Sum Tzu and the Mathematics of War: A Predictive Assistant for Warhammer 40,000

Ben Kalmin Newman
Bard College, bn4912@bard.edu

Follow this and additional works at: https://digitalcommons.bard.edu/senproj_s2017

Part of the Numerical Analysis and Scientific Computing Commons

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 4.0 License.

Recommended Citation
https://digitalcommons.bard.edu/senproj_s2017/294

This Open Access work is protected by copyright and/or related rights. It has been provided to you by Bard College's Stevenson Library with permission from the rights-holder(s). You are free to use this work in any way that is permitted by the copyright and related rights. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. For more information, please contact digitalcommons@bard.edu.
Sum Tzu and the Mathematics of War: 
a Predictive Assistant for Warhammer 40,000

Senior Project Submitted to 
The Division of Science, 
Mathematics and Computing 
of Bard College 

by 
Ben Newman 

Annandale-on-Hudson, New York 
May 2017
Acknowledgements:

Robert McGrail, for guiding me through this.
Sven Anderson, for helping me get into the department, and believing in me when even I didn’t.
To Eric, Eric, Michael, Devin, Quincy, and Romulo, for spending countless weekends play with tiny plastic men with me.
To my friends at Bard, for giving me 4 years I wouldn’t trade for the world.
To Cat, for putting up with my puns.
To Will, because 3 years won’t be long enough to forget a promise.
To Riley, because 10 years of friendship is a bond stronger than adamantium.
To Aaron “Little Abaddon”, who put up with my endless theory crafting and sulking.
To Mom and Dad, for the endless support, guidance, and love. I wouldn’t have gotten anywhere without you guys, I love you so much.
To Abe, for putting up with being the black sheep in a family of black sheep, and a better brother than I deserve.

And to Derek Nelson, a true son of Russ, may he ride again with us in the Wolf Times.
Abstract:

The purpose of this project is to classify simple strategies for the tabletop miniature war game *Warhammer 40,000*. The paper enumerates a series of strategies that are straightforward to automate. Further analysis on these simulations identify collection of proposed best and worst auto-strategies.
# Table of Contents

Introduction: ..................................................................................................................................... 1  
Background ....................................................................................................................................... 3  
   A brief History of 40,000 Years: ................................................................................................. 3  
   The Game Now: ........................................................................................................................... 3  
   The Rules: ................................................................................................................................... 5  
   The Armies: ................................................................................................................................ 6  
   How to Play: ................................................................................................................................. 7  
Movement Phase: ............................................................................................................................ 7  
Shooting Phase: ............................................................................................................................... 9  
Assault Phase: ............................................................................................................................... 11  
The Turn, In Summary: ..................................................................................................................... 12  
   An example of play: ...................................................................................................................... 14  
Strategies: ..................................................................................................................................... 23  
   Representation: ............................................................................................................................ 24  
Simulations: ................................................................................................................................... 27  
Conclusions: .................................................................................................................................. 29  
   5 Best Strategies ....................................................................................................................... 31  
   5 Worst Strategies ...................................................................................................................... 31  
   Future Research: ....................................................................................................................... 33  
Terminology: .................................................................................................................................. 34  
Bibliography: .................................................................................................................................. 35  
Indices ............................................................................................................................................... 36  
   Tables: ....................................................................................................................................... 36  
   Code: ......................................................................................................................................... 45
**Introduction:**

Warhammer 40,000 is a game that I have been passionate about since my freshmen year of high school. It has a multifaceted appeal with a unique aesthetic, a deep and complex universe that the story takes place in, and a set of constantly evolving mechanics.

That is not to say that it is flawless by any means. It is a series that at times can be buried by the weight of the complexity of the game and the vestigial elements from older edition that the game has yet to evolve past. This results in a game that people criticize for a plethora of reasons, to the extent that many of them contradict each other. The game has been criticized for being imbalanced: that games are decided in the creation of the armies so that the actual game is superfluous; and that each edition has certain strategies that are just too good to counter. It has also been criticized for being too random: that strategy and unit selection are less valuable than sheer blind luck. The objective of this project is to assess if there are any universal simple automatic strategies that perform significantly better than others, and how much random dice rolls effect the overall outcome of the game. This project has two parts, creating the simulation and analyzing the data that it creates.

The first part of the project is a simulation of the tabletop war game *Warhammer 40,000* (7th edition). In it two armies are guided by a simple automatic strategy. A simple automatic strategy is strategies that the computer can implement without having to do excessively complex calculation before acting. There are 16 of these strategies in this simulation, but given the nature of the program, more can be easily created. For these simulations work in a way that is accurate to the game, they have to be able to correctly replicate the turn order and mechanics of *Warhammer 40,000.*
Due to the dice rolling mechanic of the tabletop game, the simulation will have an element of randomness that can only be replicated by taking the probability seed, making it unlikely to organically occur. However, this means that the data collected will be unlikely to perfectly represent the mathematically average game, and thus some results will perform better or worse than they are expected to. Additionally, this means that even if one runs the same simulation with the same parameters (excluding the probability seed), they are not guaranteed to get the same result. By running the game 1,000 times for each combination of strategies (for a total of 256,000 simulations), the goal is that the data will approach the mean.

After the simulations have been run, the data will be collected in a series of spreadsheets that are included in the index. This information will be analyzed in terms of what the gross and net victories for each side are, the standard deviation for both sides per strategy, and from this, what the best and worst case scenarios, within a 95% confidence interval. From this it will be possible to derive what they most and least successful strategies are, and depending on the trends presented, attempt to extrapolate what are the elements of a successful simple strategy.
Background

A brief History of 40,000 Years:
Warhammer 40,000 (40k), one of the most popular two player miniature war games on the market, was created as a joke. No, Really.

There are many stories of how exactly 40k came to be, with many potential origin points, for the sake of consistency, we will use the version provided by Steve Tudor on Polyhedron Collider\(^3\). Created in 1987 by Games Workshop, Warhammer 40,000 was created as a joke about what would happen if the high fantasy setting of their other product Warhammer (often referred as Warhammer Fantasy Battle to avoid confusion) was set in space. The setting was a combination of tropes and clichés spanning a range from Tolkienesque fantasy (elves, orcs, and dwarfs), science fiction (with characters inspired by Judge Dredd, Alien, Starship Trooper, etc.), religion and mythology (Vulcan, Horus, Ezekiel) and even including real people (John-Claude Van Damme, Sylvester Stallone, Lionel Johnsen, and Genghis Khan, to name a few). All these sources are mixed together to form a black comedy dystopian future of the year 40,000. The rules for the game were just as eclectic, required multiple distinct types of dice, vast amounts of book keeping, and even a third player who would act as a referee between the other two.

However, over time both the setting and the rules for 40k became more developed, with the humor of the setting taking less focus compared to the drama, and the rules became more focused on balance and streamlining rather than narrative flavor.

The Game Now:
Warhammer 40,000 is a tabletop war game for two (or more) players that uses miniatures to represent soldiers in army-to-army skirmishes \[^4\]. It is a game of perfect information where both players know everything that is in the opponent’s army and there are no secrets. However, the game does use random chance to create some level of uncertainty in player actions, and to prevent players from being able to accurately predict too far in advance. Six-sided dice are used to determine the outcome of actions. At different points in the game, players are required to roll different numbers of dice and with threshold score for success also that vary. Despite being made in England, the game uses inches to measure, most likely due to 12 inches to a foot being easier to divvy up for a game involving six-sided dice.

Perhaps the most important part of the game is the models themselves. Standing on average, roughly 1 1/2” tall, these miniature figurines represent the characters and infantry that make up a player’s army. The models are estimated to be roughly 1:35 scale, though the game takes liberties, with units that are canonically different sizes having the same size model (i.e. in the universe of the game an imperial guardsman is roughly 2 meters tall whereas a Space Marine is around 3, but their models are of identical height.) These models represent a player’s army. Models come both unassembled and unpainted, and players spend hours building and painting the models, to the extent that some consider the process of creating one’s army more rewarding than the game itself.

These armies represent different factions from the universe of 40k. Armies vary in both appearance and playstyle: from the dinosaur-like hordes of the Tyranids who rely on strength in numbers to overwhelm their enemies, the graceful Eldar, a race of space elves who use hit a run tactics as well as eldritch sorcery, the egalitarian Tau, possessing advance technology and tactics that give them an advantage in ranged combat, and the Space Marines, transhuman super soldiers
who represent the greatest warriors of mankind and are an elite army focusing on quality over quantity.

While the game has been greatly refined over the years, the relative strengths of the players’ armies are still far from balanced. Various types of armies are imbalanced when compared to one another, especially when played with a “win at all costs” \cite{2} philosophy where players pick and design armies based around what is considered statistically more competitive rather than aesthetic pleasing (such as design or paint schemes) or based on backstory of the different races. The disparity between factions is most easily represented by a look at the most successful factions. In particular, the Eldar, Space Marine and Tau armies are picked significantly more frequently picked over other armies, with Eldar in particular taking 3 of the top 8 tournament positions in the Las Vegas Open 2016\cite{2}. This is also of note as all three armies are considered more oriented towards long range than melee combat, suggesting that there may be an advantage to range combat above close combat. This may lead to ranged strategies performing better in the simulation, though there are multiple other factors that could account for this.

The armies in the simulation are identical, so that differences in outcomes are a result of the strategies themselves rather than one type of army or unit being better than another, or better for a specific strategy.

**The Rules:**

Rules come from two sources, the main rulebook (colloquially the BRB or Big Red Book regardless of actual color) and the codices made for each faction in the game, with rules in the
codices having authority to override the BRB. *40k* can be described as a game where “the rules apply except for when they don’t.” For every given rule in the game, there is likely to be another rule that can override it or mitigate to some extent. For example, the morale mechanic does not affect models with the fearless rule and models with the relentless rule are unaffected by having moved before trying to shoot. As such, when describing the rules below it is not worth noting that there are exceptions, as nearly every rule has exceptions, and some even have exceptions to those.

While the game is traditionally played on either a 4’x4’ or 4’x6’ table, the game makes no strict requirement of this, nor the type of terrain or scenery in the game. Terrain has multiple effects, from slowing down or outright preventing moving through areas to providing cover for models being shot at.

**The Armies:**

The game uses asymmetric balance with players starting out with armies of different compositions that are balanced around the point cost of the army. Both armies are formed of units, that have associated point costs (as well as upgrades at additional points), with the trade-off being that the stronger a unit is, the more points it costs, meaning the player must balance the quality and quantity of the units in their army. In turn each unit is made of models that represent individual figures. Models have an array of nine characteristic scores that determine their odds of success at preforming certain actions. They also may have special rules that can be anything from bonuses to said scores to being an “independent character” allowing them to be a unit unto themselves or join another unit. While the game has requirements for what units can be considered a valid army, recent editions have made the requirements more varied: from taking
units that fulfill a specific role, to specifically named units, and even a mode called “unbound” where points are the only limiting factor.

**How to Play:**

The current edition of 40k has around 400 pages of rules[^4], so what follows is a simplification of the whole ruleset, focusing on what is necessary for understanding the creation and results of the simulation, rather than a complete overview.

Once both players have agreed to a point cost and assembled their army lists, the players chose what game type they are playing. The scope of possible games ranges from the main rulebook, an expansion book containing more game types, or one of the players’ own design. The selection may be either by agreement or randomly from a table provided in the books. These different types of games determine both how the players deploy their units at the beginning of the game and what the objectives of the game are, typically either involving holding a specific position on the board or killing enemy units. Once this is done, the players roll to see which player deploys his models first. This player will also take the first turn. Once the units are placed on the table, the player who is set to go second may attempt what is known as a “seize the initiative roll” and attempt to roll a 6 to go first, with no penalty for a failed roll. Once this is done the game may begin in earnest. Games generally last five to seven game turns depending on the game type, and each game turn consists of one player turn for each player. The turn itself is divided into four phases: the movement phase, the psychic phase, the shooting phase, and the assault phase. While the phases must be taken in order, the player is never required to take any action in any phase, except for resolving close combat in the assault phase.

**Movement Phase:**
The movement phase is perhaps the most straightforward. In this phase, models can move a distance determined by their unit type (e.g. infantry move 6”, bikes move 12”). Most of the decision making in this phase revolves around positioning oneself in range of what one wants to attack, away from what one does not, and in cover from the enemy’s shooting phase.

During the game, all models in a unit must remain within 2” of each other and cannot be within 1” of an enemy model, being unable to move if it would violate one of these rules. Additionally, units can only move as fast as the slowest model in the unit (i.e. a unit of models with 12” movement joined by a model with 6” movement can only move 6”), so players tend to prioritize putting models with the same movement speed in the same unit.

The Psychic Phase:

The newest phase to 40k, the psychic phase is also the most complex. Nearly identical to WHFB’s magic phase, the psychic phase is where the units in the game with psychic powers (those with the psyker special rule) attempt to cast their powers (also referred to as spells). These typically come in one of five forms: a direct attack (witchfire), a buff (blessing), debuff (malediction), summoning additional units (conjuration), and movement utility (generally also blessings), though there are other types.

