ONE WAY STREET

PROJECT 2

Programming and Problem Solving

Fall 2013 / Professor Ken Ross

GROUP 6

Andrew Goldin

Matt Kim

Krista Kohler

October 21, 2013

TABLE OF CONTENTS

1 *Introduction*
- **1.1** Abstract
- **1.2** Problem Statement
- **1.3** Key Terminology

2 *Strategy*
- **2.1** Baseline
- **2.2** Safe Strategy
  - **2.2.1** Dumb Player
  - **2.2.2** Improvements to Dumb Player
- **2.3** Modified Baseline

3 *Justification for Strategy*
- **3.1** General Approach to Testing and Analysis
- **3.2** Results of Semifinal Player Testing
- **3.3** Remarks about Testing

4 *Official Tournament Results and Analysis*
- **4.1** Overall Performance Analysis
  - **4.1.1** Remarks about Tournament
  - **4.1.2** Analysis of Player Rank
  - **4.1.3** Score Analysis
- **4.2** Analysis of Our Player
  - **4.2.1** Player Succeeds with Relatively Low Penalty
  - **4.2.2** Player Succeeds with Relatively High Penalty
  - **4.2.3** Player Fails
- **4.3** Analysis of the Baseline Strategy
- **4.4** Comparing Our Player with the Baseline Player
  - **4.4.1** Analysis of Our Final Player vs. the Baseline Player
  - **4.4.2** Analysis of Our Fixed Final Player vs. the Baseline Player

5 *Future Improvements*
- **5.1** Better Heuristic for Choosing Direction in Phase 2
- **5.2** Dynamic Strategy Based on Map and Time Distribution
- **5.3** Speedup of Phase 0
- **5.4** Better Handling of Zero Capacity Parking Lots

6 *Conclusion*
- **6.1** Results Summary Before Bug-Fix
- **6.2** Results Summary After Bug-Fix

7 *Appendix*
- **7.1** Individual Contributions
- **7.2** Acknowledgments

1 INTRODUCTION

1.1 ABSTRACT

In this project, our group sought to create a player that can efficiently manage the flow of traffic on a finite linear system of one way road segments, separated by parking lots of finite capacities to hold cars pending a green light in their direction of movement. For benchmarking purposes, our group first developed a simple "baseline" strategy in which we ignored all parking lots and opted to send all cars moving in one direction from start to finish before sending all cars moving in the other direction. We then constructed our own player by making a series of modifications to the baseline player in an effort to reduce the waiting time of cars in the system while still preventing car crashes and parking lot overflows. Though we tested our player vigorously under several straining conditions, the tournament results showed that our final player still had a few glitches that would cause cars to crash or parking lots to overflow in about 15% of the tournament games. Furthermore, in the games in which our player did not fail to complete the game, our success at besting the baseline player was varied. However, a small bug fix allowed us to substantially improve our success rate to almost 99%, and beat the baseline player by significant margins in a greater number of cases. Without the bug fix, our player's average rank among all groups in the tournament was close to the median. With the bug fix, our group moved to third best performance among all groups[1]. We have also identified several possible ideas for future development of our player to continue this trend in the future.

1.2 PROBLEM STATEMENT

The goal of this project was to create an agent that can intelligently control a system of street lights on a linear network of one way road segments in order to deliver a specified number of cars from each end to the opposite end as efficiently as possible. The network itself is represented as such:

There are two parking lots at each end with an infinite car-holding capacity, and between them are N road segments of various lengths. The length of each segment is an integer value that can be measured in "blocks", where a block is the length of a single car. Between two consecutive segments lies a parking lot with a specified finite capacity in which cars can wait for the traffic light governing movement in their designated direction to turn green. It is important to note that parking lots can hold cars traveling in both directions; thus, each parking lot has two traffic lights, one for each direction of traffic. If the light is green in the leftward-moving direction, for example, cars in the parking lot that seek to travel leftward will leave the parking lot; if the light is red in the leftward-moving direction, cars in the lot that wish to travel leftward will remain in the

[1] We came in fourth overall, but that was because Group 2 submitted two players, both of whom achieved identical scores and tied for first-place. Hence, we were the third best among all groups.

parking lot. Parking lots are represented as a first-in-first-out queue, such that the first car to enter a lot from a given direction will be the first one to leave the lot once the outgoing light turns green. A car can only exit a lot when the light is green and there is at least one block of space between it and the previous car to have left the lot. Figure 1 shows a visual representation of a road network in action.

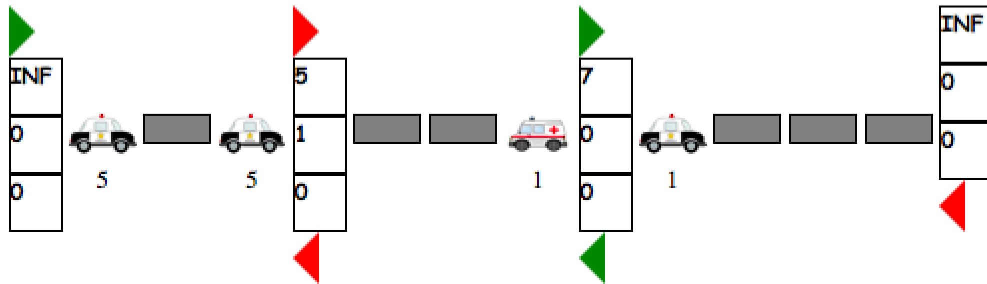

Figure 1: The simulator's representation of a linear road network. This configuration has two infinite capacity parking lots on each end, with three road segments of size 3, 3, and 4, respectively, separated by two parking lots of capacities 5 and 7. For each parking lot, the top number represents its total capacity, the middle number represents the number of right-bound cars waiting in the lot, and the bottom number represents the number of left-bound cars. The right arrows represent the lights for right-bound cars leaving each lot, and the left arrows represent the lights for left-bound cars. All police cars are right-bound and all ambulances are left-bound.

Each car that enters the system is marked with its arrival time. Cars on a road segment move at one block per time unit. The latency of a car is determined by the number of time units it takes for it to be delivered to the other side of the map. A penalty is calculated for each car that reaches its destination, and the total penalty, which is the sum of all cars' penalties, is calculated by the following formula:

$$\text{penalty} = \sum_{i}^{n} (L_i \log L_i - m \log m)$$

Where n is the number of cars, L_i is the latency of car i and m is the minimum possible latency of a car in the network. It is important to note that this function is ***superlinear*** in L, meaning that the derivative with respect to L, $n + \Sigma \log L_i$, is a function that increases with L. The significance of this fact is that high values of L result in disproportionately high penalties (in the context of this problem, car drivers are much more tolerant of short waits and are very intolerant of incredibly long waits). Thus, having a player that performs with consistently average latencies will likely have a better average penalty than a player with a combination of very high and very low latencies.

The goal of the agent is to minimize this penalty and therefore seek to minimize the latency of each individual car. An agent (or "player" as we will refer to it in this report) will fail a given system if any of the following occurs:

- At the end of any time unit, the combined number of cars in both the left and right queues of a parking lot exceeds the lot's capacity.
- Cars moving in opposite directions are present on the same road segment, causing a car crash.
- Simulator timeout, which occurs primarily when the map has reached a "deadlock" state in which no cars have moved for a significant period of time due to a lack of green lights anywhere in the system.

1.3 TERMINOLOGY

The following are definitions of some terms used throughout the report:

Segment - An individual road within the linear road network. Segments in the network are broken down into blocks and are separated by parking lots.

Block - A unit of length used to describe the length of road segments. A block is equal in length to the length of a single car.

Parking lot - Road segments are separated by parking lots, which can hold a specified number of cars traveling in either direction.

Map configuration - A representation of the configuration of the linear network, which consists of the number of segments in the system, the length of each segment, and the capacities of each parking lot.

Time distribution - A list of the arrival times (in simulator time units) of each car in a system. The absolute value of the arrival time indicates the car's arrival time, and the sign of the arrival time indicates direction (positive values represent right-bound movement while negative values represent left-bound movement). A distribution is **sparse** (light traffic) if the spaces between arrival times are relatively large, and **dense** (heavy traffic) if the spaces are relatively small.

Player - Another term for the agent (in this case our program) that controls the lights in the system.

Dumb player - The player given to each group at the start of the project. The dumb player implements a simple strategy that is prone to failure.

Batching - A tactic in which multiple cars are sent in one direction in varying size groups for a certain period of time.

Bubble - A spatial gap or time gap between cars moving in the same direction that could likely be

avoided by an improved strategy.

Deadlock - A situation in which there are no moving cars on the road for an extended period of time, often leading to simulator timeouts.

Zero-capacity parking lot - A special parking lot case in which the parking lot cannot hold any cars; i.e., it is always at capacity. A unique feature of zero-capacity parking lots is that if two cars moving in opposite directions enter the lot at the same time, they do not crash; instead, they simply switch positions and are able to pass each other.

Tournament - A series of many different games, where each game has a unique configuration and a unique distribution, on which all players are tested.

Game - A unique configuration and a unique distribution which all players were tested against in the tournament.

2 STRATEGY & EVOLUTION

2.1 BASELINE

Initially, we wrote a player that we called "baseline" that implemented a very simple strategy that we knew would pass in any map configuration or time distribution. This strategy, which did not use the middle parking lots at all, is very simple and was used primarily as a standard of comparison for our other strategies. Because it did not fail in any case, we knew that we would always be able to use it to measure the efficacy of our strategies.

The algorithm for the strategy is as follows:

1. Initially, turn all the right lights green and all the left lights red.
2. If there are no cars on any segments, turn all red lights green and all the green lights red.
3. Repeat step 2 until all cars are delivered.

In other words, the lights would continue to switch while there are no cars on the road. Essentially, this means that all of the cars from one of the end parking lots would be delivered until either there are no more cars from that end or there is a gap in the car distribution larger than the time it takes for the cars to travel across the entire segment. In this case, the direction switches and the same process continues. This strategy does not take into account the capacity of the parking lots; in fact, it does not store any cars in any parking lots aside from the end ones.

For most dense distributions, this would result in all the cars from one parking lot sent to the

other and then the direction switches once and all the cars from the other parking lot would be sent across. For distributions with a low density of cars in relation to the total length of the road, the direction would switch many times.

This strategy would never result in a failure in the simulation. This is because cars are never stored in parking lots with finite capacity (so the capacity of those parking lots can never be exceeded) and cars only ever move in one direction (i.e. all cars are only moving left or all cars are only moving right), so crashes can never occur. Hence, the only way that this strategy could conceivably fail is through a timeout in the simulator, which is assumed to be fixed based on the configurations so that it never occurs.

A problem with this solution arises through the superlinearity of the score. Since the score is directly related to $L \log L$, which grows faster as L increases, moving half the cars with low latency and then moving the rest with a high latency would likely be worse than moving all the cars with an average latency.

2.2 SAFE STRATEGY

2.2.1 DUMB PLAYER

Throughout the project, our team decided that our main goal would always be not to fail, hence the name "safe strategy". In part, this was due to the real-life consequences of failure in this situation: either a car crash in which people may be hurt or a parking lot overflow which could take hours to clear (assuming that the parking lots are packed so tightly that there is no room for any additional car to maneuver through the lot). Thus, the first strategy that we implemented (aside from our baseline) was a modification of the dumb player's strategy that was provided with the problem. This "dumb" strategy worked in the following way:

1. Initially turn all the traffic lights red.
2. Find out which parking lots are "almost full" - the number of cars in the parking lot is greater than 80% of the capacity of the parking lot.
3. For each segment:
    a. Let L be the left parking lot of this segment and let R be the right parking lot of this segment.
    b. If there is no left-bound traffic, R is not almost full, and L has pending right-bound cars, turn L's right light green.
    c. If there is no right-bound traffic, L is not almost full, and R has pending left-bound cars, turn R's left light green.
    d. If both L's right light and R's left light are green, find out which parking lot is in less danger, that is, which parking lot has a smaller ratio of cars in it to its capacity.
        i. If R is in less danger, R's left light red.

ii. Else, turn L's right light red.

This strategy guarantees that there will never be a crash because step 3d ensures that if both lights that correspond to a segment are green, one of them is chosen based on which parking lot is in more danger.

However, there is no guarantee that an overflow will never occur, especially with low capacity parking lots. Consider the following case:

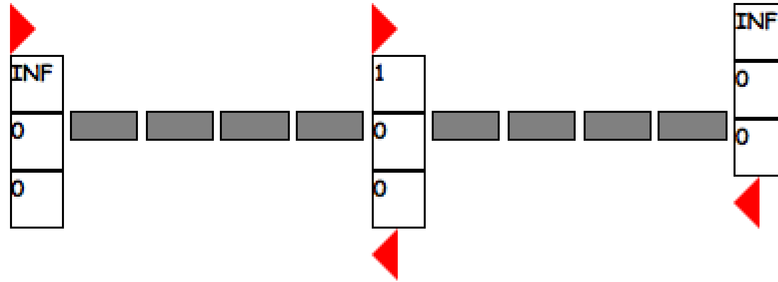

Figure 2: Configuration with low capacity parking lots in which the dumb player fails.

This configuration has one central parking lot of capacity 1 of distance 4 from each of the end parking lots. Consider a timing configuration in which a car arrives at both sides at tick 1 and another car arrives at tick 2 at the right side. In this case, both end parking lots would turn their lights green because they would see that the middle parking lot is not in danger (0 * 0.8 < 1), they have pending cars, and there is no opposite traffic, resulting in the following:

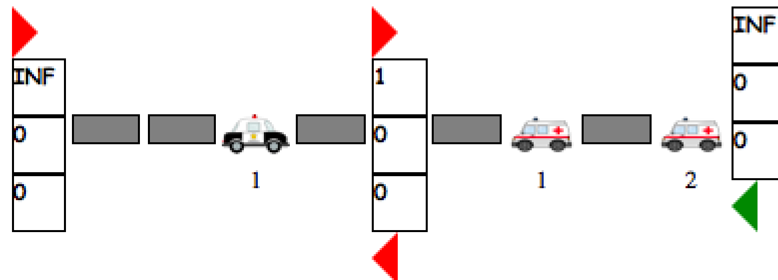

Figure 3: A continuation of Figure 2 after three time ticks.

Here we see that the configuration is destined to fail under the dumb strategy because we have three cars moving towards the parking lot of capacity 1. Since the default configuration of the lights is to keep them red, the two cars that arrived at time tick 1 go inside the central parking lot and keep them in there, resulting in an overflow.

This configuration shows a flaw of the dumb strategy: even though it avoids car crashes, the choice of 0.8 as an indicator of almost full is somewhat arbitrary. With these small parking lots, the strategy cannot tell that it should stop sending cars because it is very bucketed. Furthermore,

the calculation of whether a parking lot is in danger only takes into account the cars that are currently parked in the parking lot; it pays no attention to the cars that are headed towards the parking lot on an adjacent segment.

The last flaw of the dumb player was that it kept the lights red by default and did not allow cars to simply pass through parking lots. In practice, this meant that when a car passed through a parking lot, it had to wait an extra tick before it could actually pass through the lot, which increased the time it took for all of the cars to move through the system.

2.2.2 IMPROVEMENTS TO DUMB PLAYER

To develop our initial strategy, we aimed to create a modification of dumb player that handled these cases so that it would never cause an overflow, as well as other improvements.

