COM2108 Programming Assignment: Results

1. Introduction

In this report I hypothesise on the performance of my players playing against each other and present tests to determine whether those speculations are correct. Later, I conduct those tests and comment on the results they provide. I also include a brief discussion on the efficiency of the players.

The *implemented players* I refer to in this document are the players I designed and realized. I present and detail them in the design document.

2. Hypothesis

I hypothesise that every player will perform much better than the provided *randomPlayer*, because they do make a choices which employ at least some level of reasoning.

In terms of the players playing amongst themselves, I am inclined to theorise that there will not be any major differences. This is due to the fact, that although they use different tactics, the game of dominoes is not very complex in its nature and is heavily dependent on the dominoes which are randomly dealt. Hence, I project that more often than not some tactics would not be able to produce a domino thus reducing the difference between the players.

Despite that, I hypothesise that *beginnerPlayer* will perform the best out of all, because it exploits the highest number of tactics and the choices between those tactics are the most complex.

3. Setting up the Experiment

3.1. Testbed

Designed and implemented players' performance will be tested by playing the game of dominoes, using the 3s-and-5s variation of the game. In his adaptation of the game, the pips at the each side of the domino board are summed after each turn and checked whether they are a multiple of either 3, 5 or both. The score is awarded as follows:

Pip Total	Awarded Score
3	1
5	1
6	2
10	2
12	4
15	8
18	6
20	4
Other	0

Table 1 Scoring after player plays a domino

3.2. Measures

In this experiment, the only measures introduced are the wins and losses of players whilst playing the game of 3s-and-5s dominoes.

3.3. Baseline

randomPlayer is included in the experiment as a baseline. In this type of experiment it is hard to derive a baseline due to the inability to fully differentiate between 'bad' and 'good' play. Recognising that, *randomPlayer* is included in the experiment to act as more of a reference and starting point for all other players.

3.4. Test subjects

All implemented players presented in the design document along with the *randomPlayer*, which is provided in supplied file *DomsMatch.hs*. Below I present a table which assigns a colour to each player for the convenience whilst examining the results.

Figure 2 Colours associated with the players

3.5. Structure of Tests

- Each player plays 1000 games with every other player with the same seed;
- Out of these 1000 games, the player starts as player 1 *(P1)* 500 times and player 2 *(P2)* other 500 times. This is to make sure that being the first player does not give the player an unfair advantage, because the players seem to win more if they are placed as *P1* rather than *P2 (observation made before redaction of DomsMatch.hs on 2020/12/10)*;
- 10 different seeds are used;
- Therefore, during the experiment, each player plays 1000 games against every other player per seed, 5000 games per seed overall and 50000 games in total.

4. Experiment Results

4.1. Overall Results

Below table gives an overview of how many games has the player won and lost overall and how well each player performed against every other player by highlighting the number of wins in each scenario.

	randomPlayer	highestScoringPlayer	reachesForGoalPlayer	avoidsKnockingPlayer	saboteurPlayer	beginnerPlayer
Overall Games Won:	1171	28916	32096	22552	31772	33015
Overall Games Lost:	48351	21064	17446	27448	18228	16985
Games won against **randomPlayer**	x	9647	9410	9587	9842	9865
Games won against **highestScoringPlayer**	333	x	5661	3792	5527	5751
Games won against **reachesForGoalPlayer**	132	4339	x	3002	4848	5125
Games won against **avoidsKnockingPlayer**	413	6208	6998	x	6823	7006
Games won against **saboteurPlayer**	158	4473	5152	3177	x	5268
Games won against **beginnerPlayer**	135	4249	4875	2994	4732	x

Table 3 Summary of testing results

4.2. Distributions of won and lost games based on player

In the below chart the win and loss distribution over all played games of each player is presented. It is clearly visible that *randomPlayer* performed the worst whilst *beginnerPlayer*, *saboteurPlayer* and *reachesForGoalPlayer* are the leaders. Amongst them, *beginnerPlayer* displays the best performance, although the difference between the three is not very significant.

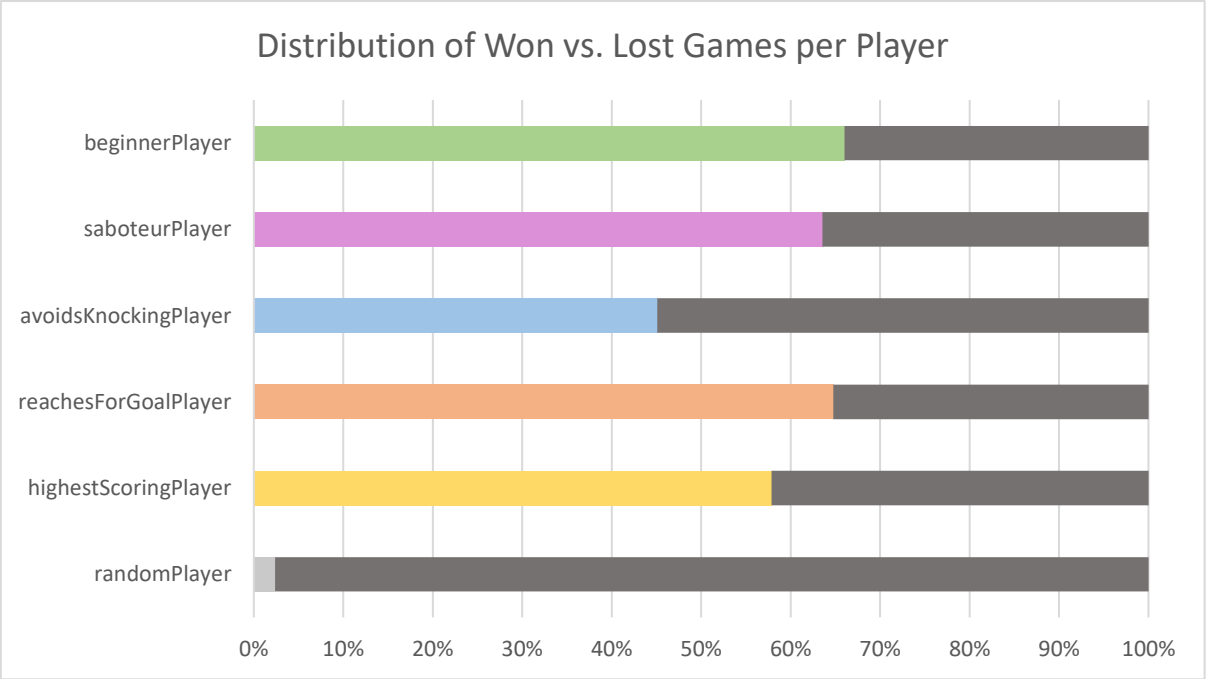

Chart 4 Distribution of wins/losses of each Player

4.3. Average wins of each player

Percentage of wins of each player over all played games is presented in this diagram. Again, three clear winners emerge: *reachesForGoalPlayer*, *saboteurPlayer* and *beginnerPlayer*. They are closely followed by the *highestScoringPlayer*. *avoidsKnockingPlayer* performs the worst out of all implemented players, but still is more effective than the *randomPlayer*.

