Simulating Robot Tournaments

Introduction and Rationale

I love robots. They dare venture where humans will not and accomplish feats we think impossible. Like NASA's Mars Rover "Perseverance," robots surge past the boundaries of possibility and are only limited by one's capacity to imagine.

I also make Mars rovers, but mine are the smaller kind made of LEGOs. Every year, a team of classmates and I create a robot to compete in international robot competitions. Similar to Nasa's Mars rover's "7 minutes of terror," my robot must complete a set of pre-programmed tasks without any humans controlling it. I can't communicate with or control my robot once I place it down on the competition board. As one would expect, this creates for exciting and suspenseful showdowns with other robots since you don't know if either robot will work, or if it will crash and burn. Either way, there's no way for us to correct its course once it starts running.

At my most recent robotics tournament, GCER[1] 2019, my team won 5th place. But, as an avid follower of math, I wondered if there was a way to find the probability my robot wins 1st at the tournament given that the tournament was exactly the same. It was to my excitement, then, when I realized this would make a perfect exploration for my Math IA.

In this exploration, I will determine a method to calculate the probability a robot wins a certain round. Then, I will discuss how to find the probability a robot wins consecutive rounds. Finally, I will simulate the GCER 2019 tournament using variables extrapolated from real-world data to conclude the probability that my robot could win the tournament.

Aim and Approach

The aim is simple: what is the probability that my robot could win GCER 2019? Firstly, we must decide what makes a "robot". A robot, *R*, generally consists of two things:

1. **Points (*p*):** A robot completes a set of tasks for each tournament round, and these tasks give the robot a certain amount of points. Each round lasts two minutes. Since no team can affect the robot after it starts, it has to have a pre-programmed strategy. This means that the robot will always score the same number of points if it succeeds. However, one number to account for a robot's "scoring potential" does not account for certain random factors during a robot's run that, in reality, make a robot's points a range of ±10. But, accounting for this situation greatly complicates the model and would not affect a robot's average point potential, so I will ignore these random points.
2. **Success rate (*s*):** No robot is perfect, so there is always a chance that a robot does not achieve its scoring potential. This is especially true at GCER since robots are not remote-controlled. The robot's success rate, *s*, will be the percent chance that it scores its points, *p*. If a robot fails at completing its run, it will score zero points. In other words, $p_n = 0$ where p_n is the score of a particular round. This reflects the "all-or-nothing" element of autonomous robots, which tend to either succeed at attaining their scoring potential or catastrophically fail during their run. Though a robot rarely scores exactly 0 on failed runs due to the random ±10 points, this assumption reduces the model's complexity and also allows for a more interesting exploration.

Secondly, a "tournament" is a group of robots in a perfect bracket with no byes. In other words:

$$R_T = 2^n, n \in Z^+,$$

where R_T is the total number of robots in a tournament and *n* is some positive integer that represents the number of rounds. We can arrange this to find *n* based on the total robots:

$$n = \log_2(R_T).$$

For example, in a tournament of 4 robots, we can find the number of rounds by setting

$$\begin{aligned} n &= \log_2(4) \\ &= 2. \end{aligned}$$

meaning we would have 2 rounds.

[1] GCER: Global Conference on Educational Robotics. Yearly robotics tournament between the best teams from around the world.

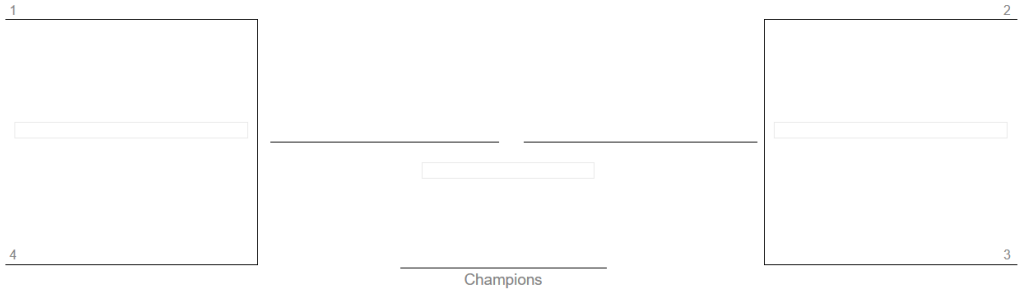


Fig. 1 Example bracket with total robots, R_T, equal to 4 from online bracket generator.

Looking at the bracket, we find this to be correct by eye.

The tournament will be single-elimination, meaning a single loss takes one out of the tournament. Every round, two robots in the bracket face off with the winner moving to the next "prong" of the bracket to face the other winners from earlier rounds. So, to win a tournament, a robot must consecutively win every round. Immediately, we know that

$$P(W_n) = P(W_{n-1} \cap W_n),$$

where $P(W_n)$ is the total probability that a robot wins a tournament of n rounds. This is true since you can only win a round if you won the round before it. This means that $P(W_n) \subseteq P(W_{n\text{-}1})$. Going further,

$$\begin{aligned} P(W_n) &= \frac{P(W_{n-1}) \cdot P(W_{n-1} \cap W_2)}{P(W_{n-1})} \\ &= P(W_{n-1}) \cdot P(W_n|W_{n-1}), \end{aligned}$$

which is true by reversing the conditional probability formula in the formula booklet. In this case, $P(W_n \mid W_{n\text{-}1})$ is the probability that a robot wins the nth round. By applying this same equation to $P(W_{n\text{-}1})$ and beyond, we get

$$P(W_n) = P(W_1) \cdot P(W_2|W_1) \cdot P(W_3|W_2) \cdot \ldots P(W_{n-1}|W_{n-2}) \cdot P(W_n|W_{n-1}).$$

