Data Exploration: Cooperation

Kayla Huang

September 16, 2021

To begin this assignment, **make sure you have downloaded all the materials in this week's folder on Canvas**. Before you begin, make sure you have the files `data_assignment_sept16.Rmd` and `data_assignment_sept16.pdf` as well as the folders `axelRod-master` and `rmd_photos` in the same folder on your computer. You will be using a Shiny app, developed by Simon Garnier and edited slightly by the TF team, that will only work if those things are all in the same place.

Next, **you must set your working directory to the source file location**. To do so, select 'Session' from the menu bar at the top of your screen, hover over 'Set Working Directory', then select 'To Source File Location'.

When knitting your RMarkdown file to a PDF, make sure to say "No" when it asks you if you would like to "Render and view this document using Shiny instead."

If you have trouble getting the Shiny app to work (or with anything else), please come to the teaching team. We are happy to help!

The Evolution of Cooperation

Axelrod's *Evolution of Cooperation* uses the construct of the Prisoner's Dilemma to illustrate how cooperation can emerge despite incentives not to. In the Prisoner's Dilemma game, players must choose whether to cooperate or defect. The payoffs for doing one or the other depend on what the other player does, but no matter if the other player cooperates or defects, it is always strictly better for players to defect.

This can be seen in the table below, which replicates the game Axelrod uses throughout his book. If player 2 cooperates, it's better for player 1 to defect, since then player 1 would receive 5 points instead of 3. If player 2 defects, player 1 definitely wants to defect and receive 1 point rather than 0. So no matter what each player expects the other to do, they will both choose to defect, yielding 1 point to each player.

P1 ↓; P2 →	C	D
C	R = 3, R = 3	S = 0, T = 5
D	T = 5, S = 0	P = 1, P = 1

But ideally, in the long run (in a repeated game) players would like to cooperate and receive 3 points on each round. Axelrod explains how strategies of cooperation can evolve even in circumstances where players are antagonists (like Allied and Axis soldiers in the trenches of World War I) or when the players are not capable of foresight (as is the case for cooperation in biological systems).

The Prisoner's Dilemma Simulator

For this week's Data Exploration assignment, you will be working with a Shiny app that simulates prisoner's dilemma games. To use it, simply run the code chunk above labeled 'setup', then run the following code (`shiny_tournament()`). Doing so will open the app in a separate window.

P.S., the "Instructions" tab in the app is broken. Don't worry if it doesn't display anything for you. Refer to this document for instructions.

P.P.S., when you close the app window, there may be some warnings or errors (like "no loop for break/next, jumping to top level"). You can just ignore them.

Setup

Now we're going to do a round-robin style tournament between strategies of your choosing, similar to the ones Axelrod conducted. All students must complete this part, as the subsequent questions are based on the tournament you conduct here.

First, choose at least 6 strategies from the menu that look interesting to you. Your task is to play each one against all the other opponents and record the results in the excel file available in the Google Drive called `prisoners_dilemma_data.xlsx`. 1. titfortat 2. punisher 3. alternator 4. cycler 5. defector 6. grumpy

Second, once you have chosen your strategies, type all the pairwise combinations of those strategies into the columns `player1` and `player2`. Make sure you type the strategies exactly as they appear in the application, including the case of the letters! Your spreadsheet should look this like after you have done so (but with the strategies you choose):

	A	B	C	D
1	player1	player2	score1	score2
2	titfortat	inverser		
3	titfortat	appeaser		
4	titfortat	foolMeOnce		
5	titfortat	handshaker		
6	titfortat	grumpy		
7	inverser	appeaser		
8	inverser	foolMeOnce		
9	inverser	handshaker		
10	inverser	grumpy		
11	appeaser	foolMeOnce		
12	appeaser	handshaker		
13	appeaser	grumpy		
14	foolMeOnce	handshaker		
15	foolMeOnce	grumpy		
16	handshaker	grumpy		
17				

Figure 1: This is what your table should look like after step 2.

Note that there are 15 ways to combine 6 elements into pairs[1], so if you don't have 15 pairs, check your work. Also note that the more strategies you choose, the more typing you will have to do.

