CS221 Project Final: DominAI

Guillermo Angeris and Lucy Li

I. INTRODUCTION

From chess to Go to 2048, AI solvers have exceeded humans in game playing. However, much of the progress in game playing algorithms have centered around perfect information games, where players' possible moves are visible to each other. We explored possible solutions that run in tractable time for Latin-American dominoes, an imperfect information game. This version of dominoes is four-player, team-based and zero-sum with relatively simple rules, making it it ideal for attack with modern algorithmic and approximation tools.

We expect humans to play at least as well as greedy—picking the tile with the greatest number of pips in their hand which is able to be played. Most novice players use this strategy, which does not consider the tiles that other players are likely to have and use that information to block opponents. We expect human to do at most as well as pure-strategy optimal, which is minimax/negamax.

We developed two negamax-based algorithms to approximate an ordering over given moves for the computer player. Both algorithms use the probabilities of all player tiles on the board to weigh evaluation scores—in general, it should be noted that the game suffers heavily from combinatorial explosion throughout the crucial opening rounds.

II. RELATED WORKS

There has been some work on reducing the problem of imperfect information to many problems of perfect information, most famously in the Perfect Information Monte Carlo (PIMC) algorithm [1], as well as a similar Monte Carlo approach used for contract bridge [2]. Other algorithms have been proposed [3], [4] which either perform similar reductions or which can solve for exact Nash equilibria.

Due to the game being relatively unknown in most places where AI research is common, there is a dearth of AI/ML work specifically focusing on dominoes. Only a few relatively old papers which touch upon the subject of reinforcement learning for strategies [5] in (comparatively) low-power computers really exist. There are also some texts that refer to the two-player variant of the domino game [6], but there is a lot less information in this game relative to the four-player one in question, as 28 dominoes are distributed over four players and we only observe 7 of them in the initial game. There are some further (non-academic) references to domino AI, though the specification seems to use a relatively naive AI which does not infer other players' dominoes[1] and is therefore unlikely to be competitive.

III. TASK DEFINITION

Given the current state of a dominoes game, our agent aims to return the optimal move in order to beat its opponents while helping its partner.

A. