Strategy in Battle Royale Games

Ao Chen

Zhenning Liu

October 15, 2021

Abstract

We create a discrete-time mathematical model for battle royale games (BRG) to investigate the game theory behind this popular class of video games. We use a generalized asymmetric prisoner's dilemma to study the microscopic decision-making scenario and successfully find the Nash equilibrium by theoretical analysis. Repeated numerical simulations are performed to investigate how the macroscopic strategy influences the game dynamics. We conclude that for some specific setup of the game, a unique optimal strategy exists, while for others the best choice is conditioned on other participants' choices. The evolution of strategies are also studied together with examples from existing BRGs including PUBG and APEX Legends.

Contents

Chapter 1

Introduction

Battle royale refers to a non-cooperative fight among many combatants which lasts until only one of them remains standing. It is often used in many fictions and films like The Hunger Games to show the humanity in an extreme environment. In recent years, many popular shooter video games, PlayerUnknown's Battleground (PUBG), Fortnite and APEX Legends included, adopt the battle royale rule. In these games, many people are scattered in a large field and their goal is to become the only survivor. There is usually a safe zone on the game map shrinking with time that forces people to fight. It requires the players to have a good shooting accuracy in combats as well as a clear strategy to determine their actions in different situations.

In this thesis, we want to study the strategy in battle royale games (BRGs) using game-theoretic tools together with computer simulations. Our purpose is to present a guidance for game players to get a higher score in the game and for game designers to improve the game.

(a) PUBG [1]

(b) APEX [2]

Figure 1.1: Popular battle royale games

With a simplified rule based on PUBG and APEX, we separate the strategy into a microscopic part and a macroscopic part.

The microscopic strategy determines the players' choice in a 1 vs. 1 combat. We assume the information during the combat is totally transparent, so the players can estimate whether they are in an advantageous situation and then make a decision of fight or retreat. The interaction in the microscopic strategy is relatively simpler because there are essentially only two players in a combat, so it will be simplified as a prisoner's dilemma which we will analyze theoretically.

On the other hand, the macroscopic strategy represents the tendency of players to be aggressive or peaceful. Avoiding fights may make it easier to get a higher rank, but being aggressive can help players to gain the killing scores and better equipment from eliminated enemies, the latter improving the winning rate in successive combats. The macroscopic dynamics is more complex because it involves the interaction among all survival players. For example, if most teams are aggressive, then the number of survivors will decrease quickly and a player can easily gain the rank score through avoiding combats. We will run numerical simulations to investigate whether an optimal macroscopic strategy exists under different game conditions, and try to explain the different player behaviours in different BRGs.

It is worth noting that the microscopic and macroscopic strategies are separated. A player may avoid combats in its macroscopic strategy but find itself advantageous in an unavoidable combat and then choose to fight until killing the enemy in the microscopic strategy. The microscopic strategy is determined by the Nash equilibrium of the prisoners' dilemma of the combat situation, while the macroscopic strategy of a player is consistent during the whole game and may only be changed after the end of one game.

Chapter 2

Modelling of the game

Obviously, we are unable to emulate a game of PUBG or APEX Legends in every detail due to the requirements of huge computational resources. In fact, it is also unnecessary to do so, since the game mechanics contains a significant amount of properties that have nothing to do with the game theory problems that we are interested in. Examples are geography of the game map, functional difference between weapons and body armors, and skill difference between players etc. We have to simplify the game to eliminate these unrelated aspects and focus on the microscopic and macroscopic strategies that dominate an idealized battle royale game. In this section we present a general modelling with some free parameters which can be used to distinguish between different battle royale video games.

2.1 Overview

Instead of simulating the real-time dynamics of a game, we split the timeline into a sequence of discrete *rounds* as the smallest unit of time. We model the 2-dimensional motions of players in BRGs mainly by a probabilistic *fight matching* module by which people are matched in pairs to fight against each other. If by any chance someone is matched, there is a *decision making* module which simulates how players make decisions to engage or to retreat. If either side decides to fight, a *battle simulator* will give a result – one player might be eliminated from the game for losing the battle. The simulation will terminate once all but one players are eliminated, which will definitely happen after certain number of rounds because the shrinking safe zone is also considered in our simulation. The active area decreases with time, so battles will appear more and more frequently and everyone will be forced to fight in the end, just like a typical PUBG or APEX Legends game. A ranking will be calculated for players based on survival time. With no doubt, the only survivor will be the rank-1 player.

2.2 Scoring

The rank-1 player is not necessarily the most successful player due to the existence of a scoring system. Players earn scores from not only the rank but also the amount of elimination. The goal of a professional player is to maximize the score. The score s_i earned by the ith player with number of killing k_i and rank r_i can be expressed by the following formula

$$s_i = s_{\text{kill}} \cdot k_i + s_{\text{rank}}(r_i) \tag{2.1}$$

where s_{kill} denotes the score awarded for each elimination and $s_{\text{rank}}(r_i)$ stands for the score given for being rank-r_i player. s_{kill} and s_{rank} are pre-defined characteristic values for every specific BRG. They largely determine the optimal choices of macroscopic strategies. If s_{kill} is significantly higher than s_{rank}, then players are more encouraged to join battles to kill other participants, and vice versa.

In our work, we keep the score for ranking constant, because what really matters is the ratio of 2 types of scores. Table 2.1 shows how much score is given for each rank that one player may end up with.

Table 2.1: Ranking scores.

r_i	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25
$s_{\text{rank}}(r_i)$	15	12	10	9	8	7	6	5	4	3	2	1	0

The ratio of s_{kill} and $s_{\text{rank}}(r_i)$ is actually a key property that distinguishes different BRGs. For example, if we use the $s_{\text{rank}}(r_i)$ values given by table 2.1, APEX Legends can be treated as a game with $s_{\text{kill}} \approx 5$ while PUBG is a game with $s_{\text{kill}} \approx 9$. A fun fact is that in top-level E-sports games, APEX participants usually tend to avoid fighting as much as possible and PUBG players are more willing to join battles. We will further investigate this topic in chapter 4.

2.3 Players

There are 25 players at the beginning of the simulation. Every player is modelled by 3 attributes: HP (hit point), maxHP (upper limit of HP) and ATK (attacking power). A player is "eliminated" if his HP is reduced to 0. In typical BRGs, there are way more attributes related to a player, but we prefer to make a generalized model and get rid of technical details. Hence, we model both health value and body armor value by the HP, and model any protective

equipment, such as helmet or high-level body armor, by the maxHP. Furthermore, all weapons and/or any other game-specific properties related to fighting strength are treated as a single ATK value.

Every player is initialized with identical attributes before a game simulation begins. We also assume all players are equally skilled. It means that, in battle simulations, the probability distribution of damage dealt is always the same for 2 players if they have the same ATK value.

The *microscopic strategy* in deciding whether a player wants to fight or retreat when he/she is chosen in a battle, is also identical for all players. However, the *macroscopic strategy* which dominates how frequent the player will encounter a battle, may differ for different players. In our simulation, we only consider 2 options of macroscopic strategy – aggressive (strategy 1) and peaceful (strategy 0). Being aggressive means the player tends to search and kill other players to earn killing score, while being peaceful means the player prefer to avoid fighting to obtain score from higher ranking.

