Are you in the top 1% of brawl stars players?

Dominik Moser, Ole Petersen

Have you ever wondered how good you are at the game Brawl Stars? Are you in the top 1% of players? In this post, we will find out by simulating the game with a million players using Julia.

Our model

Brawl stars is a multiplayer game. We will analyze the `3v3` game mode where two teams of three players each fight against each other. We are primarily interested in top players; therefore, we will assume that all brawlers are maxed out. The strength of a player is assumed to be soley determined by a single number, the skill level, which is constant over all games. Players have a trophy cound, changing over the rounds.

We simplistically assume that the game is played in discrete global rounds. For each round, each player joins with a probability of `activity_level`, which is individual for each player. If a player has an activity level of 1, they play in every round.

This leaves us with the following model:

```
using Distributions
using Random

mutable struct Player
    trophies::Int
    skill::Float64
    activity::Float64
    Player(skill_dist=Normal(0, 3),activity_dist=Uniform(0.2, 1)) = new(0,
rand(skill_dist), rand(activity_dist))
end
```

Here, we assume that the skill level of the players is distributed as $\mathcal{N}(\mu = 0, \sigma = 3)$ and the activity level is distributed as $\mathcal{U}(0.2, 1)$.

Consequently, the players that are active in a round are drawn as follows:

```
function sample_players_in_round(players::Vector{Player})
    mask = [rand(Bernoulli(p.activity)) for p in players]
    return players[mask]
end
```

Now, let us consider the outcome of a game. We assume that the strength of a team is the average skill of its players. If two teams with average skills m_1 and m_2 play against each other, we model the probability of team 1 winning as

$$P(\text{team 1 wins}) = \frac{1}{1 + \exp(m_2 - m_1)}.$$

This is implemented as follows:

```
function play(g1::AbstractVector{Player}, g2::AbstractVector{Player})
    m1 = mean([p.skill for p in g1])
    m2 = mean([p.skill for p in g2])
    p_team_1_wins = 1 / (1 + exp(m2 - m1))
    return rand(Bernoulli(p_team_1_wins))
end
```

How are the players rewarded or penalized after a game? This data is publicly accesible:

min trophies	max trophies	win trophy bonus	loss trophy penalty
0	49.0	8	0
50	99.0	8	−1
100	199.0	8	−2
200	299.0	8	−3
300	399.0	8	−4
400	499.0	8	−5
500	599.0	8	−6
600	699.0	8	−7
700	799.0	8	−8
800	899.0	7	−9
900	999.0	6	−10
1000	1099.0	5	−11
1100	1199.0	4	−12
1200	Inf	3	−12

So now, let us write a function that returns the new trophy count of the players after a game. Since this is run billions of times, we pre-compute the result in bins of 50 trophies:

```
num_bins=1200÷50+1
to_bin(trophies::Int)=1+min(trophies,1200) ÷ 50
win_trophies_by_bin=zeros(Int,num_bins)
```

```
loss_trophies_by_bin=zeros(Int,num_bins)
for trophy in 0:50:1200
    df_row=first(filter(row->row[1]<=trophy<=row[2],eachrow(trophy_changes)))
    win_trophies_by_bin[to_bin(trophy)]=df_row["win trophy bonus"]
    loss_trophies_by_bin[to_bin(trophy)]=df_row["loss trophy penalty"]
end
function get_trophy_change(trophies::Int, win::Bool)
    trophy_bin = to_bin(trophies)
                return    win    ?    win_trophies_by_bin[trophy_bin]    :
loss_trophies_by_bin[trophy_bin]
end
@assert  all(get_trophy_change.(trophy_changes[:,"min  trophies"],  true)  .==
trophy_changes[:,"win trophy bonus"])
@assert  all(get_trophy_change.(trophy_changes[:,"min  trophies"],  false)  .==
trophy_changes[:,"loss trophy penalty"])
println("Trophy change at 543 trophies after a win: ", get_trophy_change(543,
true))
println("Trophy change at 543 trophies after a loss: ", get_trophy_change(543,
false))
```

```
Trophy change at 543 trophies after a win: 8
Trophy change at 543 trophies after a loss: -6
```