Winning the first round is not conditional since there is no 0th round. This multiplication pattern continues for the number of rounds, n, in the tournament. The chance that the robot wins the tournament decreases with each round, so a tournament with more rounds signifies a smaller win chance for each robot. The single-elimination bracket of GCER 2019 had 16 robots, so we assume our hypothetical GCER has 16 robots also. Since

$$\log_2(16) = 4,$$

our GCER will have 4 rounds. To figure out how to calculate the probability of my robot winning this theoretical tournament do-over, I will analyze a "sample" tournament of 4 robots with randomly selected points and success rates with the same parameters as GCER. This is easier than immediately attempting to calculate my robot's win probability for GCER since, with a smaller tournament, I can more easily verify that my results are correct and avoid redundant calculations. As

$$\log_2(4) = 2,$$

2 rounds will take place in the sample tournament. Each robot will have an arbitrary number of points and success rate associated with it. The points, p, range from 0 to 1000. or

$$0 < p \leq 1000,$$

since this roughly reflects the scale of points in real competitions. Of course, a robot can not have a scoring potential of 0 points since all robots will score at least some amount of points or else they would not be in the competition. The robot's success rate, s, is a decimal that represents the probability that the robot scores its ideal score. Thus,

$$0 < s < 1.$$

A robot can never have a success rate of 0 or 1 since this would imply it either never succeeds or is perfect, neither of which is practical in real tournament play. For the sample tournament, all robots have success rates within the range of

$$0.125 \leq s \leq 0.875,$$

so that no one robot has too much of an individual effect on the tournament. For example, a robot with a success rate of 0.001 would essentially always lose and give its opponent an advantage, whereas a robot with a success rate of 0.999 would almost always win. Either way, it doesn't make for an interesting exploration and does not reflect the norm in real tournaments. Also, I chose not to give any robot unreasonably high success rates in combination with high point potentials

as this could make one robot completely dominate, leaving others with irrelevant win chances. This also reflects real tournament conditions since a robot that ambitiously aims for a high score tends to be riskier, meaning that no one robot has both high points and a high success rate. Finally, no robot will completely "outclass" another robot by having a higher point total and success rate to ensure a diversity of robot types, each with strengths and weaknesses.

Each robot's name, point score, and success rate will be displayed in the following table.

Robots	**Points (*p*)**	**Success Rate (*s*)**
Robot J	200	0.25
Robot K	100	0.5
Robot L	50	0.75
Robot M	250	0.125

Table 1 Robots in sample tournament.

The initial matchups will follow logically with Robot J facing Robot K in round 1 and Robot L facing Robot M in round 1. As a shorthand, a robot's points and success rate will be represented in the form (*p*, *s*), or (points, success rate). Visually, we can represent this table with the following bracket:

Fig. 2 Sample tournament bracket from online bracket generator.

To find the probability that my robot wins GCER, I will find the probability that a robot in the sample tournament wins on the smaller, 4-robot scale. I will first find the probability that a robot in this sample tournament wins a certain round and then the probability that it wins consecutive rounds. After, I will extrapolate these calculations to apply to the larger GCER tournament and determine the probability that my robot could win.

Body

Winning a Round

To analyze the chance a certain robot wins a given match, we must focus on one "side" of the bracket, either on the match between Robot J and K or the one between L and M. I chose to focus on the Robot J/K side because it seemed more natural, but focusing on Robot L/M would work just as well.

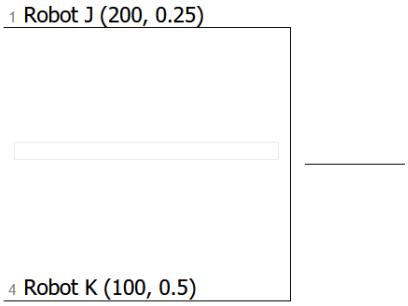


Fig. 3 Robot J vs Robot K side of bracket.

There are four discrete possible outcomes of this match:

1. Both robots fail to achieve their point score
2. Robot K succeeds and Robot J fails to achieve their point score
3. Robot J succeeds and Robot K fails to achieve their point score
4. Both robots succeed to achieve their point score

Added all together, these 4 separate outcomes add up to make the entire spread of possible outcomes for this match. We can call each outcome a "partition" B_n, where n is the partition number. The spread of all possible outcomes is the same as the "sample space", S, of the match. We can represent this geometrically where each of these outcomes represents a part of the whole by creating a square of side length 1.

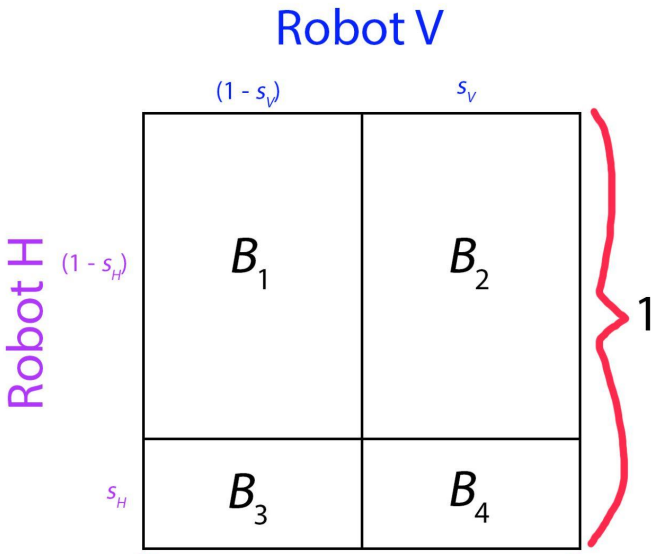


Fig. 4 Sample space, S, of Robot H vs Robot V. Created in Photoshop.

The numbering of the partitions ascends from left to right, top to bottom. Each of the four outcomes can be displayed as separate, rectangular portions of the larger square, with each side length tied to the success rates of the robots. In this case, the "hero" robot, H, is Robot J and the opposing "villain" robot, V, is Robot K. Since the square is of side length 1, s_H' and s_V' can be written as (1 - s_H) and (1 - s_V). The probability that a combination of two events s_H, s_V, and their primes is B_n. For example, B_1 is the combination of events (1 - s_H) and (1 - s_V), or the probability that both robots fail. Separating the partitions greatly simplifies the problem and allows us to more intuitively understand the numbers behind the equations.

The probability of each independent partition, B_n will add up to the total probability of all events occurring in sample space S. Therefore, the probability of winning in each partition added together equals the total probability of winning in S by the Law of Total Probability. So, we can represent the probability that the "hero" robot wins a certain match with the equation

$$P(W_H) = P(B_1)\cdot P(W_H|B_1)+P(B_2)\cdot P(W_H|B_2)+P(B_3)\cdot P(W_H|B_3)+P(B_4)\cdot P(W_H|B_4).$$

where $P(W_H \mid B_n)$ is the probability the "hero" robot wins in partition B_n. This works since $P(B_n) \cdot P(W_H \mid B_n)$ is the probability of Robot H winning in partition B_n. This is the fundamental equation for a robot winning a certain round. For simplicity, the "hero" robot will be Robot J for the sample tournament. In other words, we will follow Robot J through the tournament. Again, focusing on either robot would work just as well and result in the same findings. $P(B_1)$ through $P(B_4)$ can be found by analyzing Figure 4. For example, to find $P(B_1)$ we can say that

$$\begin{aligned} P(B_1) &= P((1-s_H)\cap(1-s_V)) \\ &= P(1-s_H)\cdot P(1-s_V). \end{aligned}$$

This is an equation in the Math HL formula booklet. We can use the "independent events" equation here since s_H and s_V are independent events, so (1 - s_H) and (1 - s_V) are also independent. This equation is confirmed by geometrically applying the area of a square formula to B_1 in figure 4. For the purposes of this paper, $P(s)$ and s will be used interchangeably.