Unlike other phases in the game, psychic powers are not guaranteed to work, and the player must attempt to cast them before they happen. This is done by first taking the number of psychic master levels the psykers have (indicated in their special rules) and then adding the result of 1d6 to it, the result being your known as your “warp charge pool” (the warp being the name for the place where psychic powers come from in the setting). Once this is done, the player declares what power they are going to use, who is going to cast it, how many warp charges they will
spend on it, subtracting that total from the warp charge pool, and then rolling that many dice to see if they cast the power. Once the dice are rolled, they compare how many of them are a 4+ and that is how many warp charges were successfully “harnessed”, if that number is equal to or greater than the number the power requires to cast, the power goes off. However, if the player rolled two or more 6s, the suffer what is known as “perils of the warp” and take a small penalty, discouraging players from putting too many psychic points in a single power.

Before the effects of the power go off, however, the opposing player may attempt to “Deny the Witch” to prevent the power from happening. This is done by rolling dice from a similarly created psychic pool to the one created for the other player, and for each 6 they roll, they subtract one successful warp charge from the other player’s successes, and if they get them below the required amount, the spell does not go off.

Unlike the other phases of the game, it is not only possible, but probable that a player does not have any psyker characters, and thus may skip this phase on his turn entirely.

**Shooting Phase:**

Once the psychic phase is complete the player moves on to the shooting phase. In this phase, the player may have any of his squads with projectile weapons fire them, as well as having other units do what is known as a “run action”.

If a unit wishes to shoot at a target, it must first check to see which, if any, of the ranged weapons are within range of the enemy and if they have “line of sight” (LoS) on the target. This is done by seeing if, from the attacking model’s perspective there is a direct uninterrupted line that can be made between the attacking model and the defender. After this, they roll to hit, rolling the number of dice determined by the weapon profile, in an attempt to get a “hit” by
rolling a static number determined by their ballistic skill (BS). Once they have hit, they then take the successes and roll to wound with those, comparing the strength characteristic of the weapon to the target’s toughness value. After that, the target may roll dice to make either make an armor save or invulnerable save (which are collectively known as a “saving throw”) depending on what armor and special rules the target has as well as the armor penetration of their weapon. If the target is at least 50% obscured by either terrain or other models not from either unit, the target gets what is known as a “cover save” as well, which they can use instead of their normal saving throw if it would be better. If the model fails its saving throw, then it is removed as a casualty.

After the saving throws have been made, another unit may shoot, repeating the process until there are no more units left that have not shot. Regardless of the outcome of the shooting attack, a unit cannot attack a different target in the shooting phase and the assault phase.

If a unit choses to run, they forfeit their shooting for the turn and instead move. They determine the distance to move by rolling one six-sided die and moving up to the number of inches shown on the die. This is known as “1d6 inches”. Units that ran in the shooting phase cannot assault in the same turn. Typically, this is done if the target is either attempting to get into cover or has no possible shooting targets. The “To Hit” table below shows how what a player needs to roll to successfully hit an opponent. For example, if they have a ballistic skill of 4, then they need a to roll a three or higher to hit. Once they have successfully hit their opponent, they consult the “To Wound table” to see what result they need to wound they enemy model. For example, if they have a weapon strength of 1, they need to roll a three or higher to wound.

<table>
<thead>
<tr>
<th>Ballistic Skill</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roll Required</td>
<td>6+</td>
<td>5+</td>
<td>4+</td>
<td>3+</td>
<td>2+</td>
</tr>
</tbody>
</table>
Assault Phase:
The assault phase can be divided into two parts, charging and close combat.

During the charging phase, units that are eligible may attempt to charge an enemy unit. Typically, units that have run or used a rule that moves them a large amount in a single turn cannot charge. Additionally, some types of weapons prevent the target from charging in the same turn they fired. If they are charging a unit, they roll 2d6 and compare that to the distance between the two units, and if it is greater, then they are considered locked in close combat.

Units that are locked in close combat cannot move or shoot. They must make close combat attacks against the unit they are locked in combat with. Unlike the other phases, both armies make attacks in the close combat phase. The order is determined by the initiative characteristic that all the models have. The number of attacks they make is determined by their attack characteristic, and they roll by comparing their own Weapon Skill value to their opponents and using the chart below. One they have rolled to hit, they roll the successes to wound and the opponent makes their saves as with the shooting phase, except that there are no cover saves. The units remain locked in melee until the other unit is wiped out.
<table>
<thead>
<tr>
<th>Weapon Skill compared to Opponent</th>
<th>Greater than</th>
<th>Equal to Or Less than</th>
<th>Less than half of opponent’s skill.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roll Required</td>
<td>3+</td>
<td>4+</td>
<td>5+</td>
</tr>
</tbody>
</table>

**The Turn, In Summary:**

Movement Phase:
1. Move Unit
2. Repeat

Psychic phase:
1. Generate Warp Charges
2. Choose Power and Target
3. Roll to Harness Warp Charges
4. Check to see if power succeeded, Resolve Perils of the Warp if necessary
5. Enemy Rolls Deny the Witch
6. Apply Effects
7. Repeat 2-6 until pool is depleted

Shooting phase:
1. Choose Target/Run Movement
2. Roll to Hit
3. Roll to Wound
4. Roll Saving Throws
5. Remove Models

Assault phase:
1. Declare charges (2d6”)
2. Check charge distance
   a. 2d6”
3. Combat phase
   i. Through initiative:
      1. Roll to Hit
      2. Roll to Wound
      3. Roll Saving Throws
      4. Remove Models

Tools Used:
To create the simulation to test the different strategies, I will be using Continuum Analytic’s Anaconda Python platform, and more specifically the Spyder module that comes with it. While there is nothing that prevents this program from being made in another language, like Java, or another python environment, Spyder comes with several particularly useful modules and libraries installed, included Numpy and Scipy, both of which are useful in data management.
An example of play:
The Warhammer 40,000 rules can be confusing for first-time learners; it is often best to watch a few games before attempting to play one. Since recording a few games would be impractical to demonstrate, instead, a theoretical demonstration of a game turn will be given. In this theoretical demonstration, Ben’s Inquisition forces will fight Will’s Ork Forces. However, since Ben is busy narrating, his good friend Aaron will be using his army. A Special thanks to Hayden Zahn for taking the pictures.

Dramatis Personæ

Will’s Ork army:
- 30 Ork boyz with choppas and sluggas (a beefy but unarmored close combat unit with poor accuracy)
- 12 Ork Stormboyz (a fast-moving close combat unit)
- 5 Ork Lootas – (A long-range Ork unit with powerful guns but low accuracy)

Aaron’s Inquisition Army:
- 6 Grey Knight terminators (a powerful and durable unit that excels in both close and long ranged fighting, but is incredibly expensive)
- 10 inquisitorial henchmen acolytes (a cheap cannon fodder ranged unit)

Aaron and Will are both busy, what with senior projects and all, so they decide to play a small game of 500 points, and the quickest game mode, meat grinder, where the objective is just to kill as many of the other guys as possible. Both players roll to see who will go first, and Will wins. After deploying their forces, the board looks like this.
[Will’s player turn]

Movement Phase: First, he moves his models in the movement phase. All his models move 6”, except for his Stormboyz who are able to move 12” and can ignore terrain because of their rokkit pack.
Psychic Phase: Because Will has no models with the psyker special rule, there is no need to do the psychic phase.

Shooting Phase: Will’s squad of Ork Lootas shoot at the inquisition henchmen first. Because their weapons type is heavy d3, two things happen. First, because they moved, they can only hit on 6’s. As his Orks only have ballistic skill 2, they would normally only hit on 5’s, so it’s no great loss. The second thing is that he must roll a d3 to determine how many hits they have, he rolls and gets 3 (which for a d3 roll means 2), 4(2), 5(3), 2(1), 1(1), so he gets 9 shots (2 + 2 + 3 + 1 + 1). He rolls to hit and rolls 4 6’s getting 4 hits -- lucky him. He then rolls to wound, and since his weapon’s strength is 7 and his target’s toughness is 3, he needs 2’s or higher to wound.
He gets 3 wounds. Now Aaron must roll his armor save to keep his models from dying. However, because the AP (armor penetration) on the Loota’s gun is 4 and the armor save of the guardsman is 5+, the gun prevents him from taking his armor save. But because they are partially obscured from the Orks, they receive a 5+ cover save, which they unfortunately all still fail. Thus, the henchmen unit takes 3 casualties.

After this, Will’s unit of Ork boyz fire their sluggas at the henchmen unit. They fire 30 times (needing 5’s to hit, since they are ballistics skill 2), and hit 15 times. Their guns are strength 4 so they need 3’s to wound the toughness 3 henchmen. They get 5 wounds, and thus Aaron must make 5 saves, at the guardsmen’s 5+. He makes 1, meaning that 4 henchmen left. The nob (the Ork equivalent of a sergeant) also fires his shoota, hitting twice, wounding once, and killing one more. Next, the Stormboyz fire their guns, which are the same as the Ork boyz’ weapons, and fire
10 times, hitting 3 times, wounding twice. Aaron rolls a 5 and 6 for his armor saves, meaning his unit takes no casualties.

Assault Phase: Now comes the assault phase, a specialty of the Orks. Will’s Stormboyz are right next to the henchmen, so he charges at them. Will Rolls 2d6 and gets a 3 and a 5, so he has a total charge range of 7” -- more than enough to get them into combat. As much as he wants to charge the Grey Knight terminators with his Ork boyz, he already shot the henchmen this turn, and can’t charge a different unit than he shot at.

After all the charges have been resolved, the close combat begins. Despite having charged, the Stormboyz have a lower initiative and thus attack the 2 henchmen. The Henchmen attack, their 2 total attacks at weapons skill 3 requiring a 5+ to hit, due to the higher weapon skill of the Orks. One succeeds. Their strength 3 compared to the Orks toughness 4 means they also need a 5+ to
wound, which they somehow make. After this, dealing a single wound, the Ork unsurprisingly fails its 6+ armor save, and thus dies. In retaliation though, the Orks have 27 attacks total, 18 of which hit on their 3+ to hit, 10 of which wound on their 4+ to wound, and the guard fail 5 of their saves, thus they are wiped out.

[Aaron’s player turn]

Movement Phase: Ignoring the Stormboyz, Aaron decides to go after the massive boyz squad. His terminators move 6” towards them.

Psychic Phase: Unlike the Orks, the Grey Knights are at no shortage of psykers. Aaron rolls a 2 for Winds of the Warp, meaning that both players start with 2 dice in their pool, Aaron adds to 2 more for having 2 units with psyker level 1. Aaron attempts to cast the power Hammerhand on his unit of terminators, which would give them +2 strength, and requires 1 charge to cast. He
rolls and gets a 1, a 2, a 4, and a 6, meaning he has 2 warp charges. Will throws his 2 dice at his Deny the Witch roll, and gets a single 6, meaning that he subtracts 1 from Aaron’s warp charge count. This leaves Aaron with 1 warp charge, so the power is still cast. Thus the Grey Knights will have +2 strength until his next psychic phase. Since Aaron has no more psychic dice in his pool, the psychic phase is over.

Shooting Phase: The Grey Knights have 6 storm bolters, a ranged weapon with an impressive rate of fire, including the inquisitor joining them. They have a ballistic skill of 4, meaning they require a 3+ to hit. Out of their 12 shots, they hit 9 times, and since their weapon’s strength is equal to the toughness of the Orks, they wound on a 4+. Which they do, 5 times, and since the weapon’s AP is better than the armor saves of the Orks, the Orks don’t get an armor save. Without anything between the Grey Knights and the Orks, no cover save either. Thus 5 wounds become 5 dead Orks.
Assault Phase: With almost no distance between them, the Grey Knights make the charge on a 2 and 3, and thus combat begins. The Grey Knights have a higher initiative, and thus go first. They have 3 attacks each, 4 on the inquisitor, and because they have weapon skill 4, the same as the Orks, they hit on a 4+. They hit 9 times. Because of their Hammerhand ability, they have strength 6, so they wound on a 2+. They wound 7 times and much like with the shooting, their melee attacks go through armor and kill 7 Orks. Now the Orks get to retaliate, the 17 Orks have 3 attacks each, for 51 attacks total. Having the same weapon skill as the Grey Knights, they hit on 4+, and 20 attacks hit. Since they have strength 3 compared to the Grey Knight’s toughness 4, they require 5+ to wound, and only 6 of these do. The Grey Knights now must make 6 saves,
but with their mighty terminator armor they only need a 2+ to succeed, which they do for all but one of the attacks, and thus a single Grey Knight brother falls.

While it would be possible to continue the game, it wound swiftly devolve into a massive quagmire that would neither be interesting to look at or to describe. So it would be best to move back to the program.
Strategies:

The goal of this program will be to compare how the different basic strategies work against one another. The strategies will be kept as simple as possible, requiring no predictive calculations other than the distance between units; this will be done to try to keep the testing as straightforward as possible. Additionally, the strategies will have all the units acting independently of each other.