Next, let us implement a round of the game. We first get the active players in this round and sort them by their trophy cound. This allows us to pair the players with similar trophy levels: We split the list of active players in chunks of size `6=2*team_size`. Since the list is sorted, each group of 6 has a similar trophy count. We then permute the players in the group randomly and assign the first three to group 1 and the last three to group 2. The trophies of the players are then updated accordingly. Each group is executed in parallel using `Threads.@threads`:

```
function step!(players::Vector{Player}, team_size::Int=3)
    players_in_round = sample_players_in_round(players)
    sorted_players = sort(players_in_round, by=p -> p.trophies)
    permutation = randperm(2 * team_size)
                Threads.@threads    for    i    in    1:(2*team_size):
(length(sorted_players)-2*team_size)
        @views begin
            shuffled_players = sorted_players[i:i+2*team_size-1][permutation]
            team1 = shuffled_players[1:team_size]
            team2 = shuffled_players[team_size+1:end]
            @assert length(team1) == length(team2) == team_size
            team1_wins = play(team1, team2)
            for p in team1
                p.trophies += get_trophy_change(p.trophies, team1_wins)
            end
            for p in team2
```

```
                p.trophies += get_trophy_change(p.trophies, !team1_wins)
            end
        end
    end
end
```

Now, let us simulate a few rounds of the game:

```
num_players = 1000000
num_rounds = 1000
players = [Player() for i in 1:num_players]
for round in 1:num_rounds
    step!(players)
end
```

First, we would like to find out the correlation between the skill level and the trophy count. Figure 1 shows the result of a simulation with 1 million players. It is visible that a higher skill level is clearly correlated with a higher trophy count.

```
using Plots
players_to_plot = players[1:10000]
scatter([p.skill for p in players_to_plot], [p.trophies for p in
players_to_plot], label="Skill vs Trophies", xlabel="Skill", ylabel="Trophies",
title="Skill vs Trophies",alpha=0.1, legend=false)
```