2.4 Fight matching

In round t, for every player that is not in a battle, say the ith, there is a probability $p_t(i)$ to be chosen by the fight matching module to join a battle. This probability is affected by the macroscopic strategy as well as the density ρ of remaining players in the safe zone, since higher density implies higher probability of encountering each other. Quantitatively, $p_t(i)$ can be expressed by

$$p_t(i) = \begin{cases} 0.1 + m\rho, & \text{strategy} = 0 \\ 0.3 + m\rho, & \text{strategy} = 1 \end{cases} \tag{2.2}$$

where $m = 1$ is the density parameter that quantifies how much the density should affect the probability of being chosen.

The fight matching program runs at the end of each round. All chosen players will be matched randomly in pairs and in the next round, the decision making module will decide the behaviour of every involved player in the battle simulator.

2.5 Decision making

Once a player is chosen by the fight matching module to take part in a battle against another player, our simulation system gives 2 options to him. One is to fight and the other one is to retreat. We assume players know every attribute of the opponent, therefore this decision-making

can be modelled as an asymmetric prisoner's dilemma. We propose a universal microscopic strategy to model and solve this dilemma. It will be discussed in detail in chapter 3.

2.6 Battle simulator

A battle is defined as a game of reducing HP value based on opponent's ATK value between exactly 2 players. The battle simulator receives orders from decision-making module as a binary pair $(d_1, d_2) \in \{0, 1\}^2$ where $d_i = 0$ means player i decides to retreat, and $d_i = 1$ means he determines to fight against the enemy. The general damage formula in a certain round between player 1 and 2 is the following.

$$\mathrm{HP}_1 \leftarrow \mathrm{HP}_1 - r_{N2} \cdot d_2 \cdot \mathrm{ATK}_2 \tag{2.3}$$

$$\mathrm{HP}_2 \leftarrow \mathrm{HP}_2 - r_{N1} \cdot d_1 \cdot \mathrm{ATK}_1. \tag{2.4}$$

Here $r_{Ni} \sim \mathcal{N}(1, \sigma_{\mathrm{ATK}})$ is a random number satisfying Gaussian distribution and σ_{ATK} is its standard deviation. Such a randomness input is necessary, otherwise the simulator and the microscopic strategies would be deterministic.

It turns out that the decision will never be both sides retreating, which will be discussed in 4. Therefore, we only need to consider 2 cases. If both players choose to fight, $d_1 = d_2 = 1$, damages and remaining HPs will be computed. If both survive from the battle, the decision-making module will be called again in next round and the battle will continue. If one player chooses to retreat while the other fights, let's say $d_1 = 0, d_2 = 1$. This is actually related to the "betray" case of prisoner's dilemma. Player 1 will first receive a damage while player 2 will not. If player 1 still has $\mathrm{HP}_1 > 0$ after being damaged, the battle simulator will generate a random bit $r_{\mathrm{Escape}} \leftarrow_r \{0, 1\}$ to determine if he can successfully retreat. $r_{\mathrm{Escape}} = 1$ means it is successful, and the battle simulator terminates. If $r_{\mathrm{Escape}} = 0$, then the battle will continue in next round and the decision needs to be made again.

If in any of the above case, any side of a battle ends up with $\mathrm{HP} \leq 0$, he is eliminated in the simulation and the battle simulator terminates. We will keep an record of eliminated players to calculate how much they scored, but they will not be chosen in any further battles since they are no longer active. As an extra reward of the victory, the winner will get "weapon" (increasing ATK) and "armor" (increasing maxHP) from the eliminated player and therefore has greater opportunity in winning next battle.

2.7 Other details

2.7.1 Looting

As a rather popular feature in BRGs, it is worth considering looting in our simulation. Looting is modelled as a probabilistic increase of maxHP or ATK. Specifically, in each round, there is a probability $p_{\text{loot}} = 0.1$ for each player to have ATK value added by 10 or maxHP value added by 20.

2.7.2 Recovery from a battle

Almost all BRGs have the mechanism allowing players to gain HP by medication, bondage, recharging armor, or any specific in-game action. Our setup in the simulation is that, if a player is not in a battle for more than 1 round, then his HP is automatically restored to maxHP.

Chapter 3

Microscopic Strategy: Fight or Retreat?

3.1 Generalized asymmetric prisoners' dilemma

Table 3.1: Traditional prisoners' dilemma ($T > R > P > S$)

	cooperate	defect
cooperate	(R, R)	(S, T)
defect	(T, S)	(P, P)

Table 3.2: Generalized asymmetric prisoners' dilemma

	retreat	fight
retreat	(R_1, R_2)	(S_1, T_2)
fight	(T_1, S_2)	(P_1, P_2)

The microscopic strategy determines the choice when two teams encounters. We use the prisoners' dilemma to describe this situation. Every player can choose to retreat (cooperate) or fight (defect). Compared with traditional prisoners' dilemma, there are three major differences.

1. The combat is stochastic, so every choice corresponds to many possible situations. The reward will be given by the expectation value of the score under a specific choice.

2. The T, R, P, S values are different for different players in the asymmetric prisoners' dilemma. When both player choose to fight, for instance, the player with better equipment will have a larger chance to win.

3. $T > R > P > S$ does not always apply in the generalized prisoners' dilemma. For example, sometimes $P_1 < S_1$, which means the player prefers to avoid the combat and bear the enemy's fire instead of having a head-on fight.

To solve the first difference, we need to calculate the expected rewards under different choices, which requires (1) the expected rewards under different combat results, (2) the probability of different combat results. These quantities will be calculated in the following sections. After that, we will analyze the generalized asymmetric prisoners' dilemma to give the Nash equilibrium strategy.

3.2 Probability of different combat results

Table 3.3: Possible combat results

	retreat	fight
retreat	(stop, stop)	(stop, stop) or (lose, win)
fight	(stop, stop) or (win, lose)	(win, lose) or (lose, win)

Table 3.3 shows that there are many possible results under the same choice. "Stop" means the combat stops due to retreat, and win/lose means one player kills the other player. Except for retreat of both sides, all other situations contain two possible outcomes. We will calculate the probabilities of different situations in this section.

3.2.1 Probability of successful retreat

If one player chooses to retreat and the other player chooses to fight, the retreating player will have a probability to retreat successfully. In every round, there is a chance of successful retreat as long as the HP value has not been reduced to 0 under enemy's fire. To calculate the probability, we first need to know the damage caused by the enemy. In the n'th round, the damage $d(n)$ obeys the normal distribution

$$d(n) \sim \text{ATK} \times (1 + \delta\,\mathcal{N}(0,1)), \tag{3.1}$$

where ATK is the enemy's attack value and $\mathcal{N}(0,1)$ is the standard normal distribution. The total damage from the 1st round to the n'th round is given by

$$D(n) = \sum_{i=1}^{n} d(i) \sim \text{ATK} \times (n + \delta\sqrt{n}\,\mathcal{N}(0,1)), \tag{3.2}$$

where we have used the sum rule of normally distributed random variables [3].