Essentially, all the strategies that will be tested can be broken down into two components, do what, and to whom? In terms of action, the four types of action are: to advance and shoot the chosen unit; to attempt to charge it in close combat; to randomly decide between those two strategies; or to decide between the two based on what the attacking unit is assumed to do best by default (i.e. units that favor shooting over close combat will shoot and vice versa). As to the target of the action, the attacking unit will either chose: the closest unit to itself; the unit with the highest point cost; the closest unit only if it is within a certain range of it; or a random unit from the enemy faction. These two parts of the strategy will be mixed together to create 16 different mentalities for the units to have as displayed in the table below. Units will decide which target they will attempt to act upon at the beginning of their player turn.

<table>
<thead>
<tr>
<th>Unit Strategy Table</th>
<th>Shoot</th>
<th>Charge</th>
<th>Default</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closest</td>
<td>Shoot the Closest Unit</td>
<td>Charge Closest Unit</td>
<td>Do Default Action to Closest Unit</td>
<td>Do a Random Action to Closest Unit</td>
</tr>
<tr>
<td>Most Valuable</td>
<td>Shoot the Most Valuable Unit</td>
<td>Charge Most Valuable Unit</td>
<td>Do Default Action to Most Valuable Unit</td>
<td>Do a Random Action to Closest Unit</td>
</tr>
<tr>
<td>Closest in Range</td>
<td>Shoot the Closest Unit in Range</td>
<td>Charge the Closest Unit in Range</td>
<td>Do Default Action the Closest Unit in Range</td>
<td>Do a Random Action to Closest Unit in Range</td>
</tr>
<tr>
<td>Random</td>
<td>Shoot a Random Unit</td>
<td>Charge a Random Unit</td>
<td>Do Default Action to Random Unit</td>
<td>Do a Random Action to a Random Unit</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>----------------------</td>
<td>----------------------------------</td>
<td>------------------------------------</td>
</tr>
</tbody>
</table>

**Representation:**

To represent these strategies, a theoretical version of the game was played with two identical Space Marine armies starting in mirroring positions, as demonstrated below in Figure 6. Each army will consist of two tactical squads (the generic ranged infantry), one assault squad (the close combat squad), and one Devastator squad (the specialized ranged squad that takes penalties if they move). The game will play for the average six turns, and at the end of which, the player with the most surviving models will be considered the victor. Dice will be simulated by Python’s built-in random number generator. After the game, the result will be recorded as either a win loss or a draw, which the program will keep tally of. After the simulation has run 1,000 times, it will record an array in the form of [wins, losses, draws] to an Excel sheet, using the csv writer built into Python. The program will repeat the process until all 16 strategies have gone up against all 16 other strategies, including the one that it is using itself.
To represent the choosing of targets, the program will attempt at the beginning of each phase to choose the target that satisfies their criteria. For *Closest* it will simply take the \((x, y)\) coordinates for the unit deciding, and compare it to all the living units on the opposing side, and then take the lowest value. *In Range* will find the closest unit within a set range, in this case 24”, and only act if a unit is that distance. In the case of the Devastator squad, will not attempt to advance under any circumstances as they take a penalty for moving and shooting in the same turn and their weapons have a range of 36, meaning they will always be in firing range if they are in 24”. For the *Highest Value* calculation, the program merely checks each living unit’s point cost and goes for the highest, which in this case will be the Devastator squad by default, followed by the tactical squad, and the assault squad. *Random* will put all the living squads from the opposing side in an array and then randomly select one using Python’s random module’s choice function.

Once the target has been decided upon, the unit will move 6” towards it, or 12” in the case of the Assault squad. This is done by using the formula \((a \times (1-d)) + (d \times b)\) where \(a\) is the target unit’s
location, \( b \) is the acting unit’s location, and \( d \) is the value \((ab – t)/(ab)\) where \( ab \) is the distance between the units and \( t \) is the amount the unit moves.

After moving if necessary, most units will either attempt to shoot or charge depending on their behavior. This is due to the way that the weapons the units have function. Both the Tactical Marine and the Devastator Marine’s weapons prevent the unit from charging the same turn that they fired, meaning that they must only use one or the other depending on what the player prioritizes. The Assault Marine is the only unit that can, and does, fire its weapon before charging, though due to the weapon’s close range and low rate of fire, it is much less noteworthy than the other unit’s ranged weapons.
Simulations:

For the sake of clarity, from this point onwards, player 1 will refer to the player who goes first and controls Army 1, and player 2 is the player who goes second and controls Army 2.

Each game ran for six turns, after which both sides totaled the number of remaining models on their side, the two were compared, and whoever had the greater remaining models won, with a draw in the case they were equal. After running the simulation 1,000 times for each, the results were recorded in a spreadsheet, with each cell formatted in the form of [wins, losses, draws], for army one, which is the one that has the first turn. If one wants to see how a given strategy performs for having the second turn, simply swap the wins and losses values.

After this, the results were given a value, with victories for army One given a 1, a draw given 0, and a loss for player one given a -1. From this the net victories can be displayed on another table [see table 1 in index for these results], displayed below, with a number between -1000 and 1000, with 1000 meaning that in every simulation the first army won, a -1000 meaning that the second army always won, and a number between those representing how many more times the first army won than the second (or vice versa for negative values). See Table 2 in the indices for the net victories for each strategy.

If one wishes to get the mathematical average of the cells in the first table, simply take the net table and divide by 1,000, or more simply, just put a decimal point in front of the hundreds value, i.e. the first cell with a net score of 397 has an average of .397.
Determining standard deviation requires a slight change in terms of interpreting the results. Rather than the normal zero-sum game, a draw will be considered as a loss for both players. Doing this allows us to have a binomial win/loss, and thus one can determine both players’ probability of success as simply the number of games they won out of 1000. Thus, the formula for determining a player’s standard deviation for any matchup is

$$\text{std} = \sqrt{n(p)(1-p)}$$

where $n$ is the number of games simulated, in this case 1000 and $p$ is the probability of success (in this case wins/1000). See Table 3 for the standard deviation of player 1’s strategies and Table 6 for player 2’s strategies.

After doing this, one can either add or subtract the standard deviations to the results to get either the best case or worse case averages for the two players respectively. More accurately, the standard deviation multiplied by 1.96 will be added or subtracted so that the results are within a 95% confidence interval, and thus represent the clear majority of likely outcomes. Unfortunately, due to floating point imprecision and the fact that the draws are counted as losses, some of the worst-case scenarios are listed as negative, though this should be taken as a rounding error rather than a sign that they have a negative number of wins. The tables for the best and worst cases for player 1 are Tables 4 and 5 respectively, while the best case for player 2 is Table 7 and the worst case scenario is Table 8.
Conclusions:

For the sake of simplicity, we will be using the values from the different strategies going first.

Based on the information above, here are the, on average, best and worst strategies compared to one another, with strategies in bold losing at least 99% of the time to the given strategy in the best case or beating the given strategy in the worst case.

<table>
<thead>
<tr>
<th></th>
<th>Best Against</th>
<th>Worst Against</th>
<th>Total Victories (Going First)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closest Shoot</td>
<td><strong>Valuable Charge, Closest Charge</strong></td>
<td>Random Default</td>
<td>13428</td>
</tr>
<tr>
<td>Closest Charge</td>
<td>Random Charge</td>
<td><strong>Closest Shoot, Closest Default, In Range Shoot, In Range Charge, In Range Default, In Range Random</strong></td>
<td>1968</td>
</tr>
<tr>
<td>Closest Default</td>
<td><strong>Valuable Charge, Random Charge</strong></td>
<td>Closest Default (Still Wins)</td>
<td>12921</td>
</tr>
<tr>
<td>Closest Random</td>
<td>Random Charge</td>
<td>In Range Default</td>
<td>6753</td>
</tr>
<tr>
<td>Valuable Shoot</td>
<td><strong>Valuable Charge, Random Charge</strong></td>
<td>In Range Default</td>
<td>6366</td>
</tr>
<tr>
<td>Valuable Charge</td>
<td>Valuable Charge (still loses)</td>
<td><strong>Closest Shoot, Closest Default, Valuable Shoot, Valuable Default, in Range Shoot, in Range Charge, in Range Default, in</strong></td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Range Random, Random Shoot, Random Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------</td>
<td>----------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Valuable Default</td>
<td>Valuable Charge, Random Charge</td>
<td>In Range Default</td>
<td>7057</td>
</tr>
<tr>
<td>Valuable Random</td>
<td>Random Charge</td>
<td>In Range Default</td>
<td>3005</td>
</tr>
<tr>
<td>In Range Shoot</td>
<td>Valuable Charge, Closest Charge,</td>
<td>Closest Shoot</td>
<td>7395</td>
</tr>
<tr>
<td>In Range Charge</td>
<td>Valuable Charge</td>
<td>Closest Default, In Range Charge</td>
<td>6861</td>
</tr>
<tr>
<td>In Range Default</td>
<td>Closest Charge, Valuable Charge, Valuable Default, Valuable Random, Random Charge</td>
<td>In Range Default</td>
<td>13458</td>
</tr>
<tr>
<td>In Range Random</td>
<td>Valuable Charge</td>
<td>In Range Shoot</td>
<td>7674</td>
</tr>
<tr>
<td>Random Shoot</td>
<td>Valuable Charge, Random Charge</td>
<td>In Range Default</td>
<td>10407</td>
</tr>
<tr>
<td>Random Charge</td>
<td>Valuable Charge</td>
<td>Random Charge, Random Default, Closest Random, Valuable Shoot, Valuable Default, In Range Default, Random Shoot, Random Default</td>
<td>809</td>
</tr>
<tr>
<td>Random Default</td>
<td>Valuable Charge, Random Charge</td>
<td>In Range Default</td>
<td>10604</td>
</tr>
<tr>
<td>Random Random</td>
<td>Random Charge</td>
<td>In Range Default</td>
<td>5684</td>
</tr>
</tbody>
</table>
Based on the data, below are the five best and worst simulated strategies.

5 Best Strategies: going from best to worst

1. In Range Default
2. Closest Shoot
3. Closest Default
4. Random Default
5. Random Shoot

It is worth noting that there is only a difference of 30 victories between the first and second place strategies, and using scipy’s individual t-test, a p value of .973 (rounding to the third decimal) and thus the two ranks are not significantly different. The third place, is more different from the second with a p value of .583 (and .539 from the first), which is still similar, to the point of random change being a huge factor. Between the third and fourth the p value drops to .057, but between fourth and fifth the p value is .889, so they are significantly more interchangeable. Because of this the top 5 would more accurately be the group top 3 followed by the 4 and 5.

5 Worst Strategies: going from worst to best

1. Valuable Charge
2. Random Charge
3. Closest Charge
4. Valuable Random
5. Random Random

Much like the two best, the two worst are significantly similar, with a p value of .609. Additionally, between the second and third values the p value is .304. Between the first and third,
the p value is .168, which is debatably significant depending on where one draws the line.

Between the third and fourth value, the p value is .453. Only the p value between the fourth and fifth strategies have a p value less than .1, in this case .083. Which means, yes, the four other strategies on the list are significantly worse than doing a completely random action every time.

Among the 5 best and worst there is an obvious dichotomy, with the shooting-based strategies performing better than swapping between either shooting or charging, as is the case with random, and finally close combat performing the very worse.

From the data, one thing becomes clear, first strike potential is one of the biggest deciding factors in almost all the scenarios. Valuable Charge and Random Charge are two of the worst strategies, which comes from the fact that they have the most potential for having turns where they attack nothing, especially in the case of the valuable charge, as the most valuable unit is the Devastator squad in the back of each team, meaning that they would attempt to rush the farthest enemy, absolutely ignoring anything in the way and promptly getting mowed down or charged before they could reach their target. While this may seem like a huge strategic advantage towards one player who can go first, in the actual table top there are many factors that muddy who can strike first. For example, some armies being able to delay bringing in units so that they arrive on their own turn and within striking range of the opponent, at the risk of possibly arriving at the wrong time or place. Additionally, terrain that provides cover and blocks line of sight between units often means that players who are more successful in correctly maneuvering their units get the first attack, regardless of which player went first.

In terms of strategy, even in terms of simple automated strategies, some perform significantly better than others to a degree that cannot be attributed to random chance. Even in
their worst case scenarios, the top 5 strategies performing significantly better than all the other strategies’ best case scenario.

However, even the best average case for the most successful strategy is only winning 86% of the time. While that is nothing to scoff at, it is far from an undefeatable strategy, especially against the other top 5 strategies.

**Future Research:**
Looking forward, there are multiple areas the project can expand on.

Armies could be composed of more types of units, and from the other factions in 40k. The simulation could also be used to test different unit sizes, with the current iteration using five man units, though the unit can be increased in size to ten. Additionally, changing the way units deploy will most likely affect the outcome of the game significantly.

The program could also be expanded in terms of the complexity of strategies used, with different units using more varied strategies, using strategies that require more calculations before deciding, and strategies that have units refer to the location and actions of other units within the army, rather than all acting independently.

The simulation could also be expanded upon by attempting to add terrain, though this would require a far larger reworking of the core program than many of the other changes.

Finally, Games Workshop has announced that *Warhammer 40,000* 8th edition will be released sometime during the summer of 2017, and with it will come large changes to the core mechanics to the game, which will require an updating of the simulation if it is to remain up to date.
**Terminology:**

X+: Indicating what number X must be rolled over to succeed, i.e. a 3+ succeeds on a roll of 3, 4, 5, or 6.