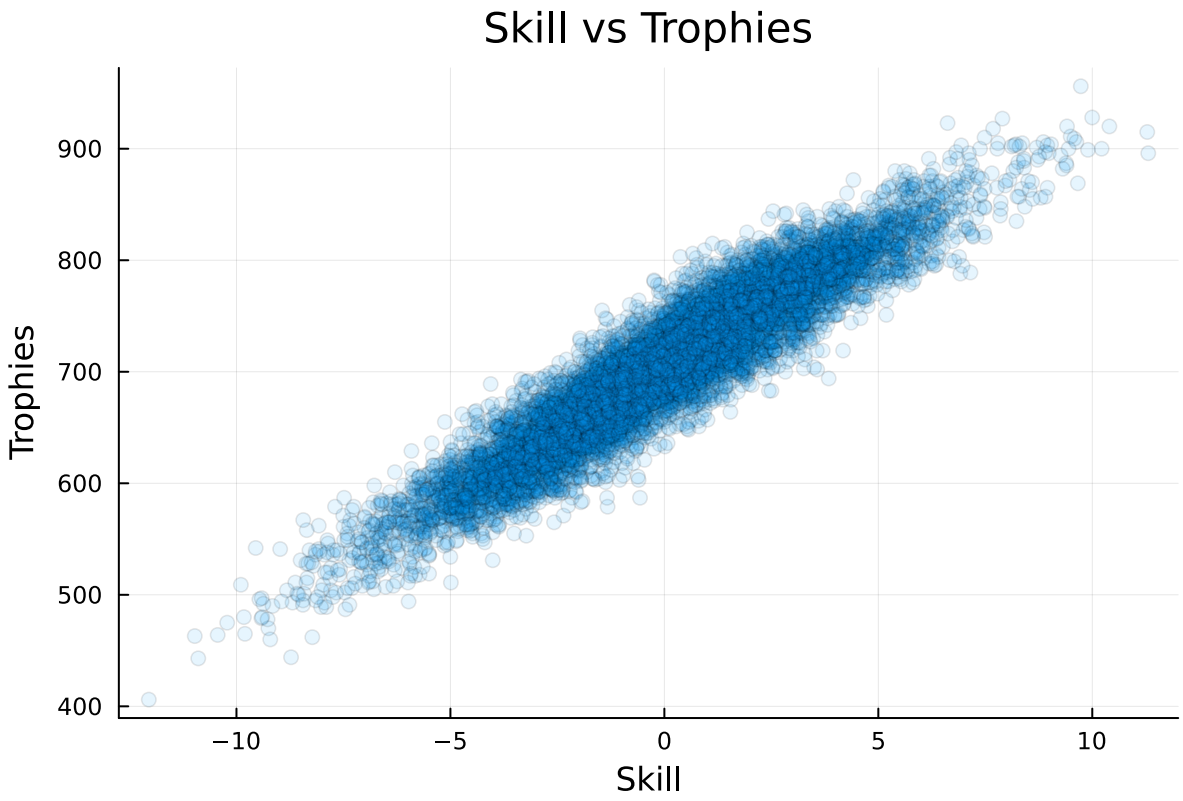

Figure 1: Skill level vs trophies for the first 10000 players

Next, let us find out what we are interested in: the top 1% of players. To do so, we sort the players by their trophy count and plot how many players have more than a certain number of trophies in Figure 2.

```
trophies=[p.trophies for p in players]
trophies=sort(trophies)
percent_better=1.0 .- (1:num_players) ./num_players
plot(trophies[1:end-1],        percent_better[1:end-1],        label="Trophies",
xlabel="Trophies", ylabel="Percentile", title="CDF of trophies", yscale=:log10,
xlims=(600,maximum(trophies)),yminorgrid=true, legend=false)
```