Third, set the app so that "Tournament Type" = "Repeated", "Number of Rounds" = 100, and "Number of Replications" = 100. Just as in Axelrod's simulation, we are playing repeated games (this is determined by the "Number of Rounds"). The "Number of Replications" changes how many times the computer plays

[1] In math terms, this results from the fact that $\binom{6}{2} = 15$

each repeated game. So in the example above, the computer will repeat the 100-round game of titfortat vs. inverser 100 times over and take the average outcome of each of those replications. This is useful because some of the strategies rely on probability (e.g. play "Defect" with probability .5) and so the outcome will be different each time. We average over many outcomes to see which strategy wins on average. Once you're done, your spreadsheet should look something like this, but with different strategies:

	A	B	C	D
1	player1	player2	score1	score2
2	titfortat	inverser	300	300
3	titfortat	appeaser	300	300
4	titfortat	foolMeOnce	300	300
5	titfortat	handshaker	101	106
6	titfortat	grumpy	300	300
7	inverser	appeaser	300	300
8	inverser	foolMeOnce	300	300
9	inverser	handshaker	3	498
10	inverser	grumpy	300	300
11	appeaser	foolMeOnce	300	300
12	appeaser	handshaker	52	302
13	appeaser	grumpy	300	300
14	foolMeOnce	handshaker	100	110
15	foolMeOnce	grumpy	300	300
16	handshaker	grumpy	122	97
17				

Figure 2: This is what your table should look like after step 3.

Of course, don't just copy these numbers, since they're made up.

Fourth, save the file as a CSV. To do so, go to File > Save As, then set the File Format to be `CSV UTF-8 (Comma delimited) (.csv)`. Make sure to save it with the name `prisoners_dilemma_data.csv`.

Now you can finally read the data you created into R using the following code and start answering the questions that follow.

```
pd_data <- read_csv("prisoners_dilemma_data.csv") %>%
  mutate(winner = case_when( # if you are interested, case_when() is a very useful
    score1 > score2 ~ player1, # function to create new variables. check out how it
    score1 < score2 ~ player2, # works by googling.
    score1 == score2 ~ "tie"
  ))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   player1 = col_character(),
##   player2 = col_character(),
##   score1 = col_double(),
##   score2 = col_double()
```

```
## )
```

```
pd_data
```

```
## # A tibble: 15 x 5
##    player1    player2    score1 score2 winner
##    <chr>      <chr>       <dbl>  <dbl> <chr>
##  1 titfortat  punisher      300    300 tie
##  2 titfortat  alternator    248    253 alternator
##  3 titfortat  cycler        273    278 cycler
##  4 titfortat  defector       99    104 defector
##  5 titfortat  grumpy        300    300 tie
##  6 punisher   alternator    296     61 punisher
##  7 punisher   cycler        384     59 punisher
##  8 punisher   defector       99    104 defector
##  9 punisher   grumpy        300    300 tie
## 10 alternator cycler        300    175 alternator
## 11 alternator defector       50    300 defector
## 12 alternator grumpy        400    150 alternator
## 13 cycler     defector       25    400 defector
## 14 cycler     grumpy        350    225 cycler
## 15 defector   grumpy        120     95 defector
```

Question 1

How do the strategies you chose rank against each other based on the number of wins? How do they rank based on the number of points won? Discuss the patterns you see here as they relate to what you read from Axelrod. Keep in mind the concepts of niceness, retaliation, forgiveness, and clarity.

```
table(pd_data$winner)
```

```
##
## alternator     cycler   defector   punisher        tie
##          3          2          5          2          3
```

In terms of wins, the strategies rank against each other in the following manner, from best to worst: 1) defector, 2) alternator, 3) cycler and punisher.

```
sum(pd_data$score1[pd_data$player1 == "titfortat"])
```

```
## [1] 1220
```

```
sum(pd_data$score1[pd_data$player1 == "punisher"], pd_data$score2[pd_data$player2 == "punisher"])
```

```
## [1] 1379
```

```
sum(pd_data$score1[pd_data$player1 == "alternator"], pd_data$score2[pd_data$player2 == "alternator"])
```

```
## [1] 1064
```

```
sum(pd_data$score1[pd_data$player1 == "cycler"], pd_data$score2[pd_data$player2 == "cycler"])
```

```
## [1] 887
```

```
sum(pd_data$score1[pd_data$player1 == "grumpy"], pd_data$score2[pd_data$player2 == "grumpy"])
```

```
## [1] 1070
```

```
sum(pd_data$score1[pd_data$player1 == "defector"], pd_data$score2[pd_data$player2 == "defector"])
```

```
## [1] 1028
```

Based on the calculations above, if we rank the strategies based on total number of points won, we see that they can be ranked from best to worst in the following manner: 1) punisher, 2) tit for tat, 3) grumpy, 4) alternator, 5) defector, and 6) cycler

Recall the following definitions: Niceness: never being the first to defect Retaliation: defecting after the other person defects Forgiveness: propensity to cooperate after retaliation Clarity: clear pattern (so that people can predict what you do)

The punisher earned the most points. This strategy is very nice (never defects first), but is very retaliatory. However, the punisher does have the capability for forgiveness (at a time in the future proportional to how many times its opponent defects). [TO DO]

The following code makes a data frame called `player_data_long` that you can use to rank the strategies based on the number of points won during the tournament. As a hint, you may want to try using `group_by()`, `summarize()`, and `arrange()` on `player_data_long`.