The probability of successful retreat in the n'th round is given by the two conditions: (1) the HP has not been reduced to 0; (2) the successful retreat happens at this round. The two conditions are irrelevant, so the probability is a product of two terms

$$P_{\text{stop}}(n) = P(D(n) < \text{HP}) \times P_{\text{run}}(1 - P_{\text{run}})^{n-1}, \tag{3.3}$$

where P_{run} is the probability of successful retreat when the player still survives. $P(D(n) < \text{HP})$ is easily determined by the cumulative distribution function of the normal distribution, which can be given by most modern scientific computing packages like SciPy [4]. The total retreat probability is the sum over all terms

$$P_{\text{stop}} = \sum_{n=1}^{\infty} P_{\text{stop}}(n). \tag{3.4}$$

It is impossible to calculate the sum of this infinite series. Fortunately, the term $P(D(n) < \text{HP})$ converges to 0 quickly when the expected damage $\mathbf{E}[D(n)] = n \times \text{ATK}$ is far larger than HP, so we can set a truncation value N and approximate $P(D(n) < \text{HP})$ with $n > N$ as 0. Based on the total damage expression Eq. (3.2), we use the famous 3σ law to give the truncation condition

$$\text{ATK} \times (N - 3\delta\sqrt{N}) \geq \text{HP}. \tag{3.5}$$

$$N \geq 4.5\delta^2 + \text{ROUND} + 3\delta\sqrt{4.5\delta^2 + \text{ROUND}}, \tag{3.6}$$

where $\text{ROUND} = \text{HP}/\text{ATK}$ is the estimated number of rounds required to deal enough damage. N is chosen to be the lowest value satisfying this inequality. The approximate retreat probability is

$$P_{\text{stop}} \approx \sum_{n=1}^{N} P_{\text{retreat}}(n). \tag{3.7}$$

3.2.2 Probability of winning a head-on combat

The head-on fight is more complex than the previous condition in which one player retreats. It is very hard to calculate the winning probability of a player, so we only roughly estimate the probability. The expected number of rounds in the combat is

$$\text{ROUND} = \min(\text{HP}_1/\text{ATK}_2,\ \text{HP}_2/\text{ATK}_1), \tag{3.8}$$

where the subscript represents different teams. The winning condition of team 1 is roughly given by the following expression

$$\text{HP}_2 - D_1(\text{ROUND}) < \text{HP}_1 - D_2(\text{ROUND}), \tag{3.9}$$

where D_1 and D_2 are defined in Eq. (3.2) with ATK given by the value of the corresponding team. Using the sum of normal distributed variables [3] again, we obtain

$$\begin{aligned} &D_2(\mathrm{ROUND}) - D_1(\mathrm{ROUND}) \\ &\sim \mathrm{ATK}_2(\mathrm{ROUND} + \delta\sqrt{\mathrm{ROUND}}\mathcal{N}_2(0,1)) - \mathrm{ATK}_1(\mathrm{ROUND} + \delta\sqrt{\mathrm{ROUND}}\mathcal{N}_1(0,1)) \\ &\sim (\mathrm{ATK}_2 - \mathrm{ATK}_1)\mathrm{ROUND} + \delta\sqrt{\mathrm{ROUND}(\mathrm{ATK}_1^2 + \mathrm{ATK}_2^2)}\mathcal{N}(0,1). \end{aligned} \tag{3.10}$$

The approximate winning probability of team 1 is

$$\begin{aligned} P_{\mathrm{win}} &\approx P(D_2(\mathrm{ROUND}) - D_1(\mathrm{ROUND}) < \mathrm{HP}_1 - \mathrm{HP}_2) \\ &= P\left(\mathcal{N}(0,1) < \frac{(\mathrm{ATK}_1 - \mathrm{ATK}_2)\mathrm{ROUND} + \mathrm{HP}_1 - \mathrm{HP}_2}{\delta\sqrt{\mathrm{ROUND}(\mathrm{ATK}_1^2 + \mathrm{ATK}_2^2)}}\right), \end{aligned} \tag{3.11}$$

with the last expression also given by the cumulative distribution function. This equation is a rough estimation allowing us to avoid the calculation of series summation as in Eq. (3.7) at the cost of introducing larger errors.

3.3 Rewards under different combat results

There are three different results for a player participating in the combat — lose, stop or win. We will analyze the expected rewards under the three combat results.

3.3.1 Lose

This is the simplest situation. The player has to quit this game and will not obtain any score in the rest of the game, which means the expected reward is 0.

3.3.2 Stop

The player will not benefit from the current combat, but will obtain scores in the successive game. We will estimate the score obtained by surviving or killing other players.

The surviving score is determined by the rank of the player, which can be estimated by the current equipment of the player. A player with better equipment is more likely to survive in the successive combats and obtain higher surviving scores. We calculate the expected rank of the player through virtual combats against all other surviving players in the game. The virtual combats, which do not really happen, are tools for us to test the combat effectiveness of the player. The winning probability of the virtual combat is given by Eq. (3.11). We will sum over all these probabilities to obtain the expected rank of the player. For example, if there are

three survivors and one of them has 30% and 70% probability of winning the other two players respectively, then the estimated rank of this player is 2. After knowing the estimated rank, we can obtain the surviving score through checking the score table.

The calculation of the killing score is a bit more complex. Considering the difficulty of beating players with better equipment, the potential killing number is roughly the number of survivors with a lower rank than the player, with the rank estimated above. However, these survivors with lower ranks may also be killed by other players, so we calculate the killing weight w_i which is given by the player's combat frequency divided by the sum of all combat frequencies, i. e.

$$w_i = \frac{p_t(i)}{\sum_j p_t(j)}, \tag{3.12}$$

with $p_t(i)$ given by Eq.(2.2). The estimated killing number is the product of this weight and the potential killing number given by the rank. This equation shows that the macroscopic strategy may have a minor impact on the microscopic strategy

3.3.3 Win

The expected rewards of the winning player is similar to the case of stopped combat. The difference is the winning player will obtain an immediate killing score and some better equipment from the enemy. With the better equipment, it is likely for the player to obtain more surviving and killing scores in the remaining game. This effect is also calculated here. We will conduct a virtual equipment update before calculating the estimated rank, which improves the rank and hence the expected killing number.

3.4 Nash equilibrium

After the great effort of calculating the probabilities and rewards, we come to the stage to obtain the R, S, T, P values. According to Table 3.2 and Table 3.3, these quantities can be calculated. For example, the reward for retreating under enemy's fire is

$$S = P_{\text{stop}} s_{\text{stop}} + (1 - P_{\text{stop}}) s_{\text{lose}}, \tag{3.13}$$

where s_{stop} and s_{lose} represent the expected scores of stop and lose, respectively.

A direct result we can obtain is that the two encountering players will not both choose to retreat. If one player choose to retreat, the rewards of the other player under different choices will be $R = s_{\text{stop}}$ and $T = P_{\text{stop}} s_{\text{stop}} + (1 - P_{\text{stop}}) s_{\text{win}}$. We always have $R < T$ due to $s_{\text{stop}} < s_{\text{win}}$.

Consequently, the two players will not choose to retreat (cooperate) at the same time, which means this is never a Nash equilibrium point.

There are three possible situations in the generalized asymmetric prisoners' dilemma, each of them corresponding to different Nash equilibrium points. We will analyze them one by one.

When $S_1 < P_1$ and $S_2 < P_2$, both players will choose to fight. It is similar to the traditional prisoners' dilemma in which both players will defect. This situation can happen even when $R_1 > P_1$ and $R_2 > P_2$, so the two encountering players have to fight under the constraint of the prisoners' dilemma even though the best choice for both of them is to retreat.

When $S_1 < P_1$ and $S_2 > P_2$, the player 1 will choose to retreat under the fire of player 2. This can happen if the player 2 has an overwhelming advantage in the combat. The player 1 have to retreat in order to survive. The situation with $S_1 > P_1$ and $S_2 < P_2$ is similar.

