GUNSLINGER

PROJECT 1

Programming and Problem Solving

Fall 2013 / Professor Ken Ross

GROUP 5

Neha Aggarwal

Priyanka Singh

Andrew Goldin

September 30, 2013

TABLE OF CONTENTS

1 PROJECT OVERVIEW

The primary goal of this project was to create an intelligent player for a game referred to as “Gunslinger”. The rules of Gunslinger are as follows:

- There are N players arranged in a circle, each armed with a gun. Each player has e friends and f enemies, where $e + f < N$ and an enemy cannot also be a friend.
- Friend relationships are symmetric, but enemy relationships do not have to be.
- On every round, each player can choose to fire a shot at another player in the circle. All shots are fired simultaneously.
- If a player is shot more than once in one round, they are dead. The game ends when 10 rounds have passed with no deaths.
- When the game ends, each player’s final score is determined by the number of enemies who are dead (1 point each), the number of friends who have survived (1 point each), and whether they themselves survived (1 point). The maximum score, then is $e + f + 1$.

Each player has access to the following information:

- A list of their friends and their enemies. A player only knows who her friends and enemies are, not the friends or enemies of any other players.
- On each round, the simulator provides each player with the previous round information (i.e. who shot whom, if at all), as well as who is still alive. The intent is for the player to use this information to decide her next move.

The ultimate goal of every player is to maximize her average score across multiple games, with a wide variety of values of N and enemy/friend configurations, with all players using the same set of round information within each game. The catch, however, is that players are not able to store or recall information about previous games. Therefore, a player’s performance is based upon how rationally they can act given only the information about the game she is presently in.

2 STRATEGY & EVOLUTION

At the start of the project, each of the 9 teams in the class was given a file for a player that implements a “dumb” strategy. That is, on every round the player will randomly choose a non-friend, and fire with an 80 percent chance of success. The goal of our team, similarly to the other teams, was to improve this “dumb” strategy by implementing (and subsequently improving upon) some rudimentary A.I. that would help our player make smarter decisions about when and who to shoot.

To try to make our player smarter, our group first established a list of conditions for the player to consider on each round when deciding whether or not (and who) to shoot:

- Who is a friend and who is not (enemy or neutral).
- Who has fired at our player in previous rounds.
- Who other players are shooting and their relationship to our player.
- Who has fired at our friends in previous rounds.

2.1 FIRST ROUND CONSIDERATIONS

The first thing we wanted to accomplish upon receiving this “dumb” player was to figure out how to more “intelligently” choose who to shoot on the first round (if anyone). We initially decided to base our decision on how many friends, enemies, and neutrals we had.

If our friends were in the majority, we would not shoot. Because there is a low level of danger in this configuration, we believed it would not be advantageous for our player to incite any action, primarily in taking precautions to not get our friends killed. If friends were in the minority, we would target a random non-friend. Additionally, if neutrals were in the majority, we would target a random neutral. The line of thinking here was that if the majority of other players were neutrals, then the group of players targeting us is likely to be concentrated in that group of neutrals, and so to prevent ourselves from getting killed we will target a player from among that group.

After the first rounds of testing during class discussion, we quickly discovered the primary flaw in our initial reasoning. Even if probability dictates that the group of players targeting ours is concentrated among the neutrals, the probability that a non-friend is targeting ours is the same for any given player. It therefore stands that we have nothing to gain by

targeting a neutral, since any enemy would be equally likely to be targeting us, and we gain points for eliminating them.

Taking these thoughts into consideration, we simplified our logic by instead always targeting a random enemy on the first round. If there were no enemy relationships, our player would not shoot. The line of thinking here is that we will always have something to gain by targeting an enemy on the first round. Since a number of other groups' players did not shoot on the first round, we may have been able to gain a first round advantage in points.

After analyzing the results of multiple games with this new implementation, we found that our player was excessively trigger-happy, and was therefore being consistently targeted after the first round, thus significantly decreasing our end score. Our final decision was to have our player remain silent on the first round to minimize provocations and allow us to make observations about other players' behaviors before deciding who to shoot.

2.2 WHO IS A FRIEND AND WHO IS NOT

The first and perhaps simplest thing to consider when deciding who to shoot is whether they are a friend, an enemy, or neutral. In our implementation, we started out with giving a small amount of weight to all non-friends, to ensure that by the end of the weighting process, no friend can have the same weight as a non-friend and become chosen as a target.

In later iterations of our player, at the same time that we decided that it may be ideal to target enemies as often as possible, we modified our logic to give extra weight to all enemies. This would ensure that two potential targets who would otherwise tie in weight would have the tie broken on the basis of enmity.

Through testing, we discovered that giving weight to neutrals on the basis of not being a friend was troublesome because it resulted in some cases where our player would regrettably target a neutral when it would have been more advantageous to target an enemy, even if the neutral player held more weight. As a result, our final modification was to have the player only give weight to non-friends who are enemies, at a default base value, and to not assign any weight to neutral players unless they could be considered a threat. We discuss ways in which a neutral can present a threat in upcoming sections.

2.3 WHO HAS FIRED AT OUR PLAYER IN PREVIOUS ROUNDS

Another consideration for this game (and to us the most important one) is to assess each round and discover if anyone fired at our player. If so, it is likely that we are an enemy of the shooter, or that they consider us a threat for some other reason. Perhaps we targeted one of their friends in the past, or we are their only non-friend remaining in the field. Regardless, it is reasonable to assume that the assailant is likely to shoot us again, and so the logical course of action is to retaliate in self-defense.

In our first implementation, we assigned the maximum weight to any player who fired at us in the previous round. After several rounds of testing and discussion, we discovered that we were getting similar results to those described in the previous section. That is to say, our player was targeting neutrals more often than desired. To address this, we performed a similar modification, assigning more weight to a player who shot us if they are an enemy. The intent, again, was to minimize the number of undesired ties that occur in the weighting process by favoring enemies.