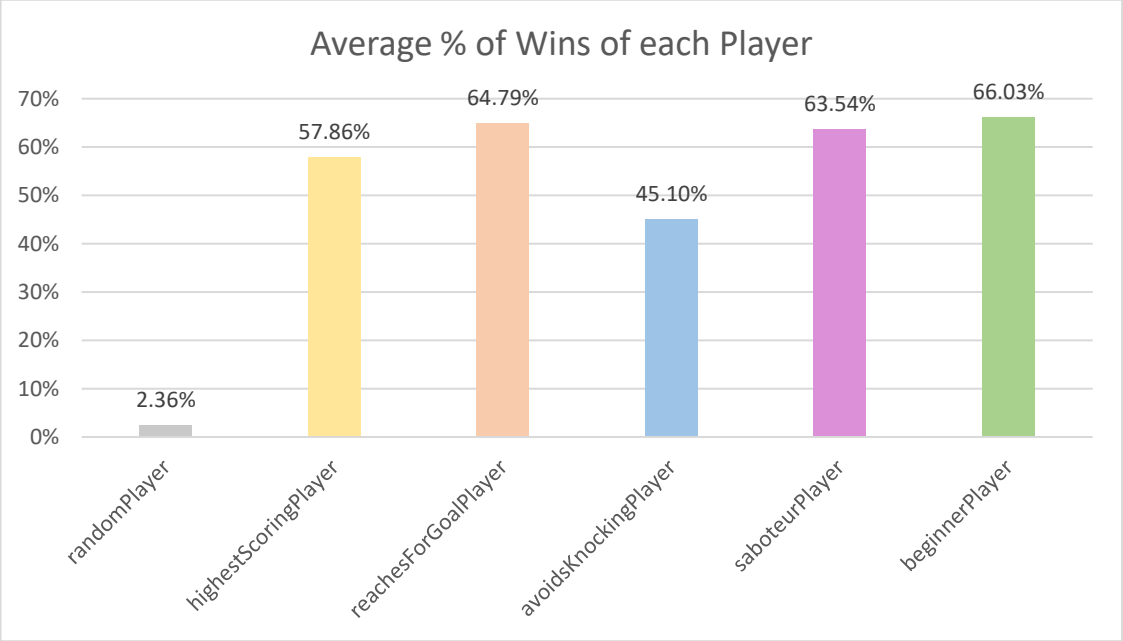

Chart 5 Percentage of games won by each Player

4.4. Comparison of best-performing players based on the provided seed

Below I examine the wins per seed of the top three players: *beginnerPlayer*, *reachesForGoalPlayer* and *saboteurPlayer*. It is apparent that the *saboteurPlayer's* performance is worse than the other two and *reachesForGoalPlayer* rivals *beginnerPlayer* quite successfully and even manages to surpass the number of wins on some seeds.