If you want, try to figure out what each line of code does. Cleaning the data and rearranging it like this is an important part of data science, not just running regressions and taking means. This is why we are leaving some of it to you, via `group_by()` and `summarize()`.

```
player1_data <- pd_data %>% select(player = player1, score = score1, opponent = player2)
player2_data <- pd_data %>% select(player = player2, score = score2, opponent = player1)
player_data_long <- bind_rows(player1_data, player2_data)
player_data_long
```

```
## # A tibble: 30 x 3
##    player     score opponent
##    <chr>      <dbl> <chr>
##  1 titfortat    300 punisher
##  2 titfortat    248 alternator
##  3 titfortat    273 cycler
##  4 titfortat     99 defector
##  5 titfortat    300 grumpy
##  6 punisher     296 alternator
##  7 punisher     384 cycler
##  8 punisher      99 defector
##  9 punisher     300 grumpy
## 10 alternator   300 cycler
## # ... with 20 more rows
```

```
player_data_long %>% group_by(player) %>% summarize(total_score = sum(score))  %>% arrange(total_score)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 2
##   player     total_score
##   <chr>            <dbl>
## 1 cycler             887
## 2 defector          1028
## 3 alternator        1064
## 4 grumpy            1070
## 5 titfortat         1220
## 6 punisher          1379
```

Note that here, the strategies are shown from worst to best.

To rank the strategies based on the number of wins, think about making use of the `winner` variable in the `pda_data` data frame.

```
pd_data %>% group_by(winner) %>% summarize(frequency=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 5 x 2
##   winner     frequency
##   <chr>          <int>
## 1 alternator         3
## 2 cycler             2
## 3 defector           5
## 4 punisher           2
## 5 tie                3
```

Question 2

Make a plot like the one below that displays how your winning strategy (according to total points) fared against the strategies it played against. Using `player_data_long` is probably easiest, but complete this question however makes most sense to you. Comment on what you find.

```
punisher_data <- filter(player_data_long, player_data_long$player == "punisher")
barplot(punisher_data$score, names.arg=punisher_data$opponent, xlab = "Opponent", ylab = "Score", main =
```

