# *CSE 202  - Project Ludo - Experimentation*

*Andrew Ghafari - A59020215*
*Chirag Dasannacharya - A59016593*
*Jay Jhaveri - A59017531*
*Niraj Yagnik - A59018067*

## I - Abstract:

In this part of the project, we decided to implement all the algorithms that we had discussed in the previous report. We also built a Ludo game that abides by our new formulation of it, with the simplifications suggested. This allowed us to run actual games between algorithms to see if our intuition and proofs will be backed by the empirical data. In addition to that, we will be running additional tests between some algorithms just for fun. All of the metrics and analysis will be provided and discussed below.

## II - Algorithm Implementation

We implement all our strategies, as described in the report, in Python, and write a game simulation to run them. All of the developed algorithms can be found inside our github repository.
Please refer to the link below:
https://github.com/JayJhaveri1906/Project-Ludo

## III - Quality Comparisons and Evaluation

We observe the following results on playing various algorithms against each other, having 4 pawns each, each over a total of 10000 games -

### A - Win Rate for Random, Random, Random, Random:

| Random | Random | Random | Random |
|---|---|---|---|
| 2535 | 2619 | 2439 | 2407 |

We observe that on having all 4 players run with a random strategy, their outcomes are fairly similar, as expected. There is a slight shift in favor of earlier players, this is more noticeable in later runs. This suggests that there is a slight advantage offered to players who start earlier.

### B - Win Rate for Fast, Fast, Fast, Fast:

| Fast | Fast | Fast | Fast |
|---|---|---|---|
| 2776 | 2470 | 2470 | 2308 |

We observe that similar to the case with 4 random players, the initial player(s) enjoy a slight advantage.

### C - Win Rate for Fast, Random, Random, Random:

| Fast | Random | Random | Random |
|---|---|---|---|
| 8835 | 391 | 409 | 365 |

We see that the Fast algorithm performs significantly better than a random algorithm, winning 88% of matches against a combination of three random opponents. We also perform the experiment with a single random opponent:

D - Win Rate for Fast, Random:

| Fast | Random |
|---|---|
| 8491 | 1509 |

The outcomes are very close to our expectation that a fast algorithm is at least 4 times better than a random algorithm. In practice, we find that it is closer to 6 times better, this could be explained by interference between random players and similar factors.

E - Win Rate for Aggressive, Fast, Fast, Fast:

| Aggressive | Fast | Fast | Fast |
|---|---|---|---|
| 4847 | 1537 | 1634 | 1982 |

The Aggressive algorithm is, indeed, found to perform significantly better than the Fast algorithm. This difference is less extreme than the corresponding difference between the Fast and Random algorithms, but is still significant. As previously, we perform a run between a single Aggressive and a single Fast player.

F - Win Rate for Aggressive, Fast:

| Aggressive | Fast |
|---|---|
| 7474 | 2526 |

The Aggressive algorithm performs 3 times better than a Fast algorithm. It is seen that raising the number of fast players (as in the previous case), causes the rate of the aggressive player’s wins to decline, rather than stay the same as was the case between Fast and random players. This can be explained by the fact that Fast algorithms cause less interference to each other, having only one active pawn at a time, than random players. This helps them accidentally ‘cooperate’ between each other relative to the aggressive player, compared to the case with Fast and Random play.

<u>G - Win Rate for Aggressive, Random:</u>

| Aggressive | Random |
|---|---|
| 9617 | 383 |

For completeness, we also run a comparison between a single instance each of the aggressive and random algorithms. We find that the Aggressive algorithm beats the random algorithm in more than 96% of cases, or a 25-fold advantage. This is much better than the 6-fold advantage seen with the Fast algorithm.

<u>H- Win Rate for Random, Defensive:</u>

| Random | Defensive |
|---|---|
| 3956 | 6044 |

Defensive beats the Random Algorithm 60% of the time. This is definitely not as good as aggressive’s performance against random but it was kind of expected. Sometimes Defensive will not risk his position by chasing a random’s pawn, which could have been killed easily given that it is ¼ likely to move. This is what Aggressive is doing, and that is what is lacking here. The results are proof of this.

<u>I- Win Rate for Fast, Defensive:</u>

| Fast | Defensive |
|---|---|
| 4899 | 5101 |

Defensive beats Fast by a very small margin, which consolidates what we have previously discussed in our last report. Defensive will perform as good as fast, and will beat it in some niche scenarios. The results of 51% for Defensive and 49% Fast confirm our intuition.

Now for a fun matchup, we are going to play Defensive vs Aggressive, and see who triumphs.

J - Win Rate for Aggressive, Defensive:

| Aggressive | Defensive |
|---|---|
| 6075 | 3925 |

Aggressive beats defensive for the majority of the time. This is probably due to the same phenomenon that was happening when defensive was playing with random, which is overestimating the risk of leaving a spot, which leaves the pawn as an easy target for aggressive. Aggressive wins 61% of the time, versus 39% for defensive.

Now we are going to level it up for 4 players.