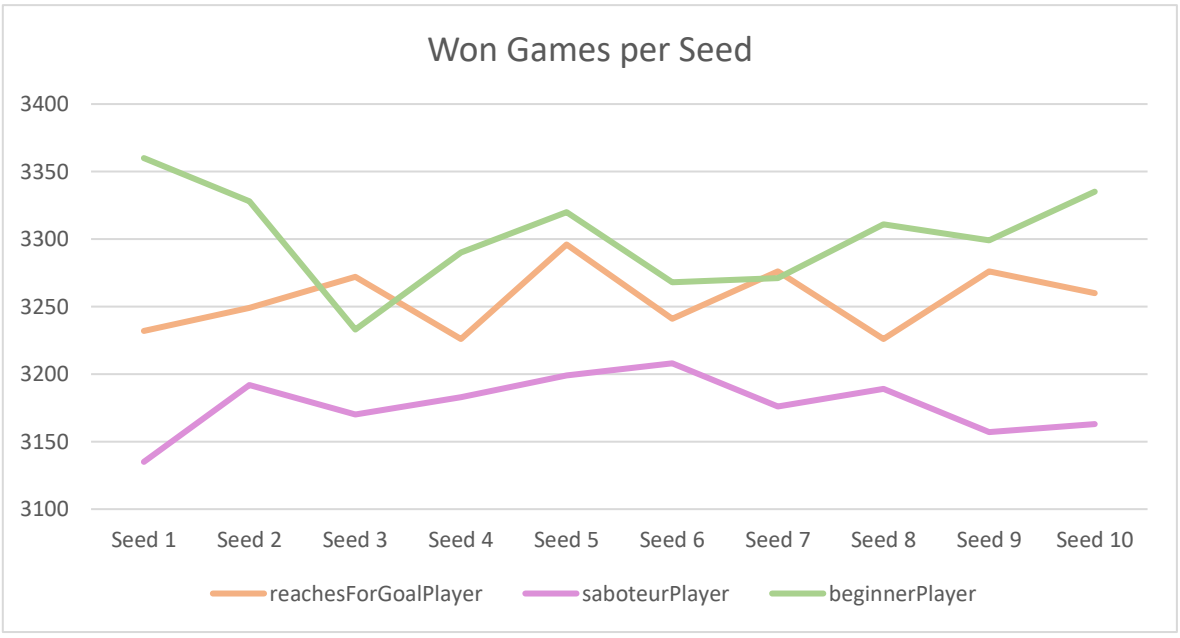

Chart 6 Won games comparison of highest scoring Players

5. Discussion of Results

As expected, the implemented players do perform better than the *randomPlayer*. However, I did not expect the gap to be so huge- the worst-performing *avoidsKnockingPlayer* won roughly 19 times more games than *randomPlayer*.

No major differences between effectiveness of the players were found. The biggest gap in performance is found comparing *avoidsKnockingPlayer* and all other players, where the former plays from 1.2 to 1.5 times worse than the other implemented players.

Despite that, there are some differences in performance and the most-winning player can be recognized. As expected, beginnerPlayer is the most effective out of all, but the performance increase is not that high and only is 1.02 times better than the second-best *reachesForGoalPlayer*.

Out of the five implemented players *avoidsKnockingPlayer* performs the worst suggesting that the strategy of keeping your options open during the game is not the one which should be focused on. As indicated by results, three clear winners emerge- *reachesForGoalPlayer*, *saboteurPlayer* and *beginnerPlayer*, whilst the later shows greatest performance out of the three.

Testing has been conducted before the final update of *DomsMatch.hs*, hence if same seed and player combinations would be tried they would provide slightly different results. The overall testing results should not be influenced by much, because as stated in *Section **0*** the unfair advantage was noted and tests are structured in such a way that the unfair advantage is eliminated or at least minimised as much as possible.