Even with this modification, there were still a number of circumstances in which our player was making questionable decisions. There are a few reasons for this. At this point we are only considering actions that take place in the previous round. If a player targeted us two or more rounds ago, we may be their enemy, but our player will cease to consider the possibility once the most recent round information arrives. As a result, our player ends up letting some shots slip by and only targets people that shot them most recently.

Armed with this knowledge, we modified our player to keep a complete history of each round of the game. To make sure not to lose track of threats, weight was assigned every round to any enemy who ever shot our player. Then, additional weight was assigned to enemies and neutrals who shot our player in only the most recent round. Finally, extra weight was assigned to players who targeted our player more than once consecutively since the previous round:

$$ShooterWeight_{i+1} = ShooterWeight_i + WeightConstant_{shooter} \times ConsecutiveShotsFired$$

As a result, our player is able to more accurately identify who may actually be targeting our player as the result of an enemy relationship, as opposed to just shooting us as a kneejerk reaction to some recent event. Using this tactic, we can more tactfully eliminate enemies while also managing other threats to our player.

2.4 WHO OTHER PLAYERS ARE SHOOTING

This started out as a minor consideration for our group's player but quickly evolved over time as we realized that it would be a major contributor to our player's predictive nature. This is because the player can use all of the shooting information from round to round to make predictions as who to who may be consistently targeted or who may retaliate against another player.

For example, say our player discovers that enemy A has shot player B. Under the assumption that, like our player, most players are prone to retaliation from being shot, we can predict that the B will fire back at A in the next round. Thus, A is likely to die if we target them, since two shots on one player is a guaranteed death.

In our early implementations we only assigned weight to enemies who had been shot by a friend in the previous round, since it seemed like a good idea to focus our efforts on helping friends get successful kills so they would survive to the end. In a later iteration, after introducing full game history tracking, we expanded this logic to include enemies who were shot by any other player. We base this on the notion that any of our enemies who are being fired upon are likely to be shot more than once, and will therefore give us a chance to swoop in for the kill. In this case, we assign more weight to enemies who have been shot by a friend in an attempt to increase our friend's chances of survival.

Additionally, to make sure that we did not provoke any neutral players unnecessarily, we would only assign weight to neutral players if they have ever tried to shoot our player specifically, and would only give them significant weight if all of our enemies have already been defeated.

While these improvements were certainly welcome, this implementation disregarded non-friend players getting shot *by* our enemies, and so our player ended up missing out on a great deal of opportunities to eliminate enemies based on the victim's likelihood to retaliate. To address this, we added some extra considerations to our weighting system in further iterations:

- If an enemy shot a non-friend, assign weight to the shooter.
- If a friend or neutral shot an enemy, assign weight to the victim.

This system ensures the maximum weight is given to enemies who have shot others *and* have been fired on at least once, since their chances of survival are the lowest.

2.5 WHO HAS FIRED AT OUR FRIENDS IN PREVIOUS ROUNDS

Since the number of friends who are left alive in any given game contributes significantly to our score, we thought it important to devote resources to protecting our friends. The idea is simple: assign weight to anyone who shoots a friend. At the onset of the project, protecting friends was our second highest priority (behind protecting ourselves), and our player would give the second highest amount of weight to anyone who shot one of our friends in the previous round. After some class discussion and contemplation, this was deemed problematic for a number of reasons.

As stated in the assignment description on the professor's website, "a friend of a friend could well be an enemy!" If one of our friends shot another one of our friends, we would give weight to that friend, meaning that we may risk shooting a friend and decreasing our final score. In later iterations of our player, we took care of this issue by only considering friends who were shot by enemies or neutrals. In the spirit of the other portions of our implementation, extra weight was given to enemies in this consideration in an attempt to further decrease the number of ties in weight between enemies and neutrals.

Even after all these enhancements, we were discovering a similar issue to what we encountered when implementing our "protect ourselves" logic. The issue being that we only concerned ourselves with actions taken solely in the previous round of the game. After including history tracking, our function for assigning weights for friend protection was also modified to take into account consecutive shots fired on each friend. That is to say, our final strategy was to assign weight to any enemies who have every shot a friend, and then assign additional weight to any enemies or neutrals who have shot that friend more than once consecutively since the previous round, with additional weight given to enemies.

The result of this essentially dictates that our A.I. will consider the safety of friends much in the same way as it considers its own safety, just on a slightly smaller scale, in order to preserve our hierarchy of priority.

The ultimate culmination of each of these four considerations results in an algorithm that allows our player to exhaustively assess its environment and make the most educated guess it can about whether to shoot, and who to shoot.

3 NOTES ON OVERALL PERFORMANCE

3.1 REVISION OF SHOOTING STRATEGY

In the first deliverable, we had based our shooting strategy more on the initial configuration of players in the game. We categorized the players as friends and non-friends. That being said, we chose to target the neutrals over enemies, if they were in majority.

However, post the class discussion, we realized the below points:

- Enemies and neutrals need to be treated distinctly. Neutrals should not be labelled as potential threats and targeted unless they do something to harm us. This harmful intent could be in the form of a neutral shooting our player or a neutral shooting a friend.
- It is rational to always shoot an enemy over a neutral irrespective of the game configuration as killing an enemy fetches us a point.

From our takeaway from class discussions, we noted that the players that got shot in a round were likely to be targeted again in the next round. So it seemed prudent to assign more weight to such players if they were our enemies. In fact, it later led us to best-exploiting the case of one of our enemies shooting another enemy in a round. We took advantage of the retaliation factor of the victim enemy and added weight to the enemy shooter.