K - Win Rate for Defensive, Fast, Fast, Fast:

| Aggressive | Fast | Fast | Fast |
|---|---|---|---|
| 2891 | 2641 | 2368 | 2100 |

The defensive algorithm is, indeed, found to perform better than the Fast algorithm. This difference is less extreme than the corresponding difference between the Aggressive and Fast algorithms, but is still good enough.

L - Win Rate for Defensive, Random, Random, Random:

| Defensive | Random | Random | Random |
|---|---|---|---|
| 4507 | 1891 | 1744 | 1858 |

The defensive algorithm is, indeed, found to perform better than the Random algorithm. This difference is less extreme than the corresponding difference between the Aggressive and Random algorithms, but is still really significant.

Now we will introduce our latest algorithm, the Mix one.

<u>M - Win Rate for Random, Mix:</u>

| Random | Mix |
|---|---|
| 1355 | 8645 |

We first ran an experiment between Random and Mix, with the latter triumphing almost 86.5% of the time. It is definitely a very high percentage, but still less than what Aggressive was able to perform. Let us now put this new algo, head to head against aggressive and defensive.

<u>N - Win Rate for Fast, Mix:</u>

| Fast | Mix |
|---|---|
| 3224 | 6776 |

The mix algorithm is also better than Fast, and beats it almost 68% of the time. This is less significant than the matchup against Random, but still a good result for Fast.

<u>O - Win Rate for Defensive, Mix:</u>

| Defensive | Mix |
|---|---|
| 3218 | 6782 |

This result is really similar to the previous one. Our mix algorithm is still beating the previous algorithms as suggested by our analysis. Let us see how it compares with Aggressive.

P - Win Rate for Aggressive, Mix:

| Aggressive | Mix |
|---|---|
| 3824 | 6176 |

Aggressive performed better than Fast and Defensive against Mix, but the latter still triumphs. This also validates what was previously discussed in our paper.

Final Round:

Q - Complete Matchup:

This is for fun. We are going to play all of our algos (without random against each other).

| Defensive | Random | Random | Random |
|---|---|---|---|
| 4507 | 1891 | 1744 | 1858 |

The defensive algorithm is, indeed, found to perform better than the Random algorithm. This difference is less extreme than the corresponding difference between the Aggressive and Random algorithms, but is still really sig

**Note about positioning:**

We see some interesting results when player orders are changed **-**

Win Rate for Aggressive, Fast, Fast, Random:

| Aggressive | Fast | Fast | Random |
|---|---|---|---|
| 5954 | 1820 | 2209 | 17 |

Win Rate for Random, Fast, Fast, Aggressive:

| Random | Fast | Fast | Aggressive |
|---|---|---|---|
| 46 | 3543 | 2989 | 3422 |

This suggests that the positioning of the players affects the outcome of the game. Even though this wasn't planned nor expected, this was still backed by empirical data. On first glance, we thought that this might be due to the specificities of each algorithms, and being surrounded by fast players, or aggressive players, or mix of these, will affect how the game run, since every player's actions are dependent on the board state at each turn, which is in itself dependent on all the player's moves. This is also something that can be discussed in further details and will be also considered as future scope of this project.

**Next Steps**

Possible next steps would include, first removing the simplifications that we introduced to the Ludo game. For example, adding back the need of a dice roll of 6 to get out of the house will bring additional complexities to the game, algorithms, and proofs. In addition to that, resorting back to the single destination as opposed to the whole runaway would also add additional constraints on completeness of some algorithms as well, most notably, Random Algorithm.

In addition to that, next steps would include coming up with new algorithms that defeat our hybrid risk-reward based algorithm, and proving its superiority over ours. One last thing, which might be the hardest, is to extend the game to more than 4 players and possibly more pawns per player. In case of choosing this route, there will be some design choices that need to be made, especially for the number of safe spots for example, that will definitely have to increase along with the rise in the number of players and pawn per player, or else the game might never end.

One last thing is how the positioning of the players' different algorithms with respect to each other affect how the game runs and what results we get. This can definitely be studied in further detail.

**Conclusions**

To conclude, we discuss a series of approaches that can serve as a valuable tool for players to optimize their strategic decision-making. By using our algorithms as a starting point and adjusting them to fit their specific situations, Ludo players can increase their chances of winning and outwitting the opponent.

In this implementation section, we transformed everything that was previously discussed, mainly the Ludo game with its rules and the algorithms proposed, into functional python code that we utilized to run experiments and check if the empirical results match our code.

We ran numerous experiments, some of it for fun, and the majority to check whether our intuition and proof were right about the performance of the algorithms. Experiments were a really fun way to implement everything that we had conceptualized, from the new game itself, to the algorithms and the matchups between them. Almost all of the experiments consolidated what we have previously said, and when we tried rotating the positions of the algorithms, this was an eye opener for us. We haven't thought about why this is happening and what might be the reason. More analysis is definitely needed in order to fully understand these reasons. Nonetheless this was really fun and entertaining.

**Final words:**

This was a very fun project to conceptualize and implement.
A big thanks to the TAs and Prof. Impagliazzo for the effort put into this course.
We hope you'll find this project as interesting as we did.