Continuing for each partition, we get this table:

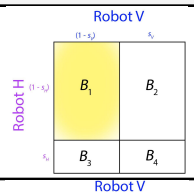
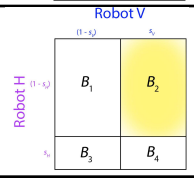
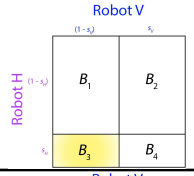
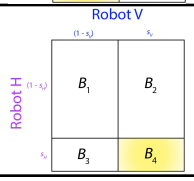
Partition	Generalized probability of partition occuring, $P(B_n)$	$P(B_n)$ for Robot J vs. Robot K, $P(s_H) = P(s_J) = 0.25$ $P(s_V) = P(s_K) = 0.50$	Sample space, S, with partition B_n highlighted
B_1	$(1 - P(s_H)) \cdot (1 - P(s_V))$	$(1 - 0.25) \cdot (1 - 0.50) = 0.375$	
B_2	$(1 - P(s_H)) \cdot P(s_V)$	$(1 - 0.25) \cdot 0.50 = 0.375$	
B_3	$P(s_H) \cdot (1 - P(s_V))$	$0.25 \cdot (1 - 0.50) = 0.125$	
B_4	$P(s_H) \cdot P(s_V)$	$0.25 \cdot 0.50 = 0.125$	

Table 2 Expressions, values, and visual representations for partitions $P(B_1)$ through $P(B_4)$.

Intuitively, we know that when one robot succeeds and the other robot fails, the first robot will win and vice versa since only one robot will be scoring anything. So,

$$P(W_J|B_2) = 0$$
$$P(W_J|B_3) = 1$$

because Robot J will always lose in partition B_2 and always win in partition B_3. "0" and "1" represent a guaranteed loss or win in a given partition respectively. This is true for all matchups of robots since, regardless of points, a robot that scores something will always beat a robot that scores 0. So, it will be assumed that $P(W_H \mid B_2) = 0$ and $P(W_H \mid B_3) = 1$ for the rest of the paper. The only case where this would not be true is if either of the robots' $p = 0$ since B_3 and B_4 would result in a tie, but this edge case has already been accounted for since I excluded robots with a p of 0. Also, it is trivial that the robot with the higher points will win if both robots succeed. In this case, Robot J wins if both robots succeed since $p_J > p_K$. Thus,

$$P(W_J|B_4) = 1.$$

$P(W_J \mid B_2)$, $P(W_J \mid B_3)$, and $P(W_J \mid B_4)$ will always be 0 or 1 since the robot will always lose or win in these partitions. This fact makes calculations easier down the line. Finding $P(W_J \mid B_1)$, though, is more complicated. This is because both robots fail in partition B_1, so there would be a tie at 0 points. There is no clear winner in B_1 like in the other partitions. We can resolve this tie using two methods commonly found in real tournament play:

1. **Flip a coin.** If the robots tie, then each robot gets a 50% chance of winning. This is the easiest method since $P(W_J \mid B_1)$ is simply 0.50.
2. **Redo the round.** In this case, the match is redone until one robot wins. This method is most commonly used at international tournaments to eliminate the chance that a team luckily wins a coin flip. A flaw with this method is that, technically, this method could go on forever if both robots keep failing. However, the probability of a game going on infinitely is extraordinarily low since

   $$\lim_{x \to \infty} ((1 - s)^x) = 0, 0 < s < 1.$$

   This means that a robot will eventually work and win. There is no possibility for an infinite game since the edge cases of s = 0 and s = 1 have already been omitted.

Because it is not too difficult, I will find the win probability for each tie resolution method and determine which is best.

Flip a coin method

If we continue following Robot J,

$$P(W_J|B_1) = 0.5$$
$$P(W_J|B_2) = 0$$
$$P(W_J|B_3) = 1$$
$$P(W_J|B_4) = 1.$$

If we plug this back into the equation for $P(W_J)$:

$$\begin{aligned} P(W_J) &= P(B_1) \cdot P(W_J|B_1) + P(B_2) \cdot P(W_J|B_2) + P(B_3) \cdot P(W_J|B_3) + P(B_4) \cdot P(W_J|B_4) \\ &= 0.375 \cdot 0.5 + 0.375 \cdot 0 + 0.125 \cdot 1 + 0.125 \cdot 1 \\ &= 0.1875 + 0 + 0.125 + 0.125 \\ &= 0.4375. \end{aligned}$$

If we use the "flip a coin" method, then Robot J has a 43.75% probability of winning against Robot K.

Redo method

Though the "flip a coin" method is a simple and easy resolution, it was much less common at GCER 2019 in particular where the "redo" method was used when both robots failed. Thus, we must derive $P(W_H)$ for when we are using the "redo" method. Since we redo the game when both robots fail,

$$P(W_H|B_1) = P(W_H).$$

So, when plugging back into the fundamental round-winning formula, we get an interesting equation with $P(W_H)$ on both sides of the equation:

$$\begin{aligned} P(W_H) = P(B_1) \cdot P(W_H) + P(B_2) \cdot P(W_H|B_2) \\ + P(B_3) \cdot P(W_H|B_3) + P(B_4) \cdot P(W_H|B_4). \end{aligned}$$

My first intuition was to continually substitute in $P(W_H)$ for itself in the equation. Since $0 < P(W_H) < 1$, the probability will eventually converge at one value instead of increasing to infinity. We can see this with the following math:

1. Substituting variables for clarity

Let

$$r = P(B_1)$$
$$u_1 = P(B_2)P(W_H|B_2) + P(B_3)P(W_H|B_3) + P(B_4)P(W_H|B_4)$$

where r is the coefficient of $P(W_H)$ and u_1 is a constant.