3.2 TAKEAWAYS AFTER BEING TRIGGER-HAPPY

One move that became a major learning point for us was entering the game as a *trigger-happy* player. Our performance and the class discussion that followed provided us useful insights into the game and steered us towards the concept of group dynamics. We made the below observations:

This is a dynamic game of incomplete information. In order to devise a rational strategy, every player tries to gain information as the game progresses. Since there are only two possible actions available to a player i.e. *shoot* and *don't shoot*, shooting in every round gives away substantial information to other players. This information can be processed by

their A.I.s to deduce relationships (especially enmity) and use it to their advantage.

By this time, almost every player had incorporated some form of collaboration feature into their strategy. For instance, if a player observed a friend shooting a common enemy, she would join hands with her to eliminate the enemy. This further put a trigger-happy player on the spot as shooting constantly led to the formation of multiple such alliances against her.

Being trigger-happy antagonized the neutral players as well. They were inclined towards eliminating the frequent shooter to ensure their own survival, to save friends etc. Perhaps motivated by long term benefits, there was a scenario where players even shot a trigger-happy friend.

The only games in which the trigger happy player performed well were the ones in which other players were trigger happy too. The players had started exhibiting similar behavior (they created similar events with variations observed in the priorities) Thus, we deduced that the key to improving our strategy was to go with the flow. Any deviation from the group behavior was proving self-destructive.

3.3 STRATEGY REFINEMENTS

We introduced the below improvements to our strategy and tested it across multiple game configurations:

- We modified our strategy to never shoot in the first round.
- We adapted the policy of not shooting an enemy unless provoked. This was a move to exploit the non-symmetric feature of enemy relationships. It helped us against retaliation from our enemies (for whom we were neutrals), their potential reinforcements (the players who would back them up, had we shot them). Ultimately, it helped us boost our performance.
- In fact, our precision and kill-rate improved substantially and almost every shot that we fired resulted in a kill.
- We tried to become smarter and use the incidence of an enemy shooting in a round, irrespective of the identity of the victim. We assigned the above enemy some weight. This decision was based on our observation that by firing a shot, a player makes herself vulnerable to getting shot in the next round.

3.4 PERFORMANCE EVALUATION ACROSS DIFFERENT DELIVERABLES

To evaluate our player g5's relative performance across different deliverables, we ran 1000 iterations for a 10 player game with below configurations.

- $N = 10, e = 2, f = 2$
- $N = 10, e = 0, f = 9$
- $N = 10, e = 9, f = 0$
- $N = 10, e = 5, f = 4$

The player list includes an instance of every group's player along with the original version of dumb player (provided to us at the beginning of the project). The line charts 1 and 2 depict the results obtained from the test runs.

Since our primary performance evaluation metric is *average score*, we have plotted two line charts that project the score differentials- *Top-Average Score – Average Score of g5* and *Average Score of g5 – Mean of the Average Score* for each of the four configurations across different deliverables.

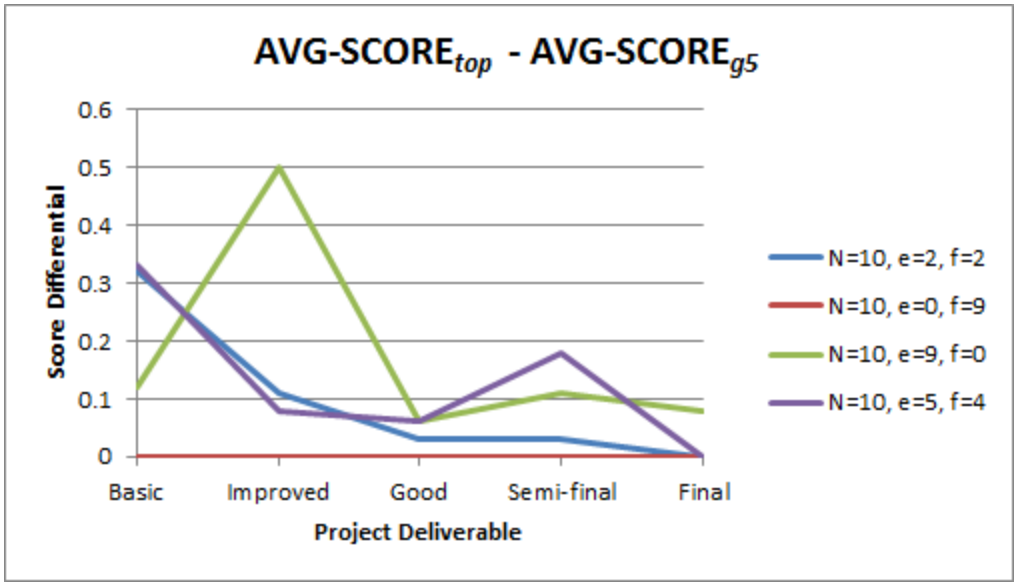


Chart 1

Looking at Chart 1, we notice that as we progress towards the final deliverable, the score differential for all configurations seems to be approaching zero. This may be attributed to the fact that our player's overall strategy improved over time and neared the top score towards the end of the project. It is worth mentioning here that we may expect a similar

behavior by other players owing to the fact that over time, all players seemed to be converging towards a common strategy of passivity. This group behavior would result into an even spread of scores.

For Chart 1, let us analyze the peak point at Improved functionality deliverable for the configuration N=10, e=9, f=0. This was the stage when our player became trigger-happy. The drastic dip in our player's performance here provides ample evidence that it is irrational to be trigger-happy (when others are not) in an all enemy scenario. The decision to shoot in a round leads to a high probability of getting killed in the next one.

$$P(Dying) = \frac{\#\ players\ likely\ to\ shoot}{\#\ players\ alive}$$

...which almost approaches 1 as each player alive is an enemy and equally likely to shoot.