When $S_1 < P_1$ and $S_2 < P_2$, both players want to retreat instead of having a direct combat. However, only one of them will retreat because of the impossibility of both retreat as mentioned above. This situation therefore has two Nash equilibrium points. In our simulations, we always choose the Nash equilibrium by forcing the player with smaller winning probability in the head-on combat to retreat.

In this chapter, we analyze the generalized asymmetric prisoners' dilemma to determine the players' choice in a combat. After some efforts of calculating the expected rewards, we summarize their choices into three situations. The most interesting situation is the one similar to the traditional prisoners' dilemma. It is worth mentioning that this situation often happens in professional competitions of PUBG or APEX, which makes the game more exciting and presents more requirements for the macroscopic strategy of the player. In the next chapter, we will utilize the conclusion in this chapter and perform some simulations to analyze the macroscopic strategy.

Chapter 4

Macroscopic Strategy: Aggressive or Peaceful?

Apart from decision-making when encountering a battle, the player's strategy also influences how frequent he meets another player and is forced to fight. In this section, we discuss how to model this macroscopic part of the strategy.

4.1 Macroscopic strategies

As is mentioned in 2, our highly simplified model only considers 2 macroscopic strategies. Strategy-0 is the peaceful way, by which the player has less probability of encountering other players, hence are less likely to be eliminated in the early game and correspondingly have less chance to gain score from killing. Strategy-1 is the other way around where players tend to search-and-destroy. Such an aggressive game-play is quite popular for casual players since they prefer to enjoy the shooting/battle content of video game. However, what a casual player like to do has nothing to do with game theory. In this work, we are more interested in what the optimal strategy is for a serious player looking for higher scores for a specific BRG. This is actually not that far from a regular player since both APEX Legends and PUBG feature a well-defined ranking system. And lots of BRGs have become a significant part of the rising E-sports universe.

4.2 Type-n games

Let us come back to the simulation. Since every player is initialized with the same attributes except a binary number representing his macroscopic strategy, they can be treated as indis-

tinguishable individuals like atoms or elementary particles. It is therefore reasonable to call a game type-n if among all 25 players, there are n players taking strategy-1 and $25 - n$ taking strategy-0. We can further conclude that, if s_{kill} is fixed, the simulation result of every type-n game is sampled from the same probability distribution $\mathcal{X}_n$. Here the *result* contains but is not limited by the scores got by players, lifetime of the game and so on.

One corollary of the above claim is that, there are at most 26 different types of simulation regarding choices of macroscopic strategies, since $n \in \{0, 1, \ldots, 25\}$. Thus, if we want to investigate the optimal strategy of a player, we simply need to simulate for all 26 scenarios and see if there is a difference in expected scores for 2 strategies.

4.3 Optimal strategy for $s_{\text{kill}} = 5$ games

We start by investigating $s_{\text{kill}} = 5$ games. This model is closely related to APEX Legends.

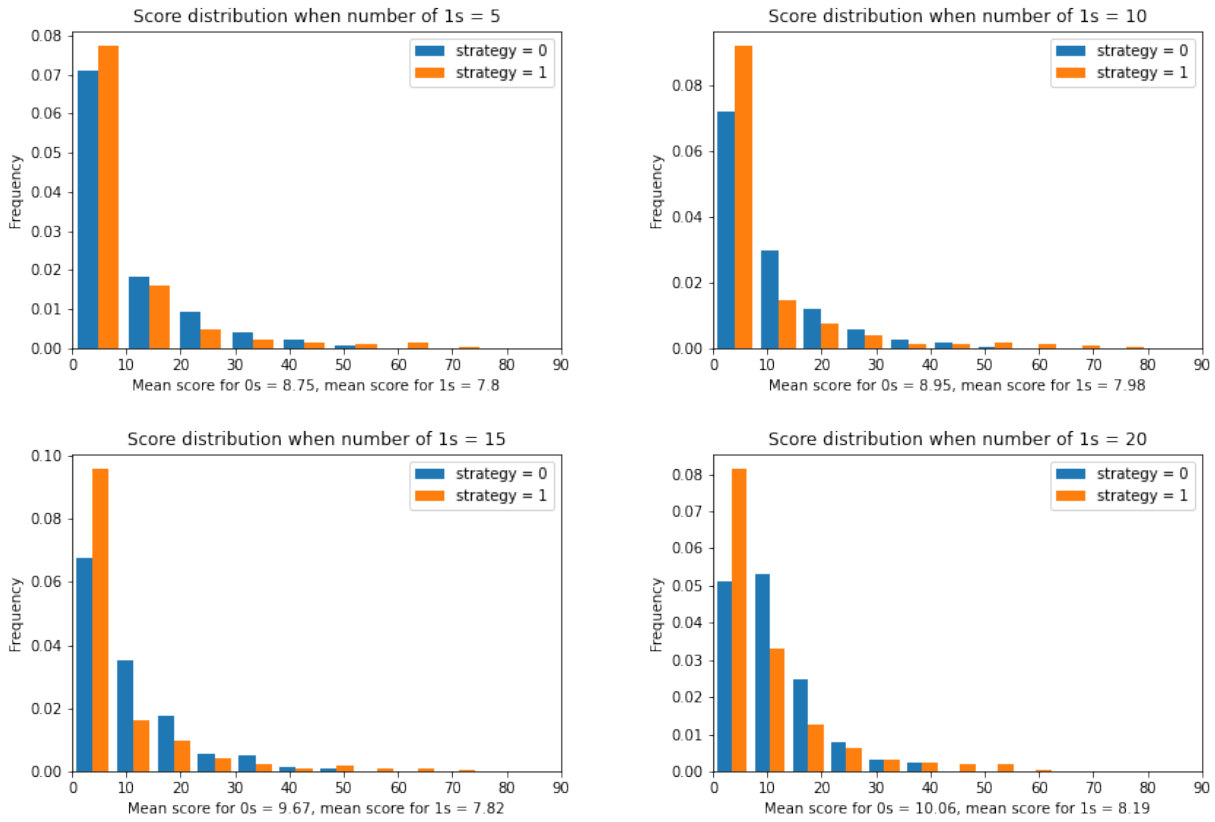

Figure 4.1: These histograms show distributions of scores earned by players choosing strategy-0 and strategy-1, respectively. The data is obtained by 500 times of simulation of type-5, type-10, type-15 and type-20 games in $s_{\text{kill}} = 5$ case.

The distribution of scores for different types of game is shown in Fig 4.1. Here we can observe some fun facts. It seems strategy-0 has higher expectation of scores in all 4 types of

games. Although strategy-1 players have better opportunity to gain extremely high scores, they are also more likely to end up with extremely low score for being eliminated in early-game battles. However, such a risk decreases if the total number of aggressive players is fewer in the game.

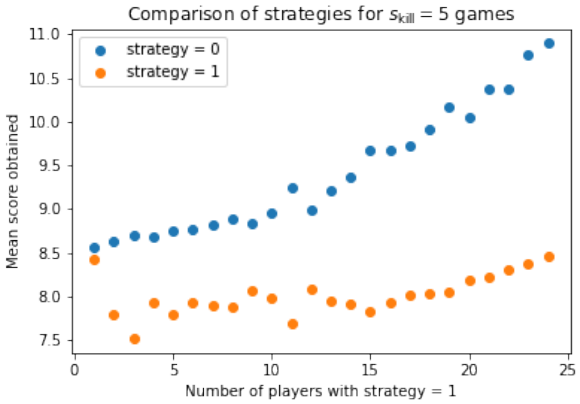

Figure 4.2: This figure shows how the mean score of different macroscopic strategy changes with the total number of strategy-1 players in the APEX-like game. The data is obtained by 500 times of simulation of each type-n game.