2. Plugging in substituted variables and rearranging

$$\begin{aligned} P(W_H) &= P(B_1) \cdot P(W_H) + P(B_2) \cdot P(W_H|B_2) \\ &\quad + P(B_3) \cdot P(W_H|B_3) + P(B_4) \cdot P(W_H|B_4) \\ &= r(P(W_H)) + u_1 \\ &= u_1 + r(u_1 + r(P(W_H))) \\ &= u_1 + r(u_1 + r(u_1 + r(u_1 + r \dots))) \\ &= u_1 + ru_1 + r^2(u_1 + r(u_1 + r \dots)) \\ &= u_1 + ru_1 + r^2u_1 + r^3u_1 + r^4u_1 \dots. \end{aligned}$$

3. Sum of infinite geometric series equation

For an infinite geometric series

$$\sum_{n=1}^{\infty} u_1(r)^n, |r| < 1,$$

the sum is

$$S_\infty = \frac{u_1}{1-r},$$

which I found in the Math HL formula booklet. Since r = $P(B_1)$ and $|P(B_1)| < 1$, we know this series converges on one answer.

4. Plugging everything back in

Therefore, the equation for calculating a robot's probability of winning a round when using the redo method is

$$P(W_H) = \frac{P(B_2)P(W_H|B_2) + P(B_3)P(W_H|B_3) + P(B_4)P(W_H|B_4)}{1 - P(B_1)}.$$

5. Simplifying

We can simplify this equation because in all cases $P(W_H \mid B_2) = 0$ and $P(W_H \mid B_3) = 1$ as stated before. This is the simplified version of the equation:

$$P(W_H) = \frac{P(B_3) + P(B_4)P(W_H|B_4)}{1 - P(B_1)}.$$

$P(W_H \mid B_4)$ is always 0 or 1 depending on the p of Robot H and V.

However, as I wrote this exploration I realized that there was a simpler method to find $P(W_H)$: solving algebraically. First, the original equation:

$$P(W_H) = P(B_1) \cdot P(W_H) + P(B_2) \cdot P(W_H|B_2) + P(B_3) \cdot P(W_H|B_3) + P(B_4) \cdot P(W_H|B_4),$$

which can be simplified to

$$P(W_H) = P(B_1) \cdot P(W_H) + P(B_3) + P(B_4) \cdot P(W_H|B_4).$$

for reasons already stated. Then, solving for $P(W_H)$:

$$P(W_H) - P(B_1) \cdot P(W_H) = P(B_3) + P(B_4) \cdot P(W_H|B_4)$$
$$P(W_H)(1 - P(B_1)) = P(B_3) + P(B_4) \cdot P(W_H|B_4)$$
$$P(W_H) = \frac{P(B_3) + P(B_4) \cdot P(W_H|B_4)}{1 - P(B_1)}.$$

This gets the same equation as the infinite geometric series method. As with all math, there will be many ways to get the same answer. Ultimately, any method that one uses to get the right answer is a “correct” method, even if a simpler method than one first thought exists. Therefore, both methods are equally valid and even serve to further validate each other.

Both equations tell us that the probability a robot wins while using the redo method is the probability that the robot wins in non-tie scenarios divided by the portion of games these non-ties occur. We can represent this visually by highlighting the top and bottom portions of the equation on the geometric representation of round outcomes shown before. Note how the green stripes are thinner in B_4 to signify that this partition only sometimes counts for this equation.

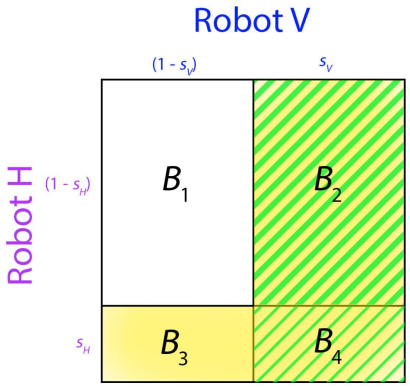


Fig. 5 Sample space of round outcomes with $P(B_2) + P(B_3) \cdot P(W_H \mid B_3)$ highlighted with green stripes and (1 - P(B_1)) highlighted in yellow.

Looking at the probability square, we see that B_1 is essentially omitted when using the redo method as it is not highlighted. This is because (1 - $P(B_1)$) = $P(B_1)$ + $P(B_2)$ + $P(B_3)$. Visually, we can see that the equation finds the probability the robot wins in B_1’ and then rescales this value to a sample space without B_1 by dividing by B_1’. This goes against my intuition since it implies that “redo”-ing rounds in the B_1 partition is the same as ignoring the B_1 partition. To test this hypothesis, I decided to approach this problem using conditional probability.

Ignoring B_1 Method

Instead of redoing each round in B_1, I want to find the probability that a robot wins given that B_1 did not occur. This will find $P(W_H \mid s_H \cup s_V)$, where s_H and s_V are the success rates of stand-in Robots H and V respectively. This should give us the same answer as the infinite geometric series method. Dealing with conditional probabilities normally necessitates Bayes’ Theorem since it relates inverse conditional probabilities $P(A \mid B)$ and $P(B \mid A)$, and this situation is no different. As a reminder, Bayes’ Theorem is

$$P(B|A) = \frac{P(B)P(A|B)}{P(B)P(A|B) + P(B')P(A|B')},$$

which is taken directly from the HL formula booklet. Plugging our values into Bayes’ Theorem, we get

$$P(W_H|s_H \cup s_V) = \frac{P(W_H)P(s_H \cup s_V|W_H)}{P(W_H)P(s_H \cup s_V|W_H) + P(W_H')P(s_H \cup s_V|W_H')}.$$

It is important to note that, by the nature of conditional probabilities, this method requires another way of finding $P(W_H)$. We can think of Bayes’ Theorem as modifying an existing win probability to remove the influence of B_1, so it needs

another method to find the initial $P(W_H)$ to be modified. When this situation arises, I will use the flip a coin method to find the initial $P(W_H)$. The bottom part of the fraction is equivalent to $P(s_H \cup s_V)$ since $P((s_H \cup s_V) \cap W_H') + P((s_H \cup s_V) \cap W_H) = P(s_H \cup s_V)$. Thus, we can simplify this equation to

$$P(W_H|s_H \cup s_V) = \frac{P(W_H)P(s_H \cup s_V|W_H)}{P(s_H \cup s_V)}.$$

This is a more practical version of Bayes' Theorem than the one in the formula booklet since we can easily find $P(s_H \cup s_V)$. This can be seen with the following diagram:

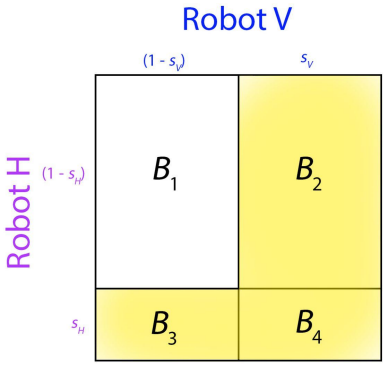


Fig. 6 Sample space of round outcomes with $s_H \cup s_V$ highlighted.

