CIS 559 PROJECT 3: GUNSLINGER

IAN SIBNER, DAVID LIAO, NIKHIL ROY

20 October 2015

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Gunslinger is an n-player game where each player is initialize with two disjoint sets of f friendly players (*friends*) and e enemy players (*enemies*). This implies that $e + f < n$ since the player will not have itself as a friend or enemy. The friend relationship is symmetric, so if A is a friend of B, then B is a friend of A; the enemy relationship, however, may be asymmetric.

Each turn, each player can fire one shot in an attempt to kill another player. However, a player is not killed unless two players shoot it simultaneously. Once dead, a player can no longer shoot. The game ends when the set of living players remains unchanged for 10 rounds in a row, whether or not players are still shooting. Each players' objective is have the highest number of points at the end of the round. Points are awarded as follows:

1. One point for surviving until the end of the round.
2. One point for each surviving friend (maximum f).
3. One point for each dead enemy (maximum e).

Players retain perfect information about who shot whom in previous rounds.

2 INITIAL INSIGHTS AND OBSERVATIONS

Immediately, our team observed several key features about the game.

1. While players are given a list of allies and enemies, there are more opponents than just those given. Players who we perceive neutrally could target us as an enemy. This lead us to believe that we might consider targeting those players as well to preserve our wellbeing.
2. A way to model the state of the game as perceived by our player would be to create a dynamic enmity graph. This would help us keep track of the relationships between all players
3. We needed to consider whether or not an aggressive or non-aggressive approach would be better. As a proactive player, targeting our enemies might lead others to

target us (if we target their allies and they choose to defend them). On the other hand, taking a retaliatory approach might limit our influence on the game end state.

4. Winning the game meant having the highest score. By killing off other players, we would be able to raise our own rank by lowering other players' score.

These insights guided our strategy throughout and allowed us to focus on the most important aspects of the game.

3 STRATEGIES AND CONCEPTS

3.1 David's Revenge

David's Revenge was our initial attempt where we tried to create a retaliatory player. The main idea was to maintain our own player's wellbeing while minimizing involvement if at all possible. The only times that David's Revenge would take action is if an arbitrary player targeted our player.

We initially construct an enmity graph. An enmity graph captures which players seem to be enemies of others. This provided us with more than enough information to design our player's decision making. We only consider any player that targeted us.

Using our enmity graph, we identify players that have targeted us and choose a player to return fire. The choice in player evolved through many designs. Initially it was the most recent player that shot us, but later on it became a weighted selection based on the number of times the player targeted us. We used a bounding "window" to make more recent hits more relevant as opposed to more temporally distant hits.

If a player stopped targeting us, as long as they are within our defined time window, we would still target them. If we had no player target us at all in the first place, we would simply take no action.

3.2 Protective Farmer

The Protective Farmer was a reaction to the results of the preliminary test run during the first several classes. XOrganism was not faring so well later on in games with other, more aggressive, players, even when it did well in the early stages of the game. As well, due to XOrganism's courteous moves, it would lose valuable coverage of the food-populated cells on the board.

The Protective Farmer was an experiment to see if these issues could be mitigated by a completely different strategy. The core idea behind the Farmer was that if there were four organisms on each adjacent cell to a cell with food, they could all 'farm' this cell. Staying off of the cell would allow it to duplicate freely, while the adjacency pattern would prevent opposing players from reaching the food source.

The initial stage of the Protective Farmer, before farming a food source, is to find a source of food that could support this configuration. In games with more organisms, many of the cells were covered up quickly, or enemy organisms prevented the Farmer from reaching other cells. The solution for this problem in Protective Farmer was almost the exact opposite of the movement strategy in XOrganism. The Farmer would move onto a new, unexplored cell on every turn (unless it needed food). It's goal was to find a cell with greater than four units of food on it as the Farmer may have needed to eat and will have to reproduce three times all without eating up the remainder of the food.

Once it found a satisfactory source food cell it transitioned into a building phase. The Farmer would look at all adjacent tiles and see if any were filled. If more than two were empty, then the farmer reproduces a wall onto one of those cells. This continues until there is only one cell left uncovered, which the Farmer simply moves to (instead of reproducing on to) to become the final wall. This Farmer organism and the newly spawned Farmer players are set to a Protective mode. This means that the organisms will stay still on their own cell unless they need food. If they do, they will attempt to move inwards. Once they replenish their energy, they move back onto the open cell they left earlier. In addition, when they move into the center cell, they also transition back to the building phase to fill any gaps in the wall that may have formed if any other Farmer organisms died off.