Xd6: Indicating how many six-sided dice (d6) must be rolled, i.e. 3d6 is 3 dice.

d3: a d6 divided by two, rounding up, so 1-2 is a 1, 3-4 is a 2, and 5-6 is a 3.

Army: The collective of all forces under a player’s control.

Army List: The in-depth list of all units and unit upgrades in a player’s army. Both players have access to each other’s lists at all times.

Unit: A collection of models that make up part of an army. These models generally have to all do the same action and must stay within 2” of each other. Forces are purchased at the unit level. Occasionally, also referred to as a squad.

Model: The individual figurines that make up a unit, the most basic level of playing piece. Models have character scores as well as special rules that define how they can act.

Player Turn: The time when a player can act. The turn is divided into 4 phases: the movement phase, the psychic phase, the shooting phase, and the assault phase.

Game Turn: The rounds of a game, consisting of one of each player’s player turns.

40k: *Warhammer 40,000.*

Warhammer Fantasy, WHFB: *Warhammer Fantasy Battle*, the game from which 40k drew inspiration.

Age of Sigmar, AoS: *Warhammer: Age of Sigmar*, the sequel game to WHFB, known for its massive simplification of the rules of the game, going from a 500 page rulebook to a 4 page one.

BRB: “Big Red Book”, the core rulebook for the game, so named because of *Warhammer Fantasy*’s 6-8th edition core books being red.

Buff/Debuff: to increase/decrease the potency of something, a term borrowed from video games.

Saving Throw: Either an armor save or invulnerable save

Threat Range: the maximum distance a unit can attempt to deal damage to another unit from. For example, a Space Marine tactical squad can move 6” and then fire their guns 24” giving them a threat range of 30”.

WAAC: *win at all costs*, the philosophy that winning is the most important goal of the game, compared to either fun or good sportsmanship. The term is a pejorative, and often hard to quantify as there is no metric for measuring how competitive someone’s intent is.

Player 1: The player who goes first.

Player 2: The player who goes second.
Bibliography:


Indices

*Tables:*

Due to the large size of the tables (17 x 17), they would not fit into a normally formatted page while remaining readable. As such, they have been included in the next section, which has been formatted in landscape rather than portrait layout. Please excuse the fluctuation in size and placement.
<table>
<thead>
<tr>
<th>Valuable Random</th>
<th>Valuable Default</th>
<th>Valuable Charge</th>
<th>Valuable Shoot</th>
<th>Closest Random</th>
<th>Closest Default</th>
<th>Closest Charge</th>
<th>Closest Shoot</th>
</tr>
</thead>
<tbody>
<tr>
<td>[43, 953, 7]</td>
<td>[585, 781, 5]</td>
<td>[165, 732, 4]</td>
<td>[73, 924, 3]</td>
<td>[642, 248, 1]</td>
<td>[34, 966, 0]</td>
<td>[1165, 781, 54]</td>
<td>[73, 924, 3]</td>
</tr>
<tr>
<td>[982, 1, 2]</td>
<td>[992, 1, 2]</td>
<td>[339, 647, 14]</td>
<td>[739, 182, 28]</td>
<td>[525, 406, 15]</td>
<td>[301, 676, 11]</td>
<td>[625, 676, 11]</td>
<td>[301, 676, 11]</td>
</tr>
<tr>
<td>[391, 597, 12]</td>
<td>[790, 182, 28]</td>
<td>[1185, 781, 54]</td>
<td>[789, 192, 10]</td>
<td>[526, 406, 15]</td>
<td>[301, 676, 11]</td>
<td>[625, 676, 11]</td>
<td>[301, 676, 11]</td>
</tr>
<tr>
<td>[301, 676, 11]</td>
<td>[526, 406, 15]</td>
<td>[301, 676, 11]</td>
<td>[789, 192, 10]</td>
<td>[526, 406, 15]</td>
<td>[301, 676, 11]</td>
<td>[625, 676, 11]</td>
<td>[301, 676, 11]</td>
</tr>
<tr>
<td>[200, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
<td>[0, 1000, 0]</td>
</tr>
</tbody>
</table>

**Table 1:** shows the win/loss/draw for all strategies compared to each other, with the rows representing player 1’s strategy and the columns representing player 2’s.
<table>
<thead>
<tr>
<th>Closest Shoot</th>
<th>Closest Charge</th>
<th>Closest Default</th>
<th>Closest Random</th>
<th>Valuable Shoot</th>
<th>Valuable Charge</th>
<th>Valuable Default</th>
<th>Valuable Random</th>
<th>In Range Shoot</th>
<th>In Range Charge</th>
<th>In Range Default</th>
<th>In Range Random</th>
<th>Random Shoot</th>
<th>Random Charge</th>
<th>Random Default</th>
<th>Random Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>397</td>
<td>982</td>
<td>396</td>
<td>798</td>
<td>499</td>
<td>1000</td>
<td>470</td>
<td>889</td>
<td>944</td>
<td>885</td>
<td>424</td>
<td>742</td>
<td>419</td>
<td>1000</td>
<td>306</td>
<td>780</td>
</tr>
<tr>
<td>272</td>
<td>965</td>
<td>212</td>
<td>720</td>
<td>423</td>
<td>1000</td>
<td>412</td>
<td>825</td>
<td>950</td>
<td>874</td>
<td>188</td>
<td>739</td>
<td>281</td>
<td>1000</td>
<td>254</td>
<td>808</td>
</tr>
<tr>
<td>-1000</td>
<td>-981</td>
<td>-1000</td>
<td>-938</td>
<td>-1000</td>
<td>-62</td>
<td>-999</td>
<td>-673</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
<td>-1000</td>
</tr>
<tr>
<td>-968</td>
<td>1000</td>
<td>-957</td>
<td>183</td>
<td>18</td>
<td>544</td>
<td>1000</td>
<td>524</td>
<td>912</td>
<td>-980</td>
<td>-946</td>
<td>-895</td>
<td>-942</td>
<td>33</td>
<td>262</td>
<td>-147</td>
</tr>
<tr>
<td>-945</td>
<td>822</td>
<td>-976</td>
<td>18</td>
<td>544</td>
<td>1000</td>
<td>524</td>
<td>912</td>
<td>-980</td>
<td>-946</td>
<td>-895</td>
<td>-942</td>
<td>33</td>
<td>262</td>
<td>-147</td>
<td>717</td>
</tr>
<tr>
<td>292</td>
<td>996</td>
<td>177</td>
<td>874</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>855</td>
<td>432</td>
<td>32</td>
<td>234</td>
<td>664</td>
<td>1000</td>
<td>632</td>
<td>950</td>
</tr>
<tr>
<td>-896</td>
<td>954</td>
<td>-897</td>
<td>219</td>
<td>593</td>
<td>1000</td>
<td>576</td>
<td>940</td>
<td>-986</td>
<td>-952</td>
<td>-624</td>
<td>-939</td>
<td>240</td>
<td>67</td>
<td>63</td>
<td>774</td>
</tr>
<tr>
<td>-114</td>
<td>895</td>
<td>-173</td>
<td>412</td>
<td>443</td>
<td>1000</td>
<td>384</td>
<td>849</td>
<td>613</td>
<td>10</td>
<td>-661</td>
<td>-235</td>
<td>84</td>
<td>1000</td>
<td>-114</td>
<td>593</td>
</tr>
<tr>
<td>-1000</td>
<td>-857</td>
<td>-1000</td>
<td>-993</td>
<td>-1000</td>
<td>106</td>
<td>-995</td>
<td>-827</td>
<td>0</td>
<td>-314</td>
<td>-1000</td>
<td>-93</td>
<td>-998</td>
<td>0</td>
<td>-1000</td>
<td>-935</td>
</tr>
<tr>
<td>-128</td>
<td>872</td>
<td>-230</td>
<td>409</td>
<td>526</td>
<td>1000</td>
<td>468</td>
<td>821</td>
<td>591</td>
<td>20</td>
<td>-670</td>
<td>-118</td>
<td>224</td>
<td>998</td>
<td>-38</td>
<td>630</td>
</tr>
</tbody>
</table>

Table 2: The net victories for each strategy.
<table>
<thead>
<tr>
<th>Closest Shoot</th>
<th>Closest Charge</th>
<th>Closest Default</th>
<th>Closest Random</th>
<th>Valuable Shoot</th>
<th>Valuable Charge</th>
<th>Valuable Default</th>
<th>Valuable Random</th>
<th>In Range Shoot</th>
<th>In Range Charge</th>
<th>In Range Default</th>
<th>In Range Random</th>
<th>Random Shoot</th>
<th>Random Charge</th>
<th>Random Default</th>
<th>Random Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.068457</td>
<td>0.9995</td>
<td>6.6245</td>
<td>4.534203</td>
<td>15.62277</td>
<td>4.638534</td>
<td>12.04259</td>
<td>0</td>
<td>1.729451</td>
<td>0</td>
<td>0</td>
<td>4.427189</td>
<td>10.45825</td>
<td>3.715373</td>
<td>11.07028</td>
</tr>
<tr>
<td>0</td>
<td>1.729451</td>
<td>0</td>
<td>4.427189</td>
<td>0</td>
<td>7.795832</td>
<td>0</td>
<td>6.826346</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14.82309</td>
<td>0</td>
</tr>
<tr>
<td>3.715373</td>
<td>0</td>
<td>4.317291</td>
<td>15.70831</td>
<td>15.72565</td>
<td>0</td>
<td>15.61278</td>
<td>10.20176</td>
<td>0</td>
<td>4.317291</td>
<td>8.014924</td>
<td>4.638534</td>
<td>11.4142</td>
<td>0</td>
<td>10.77237</td>
<td>13.90525</td>
</tr>
<tr>
<td>5.032296</td>
<td>10.66954</td>
<td>3.146427</td>
<td>15.79</td>
<td>13.71127</td>
<td>0</td>
<td>13.95618</td>
<td>7.851369</td>
<td>0</td>
<td>0</td>
<td>6.55532</td>
<td>0</td>
<td>15.8111</td>
<td>13.92232</td>
<td>15.57575</td>
<td>11.44443</td>
</tr>
<tr>
<td>15.16967</td>
<td>1.412799</td>
<td>15.59731</td>
<td>7.90639</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.676174</td>
<td>14.39274</td>
<td>15.79997</td>
<td>15.57575</td>
<td>11.93382</td>
<td>0</td>
<td>12.3554</td>
<td>5.216896</td>
</tr>
<tr>
<td>7.021111</td>
<td>5.811626</td>
<td>6.826346</td>
<td>15.57575</td>
<td>12.96935</td>
<td>0</td>
<td>13.30789</td>
<td>5.89101</td>
<td>0</td>
<td>0</td>
<td>11.82269</td>
<td>0.9995</td>
<td>15.43813</td>
<td>14.76343</td>
<td>15.79161</td>
<td>10.31305</td>
</tr>
<tr>
<td>0</td>
<td>3.443254</td>
<td>0</td>
<td>1.412799</td>
<td>0</td>
<td>15.74722</td>
<td>1.412799</td>
<td>6.956939</td>
<td>0</td>
<td>0.9995</td>
<td>0</td>
<td>13.87083</td>
<td>0.9995</td>
<td>0</td>
<td>0</td>
<td>4.937104</td>
</tr>
</tbody>
</table>