The first change to dumb player was to remove the arbitrary choice of 0.8 as the indication of whether a parking lot was almost full. Instead, we determined that deciding whether a parking lot was in danger required calculating the sum of three components:

1. The number of cars in the parking lot.
2. The number of cars on the segment to the right that are moving left.
3. The number of cars on the segment to the left that are moving right.

and comparing this sum to the capacity of the parking lot.

In other words, this sum refers to the number of cars in the parking lot and the incoming segments. This value was calculated for each of the parking lots. For each lot, if this sum was greater than the capacity of the parking lot, then the incoming lights for that parking lot would be turned off (the other checks in the dumb strategy remained, such as not turning on lights when there was opposing incoming traffic).

Essentially, this would guarantee that the number of cars in a parking lot and its surrounding segments would never exceed its capacity, so that it would never be possible for there to be more cars in a parking lot than its capacity. Although this would cause a slowdown in the movement of cars, especially those with long roads and small parking lot capacities, we implemented this strategy because it would cover a case in which the dumb player failed.

The next improvement that we implemented was a smarter behavior to choose which light to keep green when two lights pointing toward the same road segment were green. In the dumb player strategy, this was chosen based on which parking lot was more in danger. However, with this new implementation, there was no need to do this as a parking lot was either in danger or not, and strictly so. We knew exactly whether or not the parking lot capacity would be exceeded and handled that case, so we were not worried about the possibility of putting a parking lot in that

situation. Hence, we chose which parking lot's light to keep green based on which of the two parking lots had a car with an earlier arrival time (if the earliest arrival times in both lots are the same, we behaved the same as the dumb player would in this case). Because we prioritize based on arrival time, our strategy favored more average car latencies rather than extremes with both high and low latencies. Because of the superlinearity of the score calculation, this is more desirable.

A final improvement we implemented was to remove the behavior in which the dumb player turned all lights red by default. This would allow the cars to pass directly through the parking lots, and thus not spend an extra time unit waiting inside of it. Essentially, this caused us to reverse our logic (every case where we would turn a light green, we would turn it red in the negation of that statement). Furthermore, since cars pass through parking lots instantly, we had to treat a car that was about to go into a parking lot as if it were inside the parking lot already to avoid a crash.

Although this strategy accounted for overflow and made the aforementioned improvement, it introduced another problem: deadlock. Essentially, this strategy could cause the system to enter a state in which no cars could be sent across because all lights that could send cars would be turned red. For example, consider the following case:

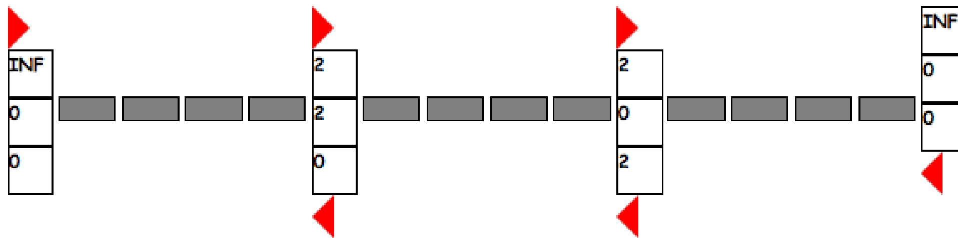

Figure 4: A deadlocked state.

Here, we have a configuration with two middle parking lots, each with capacity 2. Consider a timing configuration in which cars arrive at both sides at time ticks 1 and 2. As shown above, both parking lots were filled to capacity and nothing further occurs within the simulation. This is because for both parking lots (and the end lots) our strategy observes that the adjacent parking lot is at capacity, so it cannot turn a light green in fear of exceeding that parking lot's capacity. Since this is done on all parking lots, no cars are delivered.

This situation also occurred with zero-capacity parking lots because zero-capacity parking lots are always at capacity, so the lights that controlled cars going through them were always red.

In general, this deadlock occurred where two adjacent parking lots were filled to capacity or when there was a zero-capacity parking lot in the configuration. To address this deadlock situation, we first had to determine when this situation occurred. We reasoned that a deadlock occurred when, at the conclusion of our logic, the following occurred:

1. There were no cars on road segments.
2. There were cars in parking lots.
3. For all the right-bound cars in parking lots, their lights were turned red, and for all the left-bound cars in parking lots, their lights were turned red as well.

Essentially, this meant that no cars were moving and that there were cars that were waiting but no lights were changing color, so there would be no change from the current tick to the next.

At first, we attempted a naive strategy that would turn all the lights green in a single direction for one time unit and then let our strategy continue its course. This direction was determined by looking at all of the cars, finding the one that had arrived earliest, and prioritizing its direction of movement. This solution to the problem proved to be problematic, however, as this break from our traditional logic often resulted in parking lot overflows.

As our first priority was not to fail, we had to implement a deadlock-prevention strategy that would be able to pass in every case. Thus, rather than continuing our general logic in the case of a deadlock, we implemented the baseline strategy instead, and decided to simply send all of the cars moving in one direction and then send all of the cars moving in the opposite direction. We reasoned that, since a deadlock was an edge case occurred infrequently, defaulting to the baseline strategy would be a reasonable solution that would allow us to pass this case.

Even though we made these improvements to the dumb player strategy, we found that the baseline continued to outperform our new strategy by achieving lower penalties. To attempt to improve our strategy such that it would perform better relative to the baseline strategy, we also tried to add in logic that analyzed parking lots locally in addition to segments. In our safe strategy, we looked at each segment and analyzed whether to turn lights off for the right and left lots at either end of the segment; we sought to extend this examination by analyzing parking lots as well. A parking lot, in this case, would be examined as a parking lot and its surrounding segments. We reasoned that we should be able to determine an improved strategy based on the capacity of a central parking lot, the lengths of the segments that connect to it, and the pending cars in the two adjacent parking lots. We hypothesized that we could develop a greedy solution by solving this problem locally (for each parking lot) that would result in a globally efficient and safe solution.

However, we found that this strategy was very difficult to implement because often the solutions based on localized areas often conflicted with each other, resulting in failures due to crashes and parking lot overflows. Because we were not certain that we could improve this strategy to always avoid failures, we sought to examine other possible strategies.

2.3 IMPROVED BASELINE

Because we observed that in almost all cases our safe strategy performed more poorly than the baseline strategy, we decided to approach the problem from a different angle, especially because we noted that our safe strategy was still performing better than some of the other groups' players because we implemented a strategy to prevent deadlock. We concluded that rather than trying to create a strategy that was distinct from baseline, would not fail, and would outperform baseline, it might be more feasible to start from baseline (which we knew would not fail) and improve upon that. To take baseline and improve upon it, we first had to determine its major problems.

The first problem with baseline arises with the first movement. Basically, it only moves cars in a single direction at the start; if it chooses to send all left-bound cars first, cars will only move from the left end parking lot, and right-bound cars in the right end parking lot will accumulate there, even though they could safely proceed onto the road up to a point. Consequently, if we could improve baseline to allow cars to be stored in the middle parking lots, then it would be possible for cars to be moved in both directions at the start and then stored in the middle parking lots when they got close to crashing. From here, baseline could continue as normal, but this improvement could make a large difference in score in longer road configurations.

If, however, we were to implement an improved strategy of baseline that would store cars in the middle parking lots, a few issues would need to be solved. First, we would need to ensure that the correct number of cars would be sent out into the map so that we do not overflow the parking lots. Next, the question of which direction to send first arises. We decided to use a similar metric as in the safe strategy, where we send the direction that contains the earliest car first.

The last problem with this "improved" baseline strategy that involves storing cars in central lots is the final strategy that is implemented. In our safe strategy, if we encountered a deadlock situation, we defaulted to baseline; thus, we saw that, at the conclusion of the strategy, even though all the right-bound cars had moved past the left-bound cars that were waiting in parking lots, the left-bound cars waited until the right-bound cars were delivered before beginning to move. However, since there were no cars left of the left-bound cars on the course, it would have been possible to send them earlier to further reduce latency.

Taking these improvements we created a strategy that operated in four distinct phases, described below:

- **Phase 0:** Move cars into parking lots from both directions. Make sure to only move cars on a segment equal to the capacity of the next segment so that we cannot have lot overflow in the parking lot between the two segments. Move to Phase 1 if the rightmost, right-bound car and the leftmost left-bound car are one segment apart or if there are no cars on the road and there are cars in parking lots.
- **Phase 1:** Set all lights to red. Move to Phase 2 when there are no cars on the road and they are all in parking lots.
- **Phase 2:** Determine the car that has the earliest arrival time globally by examining the

cars, which are now all in parking lots. If it is a right-bound car, set the variable currentDir to 1, else set it to -1. This direction will now be prioritized, and we will move into Phase 3.

- **Phase 3:** Move cars in the direction currentDir. If currentDir is 1 (we are moving cars from left to right), find the rightmost parking lot such that there are no cars to the left of it on the road or in any parking lots. For this parking lot and all parking lots to the left of it, turn all the left lights green and the right lights red. Do a similar but reverse operation if currentDir is -1. Move back to Phase 0 when there are no cars on the road or if a deadlock state is entered (this is defined similarly as in the safe strategy).

One issue that this strategy does not handle the case where there are zero-capacity parking lots. In this case, it gets stuck in Phase 0 and does not move into Phase 1. Therefore, in the case in which zero-capacity parking lots exist, we decided to default to the baseline strategy to ensure that our strategy would always pass. Again, we were still hoping to create a player that would never fail, and would hopefully perform better than the baseline strategy.

When running tests with this strategy, we found that we were able to significantly improve our score over our safe strategy and still pass every case in our test suite (see section 3 for more information about testing)[2]. The results corroborate this strategy, with only a simple bug resulting in failures, which will be discussed later in this report.

3 JUSTIFICATION FOR STRATEGY

3.1 GENERAL APPROACH TO TESTING AND ANALYSIS

The rationales and logical thinking we described in Section 2 only partially guided the development of our player strategy: the other major component of our strategy justification was the result set of the test suite we imposed on all players. As was mentioned in class, this project was one in which extensive testing and regression testing were paramount; as strategies became more complex, it was easier to miss small “edge cases” in which the player would fail. Furthermore, complex strategies often resulted in improvements to certain configurations or time distributions, but it was equally important to weigh whether these complex strategies caused the player to fail or perform significantly worse in other configurations or distributions. Thus, we felt it especially important to evaluate each new version of our player’s performance relative to the previous version of the player’s performance in order to ensure that the new version was a) failing in fewer cases, b) improving our player’s score in at least some distributions, and c) not

[2] We note here that we did not fail in any of our test cases; however, we still experienced failures in the tournament. This was due to several subtleties and logical flaws in our code that did not present themselves in our test cases, even though our test cases were very conceptually similar to the tests used in the tournament.

performing worse in more cases than the number of cases in which our player is performing better than the previous player.

Throughout the process, our group prioritized the creation of a player that would not fail in any cases, if possible. In our minds, it was better to create a player that never failed and had some very bad scores than a player who had some very good scores but failed quite frequently.

A final note about our test suite is that it was very dynamic--with each new version of our player we would think of additional cases to test and, consequently, find new ways to fail. The next version of our player would correct these failures; as mentioned in Section 2, some of the early challenges we faced the case in which the dumb player fails described in Figures 2 and 3, the deadlock state that arose from our fix to the dumb player strategy, and the challenge presented by zero-capacity parking lots. In fact, in terms of the class deliverables, our first three players were focused primarily on error-prevention and not failing rather than creating a more efficient strategy; this was largely in part due to what we saw of the other groups' players at those stages, which were still failing quite frequently. Once the semi-final versions of all groups' players were developed, however, we realized that in order to perform well relative to the class, we would need not only to avoid failure, but also to come up with some ways to move cars more efficiently. As a result, the most significant testing period came between the semifinal and final versions of our player, as we strived to improve the baseline strategy without causing crashes or overflows.

We designed what we believed to be a fairly holistic test suite, at least in terms of the concepts each configuration or distribution tested, that consisted of the following types of configurations and distributions. For brevity, we will not provide the exact text of each configuration and distribution here, but rather a brief summary of the concepts it seeks to test.

Configurations:

- Standard, in which all roads have length 4 and all parking lots have size 10
- A variation of standard in which all roads have length 4 but all parking lots have size 4
- A map in which the road segments increase in size from left to right and the parking lot sizes decrease from left to right
- A map with two segments and a zero parking lot in between these segments
- A map in which the road segments vary and the parking lots alternate between capacities that are multiples of two and capacities of zero
- A map with many long roads whose lengths are relatively prime and small parking lots whose capacities are also relatively prime
- A very symmetric map with regard to both parking lot sizes and road segment lengths
- The maps that other groups submitted for their first deliverable, for the sake of completion

Timings:

- A uniform heavy distribution, in which cars arrived at both sides at every time unit
- A uniform distribution that was much more sparse, where cars arrived only every 2-3 seconds

- A uniform distribution in which the arrival times of cars was irregular/arbitrary
- A “rush hour” distribution in which right-bound traffic was heavy and left-bound traffic was light for a period of time, both sides’ traffic was sparse for a period of time, and then left-bound traffic became heavy and right-bound traffic became light
- A “sanity check” distribution in which all cars arrived first on one side and then after all of those cars arrived, cars arrived on the other side
- A second “sanity check” distribution in which cars arrived only at one side for the full time duration
- A timing in which the arrival of cars was sparse but not regular
- The time distribution we submitted for the first deliverable, which was based on the Fibonacci sequence
- The time distributions other groups’ submitted for their first deliverable, for the sake of completion

We combined these configurations and timings in many ways, particularly emphasizing the cases that would result in the most stress on the players, as these were generally more interesting and more telling, as players were more likely to fail under difficult circumstances.

3.2 RESULTS OF SEMIFINAL PLAYER TESTING

As was previously mentioned, testing for the first, second, and third versions of our player was primarily focused on identifying and fixing crash or overflow situations for our own player; because we prioritized success in all cases over having better scores but more crashes. Our analysis of the other players’ at this point was brief, as we found that many of them were still failing in several cases; thus, we believed that by creating a failproof player that we would be able to outperform our classmates.

It was between the semifinal and final versions of our player that we discovered that, to succeed relative to our classmates, we would need both not to fail and to implement some efficiency improvements. At this point, testing involved determining not only whether players failed, but also analyzing how well our player ranked relative to other game players.

As we developed our semifinal player into the final version of our player by addressing cases in which it failed and implementing improvements to enable it to perform better relative to the other players, we observed several things about the semifinal versions of other teams’ players:

- At this point, Group 2’s player was already outstanding. In our relatively exhaustive test suite, the player still did not fail at all and ranked in the top three in a majority of tests.
- While Group 9’s player was not as dominating as Group 2’s player, it still did not fail in any of our test cases.
- Group 5’s player and Group 7’s player crashed in every single test we tried, perhaps due to

some overarching bug. As a result, however, it was difficult to assess the efficacy of these teams' strategies relative to our own. Group 4's player also failed in a substantial percentage of test cases, many of which appeared to be due to deadlock

- The baseline strategy still performed very well, frequently better than our player, and often only worse than Group 2's player.

As a result of the above observations, we knew it would be difficult, if not impossible, to outperform Group 2's player, so we focused our efforts on attempting to beat the baseline strategy. With many of our attempts to beat the baseline, however, we found that we exposed ourselves to failures, so many of these attempted improvements had to be removed. As seen in Figure 5 below, we ultimately were able to submit a player that did not fail in any of our test cases, but, as shown in Figure 6, still ranked only average relative to the other players. Figures 5 and 6 also show that, fortunately, our final version player was still an improvement over the previous iterations of our player, which failed in a higher percentage of cases in our final test suite and also had worse rankings in a game in which our first, second, and third version players competed with our final-version player and all other groups' semifinal players.