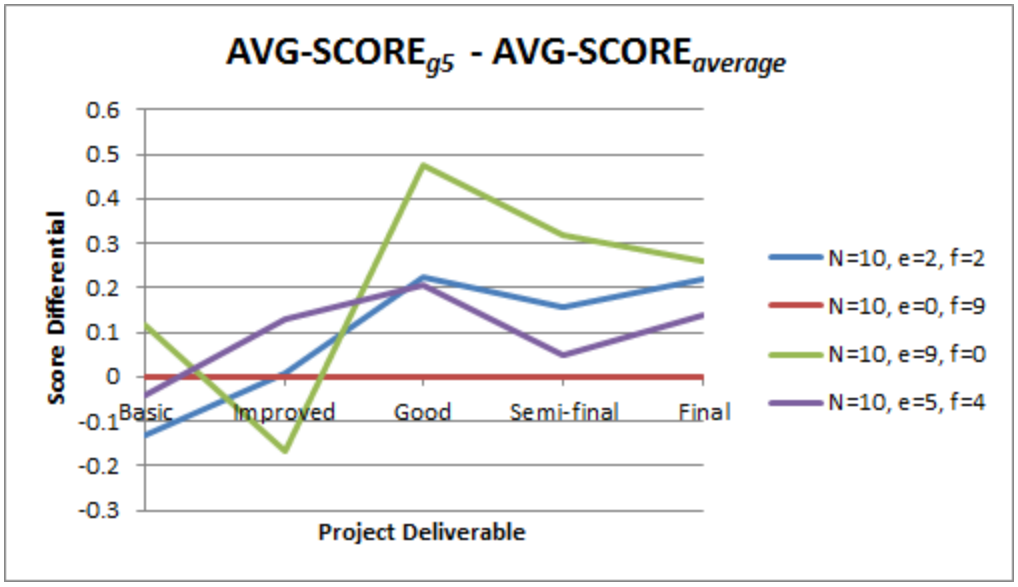


Chart 2

Chart 2 helps us in analyzing our player's performance with respect to the average performance of the group of players. Looking at the overall picture, we can deduce that from Good Functionality deliverable onwards, our player g5's performance has been fairly above average across all the configurations.

Our performance was below the average score for an enemy-saturated game during the Improved functionality deliverable (the reason for the same has been discussed during the analysis of Chart 1).

Another interesting point to note here is the convergence of the lines towards the final deliverable. This behavior could possibly be due to the convergence of the strategies employed by different players and thereby leading to an even distribution of the average scores. Towards the end of the project, almost all the players adopted the strategy of becoming passive. The idea of becoming trigger-happy was collectively frowned upon post observing the quick elimination of the trigger-happy players in a game. Ultimately, the players became frugal in taking shots, modifying their AIs to shoot only when a significant event was triggered.

4 ANALYSIS OF TOURNAMENT RESULTS

For this project, each group's final player version was placed into a multi-game tournament to see whose player achieved the highest average score across a bevy of various game configurations. The tournament itself was run in two halves: one with no "dumb" players included, and one with two different types of dumb. Each game configuration was run 2000 times with the inclusion of dumb players (1000 for each dumb player) to make the variance reasonably small. Without dumb players, games were only run 10 times because almost every configuration ended in a tie (more on this below). We have compiled results from a number of configurations in this section.

4.1 WITHOUT DUMB PLAYERS

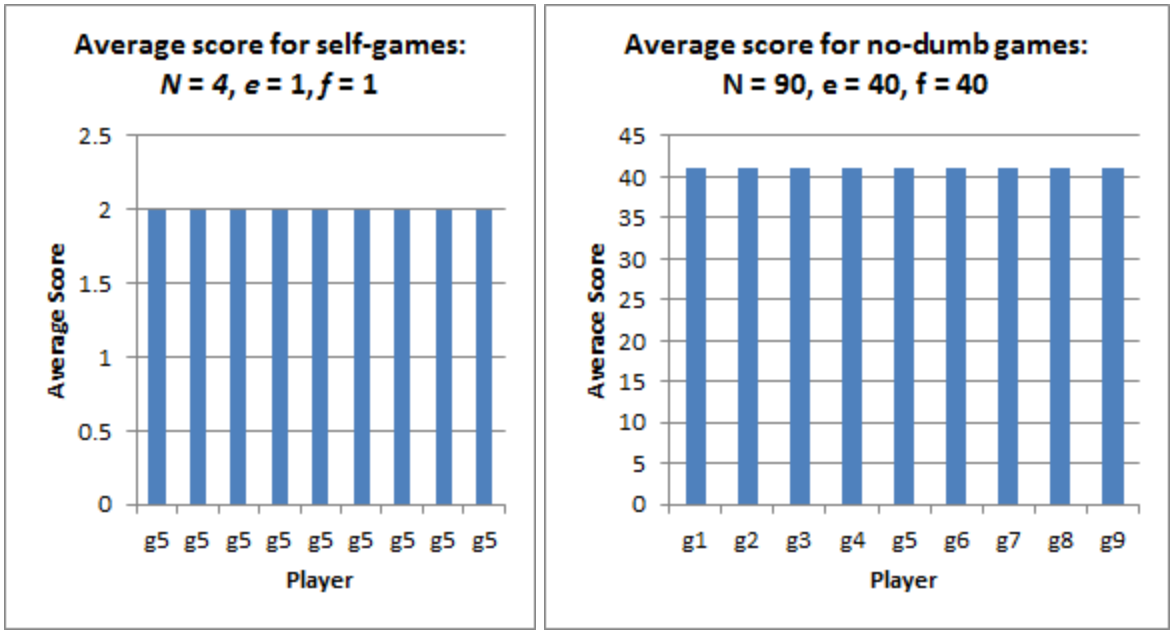
First we will take a look at results of games conducted with no dumb players involved. Specifically we will look at two cases: one in which every player in the game is our own implementation, and one in which multiple instances of each player are included in the game.