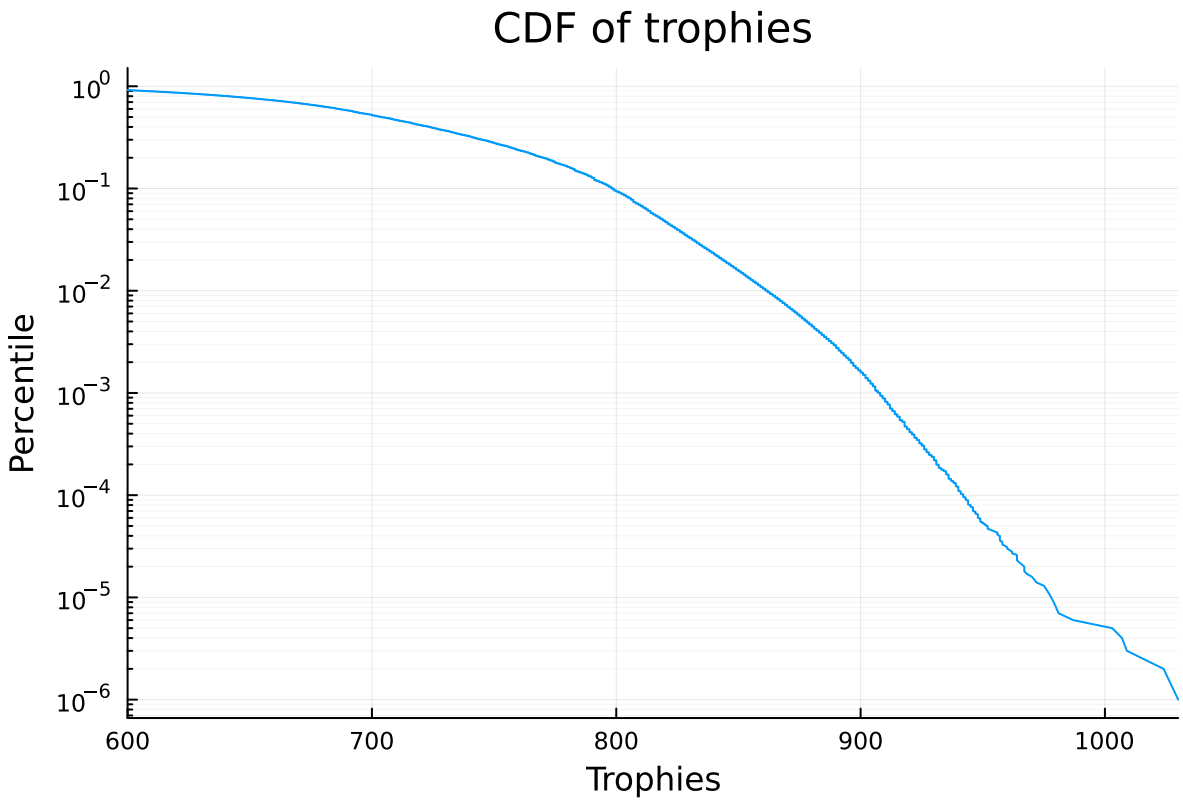

Figure 2: Fraction of players with at least a certain number of trophies

In the following table you can see that you need 864 trophies to be in the top 1% of players:

```
percentiles_to_list=[10,5,2,1,0.1,0.01,0.001]
indices=[ceil(Int,(1-percentile/100)*num_players) for percentile in
percentiles_to_list]
trophies_at_percentiles=trophies[indices]
df=DataFrame("Percentile
(%)"=>percentiles_to_list,"Trophies"=>trophies_at_percentiles)
markdown_table(df)
```

Percentile (%)	Trophies
10.0	799
5.0	819
2.0	844
1.0	864
0.1	913
0.01	976
0.001	981

Next, let us test how this curve depends on the number of players. Figure 3 shows an interesting result: If there are more players in the game, you not only need more trophies to be on a certain *absolute* position, but also to be in a certain *relative* position. Based on this, brawl stars has around a million daily active players, so the initial plot is the most relevant.

```
p=plot(xlabel="Trophies",  ylabel="Percentile",  title="CDF  of  trophies",
yscale=:log10, yminorgrid=true)
num_rounds=1000
for num_players in [100,1000, 10000, 100000, 1000000]
    players = [Player() for i in 1:num_players]
    for round in 1:num_rounds
        step!(players)
    end
    trophies=[p.trophies for p in players]
    trophies=sort(trophies)
    percent_better=1.0 .- (1:num_players) ./num_players
    plot!(p,trophies[1:end-1], percent_better[1:end-1], label="$num_players
players", xlims=(600,maximum(trophies)))
end
display(p)
```