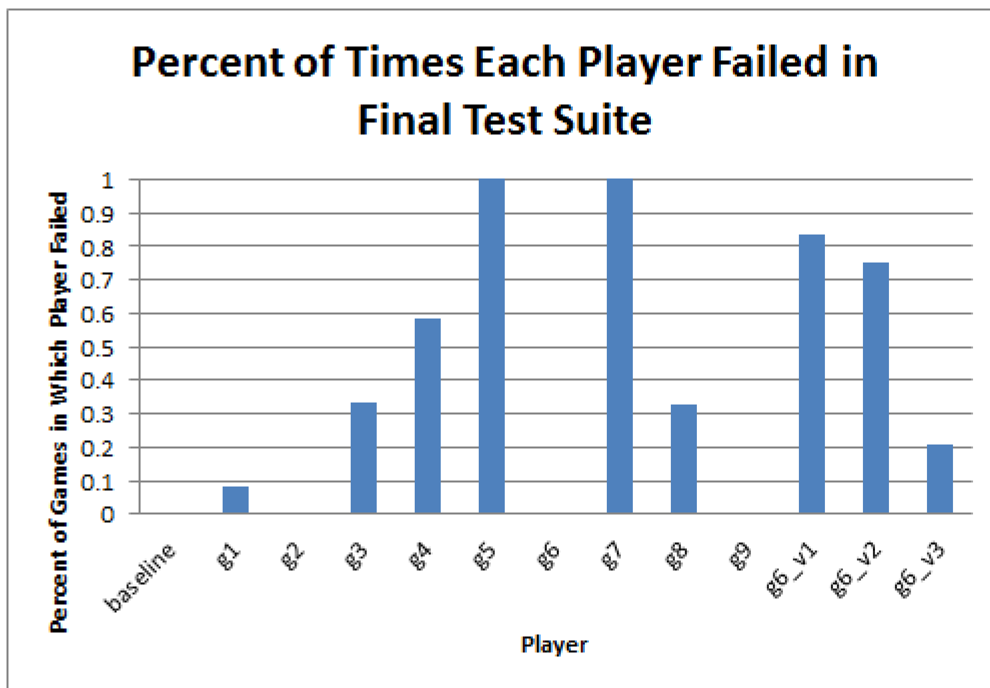

Figure 5: The percentage of times the semifinal versions of each player (and all versions of our player) failed in our final test suite. g6_v1, g6_v2, and g6_v3 refer to the simple, improved, and semifinal versions of our player, respectively.

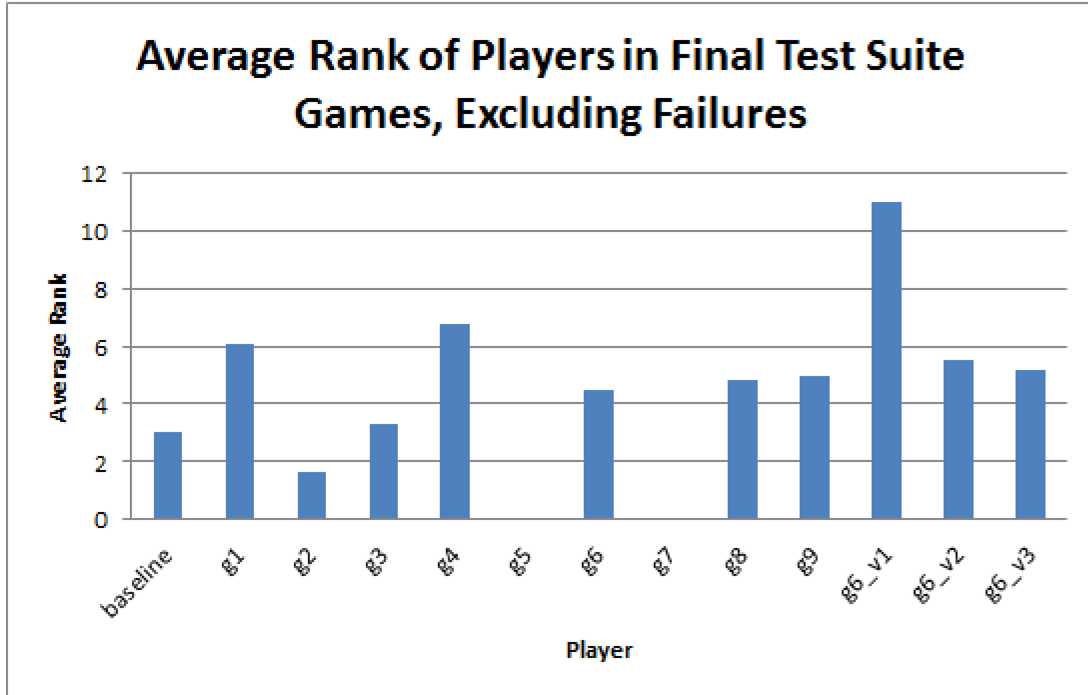

Figure 6: The average rank, excluding failures, of the semifinal versions of each player (and all versions of our player) in our final test suite. g6_v1, g6_v2, and g6_v3 refer to the simple, improved, and semifinal versions of our player, respectively.

3.3 REMARKS ABOUT TESTING

What was very frustrating for our group--and was likely frustrating for other groups as well--was the fact that our player did not fail in any of the test cases we ran, but managed to fail in a significant number of the tournament tests (which are described in more detail in section 4). As we predicted early on in the development of complex players, it can be very difficult to ensure that a player succeeds in *all* possible cases, as a success or failure may be the result of a minor detail rather than a general approach to a configuration or time distribution. These subtleties, in fact, were revealed in our testing process and enabled us to recognize and fix cases in which our player failed. However, the results of our testing were inevitably insufficient to ensure that our player succeeded in all cases. Because it was likely unfeasible to find and test all edge cases and small details that caused failures, it is possible that there was something we could have done better, in retrospect, within our high-level strategy to ensure that our player never fails. Of course, we made every effort to do this throughout the project, but we were evidently unsuccessful; thus, we are not completely sure of how this higher-level improvement to our strategy might manifest itself (however, a more in-depth description of improvements to our player can be found in section 5).

4 TOURNAMENT RESULTS ANALYSIS

4.1 OVERALL PERFORMANCE ANALYSIS

4.1.1 REMARKS ABOUT TOURNAMENT

In this section, we seek to address several notes and assumptions made in our analysis of all players' performances relative to the performances of their competitors:

- We assume that the 182 games in the tournament represent a fair and complete sample of timings and map configurations. In reality, due to small quirks with each player's implementation and/or the tests run on that player during development, the tournament may or may not be the best indicator of each player's abilities.
- There are a total of 12 players in each game, despite the fact that there are only 9 groups in the class. This is because three groups submitted additional players: Group 1 and Group 2 submitted "experimental" players, and our group, Group 6, submitted both our "improved baseline" player (g6v1) and the baseline player (g6v2). The inclusion of the baseline player was, again, for benchmarking purposes--we were interested to see which teams, if any, managed to outperform the baseline player, and we were curious to see whether our improvements to the baseline strategy did result in our player outperforming the baseline, and to what extent.
- 9 of the 12 players in the tournament failed to achieve a numerical score in at least one game, either due to a failure (i.e. a crash or a parking lot exceeding capacity), a timeout, or an inability to make the next decision within 5 CPU seconds. Together, these three situations will be referred to as failures to complete the game, or incompletions; we will not distinguish between the three situations that caused players to be unable to achieve a score. In the analysis of player performance, it is important to consider many different ways of weighting these incompletions relative to successes.
- As mentioned in the previous point, we do not distinguish between an incompletion due to a crash and an incompletion due to an inability to make a decision within 5 CPU seconds. Consequently, in an analysis in which incompletions are punished severely, there may be some argument that this lack of distinction could be unfair to a player who took more than 5 CPU seconds to make a decision; after all, in a real-life scenario, it would be better to exhaust all your resources and allow all cars to move through the map than to have any crashes or situations in which people could be harmed. However, there is no guarantee that players who took more than 5 CPU seconds to make a decision would have ultimately been able to complete the map without parking lot overflows or crashes, which is why we cannot distinguish between these two types of incompletions in a scenario in which incompletions are heavily penalized.
- A careful analysis of our player's performance showed us that 90% of our

improved-baseline player's incompletions were due to a very small oversight which caused an easily-fixable bug. In the interest of fairness, we will not analyze the bug-fixed player in our tournament analysis, as other players may have similar small-fixes that would also result in substantial improvements. We do briefly address the impact of this small fix, however, in section 4.4.2.

- To provide a holistic and fair analysis of players, we sought to consider two major metrics of performance: rank and score. Our analysis will be based on a combination of these two metrics.

4.1.2 ANALYSIS OF PLAYER RANK

To examine the overall performance of all players relative to each other, we first sought to analyze the ranks that each player achieved over all of the games. Computing an average rank, however, posed a substantial challenge given that many players were unable to obtain a score in some games due to either a failure, a timeout, or the use of more than 5 CPU seconds to make a decision about what to do next. Thus, we considered four possible methods of analyzing rank:

1. Compute a player's average rank over all games in which that player was able to obtain a score.
2. Compute a player's average rank over all games. If a player was unable to obtain a score for a particular game, assign that player a rank of 12 for that game, so that all players who failed to complete the game were given a last-place rank.
3. Compute a player's average rank over all games. If a player was unable to obtain a score for a particular game, assign that player a rank of 13 for that game. This approach would ensure that a failure would always result in a lower rank than a completion.
4. Compute a player's average rank over all games. If a player was unable to obtain a score for a particular game, assign that player a rank of 24 for that game (i.e. twice the rank of a last-place finish). This method very heavily penalized incompletions.

Below, we present graphs of players' average rank based on the four methods above.

Figure 7 below provides the average rank of each player over only the games they completed. This graph provides a fair analysis to players who created a strategy that succeeded in some cases but may have failed in many others. For reference, the graph above also shows the number of games in which each player failed to obtain a score out of the 182 total games in the tournament.

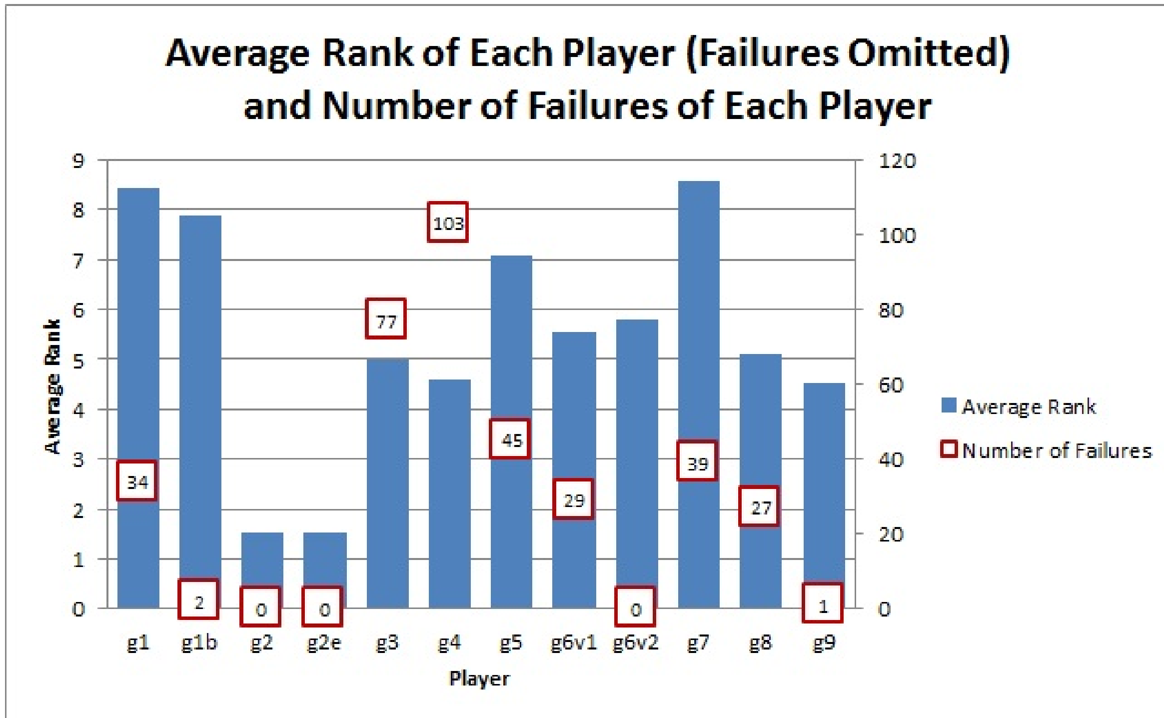

Figure 7: The average rank of each player according to Method #1, in which players' ranks were averaged only in scenarios they completed.

Several interesting things can be noted from this graph. First, both of Group 2's players achieved the lowest (and therefore best) average rank *and* they achieved zero incompletions. Thus, regardless of the weight of failed games, Group 2 performs the best of all groups with regard to rank. Similarly, Group 9's player achieved the second best average rank with only one failed game, indicating that this group also performs well with regard to rank and without regard for the weight of failures. Group 4's player had the third-best average rank, but a very high number of incompletions; its overall performance, then, is much more likely to be affected by the weighting of failures than that of any other player.

In addition to the three aforementioned groups, Groups 3 and 8 also achieved lower average ranks in this test than both our player and the baseline, but because they each had a significant number of failures, it is difficult to determine from Figure 7 alone whether they would still beat the baseline strategy if incompletions were included in calculation of average rank.

Our improved baseline player actually did outperform the baseline player if we only consider its average rank over all of the games it was able to complete. This is a strong justification for our ultimate strategy, whose goal was to present an improvement to the baseline player. Unfortunately, however, there were many players that beat baseline in this analysis, so, while we achieved our goal of improving upon the baseline strategy, we still did not perform particularly well relative to the other players, with the 7th best average rank of the 12 players. The other major downside, of course, is that our player did not outperform the baseline strategy with regard to completions; as expected, the baseline player never failed, but our "improved baseline" player

did.

The next three graphs present the average ranks achieved by each player in situations in which failures were included. As previously mentioned, we wanted to examine a few ways of weighting failures. The first way was to assign a player who failed to complete a game rank of 12 for that game, which would make a failure equivalent to a last-place completion. The second way was to assign a player who failed to complete a game a rank of 13 for that game, such that failures were always ranked lower than successes[3]. A third way to incorporate failures into rank was to penalize all failures very heavily by assigning all failures a rank of 24, which is twice the rank of a last-place completion[4]. With each graph, we have included a numerical ranking of each player (such that the player with the lowest average rank is ranked 1, the player with the highest average rank is ranked 12, etc.) for convenience, as the actual values of the ranks become less relevant when failures are accounted for, especially in the cases where the rank assigned is 13 or 24, since some players' then have an average rank that is greater than 12[5].

