.
- `map_dfr()` applies the parser to all `p1` lines and row-binds results.

Step 3

```
parse_line2 <- function(x) {
  parts <- str_split(x, "\\|", simplify = TRUE) |> as.character()
  parts <- str_trim(parts)
  parts <- parts[parts != ""]
  state <- parts[1]

  pre <- str_match(x, "R:\\s*(\\d{3,4})")[, 2]
  if (is.na(pre)) pre <- str_match(x, "(\\d{3,4})\\s*->")[, 2]

  tibble(
    state = state,
    pre_rating = as.integer(pre)
  )
}

line2_df <- map_dfr(p2, parse_line2)
```

R

Explanation

- `state <- parts[1]` assumes the first `|` field in line 2 is the state code.
- Extracts pre-rating using either `R: ####` or `#### ->` as fallback.
- Converts to integer; unmatched cases become `NA`.

Step 4

R

Explanation

- `bind_cols()` combines the parsed line-1 and line-2 tables column-wise (same order/row count).

Step 5

```
opp_long <- players %>%
  select(player_num, starts_with("round")) %>%
  pivot_longer(
    cols = starts_with("round"),
    names_to = "round",
    values_to = "cell"
  ) %>%
  mutate(
    cell = str_trim(cell),
    opp_num = as.integer(str_extract(cell, "\\d+"))
  )
```

R

Explanation

- `pivot_longer()` converts round1–round7 from wide to long: one row per player per round.
- `str_extract(cell, "\\d+")` pulls the opponent number from `W 34 / L 5 / D 12`.
- `B/H/U` have no digits → `opp_num = NA` (excluded from opponent-average later).

Step 6

```
opp_lookup <- players %>%
  select(player_num, opp_pre = pre_rating)

avg_opp <- opp_long %>%
  left_join(opp_lookup, by = c("opp_num" = "player_num")) %>%
```

```
  group_by(player_num) %>%
  summarise(
    avg_opp_pre = mean(opp_pre, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  mutate(
    avg_opp_pre = ifelse(is.nan(avg_opp_pre), NA, avg_opp_pre)
  )
```

R

Explanation:

- `opp_lookup` maps opponent player number → opponent pre-rating (`opp_pre`).
- `left_join()` attaches opponent pre-ratings to each round using `opp_num`.
- `mean(..., na.rm=TRUE)` excludes `NA` (so `B/H/U` don't affect the average).
- Converts `NaN` (possible if all rounds are `NA`) to `NA`.

Step 7

R

```
final <- players %>%
  left_join(avg_opp, by = "player_num") %>%
  transmute(
    Name = name,
    State = state,
    TotalPoints = total_points,
    PreRating = pre_rating,
    AvgOppPreRating = as.integer(round(avg_opp_pre, 0))
  )

write.csv(final, "chess_players.csv", row.names = FALSE)
```