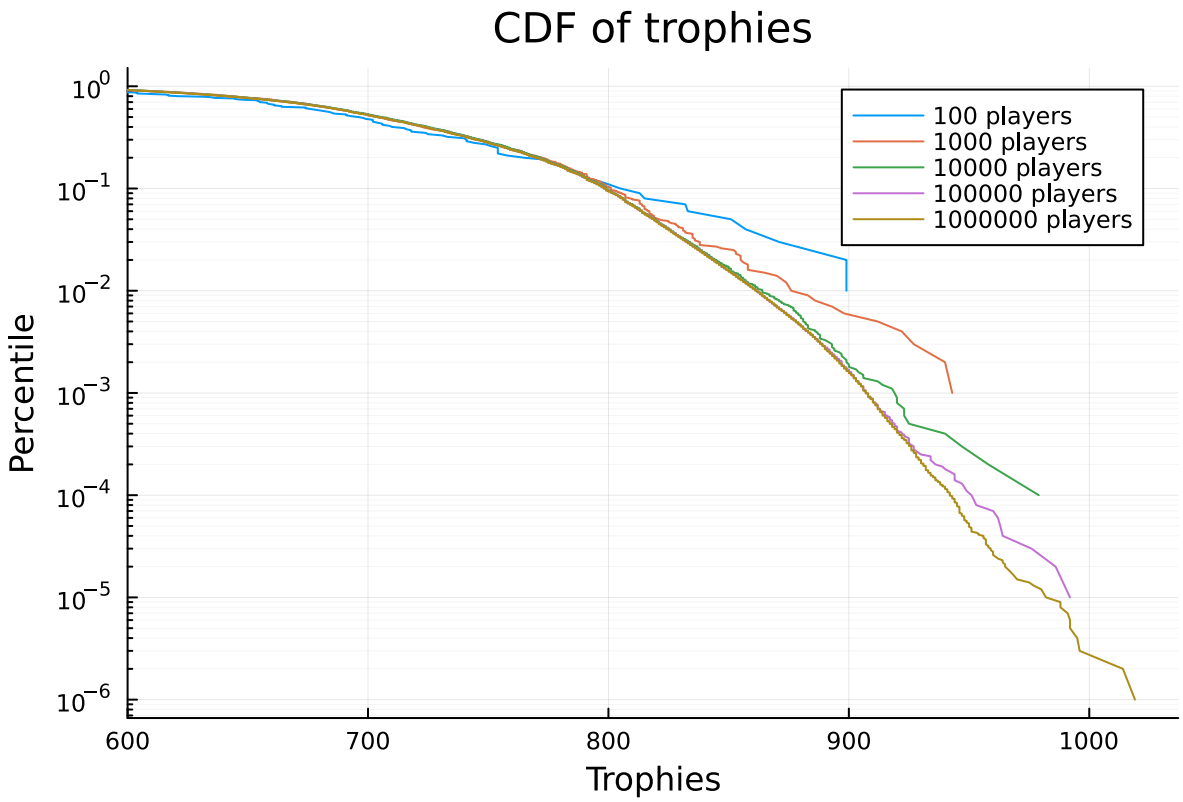

Figure 3: Fraction of players with at least a certain number of trophies for different numbers of players and a skill standard deviation of 3 after 1000 rounds

Next, let us find out how the curve depends on the standard deviations of the skill level. This model parameter is crucial since the formula for the probability of winning a game depends on the average skill level of the team members:

$$P(\text{team 1 wins}) = \frac{1}{1 + \exp(m_2 - m_1)}.$$

The standard deviation controls how likely the best players win against the worst players. Let us see how likely the top 5% of players win against the bottom 5% of players for different standard deviations, as depicted in Figure 4.

```
# The top 5% of players have a skill of 2 standard deviations above the mean,
while the bottom 5% have a skill of 2 standard deviations below the mean
p_win(std) = 1 / (1 + exp(-2*std+(-2*std)))
stds=0:0.1:3
plot(stds, p_win.(stds), label="Win probability of top 5% vs bottom 5%",
xlabel="Skill standard deviation", ylabel="Win probability", title="Win
probability of top 5% vs bottom 5% of players", legend=false)
```

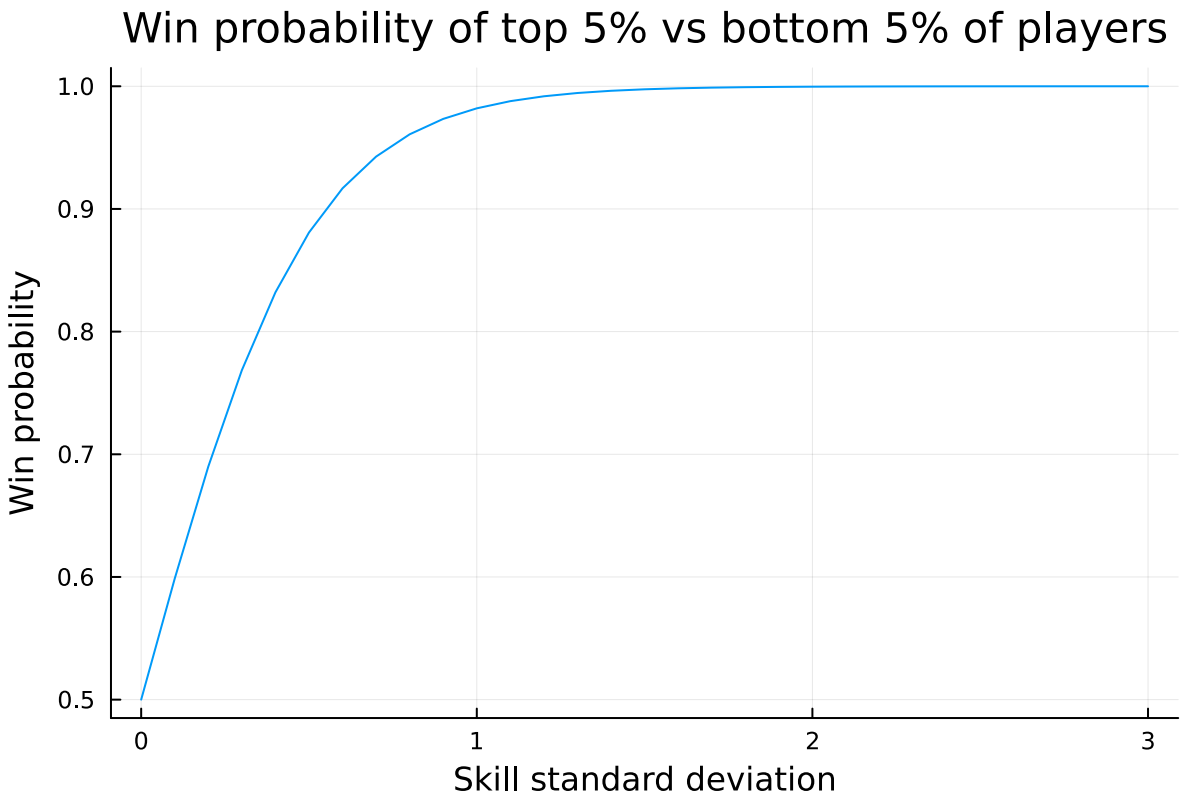

Figure 4: Win probability of the top player 5% of players vs the bottom 5% of players for different skill standard deviations