By eye, one can see that

$$P(s_H \cup s_V) = P(B_2) + P(B_3) + P(B_3).$$

If we can find $P(s_H \cup s_V \mid W_H)$, or the probability that at least one robot succeeded given that the hero robot wins, then we can find $P(W_H \mid s_H \cup s_V)$. We can find $P(s_H \cup s_V \mid W_H)$ and thus $P(W_H \mid s_H \cup s_V)$ using Bayes' Theorem. We can see this with the following math:

1. Separating $P(s_H \cup s_V \mid W_H)$ into partitions

To find $P(s_H \cup s_V \mid W_H)$, we can find the probability of falling in each portion of $s_H \cup s_V$ given that Robot H wins and add these probabilities together with the Law of Total Probability:

$$P(s_H \cup s_V) = P(B_2) + P(B_3) + P(B_3)$$
$$P(s_H \cup s_V|W_H) = P(B_2|W_H) + P(B_3|W_H) + P(B_4|W_H).$$

2. Using Bayes' Theorem on $B_n \mid W_H$ and simplifying

We can use Bayes' Theorem on each partition so we can plug in $P(W_H \mid B_n)$, a value that we know.

$$P(s_H \cup s_V|W_H) = \frac{P(B_2)P(W_H|B_2)}{P(W_H)} + \frac{P(B_3)P(W_H|B_3)}{P(W_H)} + \frac{P(B_4)P(W_H|B_4)}{P(W_H)} = \frac{P(B_2)P(W_H|B_2) + P(B_3)P(W_H|B_3) + P(B_4)P(W_H|B_4)}{P(W_H)}.$$

3. Plugging back into initial equation

$$P(W_H|s_H \cup s_V) = \frac{P(W_H)\frac{P(B_2)P(W_H|B_2)+P(B_3)P(W_H|B_3)+P(B_4)P(W_H|B_4)}{P(W_H)}}{P(s_H \cup s_V)} = \frac{P(B_2)P(W_H|B_2) + P(B_3)P(W_H|B_3) + P(B_4)P(W_H|B_4)}{P(s_H \cup s_V)}.$$

4. Simplifying (same as step 5 for inf. geo series)

Since $P(W_H \mid B_2) = 0$ and $P(W_H \mid B_3) = 1$, we can simplify the equation to

$$P(W_H|s_H \cup s_V) = \frac{P(B_3) + P(B_4)P(W_H|B_4)}{P(s_H \cup s_V)}.$$

In English, this equation is the wins that come from at least one robot working divided by the portion of times that at least one robot works. Interestingly, we actually do not need a method of finding $P(W_H)$ because $P(W_H)$ gets canceled out in the equation. By looking back at figure 4, we know that $P(s_H \cup s_V) = 1 - P(B_1)$ since the area of the square is 1. Therefore, by eye we can see that the equation found by Bayes' Theorem is identical to the sum of an infinite geometric series and algebraic equation found earlier. Thus, my hypothesis that redoing rounds in B_1 is identical to ignoring B_1 is proven. This endeavor was also useful since, with all methods arriving at the same result, all are validated.

Plugging in values for Robot J against Robot K:

$$\begin{aligned} P(W_{J,1}) &= \frac{0.125 + 0.125 \cdot 1}{1 - 0.375} \\ &= \frac{0.25}{0.625} \\ &= 0.40. \end{aligned}$$

Robot J has a 40% probability of winning round 1 when using the redo method. The notation for a robot winning a round $P(W_{H,n})$ where H is the robot and n is the round number is used for specificity. This notation will be used for the rest of the paper. I will also use the redo method to calculate round-winning probabilities from now on because it was most common at GCER 2019.

Winning Consecutive Rounds

Now that we know the probability a robot wins an individual match, we must find the probability that the robot wins multiple rounds in a row. Here is the general consecutive round-winning probability formula established farther up for reference:

$$P(W_n) = P(W_1) \cdot P(W_2|W_1) \cdot P(W_3|W_2) \cdot \ldots P(W_{n-1}|W_{n-2}) \cdot P(W_n|W_{n-1}),$$

The probability that we win round 2 given that we won round 1 essentially just means the probability that we can win round 2 once we get there. This can be shown as

$$P(W_{J,2}|W_{J,1}) = P(B_L) \cdot P(W_{J,2}|B_L) + P(B_M) \cdot P(W_{J,2}|B_M).$$

In this case, we have different partitions than the single match calculations. The partitions, B_n, are the probability that Robot J faces a certain opponent, and we multiply this by the probability Robot J beats that opponent. For example, $P(B_L)$ is the probability that the second round opponent will be Robot L and $P(W_{J,2} \mid B_L)$ is the probability that Robot J wins in this partition. The probability that Robot J faces a certain robot is the same as that robot's probability of winning round 1. So,

$$P(W_{J,2}|W_{J,1}) = P(W_{L,1}) \cdot P(W_{J,2}|W_{L,1}) + P(W_{M,1}) \cdot P(W_{J,2}|W_{M,1}).$$

This requires us to calculate the win probabilities for the entire other side of the tournament bracket. This is tedious, especially as the number of rounds and robots on the other side of the bracket increase. So, I decided to develop a simulator using Python, a programming language I use to code my robots. This will allow me to quickly calculate a robot's tournament-winning probability based on initial parameters without doing tens of repetitive round-winning calculations by hand. Since this tournament is small, I will also calculate by hand to check against my Python simulation. Since I've already shown how to do round-winning calculations, I will move through them quickly.

By hand:

1. Defining variables

$$P(W_{L,1}) = \frac{P(B_3) + P(B_4)P(W_{L,1}|B_4)}{1 - P(B_1)}$$
$$P(W_{M,1}) = \frac{P(B_3) + P(B_4)P(W_{M,1}|B_4)}{1 - P(B_1)}$$
$$P(W_{J,2}|W_{L,1}) = \frac{P(B_3) + P(B_4)P((W_{J,2}|W_{L,1})|B_4)}{1 - P(B_1)}$$
$$P(W_{J,1}|W_{M,1}) = \frac{P(B_3) + P(B_4)P((W_{J,1}|W_{M,1})|B_4)}{1 - P(B_1)},$$

where B_n is a partition of the related round.