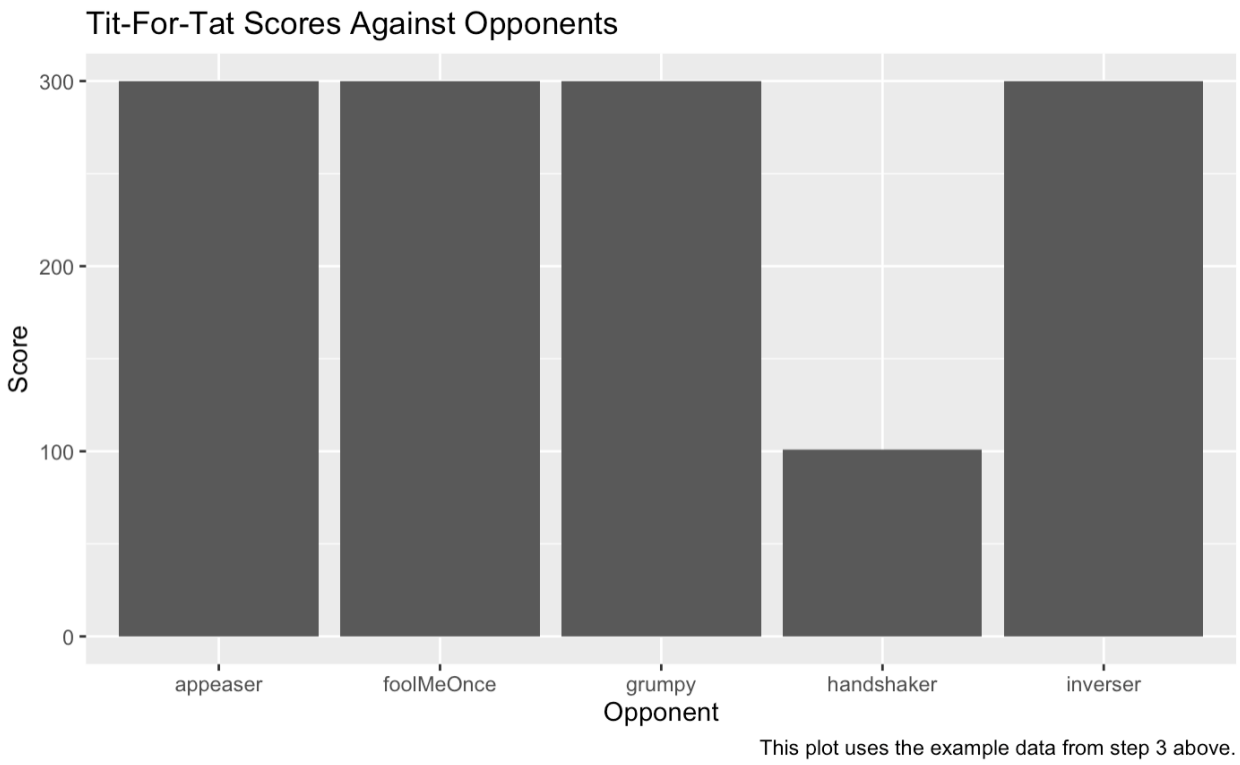
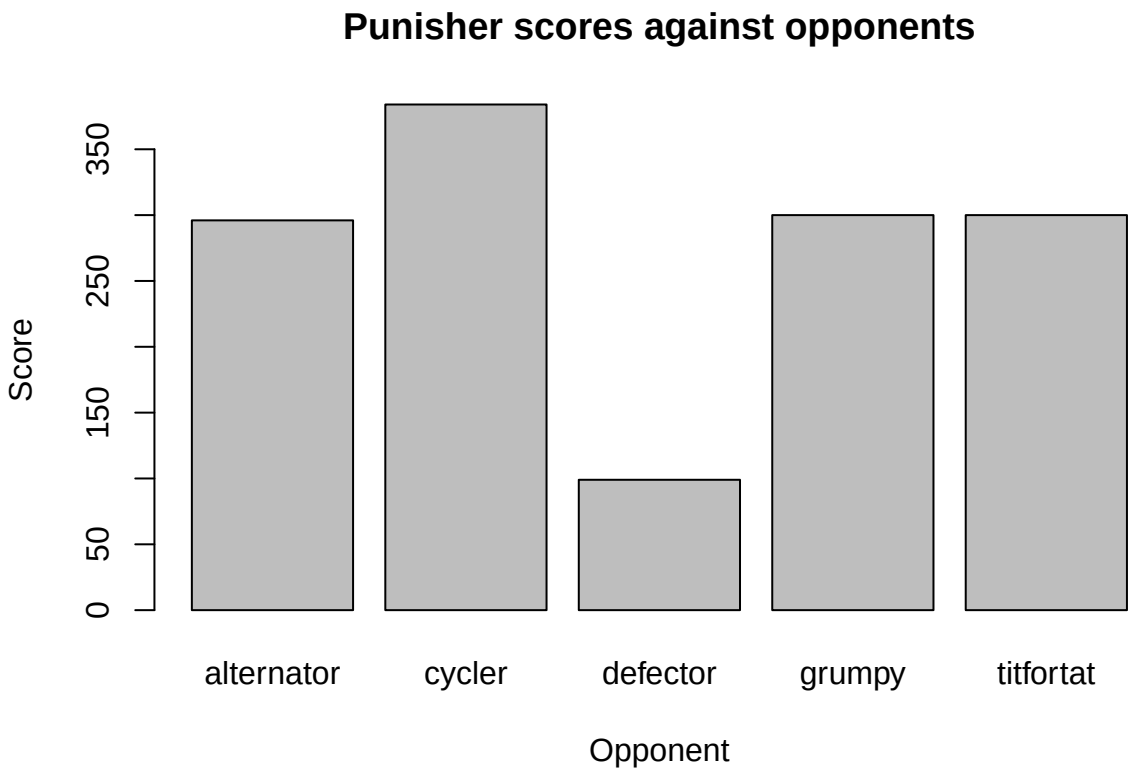