Running the simulation with different standard deviations shows that its influence is surprisingly minimal, as shown in Figure 5. The curves are almost identical except for very high trophy counts. This means our result is robust against changes in this somehow arbitrary parameter.

```
p=plot(xlabel="Trophies",   ylabel="Percentile",   title="CDF   of   trophies",
yscale=:log10, yminorgrid=true)
num_players=10000
num_rounds=1000
for std in [0.05, 0.1, 0.2, 0.5, 1, 2,4]
    players = [Player(Normal(0,std)) for i in 1:num_players]
    for round in 1:num_rounds
        step!(players)
    end
    trophies=[p.trophies for p in players]
    trophies=sort(trophies)
    percent_better=1.0 .- (1:num_players) ./num_players
   plot!(p,trophies[1:end-1], percent_better[1:end-1], label="skill std: $std",
xlims=(600,maximum(trophies)))
end
display(p)
```

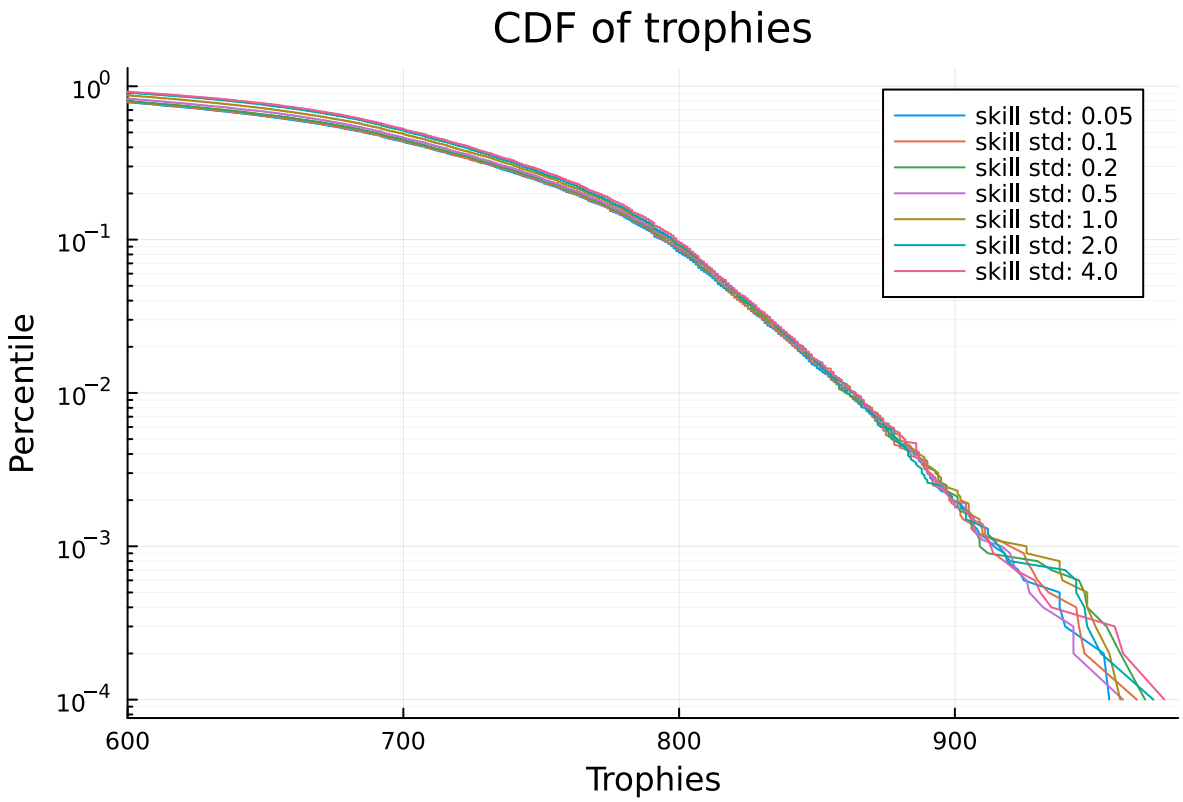

Figure 5: Fraction of players with at least a certain number of trophies for different skill standard deviations and 10000 players after 1000 rounds.