3.3 David's Revenge's Revenge

As the name suggests, David's Revenge's Revenge (DRR for short) was built directly on top of David's Revenge. Determining who to shoot is the exact same as David's Revenge: when DRR shoots, it will shoot the enemy who shot it most in previous rounds. However, unlike David's Revenge, DRR does not shoot whenever it can. It uses a logistic function in order to determine a *shooting percentage*, and then shoots with a probability equal to this function value.

The intuition behind the logistic function is that in situations with lots of friends, it is better to be conservative with shooting. The friends of the enemy you shot are likely to retaliate in subsequent rounds, so a player that is too aggressive in these situations is likely to die. However, in situations with few friends, being aggressive makes more sense because retaliation is unlikely. This applies regardless of the number of actual enemies; it generally pays off to target neutral players as well, as long as there are few friends, because it improves a players' *relative* ranking compared to other players. It was noted by Spriha that this might bring down a players' ranking in certain situations because it would eliminate another player's enemy; however, as we will see in the results section, we actually did quite well in situations where there were many enemies.

Thus we used the ratio of non-friends to players as the independent variable in the logistic function: $x = \frac{n-f}{n}$. Then the shooting percentage is calculated as $P = \frac{1}{1+e^{-\frac{x-\mu}{\beta}}}$, where β and μ govern the steepness and the mean of the distribution respectively. Initially, our logistic curve had $\beta = 0.07, \mu = 0.07$ (see Figure 1). However, we wanted to find the best-performing combination, so we systematically tried different combinations of μ and β to find out which produced the best rank. In the end, we found that $\beta = 0.06, \mu = 0.4$ (see figure 2) was the best performing. This curve was more aggressive than the original curve;

it appears that shooting more often improves performance even at relatively low non-friend to players ratio, reaching 90% at an x-value of just 60%.