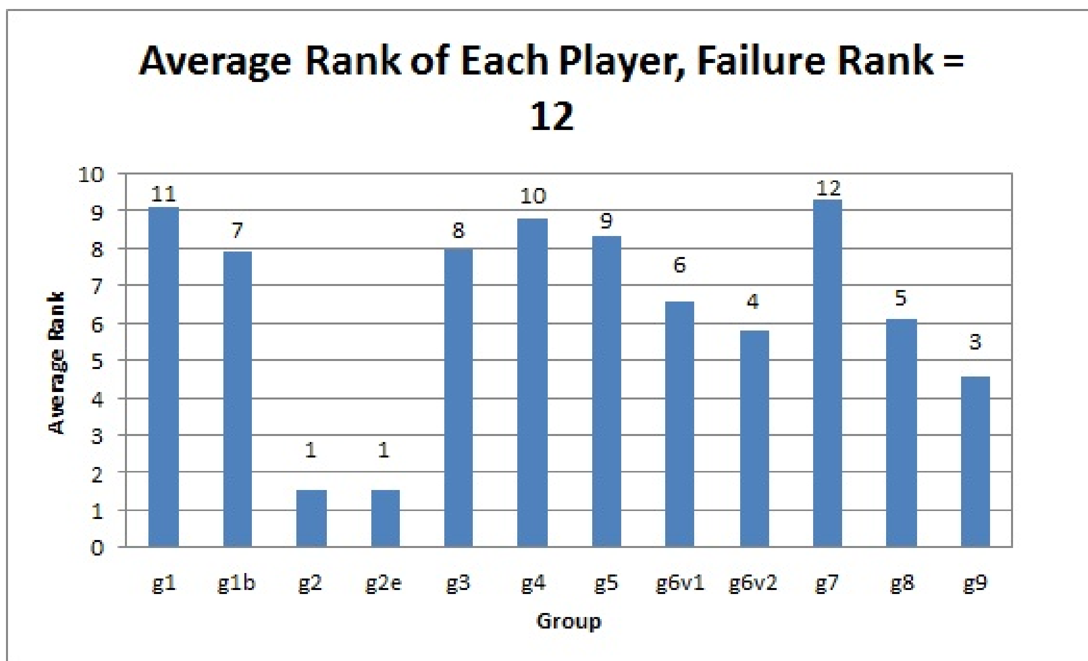

Figure 8: The average rank of each player when players who failed to complete a game were given a rank of 12 for that game. For convenience, the rank of each player's average rank is presented numerically above the corresponding bar.

[3] Consider Game A in which all players complete the game. In Game A, the player with the greatest penalty achieves a rank of 12. Now, consider Game B, in which all players succeed except player X. If Player X received a rank of 12 for that game, its failure would be indistinguishable from a last-place finish with regard to rank. Assigning a rank of 13 for failures seeks to accommodate this caveat.

[4] Though this method may seem excessively harsh, it may be the most appropriate measure for the real-life application of this problem; if one were a traffic engineer, it would be much better to make all drivers angry from long wait times but get them all through unharmed; any situations that would prevent this from happening would have to be taken quite seriously.

[5] All rankings are dependent on the presence of ties. For instance, both of Group 2's players achieved identical scores in all cases, so both players tied for Nth place, and the next score awarded was (N+2)th place.

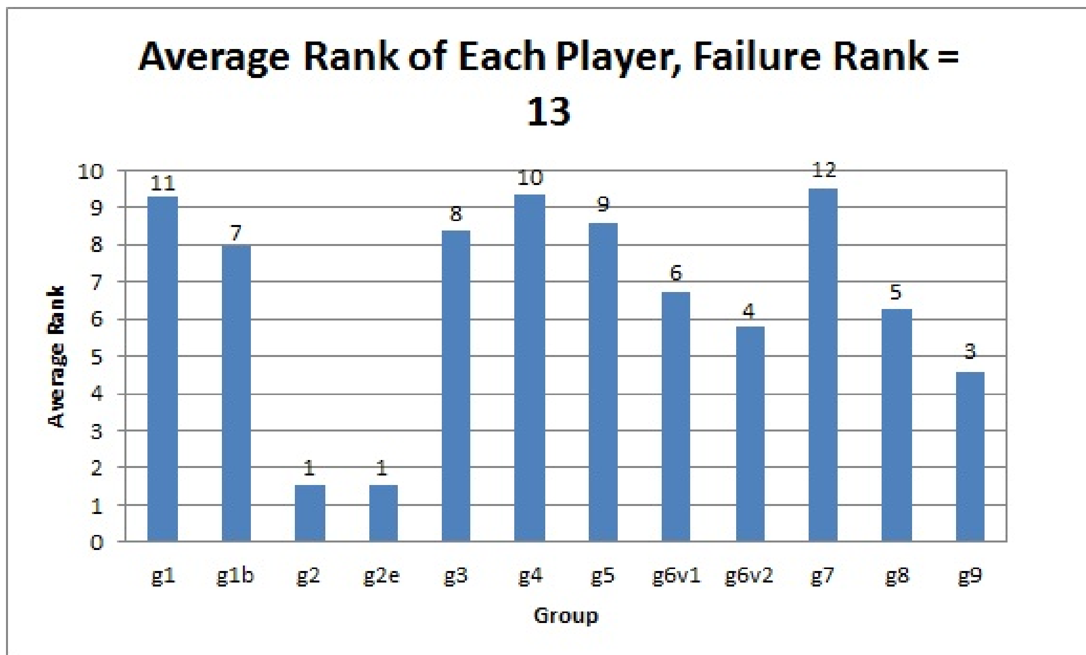

Figure 9: The average rank of each player when players who failed to complete a game were given a rank of 13 for that game. For convenience, the rank of each player's average rank is presented numerically above the corresponding bar.

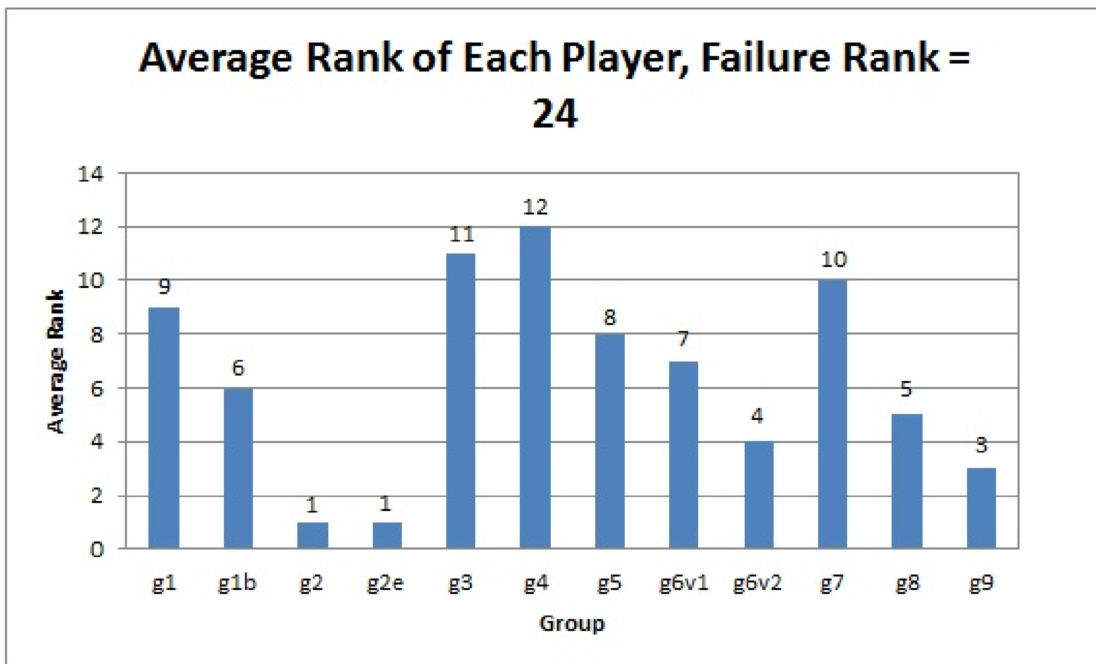

Figure 10: The average rank of each player when players who failed to complete a game were given a rank of 24 for that game. For convenience, the rank of each player's average rank is presented numerically above the corresponding bar.

The graphs above provide several interesting insights about how weighting failures can

significantly impact players' standings. First, we note that Group 2's players and Group 9's players are in first place, first place, and third place, respectively, among all ranking analyses. Thus, these players ranked extremely well overall and had low or no failures; in terms of rank, they were the best "all around" players.

Both Group 8's player and our "improved baseline" player had approximately consistent ranks of average ranks among all four ranking analyses (our players' ranks were 7, 6, 6, and 7 and Group 8's players' ranks were 6, 5, 5, and 5), even though the average rank of both players increased quite substantially over the trials (our player had an average rank of 5.54 when failures were not counted, and 8.48 when failures were weighted with rank 24; Group 8's player had average ranks of 5.11 and 7.92 for those situations). This is likely due to the fact that both of our players had a number of failures that was close to the average number of failures (the average among all players was 29.5 failures; Group 8's player had 27 failures, and our player had 29 failures). As a result (and unsurprisingly), the standings varied most among players with high numbers of failures, such as Group 4, who dropped from 4th place when failures were not counted, to 10th place when failures were ranked 12th, to 12th when failures were ranked 24th.

The baseline player's rank of average ranks improved significantly (from eighth in Figure 7 to fourth in Figures 8, 9, and 10) despite the obvious lack of change in its average rank given the lack of failures. Thus, as we predicted when devising our strategy, a majority of players did perform more poorly than baseline, regardless of how much failure was weighted. Indeed, the major advantage to the baseline strategy is that it will never fail, which makes it a relatively good player when failures are penalized.

A final interesting note is that, while we sought to provide cases where the failure is equivalent to a last-place score (Failure rank = 12) and where the failure is ranked worse than last place (Failure rank = 13), this minor difference did not have any significant effect in the rankings of the players.

4.1.3 SCORE ANALYSIS

What the ranking metric does not provide is an analysis of how well each player did in terms of score relative to the best score achieved. For instance, if the first-place player had a score of X and the second-place player had a score of 200X, this substantial win of the first-place player is not detectable by a ranking system. Similarly, in a game where the first-place player's score was X and the 12th-place player's score was X+11, the closeness of this game remains equally uncaptured by a rank analysis.

Thus, for each game, we calculated for each player the ratio of that player's score to the winning score, and averaged these ratios. Because of the nature of this analysis, failures were necessarily omitted, and results were averaged only over games which a particular player successfully completed. The results are shown in Figure 11 below.

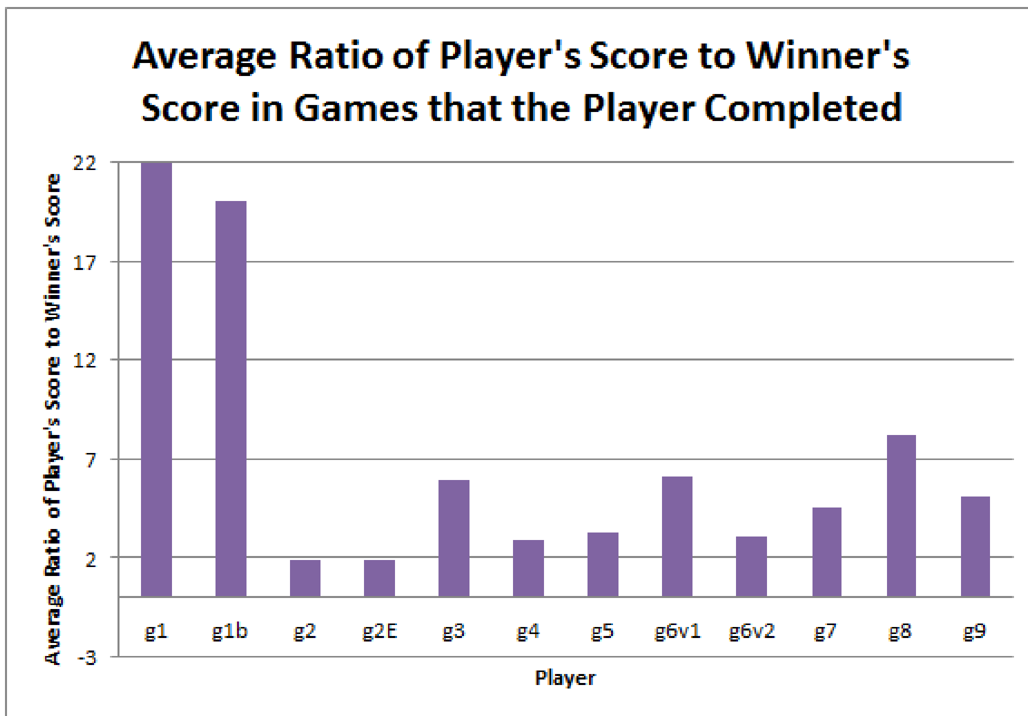

Figure 11: The average ratio of each player's score to the winner's score in the games in which that player was able to successfully complete the game.

Particularly in comparison with the ranking analysis provided in the previous section, this provides some additional interesting insights into the variance of players' strategies. Because Group 2's players won in a majority of games, they again have the lowest ratios of all other players: 1.94. The player with the second-smallest ratio, however, is Group 4 with 2.88, closely followed by the baseline (3.11) and Group 5 (3.29). Evidently, though Group 4 succeeded in a very small fraction of cases, they performed pretty well relative to the winner in the games in which they did succeed.

Furthermore, we see additional evidence that, as predicted, the baseline player again outperformed a majority of the players, and had a strategy that was relatively competitive with the strategy of the winner of each game: only Group 2 and Group 4's players performed closer to the winner more frequently than the baseline strategy, and Group 4's low ratio here is overshadowed by its massive number of incompletions. Given the 1 CPU second limit for each move in the game, it is difficult to accurately assess a player who frequently required more than 5 CPU seconds for a move within the constraints of the game. Thus, with Group 4 out of the picture, the baseline strategy is the second-best strategy by the score analysis provided in Figure 11.

Our improved baseline player ranked 9th out of 12 players in this score analysis; however, the differences between the players ranked 5th (i.e. right after baseline) and 9th, the total difference was 1.51, only 0.35 more than the difference between the difference between Player 2 and the baseline player. Furthermore, there was a substantial gap between the 9th and 10th players (6.07

vs. 8.19), and an even greater spread between the 10th and 11th players (8.19 vs. 20.00). By this analysis, our player performed more closely to the winner of each game than the players of Groups 1 and 8, and only slightly farther from the winner than the players of Groups 7, 9, and 3. Because our strategy was supposed to be an improved version of baseline, we imagine that, had we completed more scenarios, we likely would have beat the players of Groups 7, 9, and 3, given the baseline strategy's dominance.

One of the more interesting discrepancies between the rank analysis and the score analysis can be seen in Group 9's player, which ranked third in all of the ranking analyses, but dropped to rank 7th in the score analysis. This is because Group 9's player's scores are not particularly inconsistent: the player succeeds extremely quickly in certain configurations but in others succeeds at incredibly high penalties. Group 9 has the epitome of a specialized strategy; we sought to avoid specialized strategies because of their extreme performances, but it is noteworthy that Group 9 still outperformed our player both in the rank analysis and the score analysis; thus, perhaps while we were revamping our strategy, we could have thought more seriously about the possibility of developing a somewhat-specialized strategy.

4.2 ANALYSIS OF OUR PLAYER

In this section, we will observe out player in three situations: (1) the player passes the simulation with a relatively low penalty, (2) the player passes the simulation with a relatively high penalty, and (3) the player fails the simulation entirely.