Table 3: the standard deviation for player 1.
<table>
<thead>
<tr>
<th>Closest Shoot</th>
<th>Closest Charge</th>
<th>Closest Default</th>
<th>Closest Random</th>
<th>Valuable Shoot</th>
<th>Valuable Charge</th>
<th>Valuable Default</th>
<th>Valuable Random</th>
<th>In Range Shoot</th>
<th>In Range Charge</th>
<th>In Range Default</th>
<th>In Range Random</th>
<th>Random Shoot</th>
<th>Random Charge</th>
<th>Random Default</th>
<th>Random Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closest Shoot</td>
<td>725.4835</td>
<td>996.167</td>
<td>723.5363</td>
<td>914.0004</td>
<td>773.9449</td>
<td>1000</td>
<td>760.4198</td>
<td>956.4876</td>
<td>979.7423</td>
<td>955.6042</td>
<td>737.1531</td>
<td>888.0471</td>
<td>733.2658</td>
<td>1000</td>
<td>679.5629</td>
</tr>
<tr>
<td>Closest Charge</td>
<td>0</td>
<td>85.81418</td>
<td>2.95902</td>
<td>58.98402</td>
<td>29.88704</td>
<td>607.6206</td>
<td>31.09153</td>
<td>199.6035</td>
<td>0</td>
<td>6.389724</td>
<td>0</td>
<td>0</td>
<td>28.67729</td>
<td>895.4982</td>
<td>21.28213</td>
</tr>
<tr>
<td>Closest Default</td>
<td>662.8738</td>
<td>989.4619</td>
<td>632.3386</td>
<td>876.8234</td>
<td>738.1245</td>
<td>1000</td>
<td>730.3487</td>
<td>929.5588</td>
<td>981.4008</td>
<td>950.2798</td>
<td>621.4727</td>
<td>889.9123</td>
<td>667.7866</td>
<td>1000</td>
<td>652.0536</td>
</tr>
<tr>
<td>Valuable Shoot</td>
<td>189.0618</td>
<td>569.8959</td>
<td>171.0706</td>
<td>407.038</td>
<td>215.3639</td>
<td>1000</td>
<td>248.8</td>
<td>711.8401</td>
<td>511.9679</td>
<td>311.9494</td>
<td>37.04603</td>
<td>268.546</td>
<td>249.8411</td>
<td>999.9122</td>
<td>243.591</td>
</tr>
<tr>
<td>Valuable Charge</td>
<td>0</td>
<td>6.389724</td>
<td>0</td>
<td>28.67729</td>
<td>0</td>
<td>80.27983</td>
<td>0</td>
<td>62.37964</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>355.0533</td>
<td>0</td>
<td>48.69957</td>
</tr>
<tr>
<td>Valuable Default</td>
<td>204.8637</td>
<td>591.7588</td>
<td>165.7608</td>
<td>443.5176</td>
<td>368.3398</td>
<td>997.5215</td>
<td>421.245</td>
<td>815.2452</td>
<td>555.9516</td>
<td>341.7413</td>
<td>40.57311</td>
<td>277.8741</td>
<td>321.2098</td>
<td>999.9122</td>
<td>269.5832</td>
</tr>
<tr>
<td>Valuable Random</td>
<td>57.84884</td>
<td>241.5059</td>
<td>45.2327</td>
<td>188.006</td>
<td>89.12343</td>
<td>671.7143</td>
<td>94.62145</td>
<td>408.0536</td>
<td>100.0994</td>
<td>72.48757</td>
<td>9.371723</td>
<td>55.57321</td>
<td>78.059</td>
<td>839.0669</td>
<td>60.11751</td>
</tr>
<tr>
<td>In Range Shoot</td>
<td>21.28213</td>
<td>1000</td>
<td>27.46189</td>
<td>587.7883</td>
<td>478.8223</td>
<td>1000</td>
<td>451.6011</td>
<td>901.9955</td>
<td>0</td>
<td>989.4619</td>
<td>84.70925</td>
<td>987.0915</td>
<td>176.3718</td>
<td>0</td>
<td>155.1138</td>
</tr>
<tr>
<td>In Range Charge</td>
<td>35.8633</td>
<td>889.9123</td>
<td>16.167</td>
<td>504.9484</td>
<td>775.8741</td>
<td>1000</td>
<td>762.3541</td>
<td>949.3887</td>
<td>0</td>
<td>0</td>
<td>57.84884</td>
<td>0</td>
<td>527.9898</td>
<td>290.2877</td>
<td>444.5285</td>
</tr>
<tr>
<td>In Range Default</td>
<td>670.7326</td>
<td>1000.769</td>
<td>612.5707</td>
<td>948.4965</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>935.0053</td>
<td>735.2098</td>
<td>511.9679</td>
<td>616.5285</td>
<td>851.3903</td>
<td>1000</td>
<td>836.2166</td>
<td>982.2251</td>
</tr>
<tr>
<td>In Range Random</td>
<td>65.76138</td>
<td>976.3908</td>
<td>62.37964</td>
<td>616.5285</td>
<td>811.4199</td>
<td>1000</td>
<td>796.0835</td>
<td>975.5464</td>
<td>0</td>
<td>0</td>
<td>191.1725</td>
<td>2.95902</td>
<td>638.2587</td>
<td>349.9363</td>
<td>555.9516</td>
</tr>
<tr>
<td>Random Shoot</td>
<td>468.7511</td>
<td>954.7196</td>
<td>438.4612</td>
<td>723.5363</td>
<td>744.9196</td>
<td>1000</td>
<td>716.7162</td>
<td>937.7186</td>
<td>825.7457</td>
<td>522.9864</td>
<td>190.1173</td>
<td>407.0386</td>
<td>565.9143</td>
<td>1000</td>
<td>467.7433</td>
</tr>
<tr>
<td>Random Charge</td>
<td>0</td>
<td>18.74878</td>
<td>0</td>
<td>4.769085</td>
<td>0</td>
<td>485.8646</td>
<td>4.769085</td>
<td>64.6356</td>
<td>0</td>
<td>2.95902</td>
<td>0</td>
<td>287.1868</td>
<td>2.95902</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Random Default</td>
<td>458.6673</td>
<td>944.0212</td>
<td>408.0536</td>
<td>726.4569</td>
<td>783.5832</td>
<td>1000</td>
<td>759.4523</td>
<td>925.0012</td>
<td>812.3766</td>
<td>528.9901</td>
<td>185.8935</td>
<td>466.7354</td>
<td>639.245</td>
<td>1000</td>
<td>503.9451</td>
</tr>
<tr>
<td>Random Random</td>
<td>131.5466</td>
<td>649.0998</td>
<td>119.6765</td>
<td>363.2325</td>
<td>382.6207</td>
<td>935.0053</td>
<td>377.5236</td>
<td>664.8394</td>
<td>231.0669</td>
<td>137.9955</td>
<td>40.57311</td>
<td>95.71858</td>
<td>268.546</td>
<td>970.4326</td>
<td>204.8637</td>
</tr>
</tbody>
</table>

Table 4: The best average scenario for player 1.
| Closest Shoot | Closest Charge | Closest Default | Closest Random | Valuable Shoot | Valuable Default | Valuable Random | In Range Shoot | In Range Default | In Range Random | Rando Shoot | Rando Charge | Valuable Charge | Valuable Default | Valuable Random | In Range Charge | In Range Default | In Range Random | Random Charge | Random Default | Random Random |
|---------------|---------------|----------------|----------------|---------------|----------------|----------------|---------------|----------------|----------------|-------------|---------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|---------------|----------------|----------------|
| 668.5        | 165           | 983.83         | 3              | 666.46        | 37             | 875.99         | 96            | 720.05         | 51             | 1000        | 705.580       | 2              | 927.512        | 958.25         | 77            | 926.39         | 58             | 680.846        | 9              | 845.952        | 42            | 676.73        | 1000          |
| 0            | 54.185        | 82             | -0.9590        | 98            | 33.015         | 96             | 546.37         | 94            | 12.9084        | 7             | 152.396       | 0              | -0.3897        | 2              | 0              | 0              | 0              | 11.322         | 71             | 854.50         | 18            | 6.7178        | 68            |
| 603.1        | 262           | 972.53         | 14             | 571.66        | 66             | 833.17         | 66            | 681.87         | 55             | 1000        | 673.651       | 2              | 894.441        | 960.59         | 92            | 919.72         | 92             | 560.527        | 3              | 848.087        | 7              | 608.21        | 34             | 1000          |
| 153.3        | 439           | 696.29         | 13             | 128.80        | 79             | 418.17         | 79            | 276.46         | 37             | 915.29       | 291.087        | 5              | 621.456        | 296.94         | 67            | 335.16         | 67             | 39.9912        | 8              | 215.454        | 23            | 977.46        | 61             | 522.184       |
| 142.9        | 382           | 508.10         | 41             | 126.92        | 94             | 346.96         | 2             | 166.63         | 61             | 1000        | 197.2         | 654.159        | 450.03         | 21            | 256.05         | 67             | 16.9539        | 7              | 215.454        | 89            | 192.08        | 78             | 535.280       |
| 0            | -0.3897       | 2              | 0              | 11.322        | 71             | 49.720         | 17            | 0              | 35.620         | 6             | 0            | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 296.94        | 67             | 25.3004       |
| 157.1        | 363           | 530.24         | 12             | 122.23        | 92             | 382.48         | 24            | 309.66         | 85             | 360.755      | 764.754        | 84             | 494.04         | 38            | 284.25         | 87             | 19.4268        | 9              | 224.125        | 9              | 264.79        | 78             | 564.574       |
| 32.15         | 116           | 190.49         | 41             | 22.767        | 3              | 141.99         | 4             | 56.876         | 57             | 612.28       | 61.3785        | 5              | 347.946        | 45            | 343.512        | 43             | 0.62827        | 7              | 380.4267       | 9              | 47.941        | 31             | 232.813       |
| 617.7         | 868           | 1000           | 1000           | 10.538        | 11             | 526.21         | 17            | 417.17         | 77             | 1000        | 390.398        | 9              | 862.004        | 5              | 972.53         | 81             | 53.2907        | 5              | 968.908        | 82             | 131.62        | 62             | 710.745       |
| 16.13         | 67            | 848.08         | 77             | 3.8330        | 16             | 443.05         | 17            | 722.12         | 59             | 1000        | 707.645        | 9              | 918.611        | 3              | 0              | 0              | 32.1511        | 6              | 466.01         | 23             | 383.437       | 15             | 822.568       |
| 611.2         | 674           | 995.23         | 97             | 551.42        | 35             | 917.50         | 35            | 1000           | 1000          | 1000        | 1000          | 900.99         | 47             | 678.79         | 5              | 450.032        | 9              | 555.471        | 5              | 804.60         | 97             | 788.78        | 34             | 961.774       |
| 38.23         | 862           | 953.6          | 15             | 35.62         | 15             | 555.47         | 36            | 760.5          | 801            | 1000        | 743.91         | 65             | 952.45         | 36            | 0              | 0              | 144.82         | 75             | -0.9590        | 2              | 577.73         | 37             | 494.04         |
| 407.2         | 489           | 925.2          | 804            | 377.59        | 37             | 666.46         | 38            | 689.0          | 804            | 1000        | 659.28         | 38            | 904.28         | 14            | 776.2          | 543            | 461.01         | 36             | 143.88         | 27             | 346.96         | 543            | 0.8422        |
| 0             | 5.251         | 222            | -0.7690        | 9             | 0              | 424.13         | 54            | -0.7690        | 9             | 37.364       | 0              | -0.9590        | 2              | 0              | 0              | 0              | 0              | 232.81         | 32             | 0              | 0              | 0              | 0              | 15.323        |
| 397.3         | 327           | 911.9          | 788            | 347.9         | 31             | 669.54         | 168           | 730.4          | 168           | 1000        | 704.54         | 77             | 888.99         | 88             | 761.6          | 234            | 467.00         | 99             | 140.10         | 65             | 405.26         | 46             | 578.73         |
| 92.45         | 338           | 588.9          | 252            | 82.32         | 75             | 304.76         | 47            | 323.3          | 793            | 900.99       | 318.47         | 64             | 605.16         | 60             | 180.9          | 331            | 98.004         | 54             | 19.426         | 89             | 62.281         | 42             | 215.454        |

Table 5: The worst average scenario for player 1.
<table>
<thead>
<tr>
<th>Closest Shoot</th>
<th>Closest Charge</th>
<th>Closest Default</th>
<th>Closest Random</th>
<th>Valuable Shoot</th>
<th>Valuable Charge</th>
<th>Valuable Default</th>
<th>Valuable Random</th>
<th>In Range Shoot</th>
<th>In Range Charge</th>
<th>In Range Default</th>
<th>In Range Random</th>
<th>Random Shoot</th>
<th>Random Charge</th>
<th>Random Default</th>
<th>Random Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15.15094</td>
<td>0.999499</td>
<td>875</td>
<td>9.228434</td>
<td>32</td>
<td>6.892024</td>
<td>417</td>
<td>14.70183</td>
<td>662</td>
<td>2.442130</td>
<td>218</td>
<td>4.937104</td>
<td>415</td>
<td>3.715373</td>
<td>467</td>
</tr>
<tr>
<td>11.85069</td>
<td>618</td>
<td>14.05556</td>
<td>829</td>
<td>11.29158</td>
<td>979</td>
<td>15.61278</td>
<td>037</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0</td>
<td>3.968780</td>
<td>966</td>
<td>0</td>
<td>6.343185</td>
<td>32</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>12.27904</td>
<td>719</td>
<td>13.80886</td>
<td>954</td>
<td>11.13440</td>
<td>351</td>
<td>15.77691</td>
<td>063</td>
<td>0.9994995</td>
<td>875</td>
<td>15.51099</td>
<td>61</td>
<td>14.79945</td>
<td>945</td>
<td>5.565608</td>
<td>682</td>
</tr>
<tr>
<td>4.204283</td>
<td>53</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
</tr>
<tr>
<td>5.306505</td>
<td>441</td>
<td>6.692607</td>
<td>862</td>
<td>3.715373</td>
<td>467</td>
<td>15.75004</td>
<td>762</td>
<td>12.90267</td>
<td>414</td>
<td>4.638534</td>
<td>251</td>
<td>4.427188</td>
<td>724</td>
<td>7.147307</td>
<td>185</td>
</tr>
<tr>
<td>15.07312</td>
<td>177</td>
<td>1.412798</td>
<td>641</td>
<td>15.52336</td>
<td>948</td>
<td>7.451107</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.683163</td>
<td>111</td>
<td>14.12002</td>
<td>125</td>
</tr>
<tr>
<td>0</td>
<td>10.66953</td>
<td>607</td>
<td>0</td>
<td>2.230470</td>
<td>802</td>
<td>15.07312</td>
<td>177</td>
<td>1.729450</td>
<td>78</td>
<td>10.34968</td>
<td>599</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