When observing the results as a whole for this half of the tournament, there is a very standout but nonetheless interesting observation: almost all players had nearly identical average scores in every configuration (with a very low variance), and those scores were usually consistent with the number of friends present in the corresponding game. There is a very likely reasoning for this. Over the course of discussions, the class noticed that players who were trigger-happy were often eliminated from play early and could not play

to increase their own score, while more conservative players had a higher survival rate and were able to take more effective shots.

As a result of this, almost every group decided to make their players more passive, especially when considering first-round shots. Everyone had a similar general starting strategy: wait out the first round to see what happens, then start making decisions. However, since no one shoots on the first round, most players are unable to make an educated choice about who is a good target, and continue to remain silent. This continues into a feedback loop until 10 rounds have passed, at which point the game is over and everyone has the same score. Because all players are left alive, all friends remain, and no enemies are killed, every player's final score equals $f + 1$.

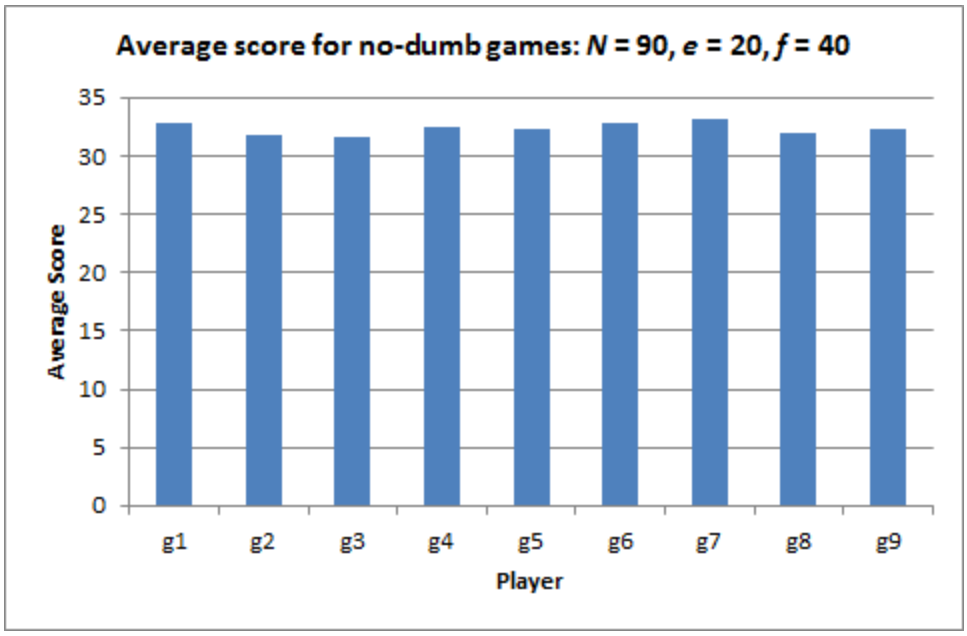
The following two charts show the average scores of players over 10 games. The chart on the left displays a game configuration in which there are 4 of the same player (specifically, our group's player), and each player has 1 friend and 1 enemy. The chart on the right shows a configuration in which there are 10 of each player, and each player has 40 friends and 40 enemies.



In the first graph it can be observed that for games that include only instances of our player, each player's average score came out to exactly $f + 1 = 2$. This is because our player never shoots in the first round, and in later rounds never shoots unless provoked. Since all players are our player, no one was ever provoked, resulting in a silent game where everyone has the same score.

A similar observation can be seen in the second graph, where despite multiple instances of every player being present, no player ever considered themselves threatened, and thus remained silent and garnered an average score of $f + 1 = 41$.

Our player technically had the 6th highest average score in this circumstance, but the differences almost seem negligible. If there were any reason for this result, it would be that our player tends to underperform in games with a relatively large value of f (more on this observation in the next section).