4.2.1 PLAYER SUCCEEDS WITH RELATIVELY LOW PENALTY

We will now look at a few configuration/distribution combinations used in the tournament in which our player performed with a relatively low penalty compared to the other groups.

Figure 12 shows a configuration (*maps/long.txt* in the tournament) comprised of four road segments of length 10, 20, 20, 10 and three parking lots of capacities 6, 7, and 8. In this configuration, the road segments are relatively long compared to the segments used in other configurations, as well as the capacities of the parking lots. The particular time distribution we are looking at is one in which right-bound traffic is a great deal heavier than left-bound traffic (*timings/heavyside.txt* in the tournament).

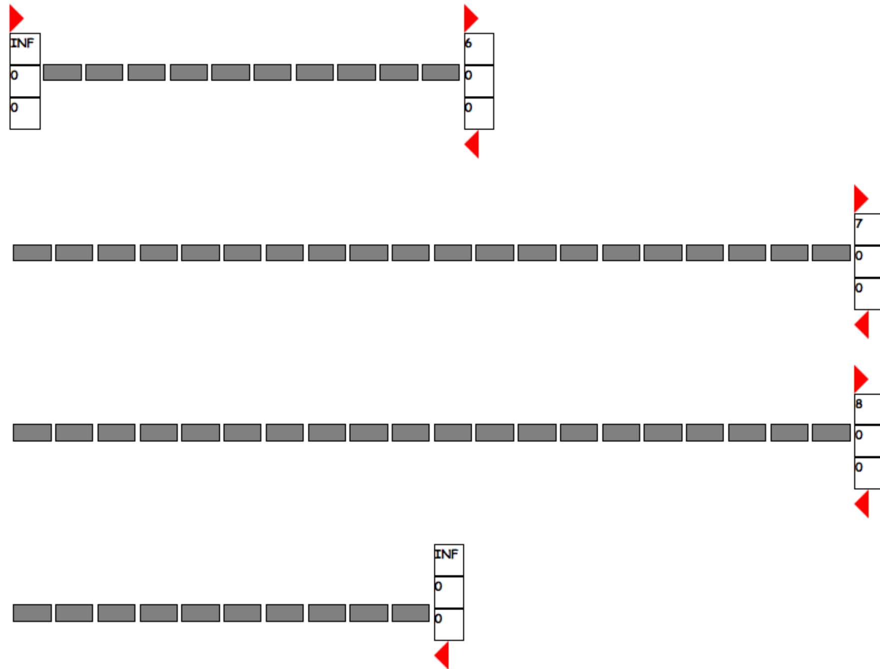

Figure 12: A visual representation of the system configuration *maps/long.txt*.

The graph in Figure 13 displays the penalty of each group's player(s) for this particular configuration and time distribution. Our primary player outperformed every group (with a penalty of 3968) except for group 3, whose penalty was 3963, only 5 points less than ours. Our player did well in this case for a few reasons.

Because of the heavy flow of traffic, some form of a batching tactic will work well. Our player batches cars in both directions as far as they can safely go, meaning that not only were we able to move the heavy right-bound traffic along smoothly, but some of the sparse left-bound cars were able to make a good amount of headway before they yielded to the right-bound cars, significantly reducing their latency. This can be contrasted with our baseline implementation (g6v2 in the graph), which waited until all right-bound traffic had subsided before sending along any left-bound cars.

Ultimately, our player moved every right-bound car along at a reasonably fast clip while finding windows of opportunity to get left-bound cars moving along the way as well (Phase 3). The reduced latency of the sparse left-bound traffic is likely the most significant factor in our low penalty.

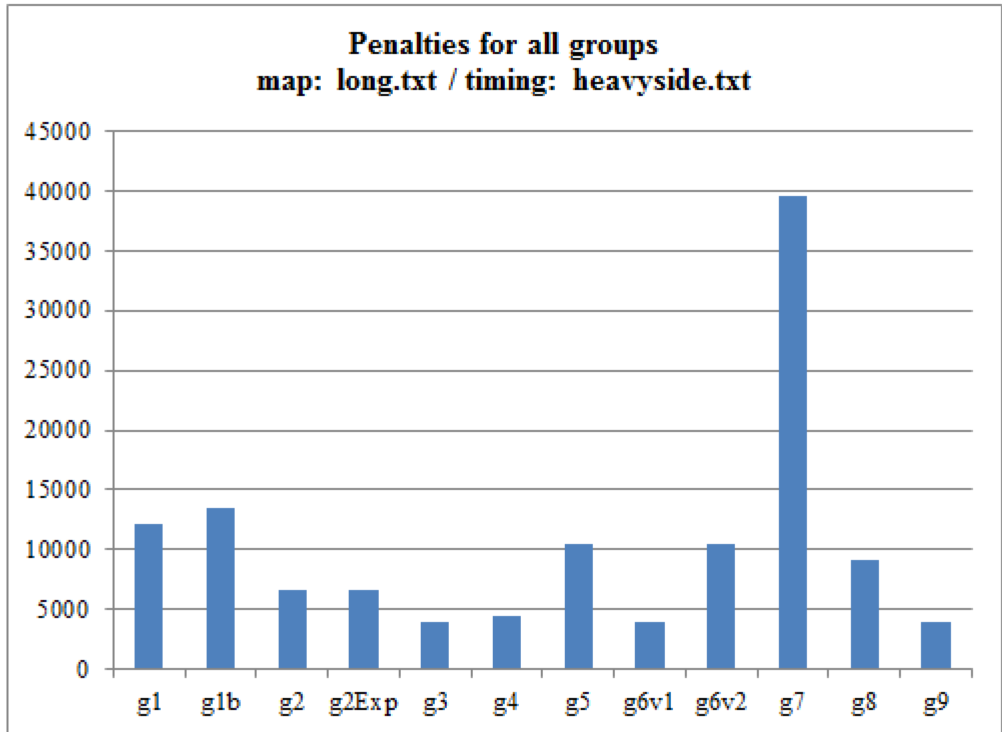

Figure 13: The performance of each group in the system using the map configuration *maps/long.txt* and the timing configuration *timings/heavyside.txt*.

Figure 14 below shows another configuration (*maps/longcenter.txt* in the tournament) in which our player performed reasonably well. It consists of six road segments of length 4, 5, 10, 4, 6, 6 and five parking lots all of capacity 7. The longer center segment is likely present to assess the behavior of players whose strategy is influenced by the goings-on near the center of the system. The time distribution used is one where the timing is uniform on both sides, and an equal number of cars arrive on each side at regular time intervals (*timings/uniform.txt* in the tournament).

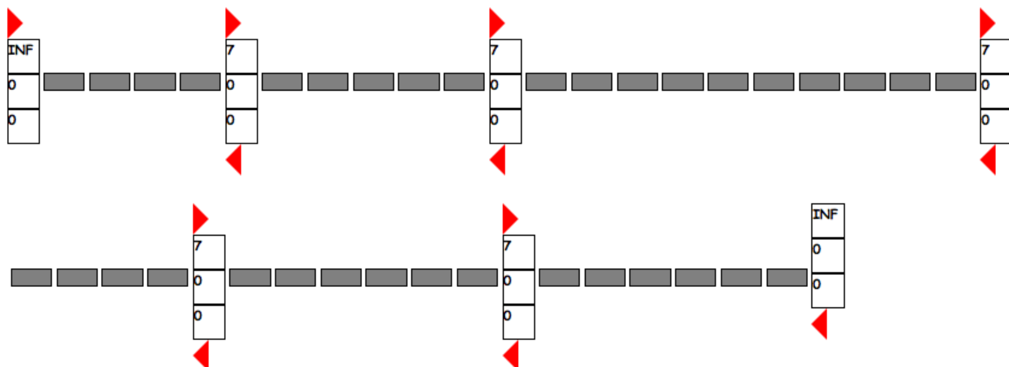

Figure 14: A visual representation of the system configuration *maps/longcenter.txt*.

Again, the graph in Figure 15 shows the penalty of each group's player(s) for this setup. Our primary player outperformed every group (with a penalty of 4851) except for group 2 (penalty of

3259), which is to be expected, since group 2 displayed superior performance in most configurations. Groups 3 and 4 failed on this system, which is why there is no penalty represented in the graph for them.

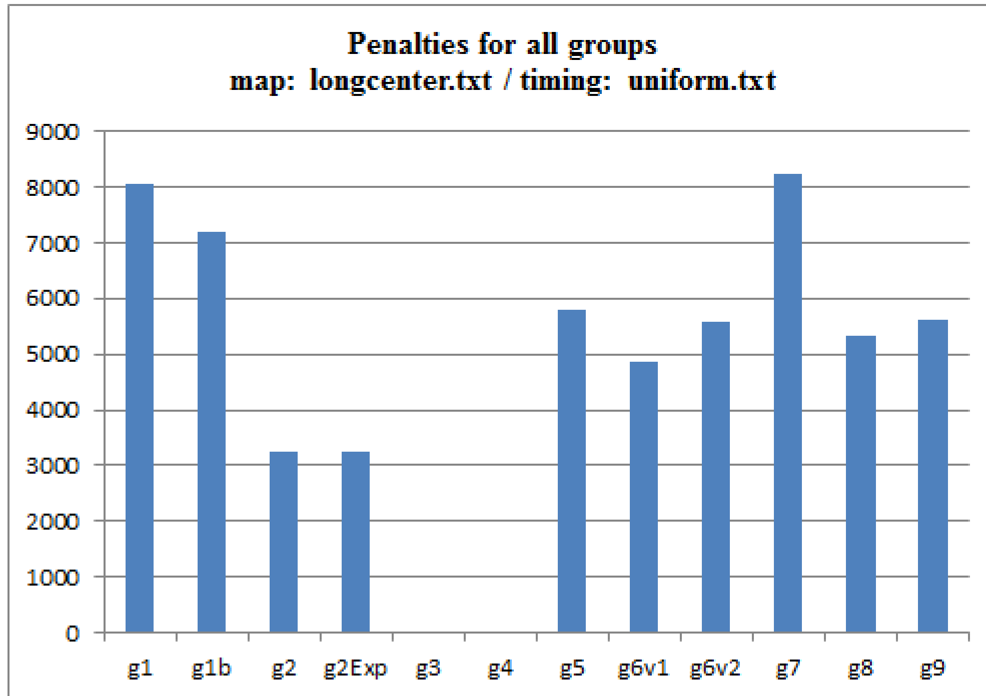

Figure 15: The performance of each group in the system using the map configuration *maps/longcenter.txt* and the timing configuration *timings/uniform.txt*. Groups 3 and 4 are shown to have no penalty, due to simulation failure.

Batching strategies tend to be superior in the presence of long road segments, since it takes a longer amount of time for cars to cross from the parking lot on one side of a long segment to the other. Constantly switching the direction of flow along a longer segment will compound the latency of every car attempting to cross it, so batching multiple cars is paramount to maintaining low latency.

In a similar vein to the last example, our player batches cars in both directions with priority based on which direction of traffic has the earliest arriving non-delivered car. For this configuration, our player was able to send a lot of cars in one direction across the long segment at once, making sure the car at the head of the moving line had the earliest arrival time in an effort to minimize the average latency of the system. It seems to have worked out, for the most part, in our favor.

Overall, since our final strategy is a modified version of our baseline implementation, which is in essence a high-volume batching strategy, it stands to reason that most systems in which high-volume batching is an advantageous tactic will ultimately lead to low latencies and decent performance from our player.

4.2.2 PLAYER SUCCEEDS WITH RELATIVELY HIGH PENALTY

Now that we have seen some tournament configurations in which our player performed relatively well, we will assess some specific configurations in which our player performed relatively poorly (i.e. with a high penalty) and attempt to draw conclusions about the results.

Figure 16 below shows a system configuration (*maps/alternate.txt* in the tournament) comprised of seven road segments of non-uniform size, and six parking lots alternating between small (5) and large (10) capacities. The purpose if this configuration was to assess player performance in a situation where parking lots have large changes in capacity at high frequencies. However, the time distribution used in the combination we will be looking at is one in which all cars arrived from only one side of the system (the left side, in this case) at semi-regular intervals (*timings/oneside.txt* in the tournament).

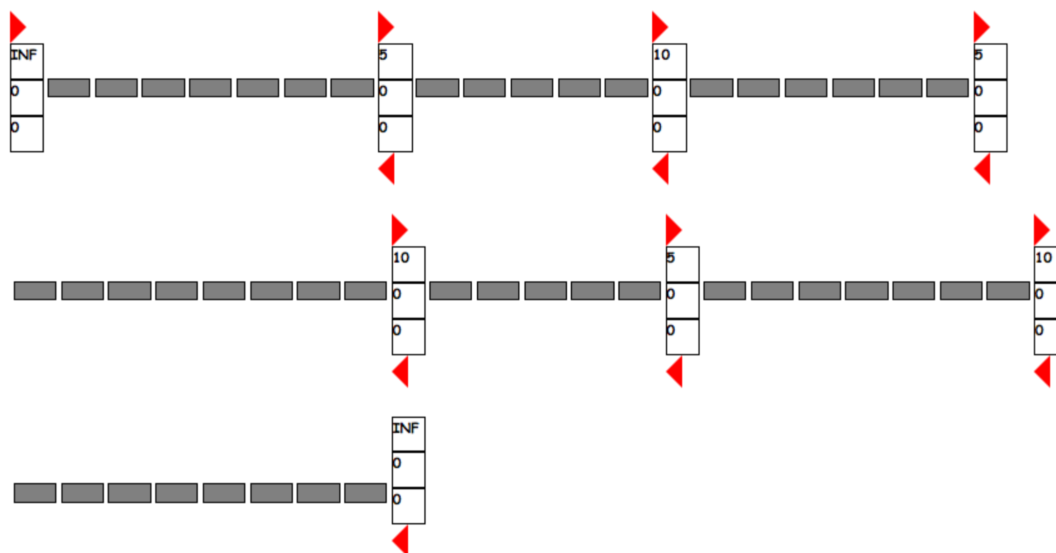

Figure 16: A visual representation of the system configuration *maps/alternate.txt*.

The obvious strategy in this situation would be to simply keep all of the lights green and move all of the right-bound cars from one side to the other in quick succession. Based on the graph in the right in Figure 17, it would seem that this is what most of the players are doing for the most part, since the scores seem to be relatively equal[6]. When running this configuration with our player, we discovered that it enters Phase 1 when the earliest car in the line approaches the second to last parking lot, turning all of the lights red and waiting for all of the cars on the road to enter a parking lot before batching them all out simultaneously. One of our criteria for entering Phase 1 is

[6] Most of the assessment of this system configuration is judged from the graph on the right in Figure 17. Since Group 1's penalty is a statistical outlier in that it is more than an order of magnitude larger than that of the other groups, it is better to work with the data from groups 2 through 9, since it better depicts the expected behavior of a player in this system, as well as how the differences in our player's implementation compare to it.

that the earliest cars in both directions are one segment apart. Extending this strategy, we discover that our implementation plays it safe if one line of cars approaches an infinite lot while in Phase 0. We anticipate that an opposite-moving car could enter the system, and therefore move into Phase 1 to prevent any possible crashes from occurring. The downside to playing it safe here is that switching to Phase 1 causes a small amount of latency at the head of the line, which compounds the further the line travels backward, causing a disproportionate increase in our player's overall penalty.