2. Setting up the equation

$$\begin{aligned} P(W_{J,2}|W_{J,1}) = &\frac{P(B_3) + P(B_4)P(W_{L,1}|B_4)}{1 - P(B_1)} \\ &\cdot \frac{P(B_3) + P(B_4)P((W_{J,2}|W_{L,1})|B_4)}{1 - P(B_1)} \\ &+ \frac{P(B_3) + P(B_4)P(W_{M,1}|B_4)}{1 - P(B_1)} \\ &\cdot \frac{P(B_3) + P(B_4)P((W_{J,1}|W_{M,1})|B_4)}{1 - P(B_1)}. \end{aligned}$$

3. Plugging numbers in

$$\begin{aligned} P(W_{J,2}|W_{J,1}) = &\frac{0.75 \cdot (1 - 0.125) + 0}{(1 - (1 - 0.75)(1 - 0.125))} \\ &\cdot \frac{0.25 \cdot (1 - 0.75) + 0.25 \cdot 0.75 \cdot 1}{1 - (1 - 0.25)(1 - 0.75)} \\ &+ \frac{0.125 \cdot (1 - 0.75) + 0.125 \cdot 0.75 \cdot 1}{(1 - (1 - 0.125)(1 - 0.75))} \\ &\cdot \frac{0.25 \cdot (1 - 0.125) + 0}{1 - (1 - 0.25)(1 - 0.125)} \\ = &\, 0.36028. \end{aligned}$$

4. Solving for $P(W_J)$

$$\begin{aligned} P(W_J) &= P(W_{J,1}) \cdot P(W_{J,2}|W_{J,1}) \\ &= 0.4 \cdot 0.36028 \\ &= 0.144111. \end{aligned}$$

Robot J has a 14.4% probability of winning the sample tournament when calculating by hand.

By Python simulator:

Instead of merely creating a calculator, I want to simulate millions of tournaments given the robots' parameters to see if my theoretical calculations reflect the real probabilities. Of course, the more tournaments you simulate, the closer you will approach the "true" probability of winning the tournament given that everything was set up correctly due to the Law of Large numbers. I will set the initial matchups, success rates, and points to the same values assigned at the beginning of the exploration. Since this is not a computer science exploration, I will only briefly dwell on the implementation.

The full Python code can be found in the appendix. I will be running it in VS Code. My implementation allows me to create a bracket based on the order I create the robots, turn on and off the 50/50 or redo method, and even randomize the starting matchups if I wanted to. Python is the best programming language for this simulation due to its simplicity, and Python is commonly used for statistical applications like this in the real world. Plugging all the robots into the Python calculator, we get 14.4%:

```
Robot L win chance: 44.8516%
Robot K win chance: 36.2637%
Robot J win chance: 14.4191%
Robot M win chance: 4.4653%
```

Fig. 7 Sample tournament results from Python simulation.

The probabilities all add up to 100% since they make up all the possible outcomes of the sample tournament. The simulated probability is ±0.01% off the probability calculated by hand. This is more than reasonable to confirm the Python simulation's effectiveness. Likely, if I ran through billions of tournaments instead of millions in the calculator, this probability would be even closer. This is impractical, though, since this program already takes 20 seconds to run, so assuming the time scales linearly, increasing the number of tournaments by 10^3 would increase the time by a factor of 10^3, or to 5.55 hours. I ran the simulation a few times, and I got a value within 0.01% each time. Interestingly, Robot L has the highest tournament-winning probability. This is not what I expected since it has the lowest *p* of all the robots, meaning it will always lose if both robots succeed. Having a high success rate seems to be the most critical factor in determining a robot's winning probability since Robot L has the highest *s* of all the robots at 0.75, which is reinforced by the fact that Robot M has the lowest win probability and also the lowest *s* at 0.125. These results also indicate that a high success rate is the best robot strategy, which I will take into account for my GCER robot this year. From now on, I will use the Python simulator to calculate win probabilities since it is much easier to use and reasonably accurate to the theoretical rates.

GCER 2019

Now, we can find the answer to the original aim. First, we must get *p* and *s* for each GCER robot. I found this by looking at the results and scores from the GCER 2019 tournament, which were posted online. I pulled numbers from the 8 documented practice rounds in the tournament called "seeding" rounds. These rounds are non-elimination and serve the purpose of giving each team a "seed" used to create the elimination bracket based on the team's average points. A "higher" seed tends to mean an easier bracket since the team faces "lower" seeded, or weaker, teams. I deemed a robot's *p* as its average points in the seeding rounds. The robot's *s* is the number of times the robot scored higher than or tied with its average points divided by the total number of seeding rounds, or 8. But, a robot scoring only a few points below its average is not the same as it completely failing, so I allowed for 50 points of leniency below the average points to count as a "successful" run. Under this rule, a robot can still score as much above its average as it wants without punishment. Thus, a robot's *s* is the number of times $p_n > p - 50$, where p_n is the points in a practice round, *n*, divided by the total seeding rounds, 8. Having exactly 50 points of leniency is arbitrary, but it could have a major effect on a team's *s* if a team barely misses the 50 point below cut off since the sample pool of seeding rounds is small. A flat amount also favors weaker teams. But, this situation is rare for this tournament in particular and weaker teams generally have less bearing on the tournament as they are normally eliminated early. As all translations from real life to my theoretical robot definitions will inevitably have compromises, this method is sufficient for our purposes.

The values attained are in the following table. My team is “Incredibots.”

GCER 2019 Robots				
Team name	Points	Success rate		Total seeding rounds
Los Altos	878	0.5		8
HTL Unic	672	0.75		
Radiant	633	0.5		
Dead Robot Society	601	0.625		
Hanalani	342	0.5		Points = Average score in 8 "seeding" rounds
Warriors	317	0.375		
Malden	309	0.5		Success rate = rounds ~50 points below average / total seeding rounds
Noble	304	0.375		
CKWA Tron	292	0.625		
Explorer Post 1010	286	0.625		
Incredibots	280	0.875		
Joker	255	0.5		
Malden Catholic	234	0.375		
G-Force	212	0.625		
CWKA Crane	201	0.5		
Guardians of GG	186	0.875		

Table 3 GCER 2019 robots in descending order of seed in Google Sheets. My team, “Incredibots,” is highlighted in orange.

By eye, one can see that a robot’s seed is entirely determined by its *p* since the seeds increase directly with points. As I played around with the Python simulation, I found that initial matchups have a significant impact on the win probability of a robot. For example,

Round 1: Incredibots win % vs Warriors : 59.3%

Round 1: Incredibots win% vs HTL Unic : 22.5%.