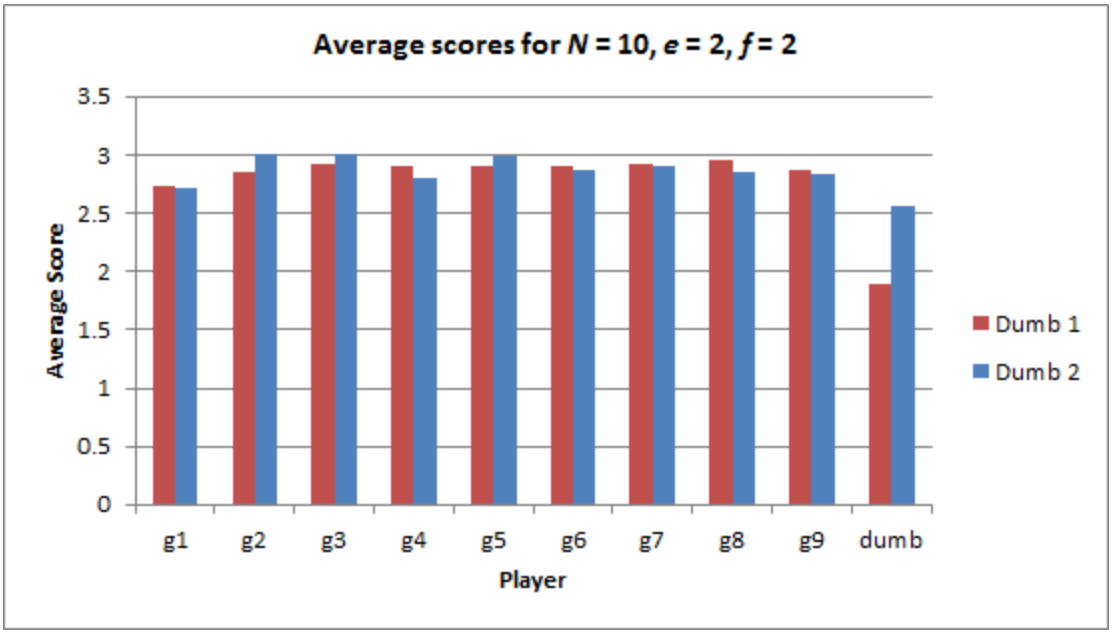
4.2 WITH DUMB PLAYERS

Because the absence of dumb players resulted in most games ending up a wash, another bevy of games was run with the inclusion of one or more dumb players to incite action in the field. Two implementations of dumb player were introduced into the mix:

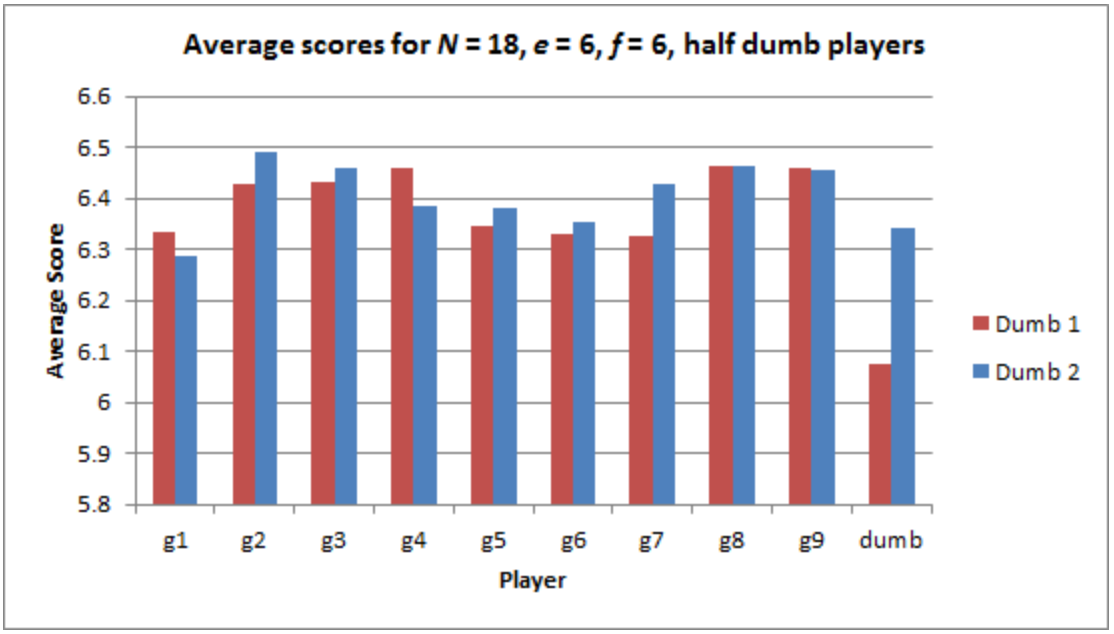
- **Dumb 1:** Always shoots a random non-friend, assuming one is alive.
- **Dumb 2:** Always shoots the first living player on their enemy list.

Because the outcomes of these games were more statistically varied, 1000 trials were run for each configuration to try to produce reasonably accurate results.

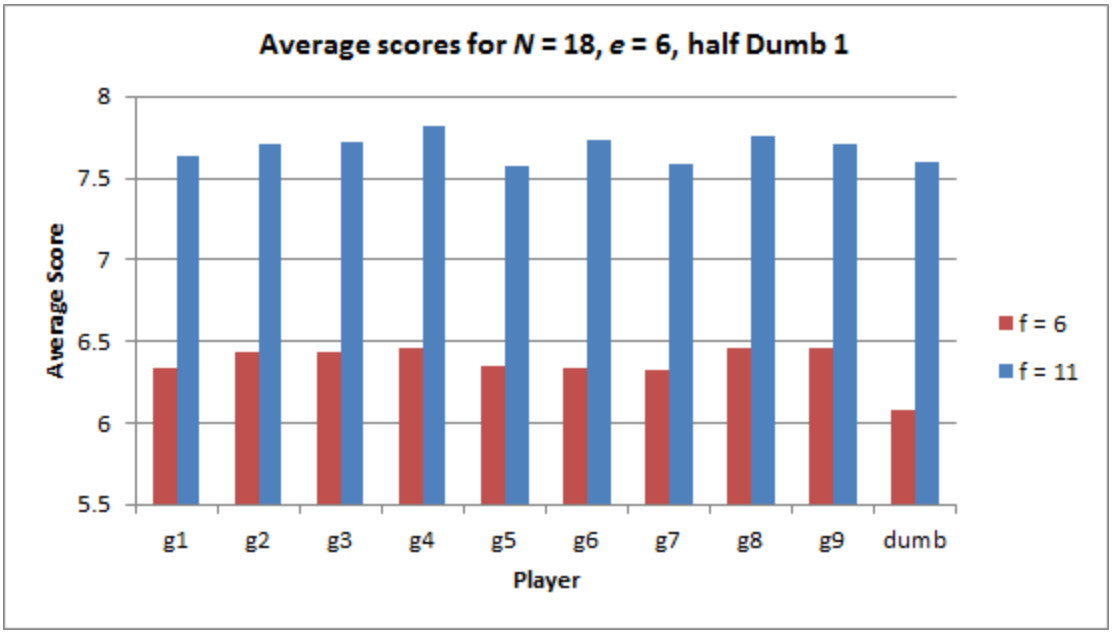
We will start out with a game that has a somewhat balanced configuration. The graph below represents the average scores for a 10-player configuration, in which one player is dumb. For both dumb implementations, all players outperformed the dumb ones, although Dumb 2 seems to have fared reasonably well in this configuration. Aside from the inclusion of the dumb players, every other player maintained an average score of between 2.5 and 3.0 (a variance of 0.017). Group 5 (our group) seems to have managed slightly better than most, but it is difficult to infer from the data alone whether the differences are meaningful.