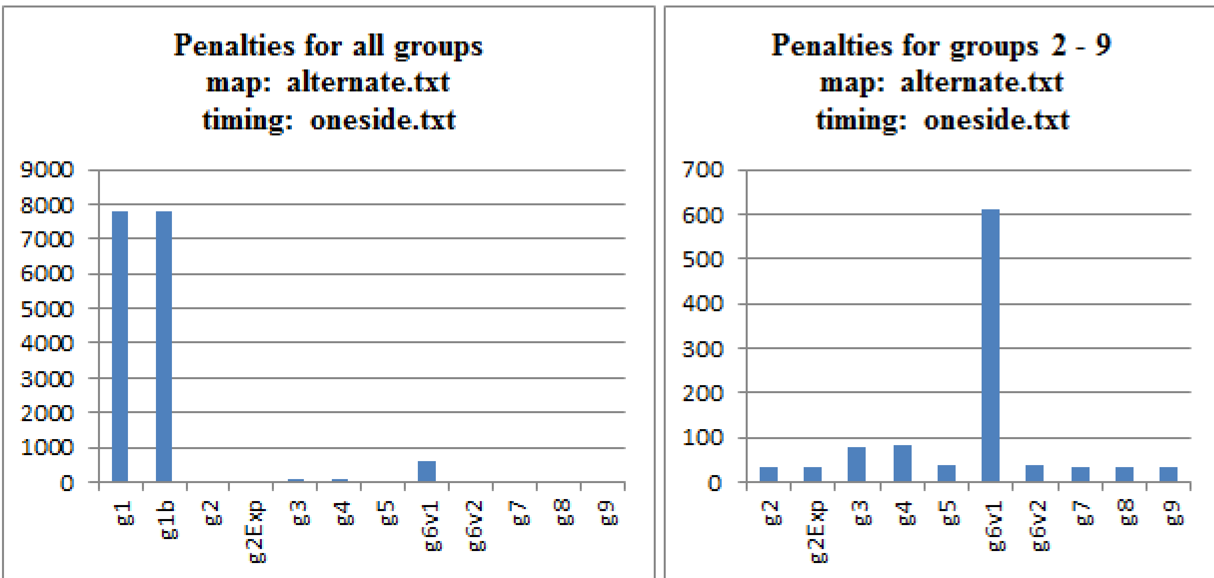

Figure 17: The performance of each group in the system using the map configuration *maps/alternate.txt* and the timing configuration *timings/oneside.txt*. While both graphs display the same data, the graph on the right omits the results from Group 1 to better show the score differentials among the rest of the players.

Through observation of the tournament results, our group found that we underperform most of the other players in any configuration that uses the one-sided time distribution. Through testing and inference, we found that the large penalties accumulated on our player for each configuration were for the same reason described above. This could perhaps be considered a "bug" in our code, and it is likely that a quick fix to account for cases where a line of cars is approaching an infinite lot may result in a great deal of speedup for our player. We address this concern in section 4.4.2 of the report.

The following case analysis is one of poor performance not due to what we would consider an oversight. The map shown in Figure 18 below (*maps/incr_decr.txt* in the tournament) contains six road segments of increasing length from 4 to 9, with five parking lots of *decreasing* capacities from 10 to 6. The aim with this configuration was to see how players performed when the lot sizes do not scale proportionately with the segment lengths.

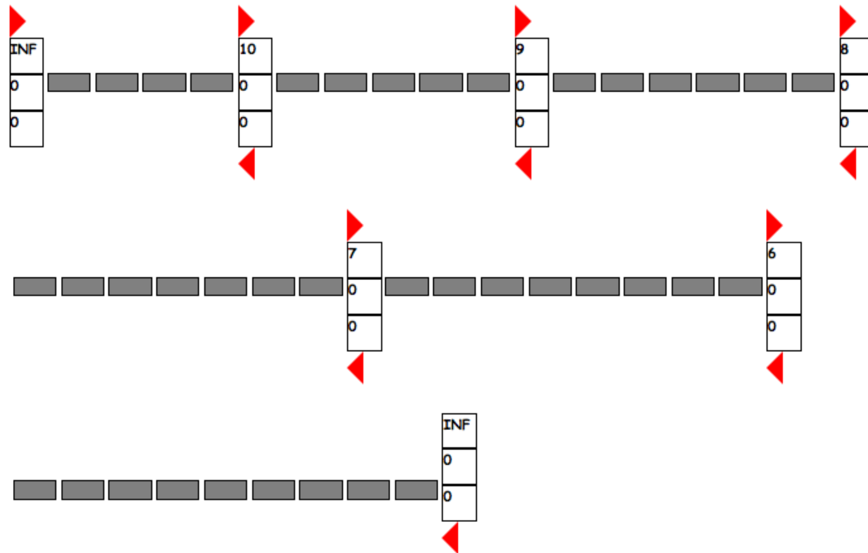

Figure 18: A visual representation of the system configuration *maps/incr_decr.txt*.

However, the reason our player performed poorly in this configuration had less to do with the map itself, but rather the time distribution used, in this case one where traffic arrived from both sides in infrequent pulses, or bursts (*timings/pulse.txt* in the tournament). The graph in Figure 19 shows the penalties for all players in this config/distribution configuration, with the exception of Group 1's first implementation, which failed the simulation.

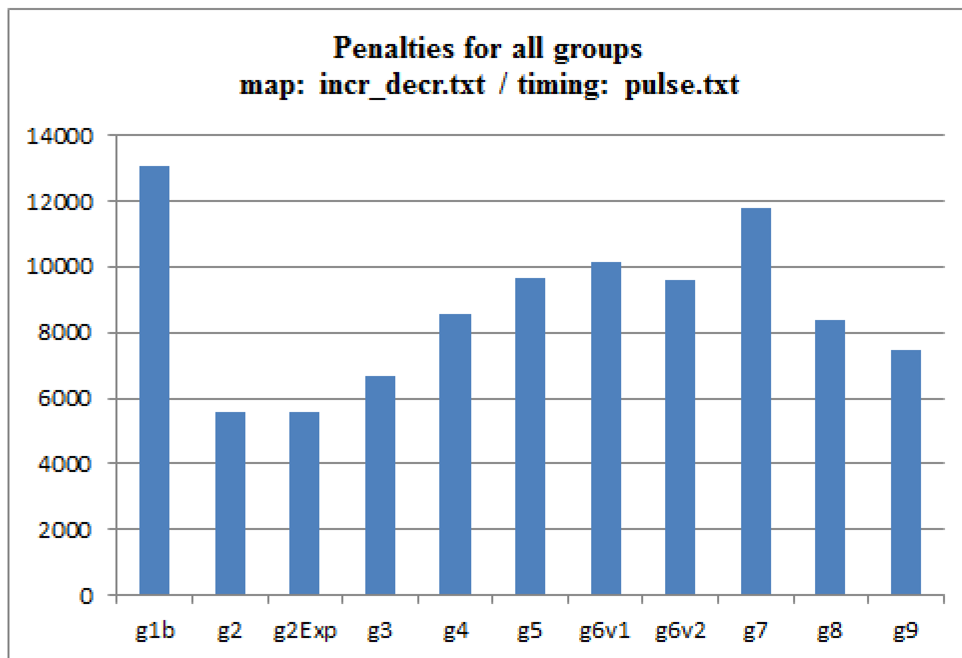

Figure 19: The performance of each group in the system using the map configuration *maps/incr_decr.txt* and the timing configuration *timings/pulse.txt*. The player g1 has been omitted from the graph due to simulation failure.

From the graph we see that we underperformed all players except those of Groups 1 and 7 with a penalty of 10150. The likely reasoning for our performance is that a distribution with infrequent pulses of traffic is not conducive to our batching strategy. At the start of the simulation, a few cars arrive on both sides, and due to our detection logic in Phase 0, all lights turn red before a decent amount of cars are able to get on the road, and we simply flush the cars that *are* on the road from the system. This means that the cars arriving at the beginning of each new traffic pulse have to wait quite a long time before they can make any significant headway, thus increasing their latencies. In fact, we observed from the tournament that this was the case in most every configuration that utilized the pulse-driven time distribution.

It is especially worth noting that we underperformed our baseline player, whom we were designing our player to try to outperform in all cases. This is because the baseline player never stops to hold up traffic from both directions like our "improved" player does. Improving our player to try to detect these infrequent traffic pulses and deal with them as a special case may have resulted in an overall lower penalty in all tournament configs using this time distribution.

4.2.3 PLAYER FAILS

Our player is designed to always err on the side of safety to avoid failure in all cases. While our player never failed due to a simulator timeout or running up the CPU clock, there were a handful of circumstances in which our player failed the simulator due to either a crash or to one or more parking lots exceeding capacity. After some testing and observation, we were able to link both cases of failure to small oversights in our implementation.

Figure 20 below shows one such configuration (*maps/more_inf.txt* in the tournament) in which our player failed the simulator for every time distribution except for *timings/oneside.txt*. It consists of 18 short road segments separated by 17 large parking lots of capacity 1000. The tournament results show us that every failure on this map was for the same reason: a car crash occurred. After testing our player on our own using various time distributions, we noticed that all of the crashes were occurring for the same reason.

As mentioned earlier, our player enters Phase 1 when the earliest cars in both directions are one segment apart. However, due to a small oversight, it does not account for the case in which two cars simultaneously arrive at a set of parking lots separated by one road segment. When this happens, both cars will believe they can pass safely to the next segment since it contains no opposite-moving cars. Thus, Phase 1 is entered only when the cars are travelling in opposite directions on the same segment.. The lights from both parking lots fail to turn red and both cars make their way onto the road segment, leading to an inevitable crash. Due to the small road segments of varying lengths and high frequency of parking lots in this map configuration, our player was unlucky enough to have encountered this situation in every time distribution used in the tournament, except for the one-sided distribution, since cars were only coming from one direction.

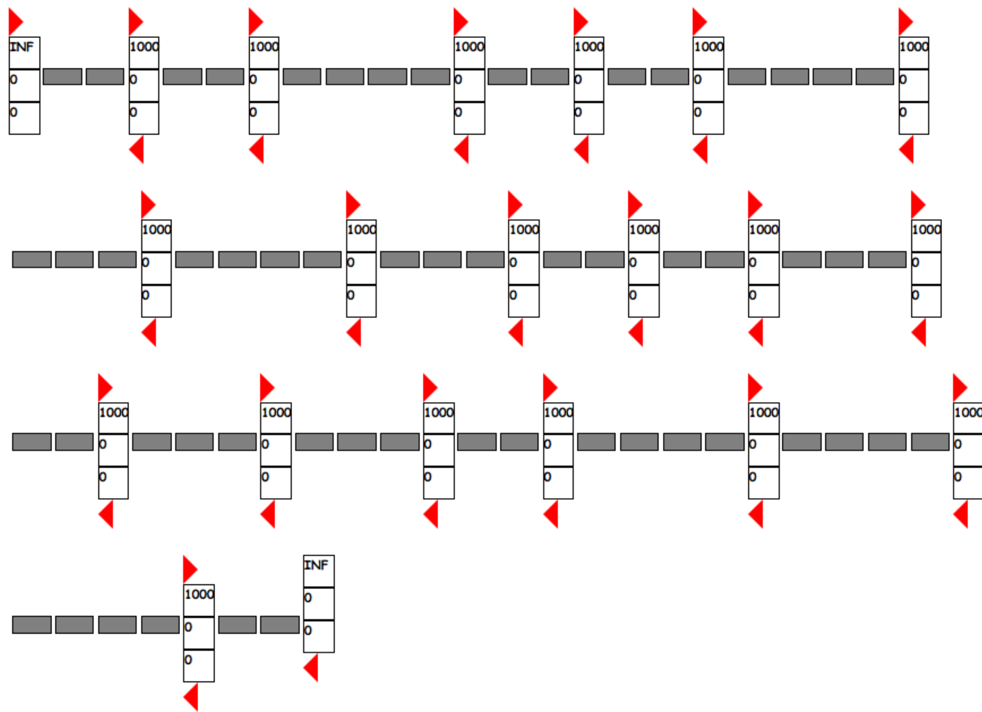

Figure 20: A visual representation of the system configuration *maps/more_inf.txt*.

Through testing and inference, we concluded that this bug may be the only reason that our player caused a car crash in any map configuration. It is likely that our player can eradicate all failures due to crashing from the configurations used in the tournament if a fix to this bug was implemented. Please see 4.4.2 for a brief discussion of the implications of this fix.

26 out of 29 of our failures in the tournament were due to car crashes, but there were also a few cases in which our player failed due to parking lot overflow. One such case can be seen in the partially displayed configuration in Figure 21 below. The full configuration (*maps/crazy.txt* in the tournament) is a *massive* system, with 100 road segments of random lengths from 5 to 10 (inclusive), and 99 parking lots of capacity 2 or 10. The section of the map we are focusing on in Figure 21 is the location at which the overflow occurred. In this map, our player failed on 3 out of the 13 time distributions used in the tournament (*timings/fib.txt*, *timings/pulse.txt*, and *timings/uniform.txt*). This particular screenshot is arbitrarily shown using *timings/fib.txt* as the distribution, as we believe the cause of overflow for all three distributions is the same.

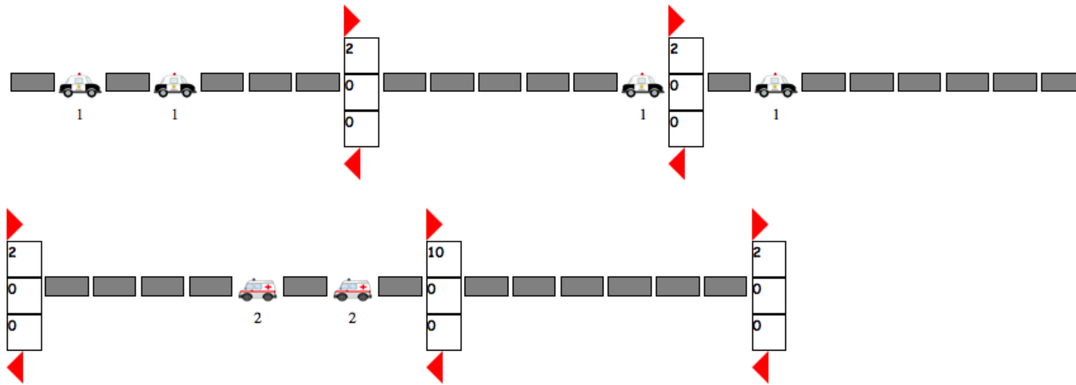

Figure 21: A small subsection of the full system configuration *maps/crazy.txt*. This section shows the portion of the map containing the parking lot that exceeded its capacity, 7 time units before the failure occurs.

We will label the parking lots from top left to bottom right with the numbers 1 - 5 for this explanation. Taking a look at the picture, we see the overflow is inevitably going to occur in the lower left parking lot (lot 3), which has a capacity of 2, and three cars heading towards it. This overflow actually occurs during Phase 1, after our player turns all the lights in the system red. During Phase 0, the first left-bound car passed through lot 4, since the parking lot was not at capacity and there were not enough cars on the segments on the either side of lot 3 to put it in danger of becoming full. However, the first right-bound car and second left-bound cars arrived respectively at lots 2 and 4 at the same time, and, due to a similar oversight to the one discussed in the failure case due to crashing, passed through their parking lots for the same reason as the first left-bound car. This caused our player to enter Phase 1, after seeing that cars were moving in opposite directions on adjacent segments.. However, this action was taken too late, as the simulator failed when the three cars entered the lot.