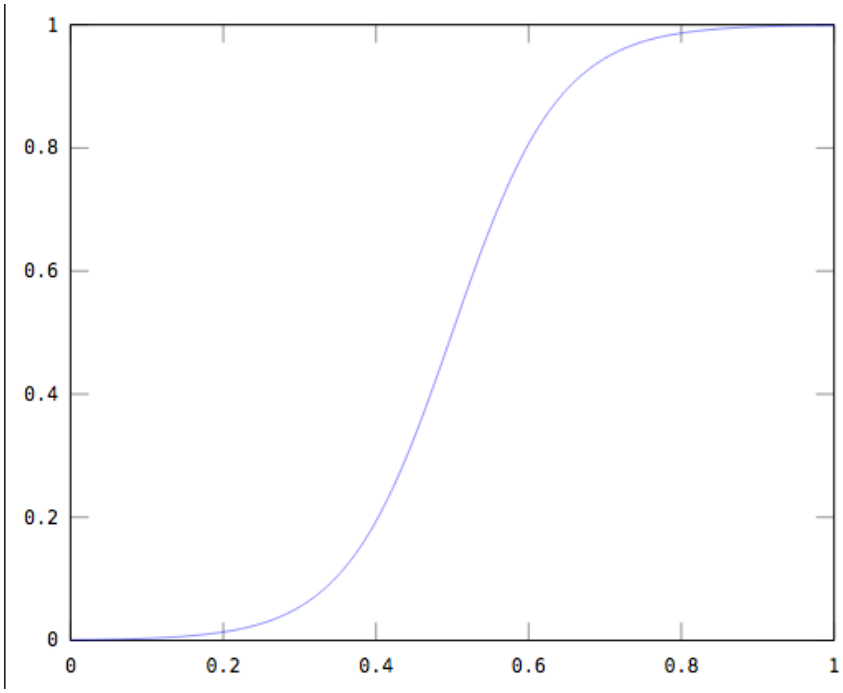

Figure 1: Initial Logistic Curve with $\beta = 0.07, \mu = 0.5$

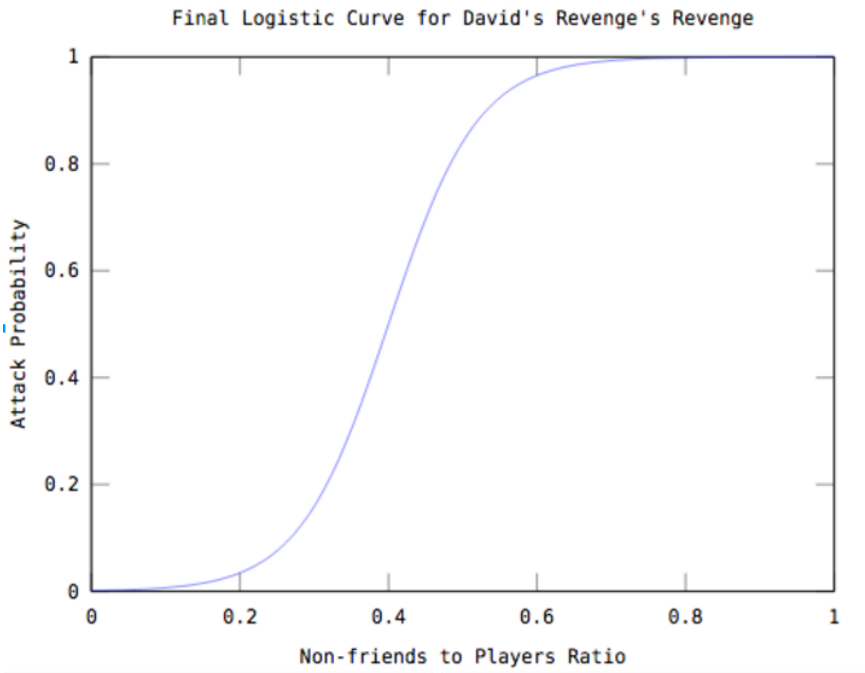

Figure 2: Final logistic curve with $\beta = 0.06, \mu = 0.4$

4 IMPLEMENTATION

4.1 David's Revenge

David's Revenge took a very simplistic approach in defining its actions. In it's final iteration, it featured a selection of a target (if one existed) based on the most hits against us in a time window. Formally, in a game of n players let p_{it} be the player in position $1 \leqslant i < n$ clockwise from our player at round t. $p_{it} \in 0, 1$ where it is 0 if it does not hit us and 1 if it does. Then at round t', our player will target $argmax_i \{P_i | P_i = \sum_{t'-k}^{t'} p_{it}\}$. Our look-back window in this case is k.

To compute this, we cache all hits from the start of the game by leveraging our enmity graph. We iterate through the past k rounds and count up how many times each player hit us throughout those k rounds. We then select the player with the largest sum and choose to hit that person. Thus if no player targeted us in a round, however within the past few rounds there had been players that targeted us, we would look to target those players.

4.2 Protective Farmer

The implementation of Protective Farmer was divided up into several different phases. The Farmer uses external state to keep track of which phase of its lifecycle it's in, but this information is only used between players during the building phase.

The first phase was classified as the SEARCHING phase. Only the first organism would start out with this state. If the cell that the Farmer was on had no units of food, it would attempt to move in an unvisited direction with food. If there was no unvisited direction with food, then it would be an unvisited direction. If there was no remaining unvisited direction, then the Farmer would attempt to move in any open direction. These choices were calculated by filtering down the set of possible choices (from open directions to unvisited directions to unvisited directions with food) and then randomly picking a remaining option. The Farmer's visited steps were calculated by keeping track of its position relative to its point of birth. Once the Farmer takes a step, its previous position is added to a set of Points (later used in determining if the Farmer is attempting to visit a cell that has already been visited). One potential drawback to this strategy is that the Farmer won't be able to figure out the global coordinates of its position and therefore could be revisiting certain cells if it wraps around an edge during its life. As well, due to the random nature of direction selection, it is possible that the Farmer could trap itself inside its own path. In this case, the Farmer will look at only open directions as it will eventually have no other options. The Farmer will only stop if is hungry and needs food, or it find an adequate food source to start farming.

Once the Farmer finds a satisfactory (food units are greater than four) food source, it transitions into the CENTER phase. In this phase, the farmer looks at the adjacent tiles and looks to see if they are filled with other Farmers in the WALL phase. For this implementation, the WALL external state is an integer from three to six, so it is possible a malicious imposter could break this strategy. If more than two of these directions are empty, the Farmer reproduces in that direction, passing along this direction information to the new Farmer organism. There are four WALL states: WEST_WALL(3), EAST_WALL(4), NORTH_WALL(5), and SOUTH_WALL(6). If there is exactly one direction that is not protected by a wall, then the Farmer moves in that direction, becoming that wall.

All of the walls have the same logic for their phase. If they reach a minimum energy threshold they will attempt to move back onto the food source (transitioning into the CENTER phase as well). Otherwise, they will remain motionless. The minimum threshold is the movement cost ν plus nine times the stay cost (1). The absolute minimum survivable energy level is one more than the movement cost, because even if the organism can move to get food, it will not survive the move. The only other action a Farmer in the wall phase can take is reproduce (with some probability, just like XOrganism) away from the food source if the cell it is originally spawned on has food.

4.3 David's Revenge's Revenge

The implementation of David's Revenge's Revenge was straightforward, given the existing code for David's Revenge. We implemented the logistic curve in another class, which was parametrized by the values of μ and β. Once we had the final values of μ and β then we simply hardcoded them in the David's Revenge's Revenge constructor, and added a check to the end comparing `Math.random()` to the value from the Logistic Curve to implement the probability calculation.