In figure 4.2, simulation results for 24 types of games are summarized together. (The reason why it is 24 instead of 26 is that the all strategy-0 case and the all strategy-1 case are extreme cases where we cannot see such a comparison.) As the figure shows, in an APEX-like BRG where $s_{\text{kill}} = 5$, strategy-0 is indeed always the better choice. If 25 equally-skilled professional players play the game together again and again, no matter what their initial experience is, their strategies will all eventually evolve to 0. This is not a surprising result because in real-life APEX Legends E-sports games, avoiding early battles and postponing inevitable fights as late as possible has become a common practice among professional players as well as high-rank players in the ranking system.

We can also see that, the advantage of strategy-0 is less significant when the majority prefers the peaceful game-play. We reasonably conjecture that, if the score assigned to each elimination is increased, the aggressive strategy should be optimal in at least some types of game. And there should be a cross-point at which 2 strategies are equally optimal. This will be investigated in next section.

4.4 Various s_{kill} values and phase transitions

On various values of $s_{\text{kill}} \in \{3, 4, 5, 6, 7, 8, 9, 10\}$, we perform repeated (500 times as usual) simulations for every possible type of game. In figure 4.3 we present the outcomes of simulations.

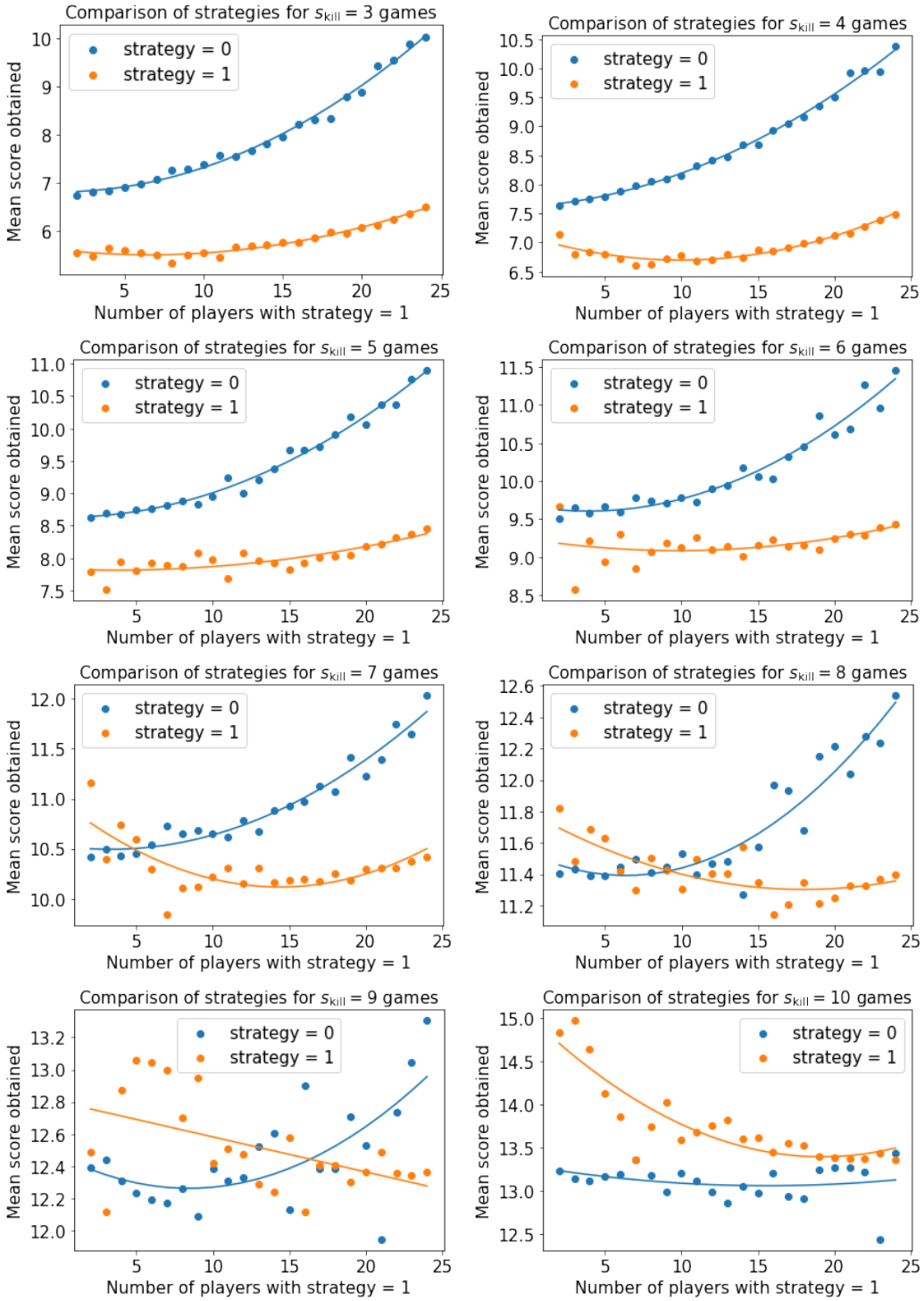

Figure 4.3: These figures show 2 macroscopic strategies' mean score variation with respect to type of the game as well as s_{kill}. Apart from the scatter diagram, the fitted quadratic curves are also plotted, such that the intersections of expected scores can be found for $s_{\text{kill}} = 7, 8, 9$.

The quadratic fitting of data points seems to be surprisingly good for extreme values of s_{skill}. And the results of simulations perfectly verifies our conjecture. The peaceful strategy is always the better one with small values of s_{skill}. But it is no longer the all-time optimal when $s_{\text{skill}} = 7$ where strategy-1 gives higher expected score for type-$n < n_c = 5$ games. Here the cross-point of 2 curves of expected scores is denoted by n_c. It can be observed that n_c increases as s_{skill} increases and it disappears again when the score for killing is equal to or higher than 10. In this case, the aggressive strategy is the optimal choice for all types of game.

n_c can be interpreted as the point where phase transition happens. When $s_{\text{skill}} \leq 6$, no phase transition exists and strategy-0 is always optimal. When $s_{\text{skill}} \geq 10$, there is still no phase transition but strategy-1 is always better. When $7 \leq s_{\text{skill}} \leq 9$, strategy-1 is optimal when $n < n_c$ and strategy-0 is optimal when $n > n_c$, which is a phase transition.

Although we cannot give an exact value, but we believe that if sufficient computational resource is given, it is possible to very accurately compute the 2 threshold values of s_{skill} where n_c disappears. But from the current data obtained, we are still able to draw a phase diagram in figure 4.4 that shows in what situation a certain macroscopic strategy is preferred.