The numbers truly start to differ when more dumb players are introduced into the mix. The graph below shows an 18-player configuration with an even distribution of friends, enemies, and neutrals. This time, half of the players in the game are of one dumb implementation. This, coupled with the larger size of *N*, resulted in a great deal more shooting amongst the players. From the data, it seems as though most of the shooting was directed at the dumb players, although the variations in performance between the other players is more noticeable. The strategies for groups 8 and 9 appear to be consistent despite the different behavior of the dumb player. Our group has underperformed in both situations. The likely cause of this is the increased number of trigger-happy players in the field. Although an increased number of players shooting means that our group has more information to work with when deciding who to target, it also means that more players are getting killed at faster rates. Since this particular configuration has the same number of friends as enemies, it stands to reason that our group may have not been so successful in saving our friends, thus decreasing our relative score.

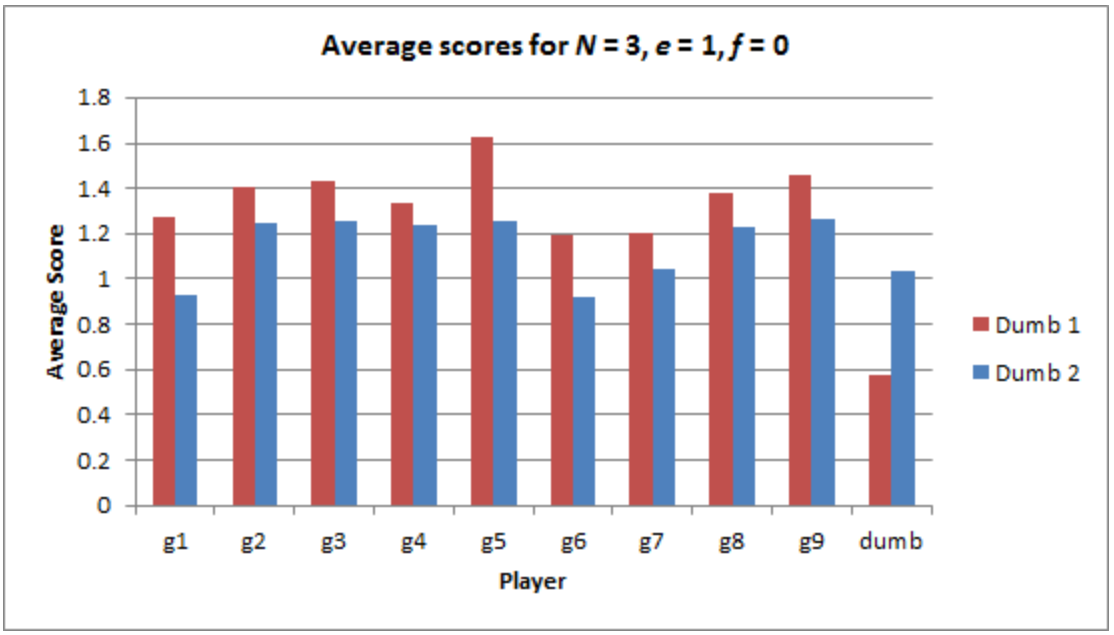


The table below further illustrates this notion by comparing the previous results with those of a series of games with the same configuration, except the number of friends is increased from $f = 6$ to $f = 11$.



Our group's underperformance relative to other groups becomes amplified, as we now fall behind Dumb 1 in terms of average score. It can be deduced that we tend towards underperformance in high-friend games, likely because we let too many friends get shot.

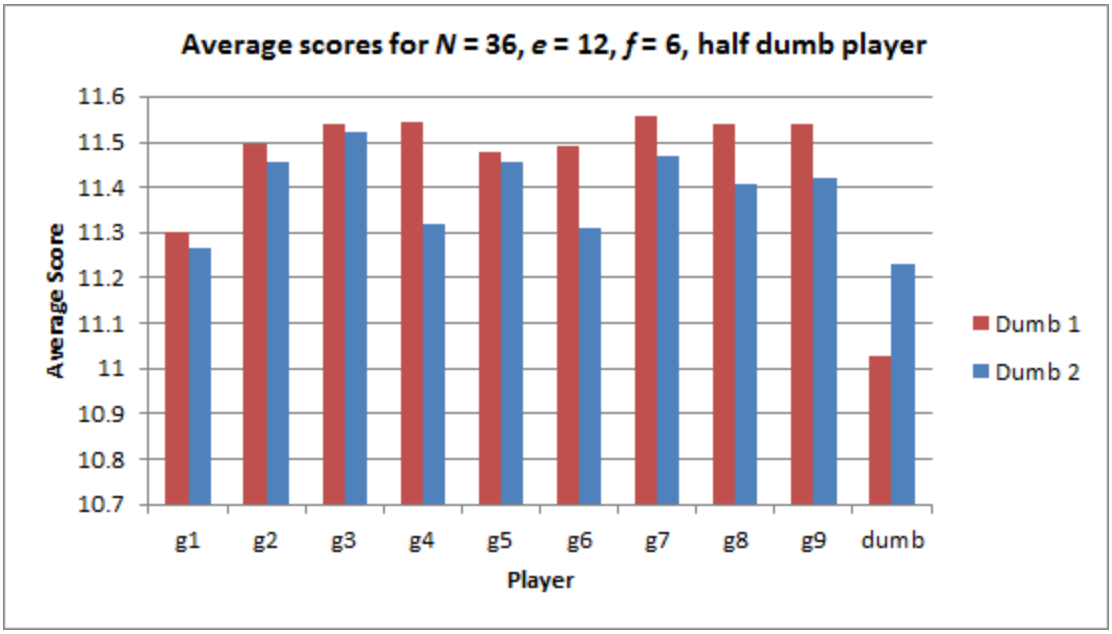
Now that we've observed our performance with mid-ranged values of *N* (by which we mean "mid-ranged" in the context of values used in the tournament), we will observe performance in cases where *N* is low and high.