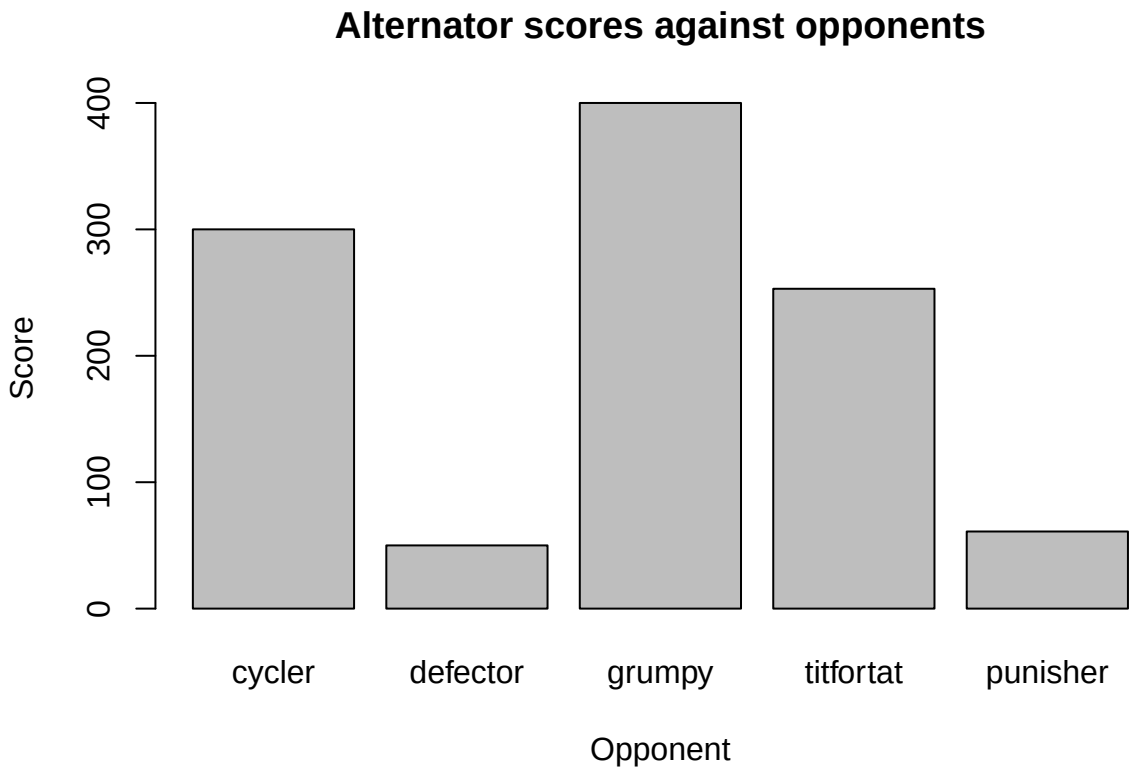
Figure 3: Here is an example of a plot you might make.



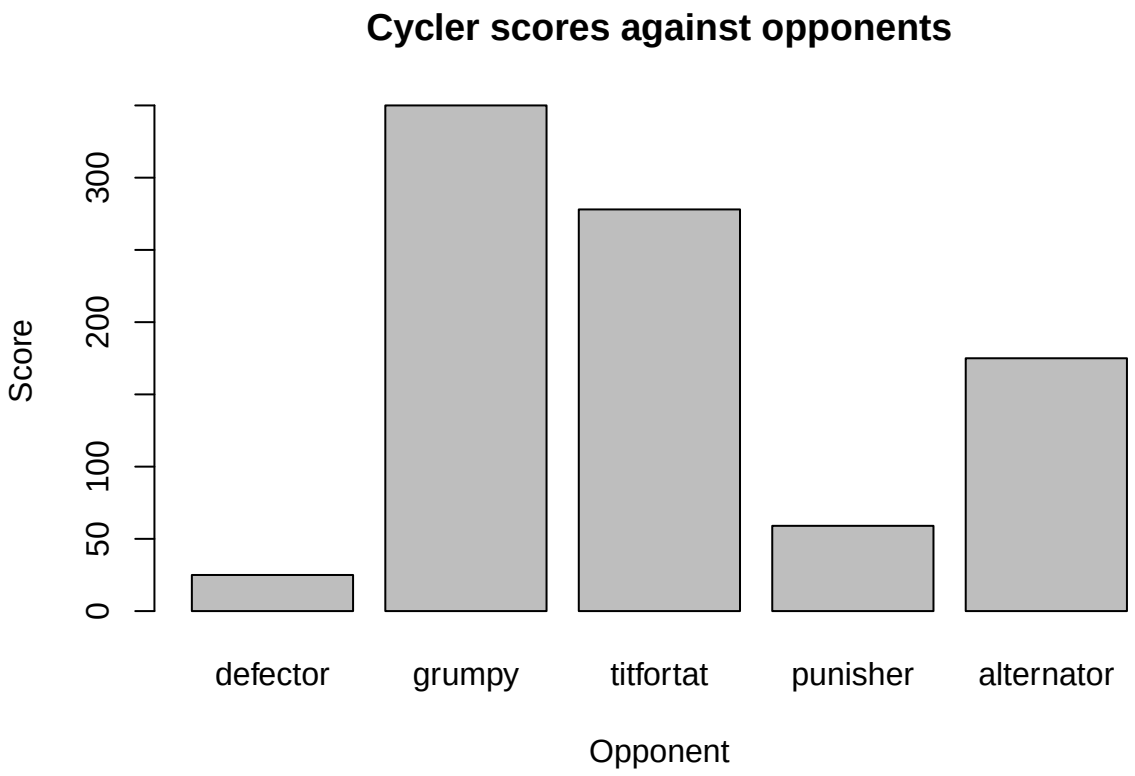
Question 3: Data Science Question

Create a plot, similar to the one above, that displays how each strategy fared against all the other strategies. Which strategies were most successful against which other ones? Comment on the patterns that you see.