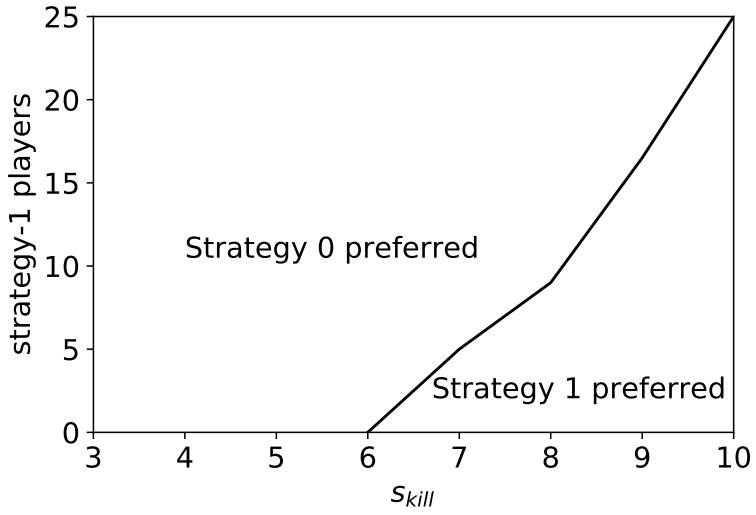

Figure 4.4: The phase diagram generated by our simulations.

It might also be interesting to consider the evolution of strategies for $7 \leq s_{\text{kill}} \leq 9$ where a phase transition exists at n_c. If 25 players play the game together once and once again. They will learn from their experience that for a type-n game, the optimal strategy depends on if $n > n_c$ or $n < n_c$. And their attractions for both strategies will be updated with more and more games played. We can conclude that there must exist an equilibrium at which every player's *attractions* a_0 and a_1 to either strategy are such that the corresponding probability p_0

of choosing strategy 0 and p_1 of choosing strategy 1 satisfies

$$25\frac{e^{\lambda a_1}}{e^{\lambda a_0}+e^{\lambda a_1}} = 25p_1 = n_c$$

where $p_1 = \frac{e^{\lambda a_1}}{\sum_i e^{\lambda a_i}}$ is the Fermi/logit function.

We discussed the relationship between $s_{\text{kill}} = 5$ and APEX Legends in last section. Here we can observe that the simulation of $s_{\text{kill}} = 9$ corresponding to PUBG contains a phase transition at $n = n_c \approx 16$. It implies that the optimal choice for players is to be aggressive if there are less than 16 players taking the aggressive strategy.

This equilibrium at $n_c = 16$ explains why E-sports teams of PUBG usually play much more aggressively than APEX Legends teams. Due to the high score given by every elimination, players are more willing to engage in battles in early-game, even though there is a risk of losing scores of ranking.

There is some enlightenment about how to make a good BRG. If an enthusiastic player with professional-level skill wants to enjoy a certain BRG, there would certainly be more fun if game mechanism and the scoring system are designed such that the phase transition point n_c exists, ideally at the half of the total number of participants. Because this allows for non-unique optimal strategy that depends on the strategies taken by other players. The game theory behind this scenario is non-trivial, unlike the APEX Legends case where the peaceful strategy is the certainly optimal one no matter what other players choose to do. Hence, we can say PUBG is a somehow a better designed game than APEX Legends, just in terms of diversity of strategies.

Chapter 5

Summary and outlook

5.1 Our results

In this thesis, we created a simplified mathematical model with a few free parameters to specify the specific battle royale video game to be modelled. The game is modelled as a sequence of discrete rounds. In every round, players are randomly chosen to fight with a probability decided by their macroscopic strategies, decisions are made following the microscopic strategy. and battles are simulated probabilistically. We managed to run multiple computer simulations based on the model and investigate the how the strategies affect the game.

The microscopic strategy dominates the decision making process of players facing a battle. The situation is modelled by a generalized asymmetric prisoner's dilemma, where the payoff of an action is computed based on expected scores if this action is executed. Thus the strategy is choosing the best option determined by Nash equilibrium.

The macroscopic strategy is modelled by 2 discrete options – peaceful or aggressive. We claim that under a certain setup of the model, there are up to $N_{\text{players}} + 1$ game-theoretic different scenarios. With the data generated by repeated simulations, we managed to draw a phase diagram that shows the optimal macroscopic strategy under certain condition and where the boundary is. We also analyzed the evolution of macroscopic strategies: in some scoring systems all players will tend to use the same strategy, while in systems with a phase transition, there will be an equilibrium between the attraction of peaceful and aggressive strategies.

We also managed to give an explanation of why top-level PUBG games involve more early-game battles than APEX Legends, by our simulations based on the model. We thus claimed that our model has the descriptive power to distinguish the 2 games. Furthermore, we suggested that a well designed battle royale game should not allow for a unique optimal strategy, because

in this case the fun of macroscopic decision-making will be reduced.

5.2 Outlook

We presented an analysis of evolution of macroscopic strategies and claimed there will be an equilibrium, but the reason why we can do it is the model is highly simplified such that brute-force simulations are sufficiently informative. It would be ideal if experience-weighted-attraction (EWA) [5] learning can be performed to show the dynamics of players' behavior and search the equilibrium for more complex models, which we did not have time to do.

Although we claimed our model is descriptive, it would be much more convincing if comparative research between real PUBG/APEX Legends E-sports game results and our simulation results can be done. As a matter of fact, academic data analysis research of both games is still missing, which makes it hard to compare the results.

References

1. *Wikipedia: PlayerUnknown's Battlegrounds* https://en.wikipedia.org/wiki/PlayerUnknown%27s_Battlegrounds.

2. *Wikipedia: Apex Legends* https://en.wikipedia.org/wiki/Apex_Legends.

3. *Wikipedia: Sum of normally distributed random variables* https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables.

4. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272 (2020).

5. Camerer, C. & Hua Ho, T. Experience-weighted attraction learning in normal form games. *Econometrica* **67,** 827–874 (1999).

Appendix A

Program

```
import numpy as np
import random
import math
from scipy.stats import norm
from matplotlib import pyplot as plt

class Team:
    def __init__(self, idd, atk, hp, stg):
        self.id = idd
        self.stg = stg
        self.hp = hp
        self.maxhp = hp
        self.atk = atk
        self.fight = -1
        self.rest = 0
        self.kills = 0
        self.die = 200
        self.rank = 0
        self.score = 0


N_teams = 25  # total number of teams joining the game
N_learn = 100  # total number of games to simulate
T = 110  # Total number of rounds (time) in one game
P_stg0 = 0.1  # (base) probablity of being selected to a battle if STRATEGY 0 is chosen
P_stg1 = 0.3  # (base) probablity of being selected to a battle if STRATEGY 1 is chosen
teams = [Team(idd=i_team, atk=50, hp=100, stg=1) for i_team in range(N_teams)]  # Initialize all teams
score_survive = list(range(12, 2, -1)) + [2] * 5 + [1] * 5 + [0] * 5
score_survive[0:2] = 15, 12
score_kill = 5
atk_uncertainty = 0.3  # std = atk * uncertainty
successful_run = 0.5  # success probability of escaping from a battle

N_current_teams = N_teams
current_time = 0
survival_teams = list(range(N_teams))


def InitTeams():
```

```
    new_teams = [Team(idd=i_team, atk=50, hp=100, stg=0) for i_team in range
   (N_teams)]
    new_survival_teams = list(range(N_teams))
    new_N_current_teams = N_teams
    return (new_teams, new_survival_teams, new_N_current_teams)


def InitArea():
    r0 = 100.0
    spdShrink = 2.0
    tShrink = 10
    tPhase1 = 20
    tPhase2 = 40
    tPhase3 = 60
    tPhase4 = 80
    tPhase5 = 90
    ifShrink = np.zeros(T, dtype=int)
    ifShrink[0:tPhase1] = 0
    ifShrink[tPhase1:tPhase1 + tShrink] = 1
    ifShrink[tPhase1 + tShrink:tPhase2] = 0
    ifShrink[tPhase2:tPhase2 + tShrink] = 1
    ifShrink[tPhase2 + tShrink:tPhase3] = 0
    ifShrink[tPhase3:tPhase3 + tShrink] = 1
    ifShrink[tPhase3 + tShrink:tPhase4] = 0
    ifShrink[tPhase4:tPhase4 + tShrink] = 1
    ifShrink[tPhase4 + tShrink:tPhase5] = 0
    ifShrink[tPhase5:] = 1
    rs = np.zeros(T, dtype=float)
    areas = np.zeros(T, dtype=float)

    for ti in range(T):
        if (ti >= 99):
            rs[ti] = 0.1
        elif (ti == 0):
            rs[ti] = r0
        elif not ifShrink[ti]:
            rs[ti] = rs[ti - 1]
        else:
            rs[ti] = rs[ti - 1] - spdShrink

    areas = rs * rs
    return areas


Areas = InitArea()


def Escape():
    rdm = random.random()
    return rdm > successful_run


def BattleRNG():
    # rng = random.random() * (AtkRateMax - AtkRateMin) + AtkRateMin
    rng = np.random.normal(loc=1, scale=atk_uncertainty, size=1)[0]
    if (rng < 0):
        rng = 0
    # print(rng)
    return rng