The table above represents the outcomes of games with only 3 players and 1 enemy, in which one player is dumb. Our player performs very well in this circumstance, being the only player to exceed an average of 1.6 points. This is likely because our player has a high rate of successfully landing kills in this configuration. We can assume that the dumb player will always fire first. If they fire at an enemy, we team up with them to kill the enemy. If they *are* the enemy, we will retaliate if they shoot us. If the enemy is Dumb 1, eventually they will shoot the other player, at which point both other players will "conspire" to kill Dumb 1. If the enemy is Dumb 2, then our chance of success is contingent upon whether the third player decides to protect us or help the dumb player. Assuming we don't die on round 1, then the probability of us getting a kill is relatively high. Because Dumb 2 is more successful at eliminating its enemies, everyone's absolute score in this configuration is lower than with Dumb 1.

The table below shows a configuration with a larger number of players, *N* = 36. Each player has 12 enemies and 6 friends, resulting in a more enemy-heavy game. Similarly to an earlier configuration, half of the players in each game are of a dumb implementation. Our group underperformed six other groups in the Dumb 1 implementation, but managed third best in the Dumb 2 implementation. Since Dumb 1's strategy is likely to needlessly antagonize other players, their performance was low, boosting everyone else's numbers.

Additionally, because of Dumb 1's unpredictable nature, our player probably had a difficult time influencing the survival rates of friends and enemies. In the Dumb 2 implementation, our player likely had a slight edge on getting kills since targeting a dumb player who is not aiming at you will not alter their behavior.



4.3 OVERALL OBSERVATIONS

Based on the data gleaned from the tournament, a handful of overall observations can be made about our performance.

- Our player performs at or above average in most mid-range configurations. This is because our player has a simple and consistent but well rounded strategy that can perform well in a wide spectrum of arrangements. Our A.I. is designed to try to make every single shot count as often as possible.
- Our player does well in games with few players and a high number of enemies, where the maximum scoring possibility is relatively low. This is because it is much easier for our player to land a higher volume of effective shots.
- Our player does not do well in games with many players and a high number of friends. These configurations make it difficult for our A.I. to decide between getting kills and saving friends, and this indecisiveness makes us more prone to dying, and losing friends.

5 FUTURE IMPROVEMENTS

We could have used round history in a better way and taken into account each player's behavioral pattern like her retaliation factor (the ratio of the number of times she retaliated to the total number of times she was shot), reinforcements (possible number of players to back her up) etc. to judge them as a likely target. A secondary weight system could have been developed to break ties on the basis of the above factors.

At present our weighting system is not very refined. It could have been worth attempting to key in statistically generated values for weights.

As of now, we maintain a constant strategy across all configurations. Perhaps making it flexible enough to incorporate various cases (including special treatment of edge cases) could have improved our overall performance..

There are certain observations that we could have used to improve our strategy. For instance, if neutrals are fewer in number, the enmity relationships tend to become symmetric. We could have tried to use such observations to further refine our strategy.

Our strategy is based upon heuristical learning rather than mathematical theorems. Perhaps we could have formulated a better strategy based upon probabilistic reasoning. If we could take into account the possible set of actions for every player in each round and assign a probability (within reasonable limits) to each possible action, we might have been able to devise a more rational strategy.

6 CONCLUSION

Our performance analysis across the tournament as well as our self-analysis across different deliverables both corroborate to the fact that our player has a simple but a well-formulated overall strategy. From the very beginning, we intended to optimize our strategy across a wider span of configurations, which we seem to have achieved. Our player performs quite well in games that have enemy-friend balanced configurations to enemy-loaded ones. However, we do not seem to do very well in games having many players and a high proportion of friends. One possible reason for this could be that our A.I. is not equipped with sufficient logic to deal with the complexities that arise with

friend-heavy games. There are a number of ways that we can think of for improving our performance in friend-heavy games like implementing a secondary weight system, refining the weights based on statistics etc. But, at this point, we have decided to adopt a generalized approach and not deep-dive into the intricacies of the particular cases.

7 APPENDIX

7.1 INDIVIDUAL CONTRIBUTIONS

NEHA

Contributed some code for the project deliverables, primarily in the priority manager, which assigns weights to players. Held chief position in discussing strategy after each class, and contributed to performance analysis of the player. For the report, contributed to the strategy section, crunched the tournament data for the charts and graphs, and created the presentation slides.

PRIYANKA

Contributed greatly toward designing and implementing strategy refinements between each class discussion. For the report, wrote the sections on overall performance analysis, self-performance evaluation, and future improvements.

ANDREW

Contributed to most of the coding for the project, creating the initial priority manager class, as well as the game history class, which contains a number of convenience functions for accessing specific metrics from the game history. For the report, contributed to the strategy section and wrote the analysis of the tournament results.

7.2 ACKNOWLEDGEMENTS

Our team would like to thank our TA Jiacheng for implementing a great simulator and running the tournament, and Professor Ross for inciting thought provoking class discussion. We would also like to thank the other groups for allowing us to use their code for testing at each project deliverable.