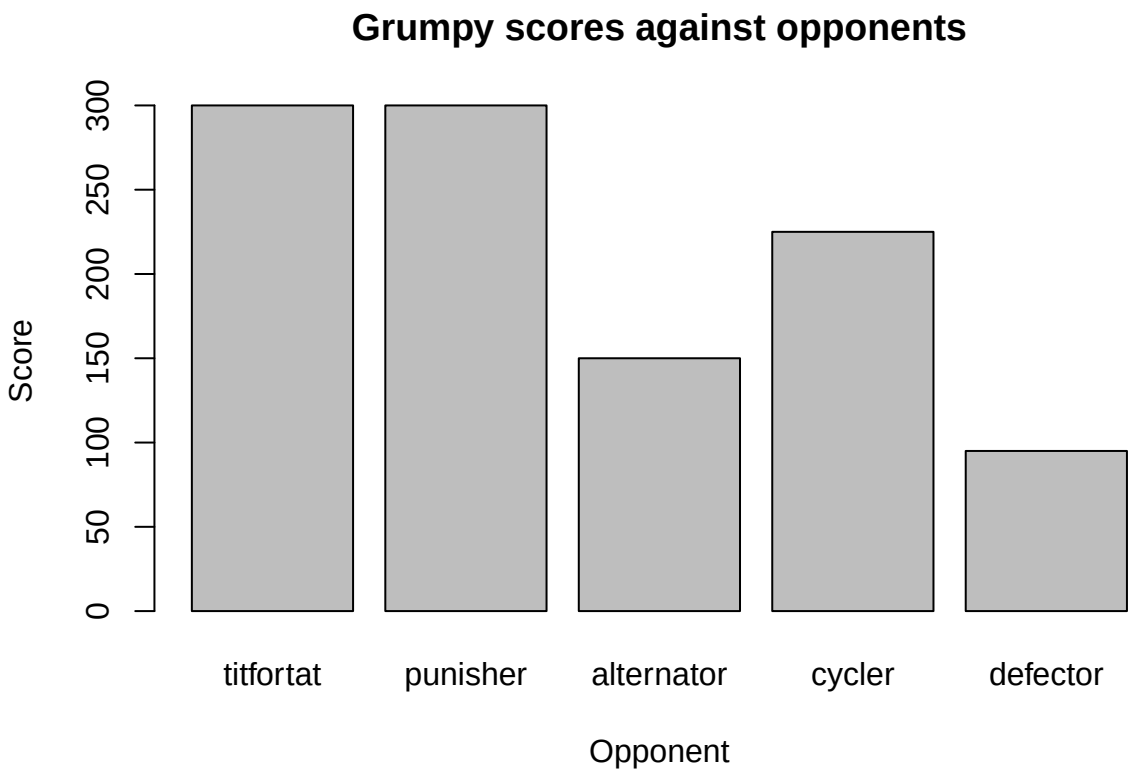
```
alternator_data <- filter(player_data_long, player_data_long$player == "alternator")
barplot(alternator_data$score, names.arg=alternator_data$opponent, xlab = "Opponent", ylab = "Score", ma
```