This is confirmed intuitively since facing a good team early means you get knocked out before you get a chance to get far. Its particularly relevant when there are multiple strong teams since these power teams can face each other early on, meaning they will eliminate each other before you have to face them. So, getting easy rounds early means you will face fewer strong teams across the whole tournament and therefore have a higher win probability. There is also a chance for upsets allowing for easier games later in the bracket. So, getting easy early rounds is important in maximizing your chances of winning a tournament. This makes sense intuitively and is why tournaments are seeded to encourage getting higher seeds.

I will use the same matchups as the original GCER 2019 to best reflect the chances I’d win given the exact same tournament. I decided not to randomize the seeds for the tournament since GCER seeds were determined by point averages, so the seeds would not change assuming we redo only the single-elimination portion of the tournament. This likely gives my robot a significant advantage since my team’s initial opponent, Warriors, is one of the easiest initial matchups for my robot due to Warriors’ low success rate. The following diagram shows the bracket.

Fig. 8 Actual GCER 2019 bracket.

Plugging this tournament into the Python simulation:

```
# Ordered based on seeding bracket                          # Seed
Los_Altos = Robot("Los Altos", 878, 0.5)                    # 1
Guardians = Robot("Guardians of the Garden Grove", 186, 0.875)  # 16

Noble = Robot("Noble", 204, 0.375)                          # 8
Tron = Robot("CWKA Tron", 292, 0.625)                       # 9

DRS = Robot("Dead Robot Society", 601, 0.625)               # 4
Malden_Cath = Robot("Malden Cath", 234, 0.375)              # 13

Hanalani = Robot("Hanalani", 342, 0.5)                      # 5
Joker = Robot("Joker", 255, 0.5)                            # 12

Unic = Robot("HTL Unic", 672, 0.75)                         # 2
Crane = Robot("CWKA Crane", 201, 0.5)                       # 15

Malden = Robot("Malden", 309, 0.5)                          # 7
EPost = Robot("Explorer Post 1010", 286, 0.625)             # 10

Radiant = Robot("Radiant HT", 633, 0.5)                     # 3
G_Force = Robot("G-Force", 212, 0.625)                      # 14

Warriors = Robot("Warriors", 317, 0.375)                    # 6
Incredibots = Robot("Incredibots", 280, 0.875)              # 11
```

```
Starting code!

HTL Unic win chance: 44.56%
Dead Robot Society win chance: 13.48%
Los Altos win chance: 12.81%
Radiant HT win chance: 5.94%
Incredibots win chance: 5.25%
CWKA Tron win chance: 5.09%
Hanalani win chance: 3.29%
Malden win chance: 2.29%
Guardians of the Garden Grove win chance: 1.99%
Explorer Post 1010 win chance: 1.98%
Warriors win chance: 0.98%
G-Force win chance: 0.82%
Joker win chance: 0.73%
CWKA Crane win chance: 0.29%
Malden Cath win chance: 0.22%
Noble win chance: 0.2%

Finished code!
```

Fig. 9 Bracket plugged into Python Simulation and GCER 2019 Python simulation results.

The results are rounded to the nearest hundredth for ease of reading. When we run the tournament in the Python calculator, we find that Incredibots, my team, has a chance of winning of 5.25%.

Analysis and Conclusion

To answer the original aim of this exploration: my team had roughly a 5% probability of winning GCER 2019 if the exact tournament were to happen again. This would mean that I should expect my team to win within the next 20 years assuming we continue bringing robots of similar strength. This generally aligns with my thinking as my team, though strong, was unlikely to beat some of the stronger teams. Interestingly, the top 5 teams in win probabilities correspond to the same teams that got top 5 in the real tournament. The real top 5 placements are compared to the top 5 from the Python simulation in the following table:

Real Top 5	Simulated Top 5
1. Los Altos	1. HTL Unic
2. HTL Unic	2. Dead Robot Society
3. Dead Robot Society	3. Los Altos
4. Radiant	4. Radiant
5. Incredibots	5. Incredibots

Table 4 Top 5 at GCER 2019 vs top 5 in simulated tournament.

Though the simulation does not predict the exact placements, the robots in the top 5 are the same. This suggests that the simulation was a success and fairly effective at predicting the effectiveness of different robots. However, the simulation puts HTL Unic as the overwhelming favorite when, in reality, Los Altos won. It also puts Dead Robot Society slightly over Los Altos for second place. This is likely because Los Altos changed its strategy to “counter” HTL Unic and Dead Robot Society in the elimination rounds, propelling them to a higher probability of winning than the simulation would predict given the assumption that their robot remained constant from the seeding rounds. This could also reveal that Los Altos’ win was a surprising upset. Indeed, though a win probability may give an idea of how strong a team is, one can only expect the true probabilities to arise after hundreds or even thousands of tournaments. This indicates that the simulation is best used to predict larger trends rather than the single winner. In other words, the Python simulation is more effective at predicting the strong teams versus the weak teams rather than foreseeing the precise winner.

Going further than the aim, I can remove the influence of the initial matchups by randomizing the tournament. This is simple with the Python simulation.

```
HTL Unic win chance: 39.59%
Los Altos win chance: 15.19%
Dead Robot Society win chance: 10.23%
Radiant HT win chance: 6.57%
Incredibots win chance: 6.57%
```

Fig. 10 Randomized GCER 2019 results, abridged to only show the win chances for the top 5 teams for concision.

The Incredibots has a 6.57% probability of winning given randomized starting matchups. I assumed that my team's initial matchups were beneficial, but the calculator refutes this as randomizing the tournament increased my team's win chance from 5.25% to 6.57%. This may be because the Incredibots' projected 2nd round matchup, Radiant, was difficult. Regardless, it shows me that making assumptions in math, even seemingly simple ones, can mislead the mathematician. Also, the top 5 placements remained the same, but the win probabilities of the top 3 were reduced except for Los Altos. This is likely because these teams had higher seeds and thus had easier brackets in the seeded tournament. The increase in win probability for Los Altos is confusing since Los Altos had the highest seed, so I expected its win probability to decrease the most. This tells me that for every trend in math, there is inevitably an exception. But, this need not get in the way of observation since, eventually, one may find a trend without any exceptions, at which point one would have happened upon a new mathematical law.