Table 6: The standard deviation for player 2.
| Leader | Closest Shoot | Closest Charge | Closest Default | Closest Random | Valuable Shoot | Valuable Charge | Valuable Default | Valuable Random | In Range Shoot | In Range Charge | In Range Default | In Range Random | Random Shoot | Random Charge | Random Default | Random Random |
|--------|---------------|----------------|-----------------|---------------|---------------|----------------|-----------------|---------------|----------------|----------------|----------------|----------------|---------------|-------------|----------------|----------------|----------------|
|        | 328.403       | 13.5215        | 327.376         | 115.3437      | 274.766       | 0              | 290.287         | 66.8857        | 34.6767       | 70.2507        | 312.979         | 145.4982        | 314.0020      | 0            | 373.4433       | 127.2376       |
|        | 1000          | 672.695        | 8              | 1000.95       | 9             | 924.0877       | 963.508         | 130.4701       | 958.2507      | 712.8156       | 1000           | 998.7866       | 1000           | 1000         | 984.6767       | 8              |
|        | 390.768       | 23.7770        | 7              | 420.2311      | 3             | 156.1802       | 315.0377        | 0              | 318.1245      | 104.4683       | 29.8870        | 75.83385        | 433.4016       | 150.8443      | 386.695         | 8              |
|        | 838.117       | 168.947        | 7              | 859.893       | 5             | 523.9873       | 697.188         | 33.48608       | 688.3816      | 315.0377       | 652.0531       | 634.3125        | 955.6042       | 775.8741      | 747.829         | 2              |
|        | 854.227       | 298.548        | 9              | 872.131       | 5             | 609.6011       | 827.651         | 0              | 796.0835      | 303.7064       | 528.990        | 724.51         | 982.2251       | 780.6937       | 798.963        | 4              |
|        | 1000          | 991.777        | 1000           | 970.4326      | 1              | 147.8379       | 1000.95         | 1000           | 749.7682      | 1000           | 1000           | 1000           | 1000          | 1000         | 1000           | 941.3252       |
|        | 839.066       | 283.049        | 7              | 876.823       | 4             | 563.9228       | 676.620         | 2.95902        | 627.4016      | 205.9149       | 490.891        | 705.0069       | 978.9086       | 772.015        | 725.483         | 5              |
|        | 966.984       | 632.338        | 6              | 977.232       | 7             | 806.6332       | 940.424         | 90.22474       | 929.5388      | 530.9903       | 929.558        | 921.3433       | 998.7866       | 966.1175      | 944.918         | 2              |
|        | 990.240       | 60.1175        | 1              | 993.282       | 8             | 486.8701       | 230.021         | 0              | 236.2892      | 31.09153       | 988.677        | 960.0087       | 954.7196       | 956.4876      | 494.909         | 9              |
|        | 378.543       | 4.76908        | 3              | 435.428       | 3             | 73.60417       | 0              | 0              | 0              | 0              | 78.059         | 302.6752       | 479.8287       | 381.6016      | 186.949         | 0              |
|        | 961.761       | 17.4674        | 3              | 960.008       | 7             | 396.8738       | 217.460         | 0              | 218.5084      | 33.48608       | 993.282        | 965.2494       | 817.1565       | 954.7196      | 397.890         | 9              |
|        | 582.822       | 57.8488        | 4              | 611.581       | 7             | 310.9196       | 301.643         | 0              | 332.51       | 88.02125       | 212.216        | 512.9702       | 851.3903       | 642.2028      | 481.841         | 1              |
|        | 1000          | 889.912        | 3              | 1000          | 1000          | 378.3433       | 1000.39         | 898.2854       | 0              | 343.791        | 1000          | 382.6207       | 1000.95        | 0              | 1000           | 972.1457       |
|        | 586.795       | 70.2507        | 3              | 638.258       | 7             | 317.0957       | 257.123         | 0              | 291.321       | 103.3771       | 220.604        | 508.9603       | 856.1173       | 584.8091      | 415.159         | 5              |
|        | 903.847       | 338.665        | 6              | 911.237       | 6             | 653.038        | 655.990         | 2              | 53.29015      | 655.0062       | 338.6656       | 801.841        | 881.5065       | 975.5464      | 926.8262        | 771.049         | 40.57311       | 829.5568       | 547.9724       |

Table 7: The best average scenario for player 2
<table>
<thead>
<tr>
<th>Closest Shoot</th>
<th>Closest Charge</th>
<th>Closest Default</th>
<th>Closest Random</th>
<th>Valuable Shoot</th>
<th>Valuable Charge</th>
<th>Valuable Default</th>
<th>Valuable Random</th>
<th>In Range Shoot</th>
<th>In Range Charge</th>
<th>In Range Default</th>
<th>In Range Random</th>
<th>Random Shoot</th>
<th>Random Charge</th>
<th>Random Default</th>
<th>Random Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>271.5969</td>
<td>2.478502</td>
<td>270.624</td>
<td>78.65635</td>
<td>221.2335</td>
<td>16.32328</td>
<td>257.021</td>
<td>104.5018</td>
<td>257.9916</td>
<td>0</td>
<td>314.5567</td>
<td>88.76242</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>613.3042</td>
<td>997.041</td>
<td>887.9123</td>
<td>936.4916</td>
<td>91.52989</td>
<td>929.7493</td>
<td>655.1844</td>
<td>1000</td>
<td>989.2134</td>
<td>1000</td>
<td>965.3233</td>
<td>6.717868</td>
<td>971.3227</td>
<td>787.7834</td>
<td></td>
</tr>
<tr>
<td>331.2313</td>
<td>8.222973</td>
<td>359.7689</td>
<td>113.8198</td>
<td>258.9623</td>
<td>0</td>
<td>261.8755</td>
<td>69.53168</td>
<td>12.11296</td>
<td>46.16615</td>
<td>372.5984</td>
<td>109.1557</td>
<td>327.3042</td>
<td>0</td>
<td>338.1091</td>
<td>74.08599</td>
</tr>
<tr>
<td>789.8829</td>
<td>125.0523</td>
<td>814.1065</td>
<td>462.0127</td>
<td>638.8114</td>
<td>14.51392</td>
<td>629.6184</td>
<td>258.9623</td>
<td>591.9464</td>
<td>573.6875</td>
<td>926.3958</td>
<td>722.1259</td>
<td>692.1708</td>
<td>0.087848</td>
<td>705.5802</td>
<td>384.4608</td>
</tr>
<tr>
<td>807.7726</td>
<td>243.4511</td>
<td>827.8685</td>
<td>548.3989</td>
<td>778.3483</td>
<td>0</td>
<td>743.9165</td>
<td>248.2936</td>
<td>467.0099</td>
<td>667.49</td>
<td>961.7749</td>
<td>727.3063</td>
<td>747.0368</td>
<td>-0.76909</td>
<td>749.118</td>
<td>381.4934</td>
</tr>
<tr>
<td>1000</td>
<td>976.223</td>
<td>1000</td>
<td>945.5674</td>
<td>1000</td>
<td>106.3621</td>
<td>997.041</td>
<td>694.2318</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>465.0107</td>
<td>1000</td>
<td>908.6748</td>
</tr>
<tr>
<td>790.9331</td>
<td>228.9503</td>
<td>833.1766</td>
<td>502.0772</td>
<td>617.3793</td>
<td>-0.95902</td>
<td>566.5984</td>
<td>158.0851</td>
<td>429.109</td>
<td>646.9931</td>
<td>957.0914</td>
<td>717.985</td>
<td>668.5165</td>
<td>-0.38972</td>
<td>727.3063</td>
<td>350.9002</td>
</tr>
<tr>
<td>944.016</td>
<td>571.6614</td>
<td>954.7673</td>
<td>755.3668</td>
<td>907.5752</td>
<td>57.77527</td>
<td>894.4412</td>
<td>469.0097</td>
<td>894.4412</td>
<td>884.6563</td>
<td>989.2134</td>
<td>939.8825</td>
<td>913.0818</td>
<td>70.4412</td>
<td>934.2386</td>
<td>664.4115</td>
</tr>
<tr>
<td>973.7596</td>
<td>0</td>
<td>966.5139</td>
<td>344.9938</td>
<td>484.0236</td>
<td>0</td>
<td>515.1411</td>
<td>73.1738</td>
<td>0</td>
<td>0</td>
<td>891.1738</td>
<td>0</td>
<td>803.556</td>
<td>0</td>
<td>832.1142</td>
<td>210.6432</td>
</tr>
<tr>
<td>960.5992</td>
<td>33.88249</td>
<td>978.7179</td>
<td>425.1299</td>
<td>179.9783</td>
<td>0</td>
<td>185.7108</td>
<td>12.90847</td>
<td>971.3227</td>
<td>931.9913</td>
<td>925.2804</td>
<td>927.5124</td>
<td>433.0901</td>
<td>-0.95902</td>
<td>530.2412</td>
<td>107.2929</td>
</tr>
<tr>
<td>319.4567</td>
<td>-0.76909</td>
<td>374.5742</td>
<td>44.39583</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47.941</td>
<td>247.3248</td>
<td>418.1713</td>
<td>322.3984</td>
<td>141.0501</td>
<td>0</td>
<td>156.1878</td>
<td>12.90847</td>
</tr>
<tr>
<td>934.2386</td>
<td>4.535267</td>
<td>931.9913</td>
<td>337.1262</td>
<td>168.5392</td>
<td>0</td>
<td>169.4911</td>
<td>14.51392</td>
<td>978.7179</td>
<td>938.7506</td>
<td>766.8435</td>
<td>925.2804</td>
<td>338.1091</td>
<td>227.02</td>
<td>431.0993</td>
<td>85.99962</td>
</tr>
<tr>
<td>521.1777</td>
<td>32.15116</td>
<td>550.419</td>
<td>255.0804</td>
<td>246.3561</td>
<td>0</td>
<td>275.49</td>
<td>55.97875</td>
<td>163.7834</td>
<td>451.0298</td>
<td>804.6097</td>
<td>581.7972</td>
<td>420.1589</td>
<td>0</td>
<td>520.1713</td>
<td>173.3012</td>
</tr>
<tr>
<td>1000</td>
<td>848.0877</td>
<td>1000</td>
<td>990.6283</td>
<td>1000</td>
<td>319.4567</td>
<td>993.6103</td>
<td>857.7146</td>
<td>0</td>
<td>286.209</td>
<td>1000</td>
<td>323.3793</td>
<td>997.041</td>
<td>0</td>
<td>1000</td>
<td>947.8543</td>
</tr>
<tr>
<td>525.2047</td>
<td>41.7493</td>
<td>577.7413</td>
<td>260.9043</td>
<td>204.8769</td>
<td>0</td>
<td>236.679</td>
<td>68.62286</td>
<td>171.3956</td>
<td>447.0397</td>
<td>809.8827</td>
<td>523.1909</td>
<td>354.8405</td>
<td>-0.95902</td>
<td>480.0172</td>
<td>157.1363</td>
</tr>
<tr>
<td>864.1523</td>
<td>281.3344</td>
<td>872.7624</td>
<td>592.962</td>
<td>596.0098</td>
<td>28.70985</td>
<td>594.9938</td>
<td>281.3344</td>
<td>750.1589</td>
<td>838.4935</td>
<td>952.4536</td>
<td>891.1738</td>
<td>716.9503</td>
<td>19.42689</td>
<td>780.4432</td>
<td>486.0276</td>
</tr>
</tbody>
</table>

| Table 8: The worst average scenario for player 2. |
Code:

Below is the code that was used for running the simulation. It imports two JSON dictionaries that contain the values for all the units and models in both armies. Special thanks to planetB.ca for having a SyntaxHighlighter for word that allows putting code in Microsoft Word while maintain indentation.

```python
from character import *
import numpy
import math
import json
from random import *
import csv
import scipy
import scipy.stats

with open('Army1.json', 'r') as f:
    army1 = json.load(f)  # creates player one's army
f.close()

with open('Army2.json', 'r') as f:
    army2 = json.load(f)  # creates player 2's army
f.close()

Armies = {'One': army1, 'Two': army2}

lockedInCombat = []

def toHitRanged(attacker):
    # the to hit calculation for ranged weapons
    return rollOverD6((7 - attacker['ballisticSkill']))

def toHitRangedHigh(attacker):
    # the to hit for ballistic skill values above 7, unused
    return (rollOverD6(12 - attacker['ballisticSkill']))

def toHitMelee(attacker, defender):
    # the to hit calculation for close combat
    if (attacker['weaponSkill'] > defender['weaponSkill']):
        return rollOverD6(3)
    elif ((attacker['weaponSkill'] * 2) < (defender['weaponSkill'])):
        return rollOverD6(5)
    else:
        return rollOverD6(4)

def toWound(attacker, defender, weapon):
    # The to wound calculations
    if (weapon['strength'] > ((defender['toughness']) +1)): return rollOverD6(2)
    elif (weapon['strength'] == (defender['toughness']) +1): return rollOverD6(3)
    elif (weapon['strength'] == (defender['toughness']) -1): return rollOverD6(4)
    elif (weapon['strength'] == (defender['toughness']) -3)):
        return rollOverD6(5)
    else:
        return (weapon['strength'] > (defender['toughness'] - 1)) and (weapon['strength'] >=(defender['toughness'] -3)))
```
def makeSaves(defender, wounds, weapon):
    # the calculations for making an armor save
    if ((weapon['ap'] > defender['armorSave']) and (defender['armorSave'] < defender['invulnerableSave'])):
        return (wounds - (wounds * rollOverD6(defender['armorSave'])))
    else:
        return (wounds - (wounds * rollOverD6(defender['invulnerableSave'])))

def shootingAttack(attacker, defender, weapon):
    # the act of shooting for a single model
    if ((Armies[attacker['inArmy']][attacker['inSquad']][\"Moved\"] == True) and (attacker[\"rangedWeapon\"]\"type\"] == \"Heavy\")):
        hits = rollOverD6(6)
        wounds = rollIDice(toWound(attacker, defender, weapon), hits)
        saves = makeSaves(defender, wounds, weapon)
        results = saves
    elif((measureDistance(Armies[attacker[\"inArmy\"]][attacker[\"inSquad\"]][\"Location\",
                                      Armies[defender[\"inArmy\"]][defender[\"inSquad\"]][\"Location\"])
                            <= weapon[\"range\"] and (attacker[\"rangedWeapon\"]\"type\"] == \"Rapid Fire\")):
        hits = rollIDice(toHitRanged(attacker),(weapon[\"rof\"]*2))
        wounds = rollIDice(toWound(attacker, defender, weapon), hits)
        saves = makeSaves(defender, wounds, weapon)
        results = saves
    else:
        hits = rollIDice(toHitRanged(attacker),weapon[\"rof\"])
        wounds = rollIDice(toWound(attacker, defender, weapon), hits)
        saves = makeSaves(defender, wounds, weapon)
        results = saves
    return results

def closeCombatAttack(attacker, defender, weapon):
    # a single close combat attack
    hits = attacker[\"attacks\"] * toHitMelee(attacker, defender)
    wounds = toWound(attacker, defender, weapon) * hits
    saves = makeSaves(defender, wound, weapon)
    results = saves
    return results

def rollOverD6(successNum):
    # actually roll for success
    if ((successNum <= 6) and (successNum > 1)):
        roll = rollID6()
        needed = successNum
        if ( needed <= roll):
            return (1)
        else:
            return(0)
    elif (successNum < 2):
        roll = rollID6()
        needed = 2
        if ( needed <= roll):
            return (1)
        else:
            return(0)
    else:
        return (0)
def rollDice(sucessNum, times):  # the function for rolling multiple dice, returning a value equal 1*number of successes.
    total = 0
    for x in range(times):
        total += rollOverD6(sucessNum)
    return total

def rollOverD6Stat(sucessNum):  # gives the mathematical average roll as a decimal, unused.
    if ((sucessNum <= 6) and (sucessNum > 1)):
        return ((7 - sucessNum) / 6)
    elif (sucessNum < 2):
        return (5/6)
    else:
        return 0.0

def rolld6():  # returns an integer from 1 to 6, inclusive, represents rolling a single die
    return randint(1,6)

def squadFire(squad, defender):  # the action of firing for an entire squad, including removing causalties
    total= 0
    a = len(squad["Squad"])
    for x in range (0,a):
        if ((measureDistance(squad["Location"],defender["Location"])) <= squad["Squad"] [x]["rangedWeapon"] ["range"]["range"][":"]):
            total = total + (shootingAttack(squad["Squad"] [x], defender["Squad"] [0], squad["Squad"] [x]["rangedWeapon"]))
    for stiff in range(total):
        if (defender["isDead"] == False):
            x = defender["Squad"][0].pop(0)
        if (len(defender["Squad"]) <= 0):
            defender["isDead"] = True

def measureDistance(start,stop):  # gives a value that represents the distance between two points on a 2d graph, representing the distance between the models in inches
    result = math.sqrt(((start[0]-stop[0])*(start[0]-stop[0]))+((start[1]-stop[1])*(start[1]-stop[1])))
    return result

def moveSix(start, stop):  # Moves the model 6" close to their target, or directly on top of them if less than 6" away
    if (measureDistance(start["Location"], stop["Location"])) < start["Movespeed"]:
        start["Location"] = stop["Location"]
        return (stop["Location"] [0], stop["Location"] [1])
    else:
        t = (measureDistance(start["Location"], stop["Location"])) / measureDistance(start["Location"],stop["Location"])
        x = ((stop["Location"] [0])*(1-t)+(t * start["Location"] [0]))
        y = ((stop["Location"] [1])*(1-t)+(t * start["Location"] [1]))
        start["Location"] = [x,y]
    return [x,y]

def closeCombatTurn(unitsInCC):  # simulates the entire fight subphase of the assault phase, for both sides
    combatants = []
    dead = []
for unit in unitsInCC:
    for model in unit["Squad"]:
        combatants.append(model)
for step in range(10, 1, -1):
    for x in range(len(combatants)):
        model = combatants[x]
        if model["initiative"] == step:
            target = Armies[model["inArmy"]][model["inSquad"]]["inCCWith"]
            hits = rollDice(toHitMelee(model, target), model["attacks"])  
            if hits > 0:
                wounds = rollDice(toWound(model, target, model["meleeWeapon"]), hits)
                if wounds > 0:
                    deaths = makeSaves(target, wounds, model["meleeWeapon"])
                    target = Armies[model["inArmy"]][model["inSquad"]]["inCCWith"]
                    for x in range(deaths):
                        dead.append(Armies[target["inArmy"]][target["inSquad"]]["Squad"][x])
                    for stiff in dead:
                        if (Armies[stiff["inArmy"]][stiff["inSquad"]]["isDead"] == False):
                            # check for dead
                            Armies[stiff["inArmy"]][stiff["inSquad"]]["Squad"]["inCCWith"].pop(0)
                            if (len(Armies[stiff["inArmy"]][stiff["inSquad"]]["Squad"])) <= 0:
                                Armies[stiff["inArmy"]][stiff["inSquad"]]["isDead"] = True
                                for squad in lockedInCombat:
                                    if (id(squad) == id(Armies[stiff["inArmy"]][stiff["inSquad"]])):
                                        try:
                                            lockedInCombat.remove(Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"])
                                        except:
                                            pass
                                        try:
                                            lockedInCombat.remove(Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"])
                                        except:
                                            pass
                                        try:
                                            Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"] = None
                                        except:
                                            pass
                                if (Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"] != None):
                                    False
    for unit in unitsInCC:
        for model in unit["Squad"]:
            combatants.append(model)
        model = combatants[x]
        if model["initiative"] == step:
            target = Armies[model["inArmy"]][model["inSquad"]]["inCCWith"]
            for step in range(10, 1, -1):
                for x in range(len(combatants)):
                    model = combatants[x]
                    if model["initiative"] == step:
                        target = Armies[model["inArmy"]][model["inSquad"]]["inCCWith"]
                        hits = rollDice(toHitMelee(model, target), model["attacks"])  
                        if hits > 0:
                            wounds = rollDice(toWound(model, target, model["meleeWeapon"]), hits)
                            if wounds > 0:
                                deaths = makeSaves(target, wounds, model["meleeWeapon"])
                                target = Armies[model["inArmy"]][model["inSquad"]]["inCCWith"]
                                for x in range(deaths):
                                    dead.append(Armies[target["inArmy"]][target["inSquad"]]["Squad"][x])
                                for stiff in dead:
                                    if (Armies[stiff["inArmy"]][stiff["inSquad"]]["isDead"] == False):
                                        # check for dead
                                        Armies[stiff["inArmy"]][stiff["inSquad"]]["Squad"]["inCCWith"].pop(0)
                                        if (len(Armies[stiff["inArmy"]][stiff["inSquad"]]["Squad"])) <= 0:
                                            Armies[stiff["inArmy"]][stiff["inSquad"]]["isDead"] = True
                                            for squad in lockedInCombat:
                                                if (id(squad) == id(Armies[stiff["inArmy"]][stiff["inSquad"]])):
                                                    try:
                                                        lockedInCombat.remove(Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"])
                                                    except:
                                                        pass
                                                    try:
                                                        lockedInCombat.remove(Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"])
                                                    except:
                                                        pass
                                                    try:
                                                        Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"] = None
                                                    except:
                                                        pass
                                                if (Armies[stiff["inArmy"]][stiff["inSquad"]]["inCCWith"] != None):
                                                    False
    def charge(unit, target): #has the unit attempt to charge the target in the close combat phase
        chargeRange = rolld6() + rolld6()
        if (measureDistance(unit["Location"], target["Location"])) <= chargeRange:
            unit["Location"] = target["Location"]
            unit["inCC"] = True
unit["inCCWith"] = target
target["inCC"] = True
if (target == None):
    target["inCCWith"] = unit
lockedInCombat.append(unit)
lockedInCombat.append(target)
else:
    pass

def findClosest(unit): # finds the closest unit to the given unit
closest = None
if (unit["Squad"][0]["inArmy"] == "One"):
    for x in Armies["Two"][x]:
        squad = (Armies["Two"][x])
        if ((closest == None) and (squad["isDead"] == False)):
            closest = squad
elif ((closest != None) and ((measureDistance(closest["Location"],unit["Location"])) > measureDistance(unit["Location"],
            squad["Location"])) and (squad["isDead"] == False)):
    closest = squad
return closest

def findValue(unit): # finds the most valuable unit for the given unit
valuable = None
if (unit["Squad"][0]["inArmy"] == "One"):
    for x in Armies["Two"][x]:
        squad = (Armies["One"][x])
        if ((valuable == None) and (squad["isDead"] == False)):
            valuable = squad
elif ((valuable != None) and (squad["Points"] > valuable["Points"])) and (squad["isDead"] == False)):
    valuable = squad
return valuable

def chooseRandomUnit(unit): # selects a random enemy unit for the given unit
options = []
if (unit["Squad"][0]["inArmy"] == "One"):
    for x in Armies["Two"]:
squad = Armies["Two"][x]
if (squad["isDead"] == False):
    options.append(x)
if (len(options) > 0):
    return Armies["Two"][choice(options)]
if (unit["Squad"][0]["inArmy"] == "Two"):
    for x in Armies["One"]:
        squad = Armies["One"][x]
        if (squad["isDead"] == False):
            options.append(x)
    if (len(options) > 0):
        return Armies["One"][choice(options)]
    return None

def chooseRandomAction(unit, target): # randomly chooses either to shoot or move for a given unit
    option = choice(["shoot", "charge"])
    if(option == "shoot"):
        if (unit["inCC"] == False):
            squadFire(unit, target)
    if(option == "charge"):
        if (unit["inCC"] == False):
            charge(unit, target)

def defaultAction(unit, target): # decides the default action for a given unit, in the future the default
    # behavior could be moved to a property of the unit
    if (unit["Name"] == "Devistator Squad"):
        squadFire(unit, target)
    if (unit["Name"] == "Tactical Squad 2"):
        squadFire(unit, target)
    if (unit["Name"] == "Tactical Squad 1"):
        squadFire(unit, target)
    if (unit["Name"] == "Assault Squad"):
        if (measureDistance(unit["Location"], target["Location"]) <= 12):
            charge(unit, target)

def playerTurn(player, target, action): # a single turn for a player, having movement followed by
    # shooting/charging, then close combat, order first by target type then action type
    # Closest
    if target == "Closest":
        if action == "Shoot":
            for unit in Armies[player]:
                squad = Armies[player][unit]
                if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                    mark = findClosest(squad)
                    if (mark != None):
                        moveSix(squad, mark)
            for unit in Armies[player]:
                squad = Armies[player][unit]
                if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                    mark = findClosest(squad)
                    if (mark != None):
                        squadFire(squad, mark)
            closeCombatTurn(lockedInCombat)
        if action == "Charge":
            for unit in Armies[player]:
                squad = Armies[player][unit]
                if ((squad["inCC"] == False) and (squad["isDead"] == False)):
mark = findClosest(squad)

if (mark != None):
    moveSix(squad, mark)