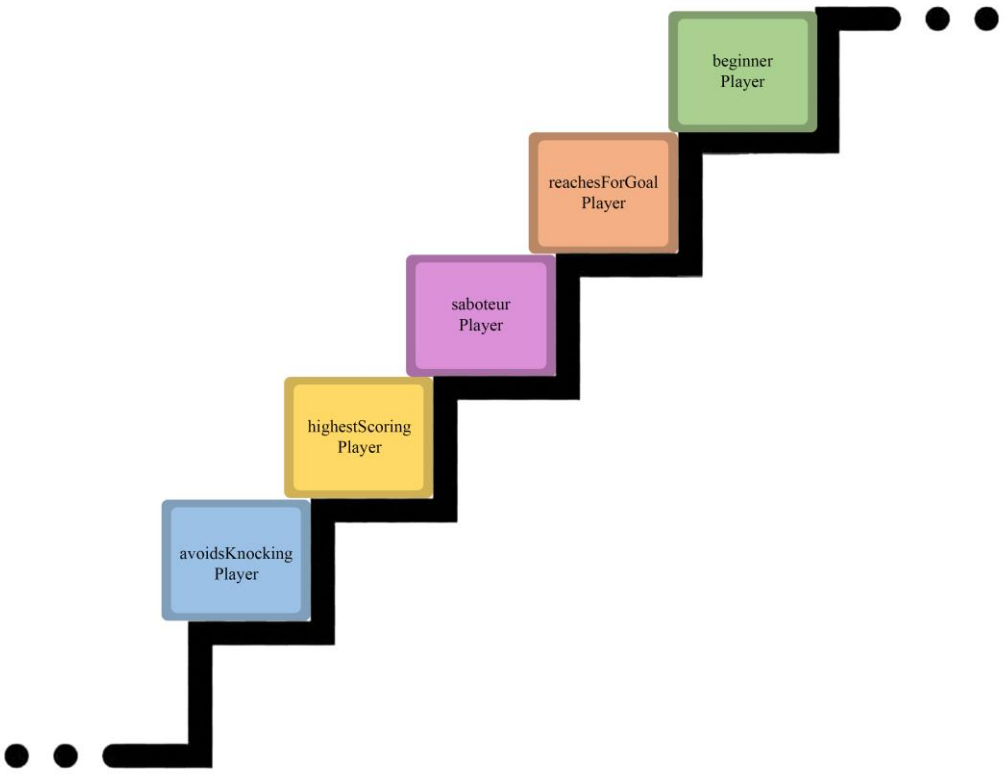

Figure 7 Players visualisation based on their performance

6. Efficiency of Players

6.1. Testing Efficiency

For efficiency testing I have played all players against randomPlayer for 1000 games with 10 different seeds. Time measurement was retrieved by using *:set +s* command in *ghci*. Although this perhaps is not the most sophisticated efficiency measure, it does allow rough comparison between the players.