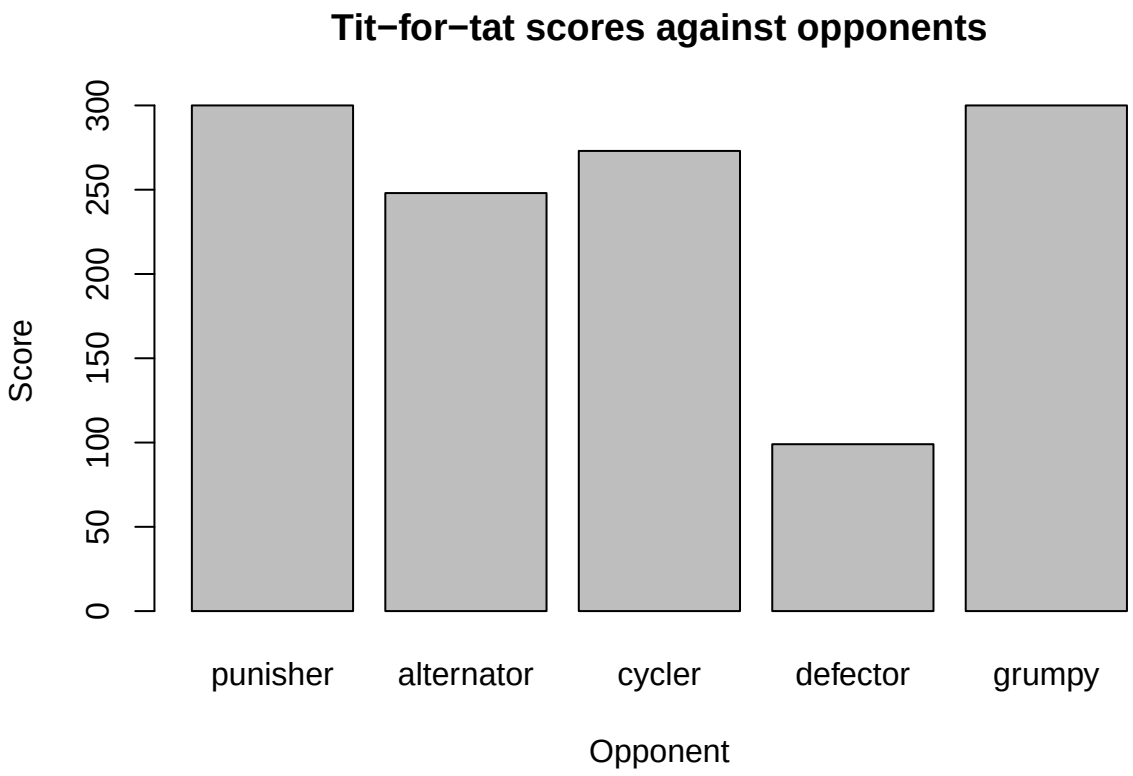

```
cycler_data <- filter(player_data_long, player_data_long$player == "cycler")
barplot(cycler_data$score, names.arg=cycler_data$opponent, xlab = "Opponent", ylab = "Score", main = "Cy
```



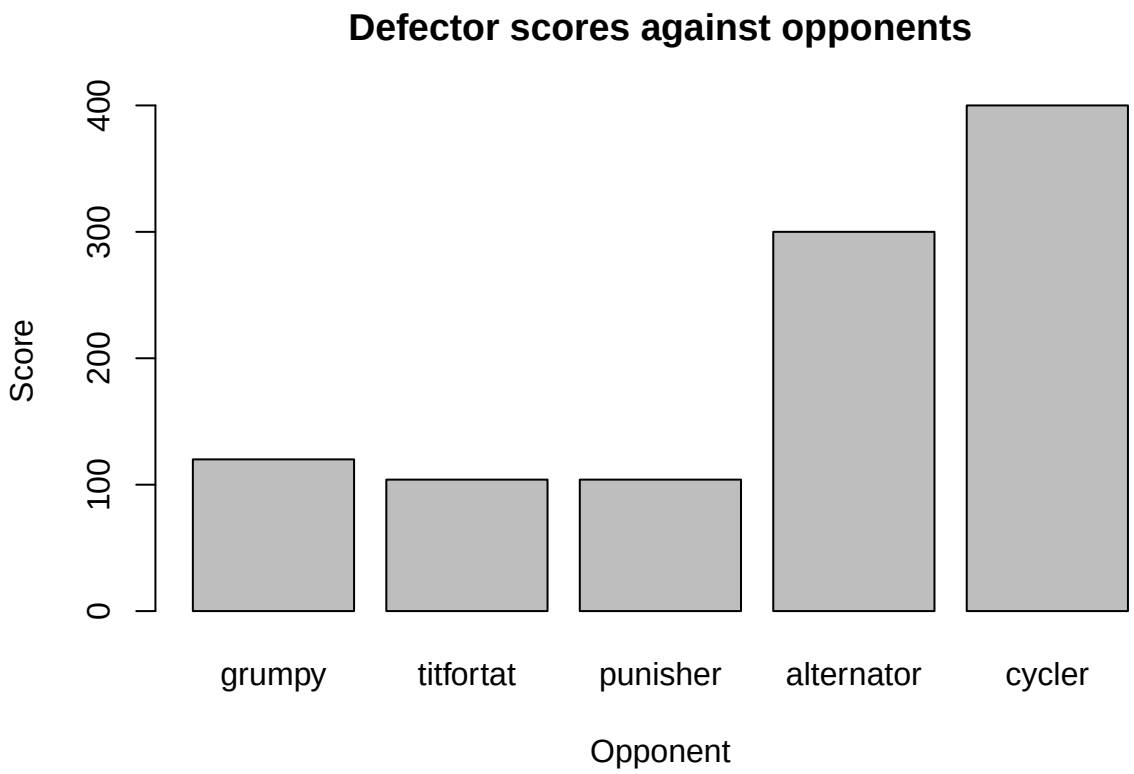

```
grumpy_data <- filter(player_data_long, player_data_long$player == "grumpy")
barplot(grumpy_data$score, names.arg=grumpy_data$opponent, xlab = "Opponent", ylab = "Score", main = "G
```