Finally, we find in Figure 6 that playing more actively does not influence the trophy count significantly.

```
players_to_plot = sample(players, 10000)
scatter([p.activity for p in players_to_plot], [p.trophies for p
in players_to_plot], label="Activity vs Trophies", xlabel="Activity",
ylabel="Trophies", title="Activity vs Trophies", alpha=0.1)
```

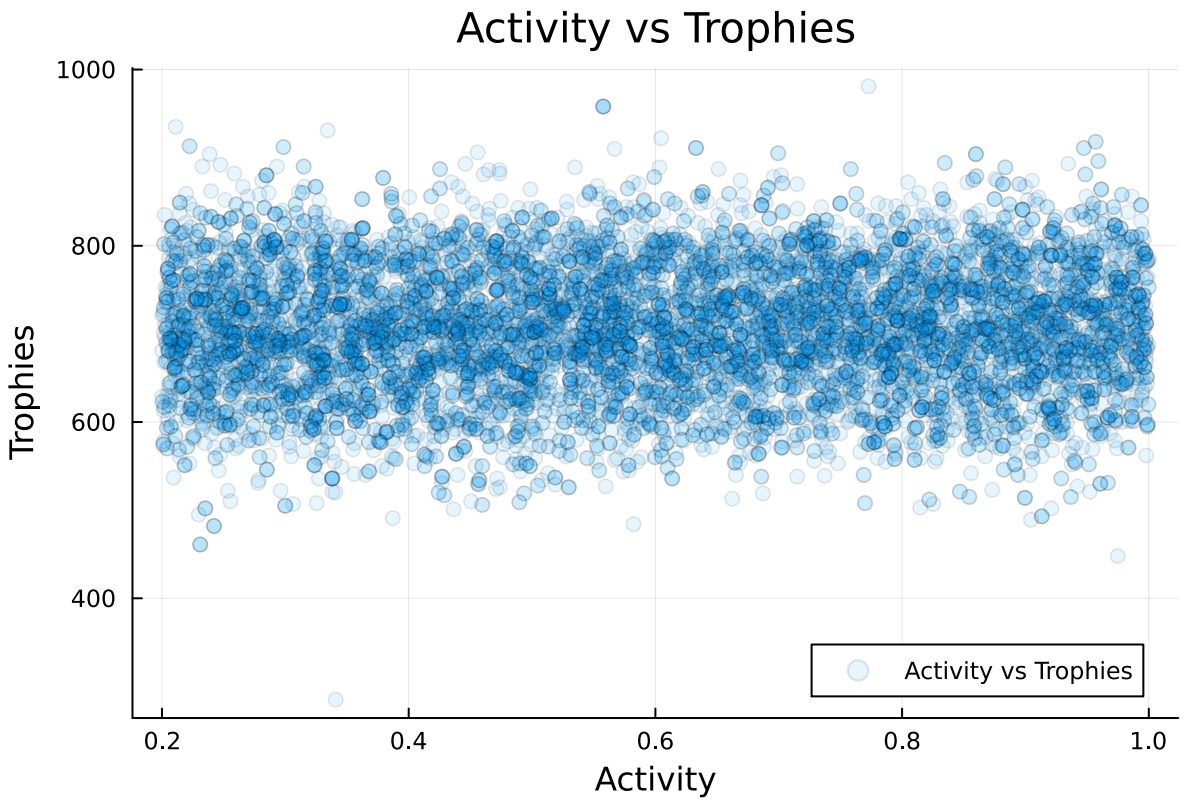

Figure 6: Activity level vs trophies for the first 10000 players

To conclude, we found that you need 864 trophies to be in the top 1% of players in Brawl Stars. This result is robust against changes in the standard deviation of the skill level and the activity level of the players. We hope you enjoyed this post and learned something new about Brawl Stars! And remember, it is just a game; playing too much does not help in our simulation.