6.2. Time taken by each player per seed

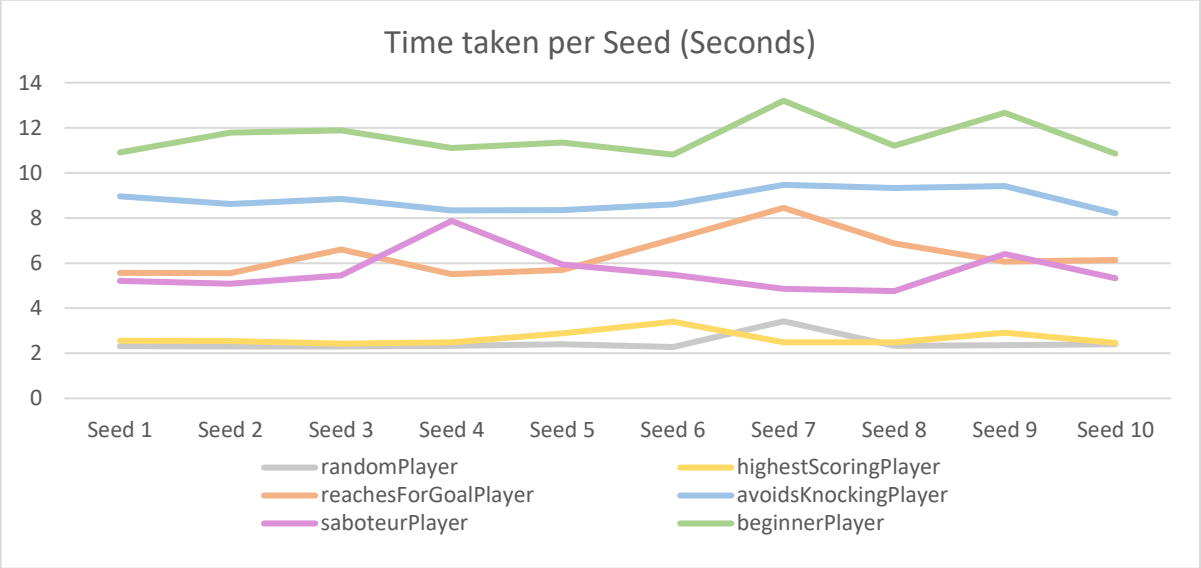

Chart 8 Efficiency of Players based on Seed

6.3. Time taken by each player on average

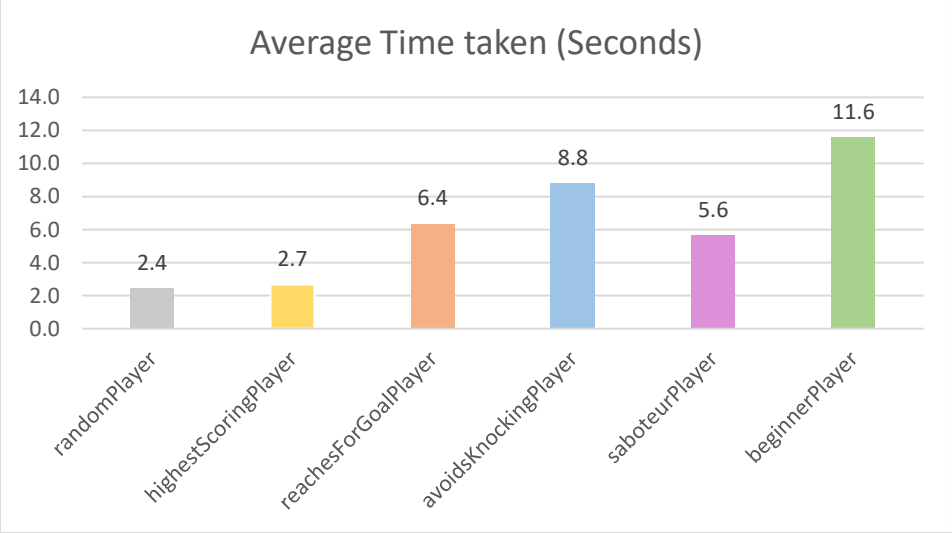

Chart 9 Average efficiency of Players

6.4. Efficiency Discussion

In terms of efficiency unfortunately the best-performing *beginnerPlayer* is the most inefficient out of all. However, this is understandable considering that the player makes much more comparisons than the other players.

The worst-performing implemented *avoidsKnockingPlayer* is also second-worst in efficiency which prompts to consider the worth of the player in general.

Efficiency of the *highestScoringPlayer* comes as a nice surprise- although it is much more competent than the *randomPlayer* in performance, their average efficiency differs by only 0.3 seconds and on some seeds the former performs even better.

7. Conclusions

7.1. Performance

Referring to the performance results in *Section **4*** and results discussion in *Section **5*** it seems that the best strategy in the game of 3s-and-5s dominoes is playing the highest scoring domino whilst taking into account the opponent's state and trying to block them if they are doing better or are near the end of the game. Each *reachesForGoalPlayer* and *saboteurPlayer* incorporates one or the other strategy whilst *beginnerPlayer* combines them both and hence performs the best out of all players.

I do believe that the performance results could be somewhat improved by incorporating the aspect of looking at the History to determine what dominoes the opponent might have and thus play the highest scoring dominoes which would not allow the opponent to score more in the following turn. However, as mentioned before, due to the nature of the game I believe the performance would not improve in a major way- the dominoes are still dealt randomly and luck is quite a big factor in this game.

7.2. Efficiency

Taking into the account the efficiency discussed in *Section **6***, the strategy to avoid blocking, as *avoidsKnockingPlayer* is behaving, is both quite inefficient and does not bring desirable performance. If the efficiency is important, the best performance-efficiency trade-off is captured in the *highestScoringPlayer*, because it is almost as efficient as *randomPlayer* but performs roughly 25 times better than it and performs only 1.14 times worse than the best-performing *beginnerPlayer*.

Although there are attempts to improve efficiency by avoiding the search for a domino if the game situation is not right (e.g. Tactics *playExactScore* & *playSecondBest*), the efficiency could definitely be improved at least a bit by using *foldr* or *foldl* instead of basic recursion in relevant functions.

7.3. Closing statement

In conclusion, *beginnerPlayer* is the most advanced out of all players. Even though its efficiency is somewhat lacking compared to the other players, the play time is still reasonable enough to be used in the real-world. Unfortunately, *avoidsKnockingPlayer* is the worst player out of all, based on the lacking performance combined with second-highest running time.

Figure 10 Awards ceremony of winning implemented Players