Evaluation and Extension

Evaluation

- The simulation is an enormous simplification of real robots. Though it depicts a robot generally effectively, robots rarely get exactly 0 when they fail. Normally, the robots score within the random range of $0 \leq p_n \leq 10$ points when they do not succeed instead of 0, which means that robots almost never tie in real competition. However, this does not make much of a difference because teams normally agree to a redo if both of their robots fail anyways.
- Generally, the win probability of a robot is only demonstrated over thousands or millions of games, so a high win probability does not equate to a guaranteed win. This means that HTL Unic getting 2nd place in the real GCER 2019 does not discount the results of the simulation, which put them as a clear favorite.
- This simulation does not take into account how robots can change during the tournament. Even though only one robot may be in play at a time, many teams have multiple programmed: one that scores highly to beat other top teams and one has a high success rate to beat lower-level teams. This means that better teams tend to have higher win probabilities than is demonstrated by only one of their robots.
- Teams that demonstrate a high consistency tend to have higher win probabilities. This is demonstrated clearly in the sample tournament, where Robot L had the highest tournament win probability due to it having a success rate of 0.75 despite having the lowest point potential in the tournament. However, this trend becomes weaker as the number of robots increases with teams such as Los Altos having a high win probability despite having a middling success rate of 0.5. That said, HTL Unic had the highest win probability and also had a high s at 0.75. Regardless, this concept proves my intuition about strong robot strategies having high success rates and should prove useful in my team's GCER robot strategy planning.

Extension

- I could add a "minimum points" to address a situation where a robot, on failure, does not score exactly 0. This would reflect how some robots "fail" better than others.
- I would love to use this Python calculator to find an optimal strategy based on our scouting of other teams for this year's GCER strategy planning. This way, I could maximize my probability of winning this year's GCER.
- I would like to find what would happen if GCER was a double elimination style tournament, where a robot is only eliminated after losing two rounds. This would likely increase the win probability of stronger, more consistent teams since it is harder to "upset" them out of the tournament.

Bibliography

"Bayes theorem." *Youtube*, uploaded by 3Blue1Brown, 22 Dec. 2019, https://youtu.be/HZGCoVF3YvM.

4 Team Single Elimination Printable Tournament Bracket, www.printyourbrackets.com/4teamsingleelimination.html.

"GCER 2019 Resources: KISS Institute for Practical Robotics." *GCER 2019 Resources | KISS Institute for Practical Robotics*, KISS Institute for Practical Robotics, July 2019, www.kipr.org/gcer/about-gcer/gcer-resources/gcer-2019-resources.

"Perseverance Mars Rover Landing- Inside Story." *Youtube*, uploaded by Mark Rober, 14 Feb. 2021, https://youtu.be/tH2tKigOPBU.

Pishro-Nik, Hossein. "Law of Total Probability: Partitions: Formulas." *Law of Total Probability | Partitions | Formulas*, 24 Aug. 2014, www.probabilitycourse.com/chapter1/1_4_2_total_probability.php.

Appendix

Full Python tournament simulator code:

```
import random
import math

robots = []
class Robot:
    def __init__(self, name, points, success_rate):
        self.name = name
        self.success_rate = success_rate
        self.points = points
        self.wins = 0
        self.win_chance = 0
        robots.append(self)

    def __repr__(self):
        return repr((self.name, self.success_rate, self.points, self.wins, self.win_chance))


# Ordered based on seeding bracket                              # Seed
Los_Altos = Robot("Los Altos", 878, 0.5)                        # 1
Guardians = Robot("Guardians of the Garden Grove", 186, 0.875)  # 16

Noble = Robot("Noble", 204, 0.375)                              # 8
Tron = Robot("CWKA Tron", 292, 0.625)                           # 9

DRS = Robot("Dead Robot Society", 601, 0.625)                   # 4
Malden_Cath = Robot("Malden Cath", 234, 0.375)                  # 13

Hanalani = Robot("Hanalani", 342, 0.5)                          # 5
Joker = Robot("Joker", 255, 0.5)                                # 12

Unic = Robot("HTL Unic", 672, 0.75)                             # 2
Crane = Robot("CWKA Crane", 201, 0.5)                           # 15

Malden = Robot("Malden", 309, 0.5)                              # 7
EPost = Robot("Explorer Post 1010", 286, 0.625)                 # 10

Radiant = Robot("Radiant HT", 633, 0.5)                         # 3
G_Force = Robot("G-Force", 212, 0.625)                          # 14

Warriors = Robot("Warriors", 317, 0.375)                        # 6
```

```
Incredibots = Robot("Incredibots", 280, 0.875)                  # 11
"""
# Sample tournament robots
RobotJ = Robot("Robot J", 200, 0.25)
RobotK = Robot("Robot K", 100, 0.5)
RobotL = Robot("Robot L", 50, 0.75)
RobotM = Robot("Robot M", 250, 0.125)
"""

# Battles robot1 and robot2. Returns True if
# robot1 wins and False if robot2 wins.
def battle(robot1, robot2):
    using_redo_method = True  ## Comment this line to use flip a coin method.
    points_scored1 = 0
    points_scored2 = 0
    # Robot scores points based on success rate
    if random.random() < robot1.success_rate:
        points_scored1 = robot1.points
    if random.random() < robot2.success_rate:
        points_scored2 = robot2.points
    if points_scored1 == points_scored2:
        if using_redo_method:
            return(battle(robot1, robot2))
        else:
            if(random.randint(0, 1) == 0):
                return True
            else:
                return False
    return(points_scored1 > points_scored2)


def do_tournament_between(tourneyRobots):
    winners = tourneyRobots.copy()
    #random.shuffle(winners)  ## Uncomment this line to randomize tournament seeds.
    while len(winners) > 1:
        for robot in range(0, int((len(winners)) / 2), 1):
            # Adds winning robot to the winning list and
            # losers to the losers list.
            if battle(winners[robot], winners[robot + 1]):
                del winners[robot + 1]
            else:
                del winners[robot]
```

```
    return winners[0].name


def main():
    print("Starting code!")
    print()
    total_tournaments = 0
    tourney_robots = robots.copy()
    for robot in tourney_robots:
            robot.wins = 0
    for i in range(0, 10000000):
        winner = do_tournament_between(tourney_robots)
        for robot in tourney_robots:
            if robot.name == winner:
                robot.wins += 1
        total_tournaments += 1
    for robot in tourney_robots:
        win_chance = int(robot.wins / total_tournaments * 1000000) / 10000
        robot.win_chance = win_chance
    tourney_robots.sort(key=lambda roboguy: roboguy.win_chance, reverse=True)

    for robot in tourney_robots:
        print(robot.name + " win chance: " + str(robot.win_chance) + "%")

    print()
    print("Finished code!")


main()
```