    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"]) <= 12):
                    charge(squad, mark)
                closeCombatTurn(lockedInCombat)

    if action == "Default":
        for unit in Armies[player]:
            squad = Armies[player][unit]
            if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                mark = findClosest(squad)
                if (mark != None):
                    moveSix(squad, mark)
                    for unit in Armies[player]:
                        squad = Armies[player][unit]
                        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                            mark = findClosest(squad)
                            if (mark != None):
                                defaultAction(squad, mark)
                closeCombatTurn(lockedInCombat)

        if action == "Random":
            for unit in Armies[player]:
                squad = Armies[player][unit]
                if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                    mark = findClosest(squad)
                    if (mark != None):
                        moveSix(squad, mark)
                for unit in Armies[player]:
                    squad = Armies[player][unit]
                    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                        mark = findClosest(squad)
                        if (mark != None):
                            chooseRandomAction(squad, mark)
            closeCombatTurn(lockedInCombat)

        if target == "Valuable":
            if action == "Shoot":
                for unit in Armies[player]:
                    squad = Armies[player][unit]
                    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                        mark = findValue(squad)
                        if (mark != None):
                            moveSix(squad, mark)
                    for unit in Armies[player]:
                        squad = Armies[player][unit]
                        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                            mark = findValue(squad)
                            if (mark != None):
                                squadFire(squad, mark)
                closeCombatTurn(lockedInCombat)

            if action == "Charge":
                for unit in Armies[player]:
                    squad = Armies[player][unit]
                    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                        mark = findValue(squad)
                        if (mark != None):
                            squadFire(squad, mark)
moveSix(squad, mark)

for unit in Armies[player]:
    squad = Armies[player][unit]
    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
        mark = findValue(squad)
        if (mark != None):
            if (measureDistance(squad["Location"], mark["Location"])) <= 12):

            charge(squad, mark)
            closeCombatTurn(lockedInCombat)

for action in "Default", "Random":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findValue(squad)
            if (mark != None):
                moveSix(squad, mark)

for unit in Armies[player]:
    squad = Armies[player][unit]
    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
        mark = findValue(squad)
        if (mark != None):
            defaultAction(squad, mark)
            closeCombatTurn(lockedInCombat)

for target in "inRange":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"])) <= 24):

                    squadFire(squad, mark)
                    closeCombatTurn(lockedInCombat)

for action in "Charge":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"])) <= 24):

                    moveSix(squad, mark)
                    for unit in Armies[player]:
                        squad = Armies[player][unit]
                        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                            mark = findClosest(squad)
                            if (mark != None):
if (measureDistance(squad["Location"], mark["Location"]) <= 12):
    charge(squad, mark)
    closeCombatTurn(lockedInCombat)
if action == "Default":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)) and (unit != "Bob"):
            mark = findClosest(squad)
            if (mark != None):
                moveSix(squad, mark)
        for unit in Armies[player]:
            squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"]) <= 24):
                    defaultAction(squad, mark)
                    closeCombatTurn(lockedInCombat)
if action == "Random":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"]) <= 24):
                    moveSix(squad, mark)
        for unit in Armies[player]:
            squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = findClosest(squad)
            if (mark != None):
                if (measureDistance(squad["Location"], mark["Location"]) <= 24):
                    chooseRandomAction(squad, mark)
                    closeCombatTurn(lockedInCombat)
if target == "Random":
    if action == "Shoot":
        for unit in Armies[player]:
            squad = Armies[player][unit]
            if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                mark = chooseRandomUnit(squad)
                if (mark != None):
                    moveSix(squad, mark)
        for unit in Armies[player]:
            squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = chooseRandomUnit(squad)
            if (mark != None):
                squadFire(squad, mark)
                closeCombatTurn(lockedInCombat)
if action == "Charge":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = chooseRandomUnit(squad)
            if (mark != None):
if (measureDistance(squad["Location"],mark["Location"])) <= 12):
    charge(squad, mark)

for unit in Armies[player]:
    squad = Armies[player][unit]
    if ((squad["inCC"] == False) and (squad["isDead"] == False)):
        mark = chooseRandomUnit(squad)
        if (mark != None):
            charge(squad, mark)
    closeCombatTurn(lockedInCombat)
if action == "Default":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = chooseRandomUnit(squad)
            if (mark != None):
                moveSix(squad, mark)
        for unit in Armies[player]:
            squad = Armies[player][unit]
            if ((squad["inCC"] == False) and (squad["isDead"] == False)):
                mark = chooseRandomUnit(squad)
                if (mark != None):
                    defaultAction(squad, mark)
        closeCombatTurn(lockedInCombat)
if action == "Random":
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = chooseRandomUnit(squad)
            if (mark != None):
                moveSix(squad, mark)
    for unit in Armies[player]:
        squad = Armies[player][unit]
        if ((squad["inCC"] == False) and (squad["isDead"] == False)):
            mark = chooseRandomUnit(squad)
            if (mark != None):
                chooseRandomAction(squad, mark)
        closeCombatTurn(lockedInCombat)

def gameTurn(stratOne,stratTwo): #a single game turn, made of two player turns
    playerTurn("One",stratOne[0], stratOne[1])
    playerTurn("Two",stratTwo[0], stratTwo[1])

def runGame(stratOne, stratTwo): #runs six game turns and then decides the winner
    gameTurn(stratOne,stratTwo)
    gameTurn(stratOne,stratTwo)
    gameTurn(stratOne,stratTwo)
    gameTurn(stratOne,stratTwo)
    gameTurn(stratOne,stratTwo)
    totalOne = 0
    totalTwo = 0
    for unit in Armies["One"]:
        total = len(Armies["One"][unit]["Squad"])
        totalOne += total
    for unit in Armies["Two"]:
        total = len(Armies["Two"][unit]["Squad"])
        totalTwo += total
    if (totalOne > totalTwo):
        return "w"
    elif (totalOne < totalTwo):
        return "l"
def resetArmies(): # resets the armies to their original composition and clears the combat
    # in order to run the simulation again
    with open('Army1.json', 'r') as f:
        army1 = json.load(f)
        f.close()
    with open('Army2.json', 'r') as f:
        army2 = json.load(f)
        f.close()
    Armies['One'] = army1
    Armies['Two'] = army2
    lockedInCombat.clear()

def getData(): # runs all strategies against each other, for a total of 256,000 test
    with open('results.csv', 'w', newline='') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        target = ['Closest', 'Valuable', 'InRange', 'Random']
        action = ['Shoot', 'Charge', 'Default', 'Random']
        for mark in target:
            for verb in action:
                line = []
                for mark2 in target:
                    for verb2 in action:
                        count = 0
                        win = 0
                        loss = 0
                        draw = 0
                        for x in range(1000):
                            resetArmies()
                            result = runGame([mark, verb], [mark2, verb2])
                            if (result == 'w'):
                                win += 1
                            elif (result == 'l'):
                                loss += 1
                            elif (result == 'd'):
                                draw += 1
                            total = [win, loss, draw]
                            count += 1
                            print(str(count) + ' for ' + mark + ' ' + verb + ' vs. ' + mark2 + ' ' + verb2)
                            line.append(total)
                            print(line)
                            scorewriter.writerow(line)

def getSpecific(mark, verb, mark2, verb2): # gets the data for a specific match up of strategies
    count = 0
    win = 0
    loss = 0
    draw = 0
    for x in range(1000):
        resetArmies()
        result = runGame([mark, verb], [mark2, verb2])
        if (result == 'w'):
            win += 1
        elif (result == 'l'):
            loss += 1
        elif (result == 'd'):
            draw += 1
        total = [win, loss, draw]
        count += 1
        print(str(count) + ' for ' + mark + ' ' + verb + ' vs. ' + mark2 + ' ' + verb2)
loss += 1
if (result == "d"):
    draw += 1
    count += 1
    print(str(count) + "for " + mark + " + verb + " vs. " + mark2 + " + verb2)
total = [win, loss, draw]
print(total)

def getDataSelect(mark, verb):
    # gets the data for a single strategy against all other
    with open('results.csv', 'w', newline='') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        target = ["Closest", "Valuable", "InRange", "Random"]
        action = ["Shoot", "Charge", "Default", "Random"]
        line = []
        for mark2 in target:
            for verb2 in action:
                count = 0
                win = 0
                loss = 0
                draw = 0
                for x in range(1000):
                    resetArmies()
                    result = runGame([mark, verb],[mark2,verb2])
                    if (result == "w"):
                        win += 1
                    if (result == "l"):
                        loss += 1
                    if (result == "d"):
                        draw += 1
                    count += 1
                    print(str(count) + "for " + mark + " + verb + " vs. " + mark2 + " + verb2)
                total = [win, loss, draw]
                print(total)
                line.append(total)
                print(line)
        scorewriter.writerow(line)

def getDataFour(mark, verb, mark2):
    # gets the data for a single strategy against a specific strategy in terms
    # of target type, for a total of 4 sets of tests
    with open('results.csv', 'w', newline='') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        action = ["Shoot", "Charge", "Default", "Random"]
        line = []
        for verb2 in action:
            count = 0
            win = 0
            loss = 0
            draw = 0
            for x in range(1000):
                resetArmies()
                result = runGame([mark, verb],[mark2,verb2])
                if (result == "w"):
                    win += 1
                if (result == "l"):
                    loss += 1
                if (result == "d"):
                    draw += 1
count += 1

print(str(count) + " for " + mark + " " + verb + " vs. " + mark2 + " " + verb2)

total = [win, loss, draw]

print(total)
line.append(total)

print(line)

scorewriter.writerow(line)

def getDeviation():  # reads a excel sheet containing the net result of all the sets of
tests and creates a sheet that gives the standard deviation for the sets

    a = []
    results = []

    with open('netresults.csv', 'r') as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))
        csvfile.seek(0)
        resultReader = csv.reader(csvfile, dialect)
        for row in resultReader:
            a.append(row)

    for x in a:
        for y in x:
            z = (int(y)/1000)
            deviant = math.sqrt(1000 * (z) *(1-z))
            results.append(deviant)

    print(results)

    with open('devresults.csv', 'w', newline='') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')

        for x in range(16):
            line = []
            for y in range(16):
                z = (16 * x) + y
                line.append(results[z])
            print(line)
        scorewriter.writerow(line)

def getMean():  # reads the raw results and creates a mean value for each set of results.

    a = []
    totals = []
    huge = []
    results = []

    with open('results.csv', 'r') as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))
        csvfile.seek(0)
        resultReader = csv.reader(csvfile, dialect)
        for row in resultReader:
            a.append(row)

    for x in a:
        for y in x:
            b = eval(y)
            win = b[0]
            loss = b[1]
            draw = b[2]
            totals.append([win, loss, draw])

    for w in totals:
        sublist = []
        for x in range(w[0]):
sublist.append(1)
for y in range(w[1]):
    sublist.append(-1)
for z in range(w[2]):
    sublist.append(0)
huge.append(sublist)
for x in huge:
    rMean = numpy.mean(x)
results.append(rMean)
print(results)
with open('meanresults.csv', 'w') as csvfile:
    scorewriter = csv.writer(csvfile, dialect='excel')
    for x in huge:
        rSum = sum(x)
        results.append(rSum)
        print(results)

def getNet():
    # reads the raw results and creates a sheet containing the net victories for a strategy based on the results, currently calibrated to give player 2's net victories.
    a = []
    totals = []
    results = []
    with open('results.csv', 'r') as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))
        csvfile.seek(0)
        resultReader = csv.reader(csvfile, dialect)
        for row in resultReader:
            a.append(row)
    for x in a:
        for y in x:
            b = eval(y)
            win = b[0]
            loss = b[1]
            draw = b[2]
            totals.append([win, loss, draw])
    for w in totals:
        sublist = []
        for x in range(w[0]):
            sublist.append(0)
        for y in range(w[1]):
            sublist.append(1)
        for z in range(w[2]):
            sublist.append(0)
        huge.append(sublist)
    for x in huge:
        rSum = sum(x)
        results.append(rSum)
    with open('netresults.csv', 'w') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        for x in range(16):
            line = []
            for y in range(16):
                z = (16 * x) + y
                line.append(results[z])
            print(line)
            scorewriter.writerow(line)

    with open('meanresults.csv', 'w') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        for x in huge:
            rMean = numpy.mean(x)
            results.append(rMean)
    results = []
    with open('meanresults.csv', 'w') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        print(results)
def getBest():  # takes the net results and the standard deviation results and creates a sheet for the best value within a 95% confidence interval
    a = []
    b = []
    results = []
    with open('netresults.csv', 'r') as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))
        csvfile.seek(0)
        resultReader = csv.reader(csvfile, dialect)
        for row in resultReader:
            a.append(row)
    f.close()
    with open('devresults.csv', 'r') as csvfile2:
        csvfile2.seek(0)
        resultReader2 = csv.reader(csvfile2)
        for row in resultReader2:
            b.append(row)
    f.close()
    for x, g in zip(a, b):
        for y in zip(x, g):
            foo = float(y[1])
            z = foo * 1.96
            best = float(y[0]) + z
            results.append(best)
    print(results)
    with open('bestresults.csv', 'w', newline='') as csvfile:
        scorewriter = csv.writer(csvfile, dialect='excel')
        for x in range(16):
            line = []
            for y in range(16):
                z = (16 * x) + y
                line.append(results[z])
            print(line)
            scorewriter.writerow(line)

def getWorst():  # takes the net result and the standard deviation and creates a sheet for the worst-case scenario within a 95% confidence interval
    a = []
    b = []
    results = []
    with open('netresults.csv', 'r') as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))
        csvfile.seek(0)
        resultReader = csv.reader(csvfile, dialect)
        for row in resultReader:
            a.append(row)
    f.close()
    with open('devresults.csv', 'r') as csvfile2:
        csvfile2.seek(0)
        resultReader2 = csv.reader(csvfile2)
        for row in resultReader2:
            b.append(row)
    f.close()
    for x, g in zip(a, b):
        for y in zip(x, g):
            foo = float(y[1])
            z = foo * 1.96
            worst = float(y[0]) - z
results.append(worst)
print(results)
with open('worstresults.csv', 'w', newline='') as csvfile:
    scorewriter = csv.writer(csvfile, dialect='excel')
    for x in range(16):
        for y in range(16):
            z = (16 * x) + y
            line.append(results[z])
        print(line)
        scorewriter.writerow(line)

def getTTest(a, b):
    # runs and returns the value of the ttest for two sets of data
    result1 = scipy.stats.ttest_ind(a, b)
    print(result1)
getData()