Though we were able to determine and quickly apply the simple fix required to alleviate the failures our player experienced due to crashes, the ultimate solution to the overflows was not as immediately apparent or straightforward. Because the majority of our failures were caused by the crashes, we have decided to focus only on this fix, which is discussed in section 4.4.2.

4.3 ANALYSIS OF THE BASELINE STRATEGY

To summarize what we observed in 4.1, the baseline strategy, as expected, performed better than a substantial number of players. It was one of only three players that never failed to complete a game (the other of which were both Group 2’s players) which, in our minds, indicates that it is a much more versatile player than the 9 players which failed on at least one occasion. After all, that is part of the reason we sought to improve upon the baseline strategy rather than attempt to improve our safe strategy: we heavily prioritize the importance of completing each game. In the rank analysis in which failures were penalized, the baseline strategy ranked 4th, behind Group 2’s players and Group 9’s player (who had only one failure and was thus not radically impacted by

the single failure). In the score analysis, the baseline player outperformed Group 9's player and performed behind Group 2's players and Group 4's player. However, we do not place too much emphasis on performing slightly worse than Group 4 in this analysis because Group 4's good rank in this analysis is offset by its high number of failures. That said, we believe that the baseline player is, in effect, second only to Group 2's players. We greatly commend Group 2 for developing a player that both never crashes and beats the baseline strategy.

4.4 COMPARING OUR PLAYER WITH THE BASELINE PLAYER

4.4.1 ANALYSIS OF OUR FINAL PLAYER VS. THE BASELINE PLAYER

While the comparison of our player with the other groups' players is important to discuss, we feel our analysis would be incomplete without further discussing and summarizing our player's performance relative to the baseline player since our goal was to create a player that utilized an improved baseline strategy.

The first noteworthy point, of course, is that our player did not match the baseline player's zero failures, and this is a significant failure of our player. While we rigorously tested our player in several different types of scenarios, our tests, in some way, did not completely align with the tournament games (please review Section 3, Strategy Justification, for additional information about the types of testing we completed throughout the development of our player). This was one of the major challenges that many groups seemed to struggle with: no matter how much testing is done, there seemed to be a variety of edge cases in which complex strategies would fail due to a minute detail that is easy to overlook (for instance, during our testing, we discovered that our player once failed due to the fact that the sum of the numbers of cars in two adjacent parking lots was equivalent to the capacity of a third parking lot; had we not tested with precisely those parking lot capacities, we might not have observed this phenomenon).

The second noteworthy point is that even if we only examine the cases in which our player did not fail, our player still did not outplay the baseline player overall with regard to the score analysis or the average rank analysis. We speculate that this is due largely in part to the small bug explained in 4.2. In short, our player was not tested in cases that were similar enough to the tournament cases for us to recognize the player's shortcomings and fix them prior to the submission.

We did have a few small victories over the baseline player, however. In the games in which our player did not fail, we achieved a higher percentage of top-4 finishes and a lower percentage of bottom-4 finishes than the baseline player did, as shown in the table in Figure 21. We hypothesized two reasons for this feat. This is likely due to the fact that the improvements that we did make to baseline allowed us to do better than baseline in certain cases, such as those mentioned in 4.2.1. Our skeptical nature, however, led us to wonder whether we considered is

that it is possible that we may have failed to complete the game in cases in which the baseline player performed poorly. A data check, however, fortunately indicated that in the games in which we failed, the baseline strategy never did worse than fifth-to-last (see Figure 22); thus, it is unlikely that completion of these games would have led us to have a higher number of bottom-4 finishes. Thus, we feel as though we did make a few improvements over the baseline player, even though some of the metrics we analyzed did not reflect any significant improvement.

Player	**Percent of Top-4 Finishes**	**Percent of Bottom-4 Finishes**
g6v1 - improved baseline	39.22	20.26
g6v2 - baseline	25.82	24.18

Figure 21: Percent of top-4 and bottom-4 finishes for our player and the baseline player.

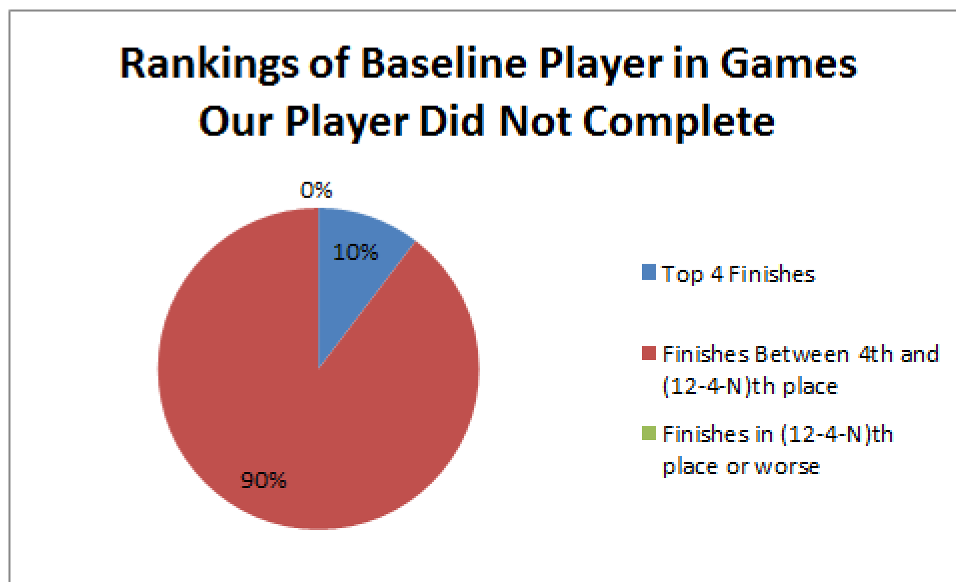

Figure 22: Brief analysis of the baseline player's rankings in games in which our player failed to complete the game. The significance of (12-4-N)th place is that this this rank represents bottom-four finish, where N is the number of players who completed the game.

Someone who views the above figures may be confused by the discrepancy between the fact that our player had a higher percentage of top-4 finishes and a lower percentage of bottom-4 finishes than baseline, but still ranked worse overall. The reason for this is that these buckets (top-4, bottom-4, neither-top-4-nor-bottom-4) do not provide a full picture of the player. For instance, while we had more top-4 finishes than the baseline, the baseline had a higher percentage of top-2 finishes and our player had a higher percentage of fourth-place finishes. Similarly, while we had a lower percentage of bottom-4 finishes, the baseline's bottom-4 finishes were concentrated in the

(12-4-N)th place while our bottom-4 finishes were more heavily concentrated in the (12-4-N)-1, (12-4-N)-2, and (12-4-N)-3 places. Thus, these differences in distribution of the various ranks accounts for the fact that, on average, the baseline player ranked more highly than we did.

4.4.2 ANALYSIS OF OUR FIXED FINAL PLAYER VS. THE BASELINE PLAYER

As was previously mentioned, 90% of our failures in the tournament were due to a minor oversight that did not present itself through our general testing process. To reiterate, our player's implementation does not account for the scenario in which two cars simultaneously arrive at parking lots separated by one road segment, and both cars believe that they can pass safely to the next segment since it contains no opposite-moving cars. They each decide to proceed onto the road, and this causes an inevitable crash.

Because the bug required a very simple fix, we were curious to see how that improvement would have affected our player's tournament scores, particularly with regard to how our player performed relative to the baseline player.

Below, we provide the same graphs we provided for the analysis in sections 4.1.2 and 4.1.3 with revisions to show our fixed player, denoted as g6v1-revised.

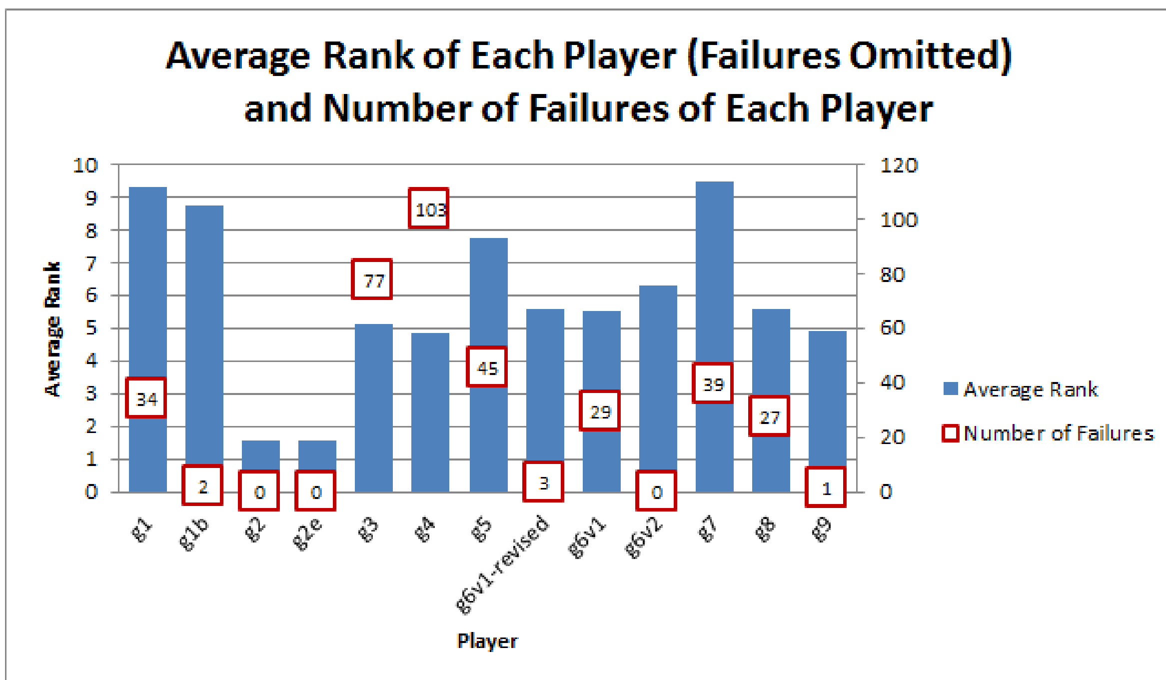

Figure 23: Average rank of all players, including our fixed player, when failures are not ranked.

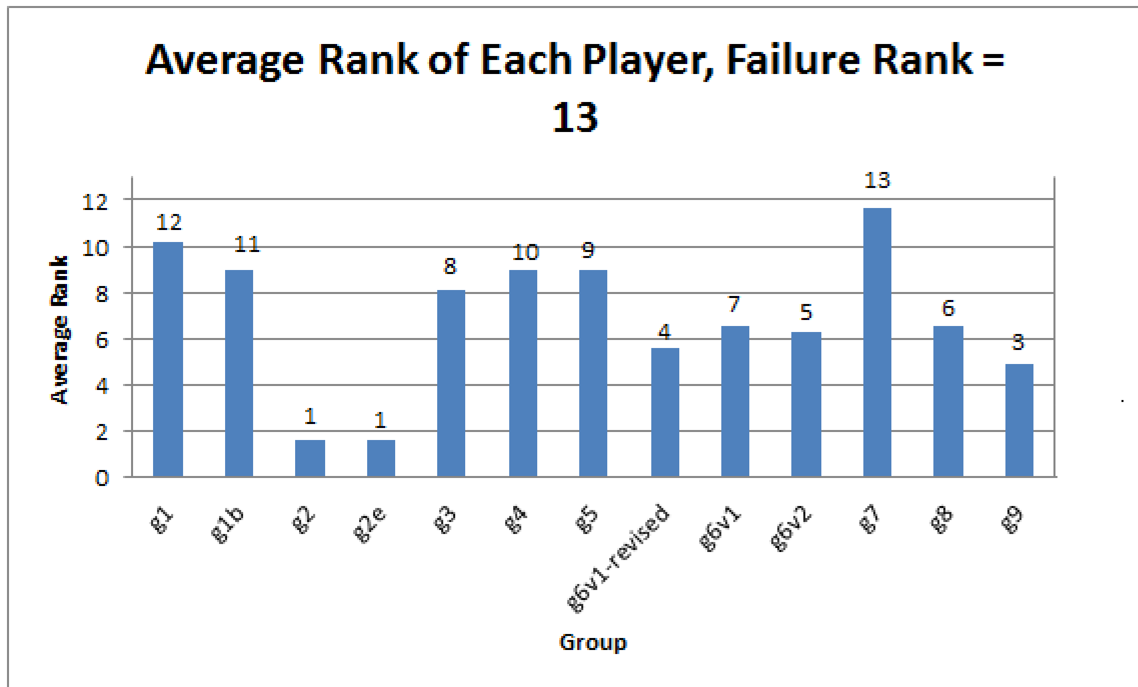

Figure 24: Average rank of all players, including our fixed player, when failures are ranked 13th[7]. As in 4.1.2, the rank of the player's average rank is displayed above the appropriate bar for convenience.

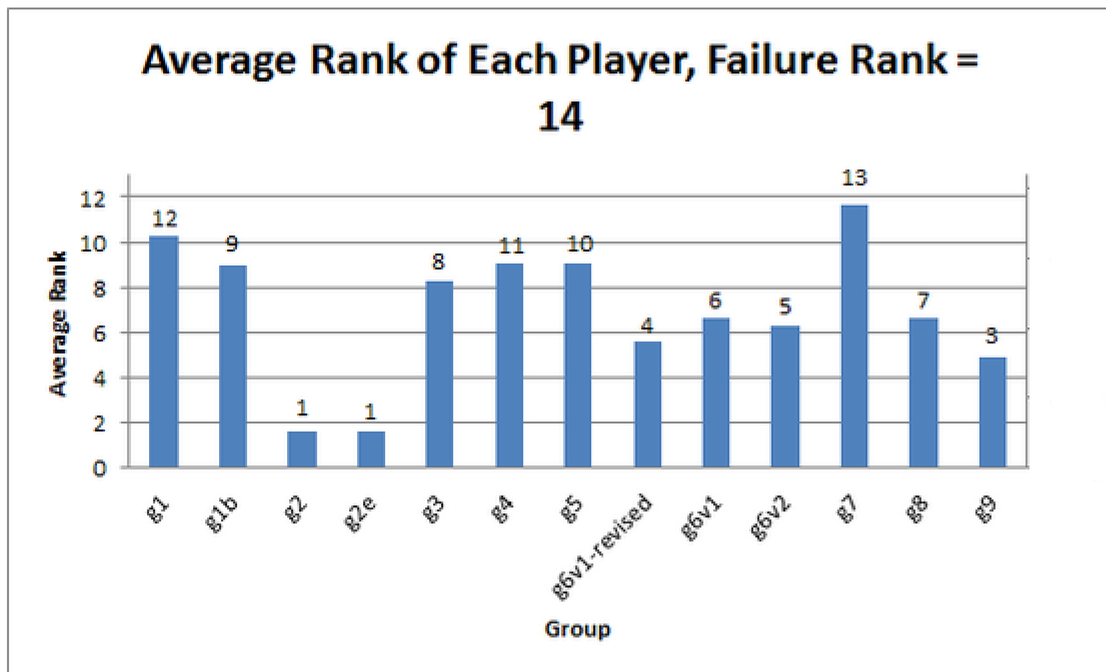

Figure 25: Average rank of all players, including our fixed player, when failures are ranked 14th[8]. As in 4.1.2, the rank of the player's average rank is displayed above the appropriate bar for convenience.