The more interesting part of the implementation of DRR was *determining* the best values of μ and β. To do this, we first modified the player to take in parameters from a file so that

it would be easier to automate changing the variables. We also modified the tournament code so that it would only print out the average ranking (which was the metric we were using to measure performance). Then we wrote two JavaScript programs: the first collected all the results for all the parameter combinations, and the second parsed the text results to determine the best-performing combination.

5 RESULTS

Our results were mixed in the end-of-class tournaments. In the free-for-all tournaments, we came within the top 3 (by average rank) in the following configurations:

- 6 enemies, 0 friends
- 6 enemies, 1 friend
- 17 enemies, 0 friends

In other configurations, we generally came in somewhere in the middle; however, this is not very impressive considering that the dumbest players (including the totally-random "dumb player") were active in this tournament.

In the best-player tournaments, we came within the top 2 (by average rank) in these configurations:

- 1 enemy, 0 friends
- 2 enemies, 0 friends
- 2 enemies, 1 friend

There were some configurations in which we did relatively poorly, however. Here are those in which we came in last:

- 5 enemies, 0 friends
- 4 enemies, 1 friend
- 4 enemies, 0 friends
- 3 enemies, 0 friends
- 0 enemies, 2 friends

There are some interesting patterns here. In free-for-all play, we do best in situations where we have no friends (kind of like real life). This makes sense because we don't take advantage of friends very well; we don't have any code for defending them. Furthermore, having more friends makes our player less agressive, and because of the randomness of a free-for-all this can often result in dying too early.

The best-of play is also interesting. We seem to do best in situations with *more neutral players and now friends*, and poorly in situations where we have a lot of enemies or no enemies. Observing a few games, it appears that our strategy of being aggressive in situations with lots of neutral players was quite helpful: it's pretty safe to aggressively target neutral players if they don't have friends. Situations in which targeting neutal players hurts our rank (as Spriha suggested might happen) seem to be fairly rare in these configurations, given that our player was fairly successful.

In the configurations that we did poorly in, we generally had a lot of enemies and few friends. This was somewhat surprising, because we thought that our aggressive strategy in these situations would help us get some kills before being killed ourselves. However, this does make sense when we consider that many of our enemies would target players that shot them if they survived. Even if they had other enemies, they would attack us in *retalation*, which meant that we would die earlier and therefore have a lower score. Perhaps if we had had time to take enemies and neutral players into account separately, we could have done better in these situations. Nevertheless, it's worth noting that the average ranks in many of these mostly-enemy configurations are within a very small range for all teams (0.2), implying that our performance was not actually as far below the other teams as the last-place finish implies.

6 CONTRIBUTIONS

We had two main contributions to the class strategy and discussion:

1. **Expected value and probabilistic play -** Nikhil was the first to try and explore the value of probabilistic play, first with the strategy of Bayesian updating and then with his geometric distribution-based Expected Value Player.
2. **Northward tendency -**

7 FUTURE DIRECTIONS AND LIMITATIONS

7.1 Faster Initial Spread

Our players tended to do best on larger boards because of its high energy density. It tended not to overextend, unlike some other organisms which would tend to start dying frequently due to very aggressive population growth. But on small maps, this aggressive overextension was actually beneficial to these organisms, because they could "choke out" less aggressive competitors and dominate all food resources. Once their opponents went extinct, their organisms would settle into a steady state of lower population and energy compared to their initial population explosion. However, by that time, it was too late for slower-growing competitors like SimplePlayer.

Although an organism might not know the size of the board it starts on, we believe that our SimplePlayer could benefit from employing a more aggressive strategy *only during the first part of the game*. The ideal length of this "aggressive period" might vary based on known map conditions, but based on our observations of population explosions in configurations where the total grid size was less than 1000, we believe that around 100 turns of aggressive expansion is usually sufficient to gain dominance over a large portion of the map. Once this period is up, we would switch to our current, more conservative strategy to take advantage of its natural tendency towards higher energy density. This would make our SimplePlayer much harder to choke out at the beginning, but would still allow it to take advantage of its effective conservative strategy after the intial territory grab. Following the initial population explosion, organisms that maintained aggression would tend to move more and use more energy, which would allow our less aggressive organisms to slowly chip away at their territory while maintaining a high average energy level.

8 ACKNOWLEDGMENTS

The idea of an enmity graph is attributed to Group 4 who initially coined the term. Our first implementation leveraged their graph code that was already written and we extended their ideas. We would also like to thank Group 3 for hinting us that using a time window (like short-term memory) might be worth a look.

9 CONCLUSION

Overall we were happy with the progression of this project, particularly our SimplePlayer's performance on large boards or boards with high movement costs. We ended up with the highest population and overall energy in several of the tournament configurations, showing that good results could be achieved even with comparatively simple heuristics.