```

```
99
100 def AtkIncrease(team_give, team_receive, in_place=True):
101     new_atk = max(team_receive.atk, team_give.atk) * 1.1
102     if in_place:
103         team_receive.atk = new_atk
104     return new_atk
105
106
107 def HPIncrease(team_give, team_receive, in_place=True):
108     new_hp = max(team_receive.maxhp, team_give.maxhp)
109     if in_place:
110         team_receive.maxhp = new_hp
111     return new_hp
112
113
114 def BattleSimulator(teamA, decisionA, teamB, decisionB):
115     teamA.rest = 0
116     teamB.rest = 0
117
118     BattleFlag = 0
119     EscapeFlag = 0
120
121     if (decisionB != decisionA):
122         BattleFlag = 1
123         EscapeFlag = Escape()
124     elif (decisionA == 1) and (decisionB == 1):
125         BattleFlag = 1
126
127     teamA.fight = -1
128     teamB.fight = -1
129
130     if BattleFlag:
131
132         win = 0
133         teamA.hp -= teamB.atk * decisionB * BattleRNG()
134         teamB.hp -= teamA.atk * decisionA * BattleRNG()
135         if (teamA.hp <= 0) and (teamB.hp <= 0):
136             if teamA.hp < teamB.hp:
137                 teamB.hp = 1
138                 win = 2
139             else:
140                 teamA.hp = 1
141                 win = 1
142         elif teamB.hp <= 0:
143             win = 1
144         elif teamA.hp <= 0:
145             win = 2
146         else:
147             win = 0
148
149         if win == 1:
150             survival_teams.remove(teamB.id)
151             teamA.kills += 1
152             teamB.die = current_time
153             AtkIncrease(teamB, teamA)
154             HPIncrease(teamB, teamA)
155         elif win == 2:
156             survival_teams.remove(teamA.id)
157             teamA.die = current_time
158             teamB.kills += 1
```

```
            AtkIncrease(teamA, teamB)
            HPIncrease(teamA, teamB)
        else:
            if (not EscapeFlag):
                teamA.fight = teamB.id
                teamB.fight = teamA.id


def WinRate(hpA, atkA, hpB, atkB):
    roundsA = hpB / atkA
    roundsB = hpA / atkB
    rounds = math.ceil(min(roundsA, roundsB))
    return norm.cdf((rounds * (atkA - atkB) + hpA - hpB)
                    / (atk_uncertainty * math.sqrt(rounds * (atkA**2 + atkB
    **2))))

def DecisionMaking(teamA, teamB):
    # haven't considered third team

    #combat_effectiveness = np.array([teams[id].hp * teams[id].atk for id in
     survival_teams])
    ceA = teamA.hp * teamA.atk
    ceB = teamB.hp * teamB.atk
    #argsort = combat_effectiveness.argsort()
    #argsort = np.flip(argsort) # sort in descending order
    #combat_effectiveness = combat_effectiveness[argsort]
    P_stg = [P_stg1 if teams[id].stg else P_stg0 for id in survival_teams]
    P_total = sum(P_stg)
    Pa = (P_stg1 if teamA.stg else P_stg0) / P_total
    Pb = (P_stg1 if teamB.stg else P_stg0) / P_total

    # case run
    #rankA = np.searchsorted(-combat_effectiveness, -ceA)
    #rankB = np.searchsorted(-combat_effectiveness, -ceB)
    rankA = sum([1 - WinRate(teamA.hp, teamA.atk, teams[id].hp, teams[id].
    atk)
                 for id in survival_teams if id != teamA.id])
    rankB = sum([1 - WinRate(teamB.hp, teamB.atk, teams[id].hp, teams[id].
    atk)
                 for id in survival_teams if id != teamB.id])
    killA = (N_current_teams - rankA - 1) * Pa
    killB = (N_current_teams - rankB - 1) * Pb
    Arun = killA * score_kill + score_survive[int(rankA + 0.5)]
    Brun = killB * score_kill + score_survive[int(rankB + 0.5)]

    # case Alose Bwin
    Alose = score_survive[N_current_teams - 1]
    atkBwin = AtkIncrease(team_give=teamA, team_receive=teamB, in_place=
    False)
    hpBwin = HPIncrease(team_give=teamA, team_receive=teamB, in_place=False)
    #ceBwin = atkBwin * hpBwin
    #rankBwin = np.searchsorted(-combat_effectiveness, -ceBwin) # find new
    rank
    rankBwin = sum([1 - WinRate(hpBwin, atkBwin, teams[id].hp, teams[id].atk
    )
                 for id in survival_teams if id != teamB.id and id != teamA.
    id])
    killBwin = 1 + (N_current_teams - rankBwin - 2) * Pb
    Bwin = killBwin * score_kill + score_survive[int(rankBwin + 0.5)]