---

[7] Adjusted from 12 in 4.1.2 to 13 to account for the additional player.
[8] Again, adjusted from 13 in 4.1.2 to 14 to account for the additional player.

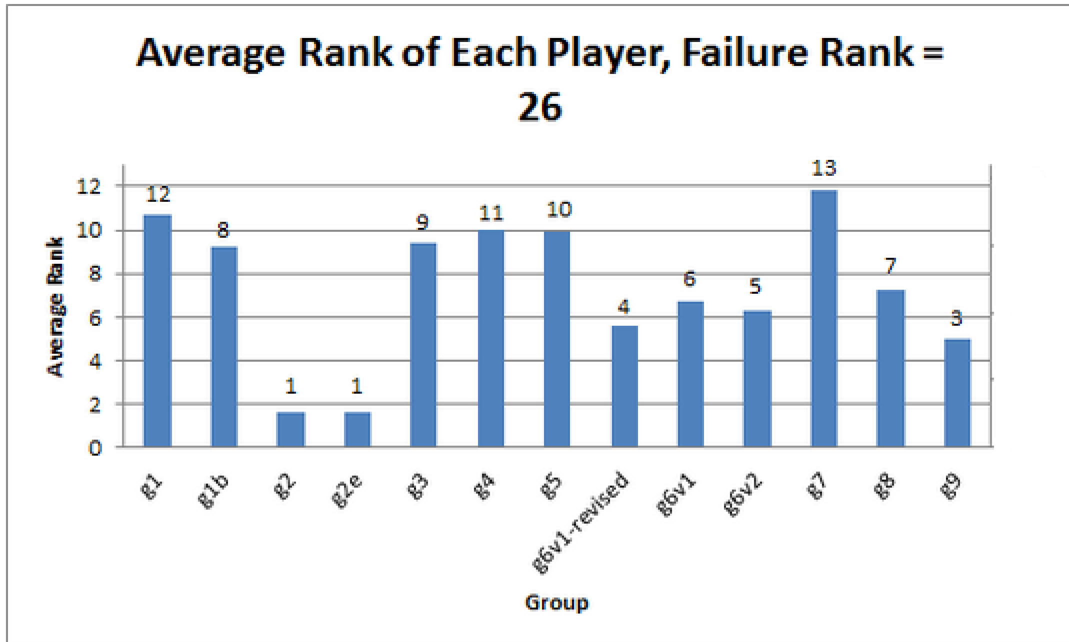

Figure 26: Average rank of all players, including our fixed player, when failures are ranked 26th[9]. As in 4.1.2, the rank of the player's average rank is displayed above the appropriate bar for convenience.

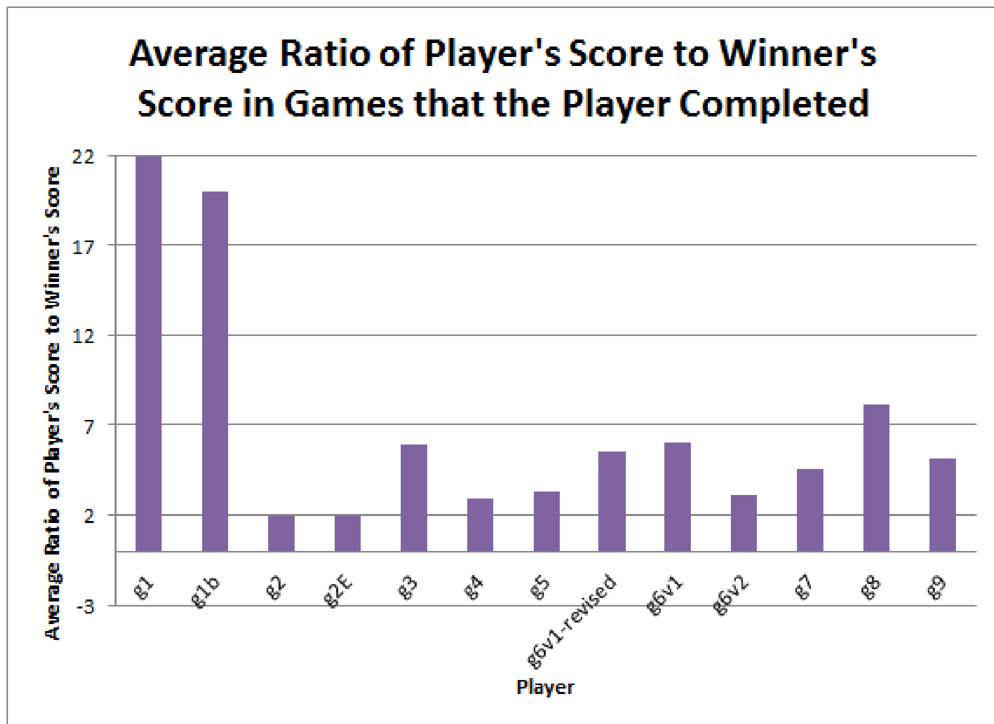

Figure 27: The average ratio of each player's score, including our revised player, to the winner's score in the games in which that player was able to successfully complete the game.

[9] As with the previous two graphs, the "double weighted" failure rank has increased from 13 to 26 to account for the additional player.

As can be seen from the above graphs, the additional successes only improved our player's standings. Though we were still not able to beat Group 2's players or Group 9's player by the metrics above, we were able to substantially improve our performance and were the third best group with regard to ranking, and we were able to improve substantially relative to the baseline. More importantly with regard to rank, we beat the baseline player in every method of failure-weighting, despite the fact that, as expected, we still had three failures after our bug-fix. In the score analysis, we also improved our standing significantly, though we were still unable to beat the baseline; thus, it is clear that, while this fix substantially improved our player, we were still not able to beat the baseline's score in every single game in the tournament.

Additionally, a more direct comparison of the baseline player to our players shows that g6v1 beat the baseline player in 42 of the 182 games and tied it in 39 of the games. Our g6v1-revised player beat the baseline player in 51 of the 182 games, and continued to tie it in 39 games. Because g6v1-revised was able to complete 26 games that the g6v1 player could not, however, this translates into the fact that the revised player beat the baseline player in only 9 of these 26 games.

Though the 26 games did not demonstrate that our revised player beat the baseline in every case, in the 9 games when it did beat the baseline, it beat it by a substantial margin, as shown in the figure below. Also importantly, g6v1-revised never performed more poorly than g6v1 in the games that g6v1 was able to complete.

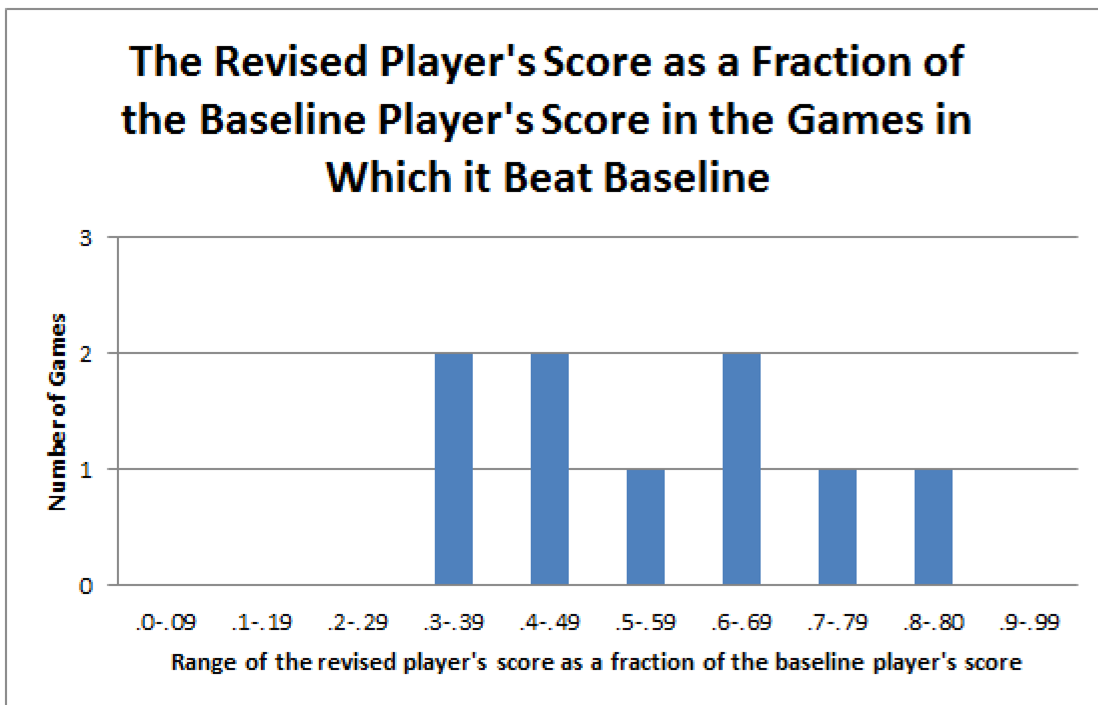

Figure 28: In the 9 games in which the revised player beat the baseline player, the revised player's score as a fraction of the baseline player's score.

In short, the bug fix did allow us some substantial improvements, and we were able to see more cases in which our player outperformed the baseline player: in 5 of the 9 games, the revised player's score was less than half of the baseline player's score. However, our player would still need to undergo additional significant improvements to beat the baseline player in all cases, some of which are described in the following section.

5 FUTURE IMPROVEMENTS

5.1 BETTER HEURISTIC FOR CHOOSING DIRECTION IN PHASE 2

One improvement that could be made to our player is a better mechanism for choosing which direction to start the traffic moving in Phase 2. Right now, the measurement is the direction that has the contains the earliest car. However, this method can be suboptimal in many traffic situations with unbalanced traffic. For example, consider the following scenario: one car arrives at the left at time tick 1, and 10 cars arrive at the right at time tick 2. With our current method, we see that we would prioritize the single car over the 10 cars and this would result in a very high penalty.

One way to address this is to account for the number of cars that are moving in either direction. In fact, we can calculate the expected penalty that is associated with sending the cars right first and sending the cars left first assuming that no other cars arrive and choose which direction to send the cars based on this. In the previously mentioned example, we would be able to see that our penalty for sending the single car first would be much higher and we would then send the 10 cars left first.

5.2 DYNAMIC STRATEGY BASED ON MAP AND TIME DISTRIBUTION

Another improvement that could be made is to use a multitude of strategies for different types of roads. Rather than trying to build a general strategy that would work decently at many maps, it may have been more useful to develop many strategies that are each optimized for a single configuration. For example, with uniform maps, it may have been useful to implement a strategy that makes use of this property in order to allow for car synchronization. Furthermore, we may have been able to let our strategy run for a certain amount of time and then make inferences about the timing configuration of the cars after that. Using this knowledge, it could have been possible to select a specific strategy.

5.3 SPEEDUP OF PHASE 0

During Phase 0 of our player, we only send as many cars on a segment as the next parking lot can hold. This is problematic for configurations with long roads and low-capacity parking lots, as we are sending a very small number of cars relative to which the road can hold. We do this movement because we want to guarantee that in moving from Phase 0 to Phase 1 (in which all the cars on the road are put in the closest parking lot), we do not overflow a parking lot. However, we can be smarter about how we partition cars when moving from Phase 0 to Phase 1. Given the number of cars that arrive at the two end lots (as well as their arrival times), we can calculate where we should stop all the cars (for example, if this is a uniform car distribution, this would be at the parking lot closest to the middle of the road), and batch out our cars accordingly. This would lead to significant speedup and would allow us to perform much better on configurations with long roads and low capacity parking lots.

5.4 BETTER HANDLING OF ZERO-CAPACITY PARKING LOTS

Our implementation defaults to the baseline strategy when a zero-capacity parking lot is encountered. This strategy may not be desirable when our improved baseline player would outperform it. Since the only phase that moves cars into parking lots, and thus would have to handle zero-capacity parking lots, is Phase 0 (Phase 1 also moves cars into parking lots, but those cars are determined by the actions in Phase 0), we could adopt a modified strategy that would locally handle zero-capacity parking lots and continue the rest of our strategy the same way.

6 CONCLUSION

6.1 RESULTS SUMMARY BEFORE BUG-FIX

To reiterate, our goals for our player throughout the project were never to fail and to beat or match the baseline strategy in all games. We tried out several different strategies throughout the design process to achieve these goals, and ultimately came up with a final player whose strategy was to be an improvement upon the basic baseline design. Though we tested our player in a multitude of test cases, in which we used many maps and timing distributions that were conceptually similar to those used in the tournaments, some of the games in the tournaments resolved subtle bugs or logic difficulties that our player and testing did not resolve, and we failed in about 15% of trials. In the games in which we did not fail, we achieved our goal of beating

baseline only some of the time; in our ranking analysis, we achieved an average ranking better than that of the baseline only when we did not include a weighting for our incompletions. However, we did achieve a higher percentage of top-4 finishes and a lower percentage of bottom-4 finishes than the baseline player, which means that in at least some maps we were able to achieve a significant improvement over the baseline player.

With regard to the other players, we were very impressed by Group 2's players' performances and were also impressed that Group 9 achieved a solid rank overall with only one failure. Our player generally performed better than Group 1's players, Group 4's player (when failures were accounted for), and Group 7's player: how we performed relative to Group 5's player and Group 8's player were more sensitive to the specific type of analysis. On balance, we conclude that our player was approximately an average player, with a few great victories and a few significant defeats.

6.2 RESULTS SUMMARY AFTER BUG-FIX

We noticed that almost all of our failures in the tournament were due to a minor oversight which allowed two cars that arrived at adjacent parking lots at the same time to travel onto the road between the lots, resulting in an inevitable crash. This problem required a very small fix in our logic, so we fixed the issue and re-ran all tests to see what effect this had on the results. While the fix did not resolve the three failures due to parking lot overflow, it did indeed fix the other 26 failures from crashing. Our analysis of the revised player shows that it was much more competitive with all other players and beat the baseline more frequently than our original player. In the tournament with the revised player, our average rank improved substantially and was only behind Group 2's players and Group 9's player. Furthermore, in 9 of the 26 games our revised player was able to complete, we beat the baseline player; in over half of these games, our player obtained a score that was half or a smaller fraction of the baseline's score.

With our bug-fixed player, we were relatively close to achieving our goals: we reduced our failures to only 1.5% of tournament games, and we beat the baseline's score by significant margins in a variety of configurations and time distributions.

7 APPENDIX

7.1 INDIVIDUAL CONTRIBUTIONS

All group members contributed to development of the overall strategy and to the compilation of the report.

KRISTA
Krista designed and wrote the test suites for regression testing of our player and other groups' players, in addition to writing a bit of code for the safe strategy. She also did the statistical and numerical portions of the tournament analysis.

ANDREW
Andrew contributed some code to our player, did some strategy development, and provided the analysis of our player's good scores, bad scores, and failures in the tournament.

MATTHEW
Matthew wrote the majority of the code for our player for this project, particularly with regard to the baseline and improved baseline strategies.

7.2 ACKNOWLEDGEMENTS

We would like to thank Jiacheng for implementing and improving the simulator throughout the duration of the project and for running the official tournament.

We would also like to thank Professor Ross for creating this complex and interesting project and for inciting thought-provoking class discussion.

We thank the other groups for creating and improving their players throughout the project so that we could learn from their successes and failures and test their players against our own. In particular, we would like to acknowledge Group 2, whose Semifinal and Final implementations were truly outstanding and served as an inspirational and motivating opponent.