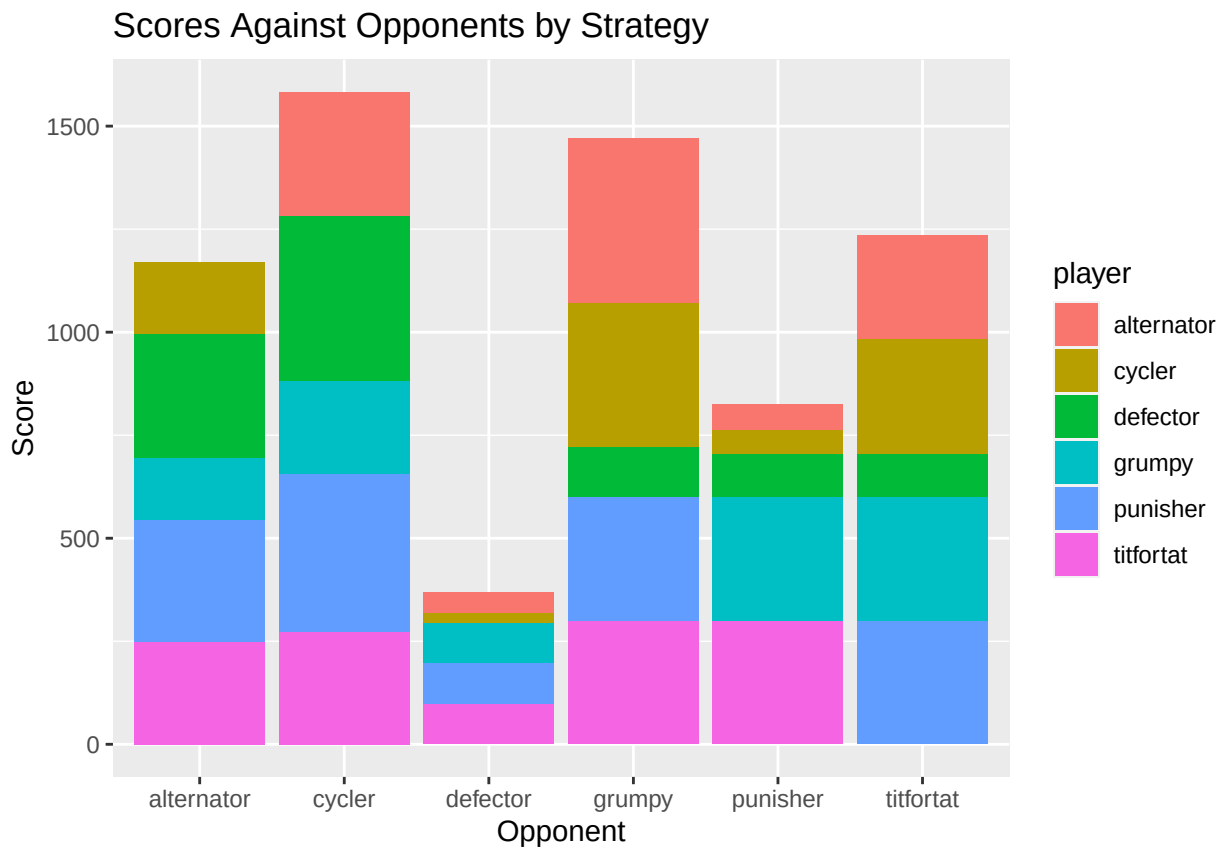

```
titfortat_data <- filter(player_data_long, player_data_long$player == "titfortat")
barplot(titfortat_data$score, names.arg=titfortat_data$opponent, xlab = "Opponent", ylab = "Score", mai
```




```
defector_data <- filter(player_data_long, player_data_long$player == "defector")
barplot(defector_data$score, names.arg=defector_data$opponent, xlab = "Opponent", ylab = "Score", main =
```




```
player_data_long %>%
  ggplot(aes(opponent, score, fill=player)) +
  geom_col() +
  labs(title = "Scores Against Opponents by Strategy",x = "Opponent",y = "Score")
```



Question 4

What is the main difference between the game you played here and the tournament Axelrod implemented? How do you think this difference might matter? The main difference between our simulated game and the one played here was that Axelrod's tournament contained many, many more players.