```

```
    # case Awin Blose
    Blose = score_survive[N_current_teams - 1]
    atkAwin = AtkIncrease(team_give=teamB, team_receive=teamA, in_place=
False)
    hpAwin = HPIncrease(team_give=teamB, team_receive=teamA, in_place=False)
    #ceAwin = atkAwin * hpAwin
    #rankAwin = np.searchsorted(-combat_effectiveness, -ceAwin) # find new
rank
    rankAwin = sum([1 - WinRate(hpAwin, atkAwin, teams[id].hp, teams[id].atk
)
                for id in survival_teams if id != teamA.id and id != teamB.
id])
    killAwin = 1 + (N_current_teams - rankAwin - 2) * Pa
    Awin = killAwin * score_kill + score_survive[int(rankAwin + 0.5)]

    roundsA = teamB.hp / teamA.atk
    roundsB = teamA.hp / teamB.atk
    # A0 B0
    Ra = Arun
    Rb = Brun
    # A1 B0
    NmaxA = math.ceil(4.5 * atk_uncertainty ** 2 + roundsA
                      + 3 * atk_uncertainty * math.sqrt(4.5 *
atk_uncertainty ** 2 + roundsA))
    Prun = sum([(1 - successful_run) ** (i-1)
                * norm.cdf((roundsA - i) / (atk_uncertainty * math.sqrt(i)))
                for i in range(1, NmaxA + 1)]) * successful_run
    Pwin = 1 - Prun
    Ta = Awin * Pwin + Arun * Prun
    Sb = Blose * Pwin + Brun * Prun
    # A0 B1
    NmaxB = math.ceil(4.5 * atk_uncertainty ** 2 + roundsB
                      + 3 * atk_uncertainty * math.sqrt(4.5 *
atk_uncertainty ** 2 + roundsB))
    Prun = sum([(1 - successful_run) ** (i-1)
                * norm.cdf((roundsB - i) / (atk_uncertainty * math.sqrt(i)))
                for i in range(1, NmaxB + 1)]) * successful_run
    Pwin = 1 - Prun
    Sa = Alose * Pwin + Arun * Prun
    Tb = Bwin * Pwin + Brun * Prun
    # A1 B1
    #rounds = math.ceil(min(roundsA, roundsB))
    #PA = norm.cdf((rounds * (teamA.atk - teamB.atk) + teamA.hp - teamB.hp)
    #              / (atk_uncertainty * math.sqrt(rounds * (teamA.atk**2 +
teamB.atk**2))))
    PA = WinRate(teamA.hp, teamA.atk, teamB.hp, teamB.atk)
    PB = 1 - PA
    Pa = Awin * PA + Alose * PB
    Pb = Bwin * PB + Blose * PA

    if Pa > Sa and Pb > Sb:
        Afight = True
        Bfight = True
    elif Pa < Sa and Pb > Sb:
        Afight = False
        Bfight = True
    elif Pa > Sa and Pb < Sb:
        Afight = True
        Bfight = False
    else:
```

```
        Afight = (ceA > ceB)
        Bfight = not Afight
    return (Afight, Bfight)


def GetDensity():
    return float(N_current_teams) / Areas[current_time]


def SelectRNG(stg, hp):
    density_factor = 1.0
    RNG = random.random() + GetDensity() * density_factor
    if ((stg == 0) or (hp < 20)):
        return RNG > (1 - P_stg0)
    else:
        return RNG > (1 - P_stg1)


def LootRNG():
    hpinc = 0
    atkinc = 0
    lootrate = 0.1

    rnghp = random.random()
    rngatk = random.random()
    if (rngatk < lootrate):
        atkinc = 10
    if (rnghp < lootrate):
        hpinc = 20
    return (hpinc, atkinc)


def SelectFights():
    newFightTeams = 0
    selected_teams = []
    for team in teams:
        if ((team.fight == -1) and (team.hp > 0)):
            if SelectRNG(team.stg, team.hp):
                # print(team.id)
                selected_teams.append(team.id)
                newFightTeams += 1
    # print(selected_teams)
    if ((newFightTeams % 2) == 1):
        newFightTeams -= 1
        selected_teams.pop()
    rem = newFightTeams
    for i in range(newFightTeams):
        p = random.randint(1, rem)
        selected_teams[p - 1], selected_teams[rem - 1] = selected_teams[rem - 1], selected_teams[p - 1]
        rem -= 1
    # print(selected_teams)
    for i_team in range(int(newFightTeams / 2)):
        teams[selected_teams[i_team * 2]].fight = selected_teams[i_team * 2 + 1]
        teams[selected_teams[i_team * 2 + 1]].fight = selected_teams[i_team * 2]


def Simulator(simu_id=""):
```

```
    # print("Simulating "+simu_id)

    global N_current_teams
    global current_time

    # f2 = open("survivalteams_"+simu_id+".csv","w")
    # f3 = open("battles_"+simu_id+".csv","w")

    # f2.write("round,team.id,team.maxhp,team.hp,team.atk,team.fight\n")
    # f3.write("round,teamA.id,teamA.hp,teamA.atk,teamB.id,teamB.hp,teamB.atk,decisionA,decisionB\n")
    for i_round in range(T):
        current_time = i_round

        # print("round",current_time,"teams=",N_current_teams,"Area=",Areas[current_time],"Density=",GetDensity())
        for team_id in survival_teams:
            team = teams[team_id]
            # f2.write(str(i_round)+','+str(team.id)+","+str(team.maxhp)+','+str(team.hp)+','+str(team.atk)+','+str(team.fight)+"\n")

        considered = np.zeros(N_teams, dtype=bool)

        for team_id in survival_teams:
            team = teams[team_id]
            if (considered[team.id]):
                continue
            if team.fight != -1:
                considered[team.fight] = True
                (decisionA, decisionB) = DecisionMaking(team, teams[team.fight])

                teamA = team
                teamB = teams[team.fight]
                # f3.write(str(current_time)+","+str(teamA.id)+","+str(teamA.hp)+","+str(teamA.atk)+","+str(teamB.id)+","+str(teamB.hp)+","+str(teamB.atk)+","+str(decisionA)+","+str(decisionB)+"\n")

                BattleSimulator(team, decisionA, teams[team.fight], decisionB)
                N_current_teams = len(survival_teams)  ### Here maintain number of remaining teams
            else:
                team.rest += 1

                (hpinc, atkinc) = LootRNG()  ### Looting
                team.maxhp += hpinc
                team.atk += atkinc
                team.hp = team.maxhp
            # if (team.rest > 1):    ### Recovering from last battle
            #  team.hp = team.maxhp
        SelectFights()  ### Select teams for next battles

    teams.sort(key=lambda x: x.die, reverse=True)
    cur_rank = 0
    for team in teams:
        team.rank = cur_rank
        team.score = team.kills * score_kill + score_survive[cur_rank]
        cur_rank += 1
    teams.sort(key=lambda x: x.id, reverse=False)
```

```
    '''
    f = open("scoreboard_"+simu_id+".csv","w")
    f.write("team.id,team.stg,team.die,team.rank,team.kills,team.score\n")
    for team in teams:
      f.write(str(team.id)+","+str(team.stg)+","+str(team.die)+","+str(team.rank)+","+str(team.kills)+","+str(team.score)+"\n")
    f.close()'''
    # f2.close()
    # f3.close()


### Brute-force simulations
N_rounds = 10
S_0s = np.zeros(N_teams + 1, dtype=float)
S_1s = np.zeros(N_teams + 1, dtype=float)

for gametype in [1]:
    print("gametype=", gametype)
    gametype = 15
    score_0s = []
    score_1s = []
    for i_round in range(N_rounds):
        (teams, survival_teams, N_current_teams) = InitTeams()
        for i_team in range(N_teams):
            if i_team < gametype:
                teams[i_team].stg = 1
        Simulator()
        for team in teams:
            if team.stg == 0:
                score_0s.append(team.score)
            else:
                score_1s.append(team.score)
    score_0s = np.array(score_0s)
    score_1s = np.array(score_1s)

    plt.figure(0)
    plt.hist(score_0s)
    plt.title(
        "Score distribution for strategy = 0 when number of 1s = " + str(gametype)
    )
    plt.xlabel("Mean score = " + str(np.mean(score_0s)))
    plt.ylabel("Number")
    plt.savefig("type" + str(gametype) + "game_0.png")
    plt.close()

    plt.figure(1)
    plt.hist(score_1s)
    plt.title(
        "Score distribution for strategy = 1 when number of 1s = " + str(gametype)
    )
    plt.xlabel("Mean score = " + str(np.mean(score_1s)))
    plt.ylabel("Number")
    plt.savefig("type" + str(gametype) + "game_1.png")
    plt.close()

    plt.show()

    print(np.mean(score_0s), str(np.mean(score_1s)))
```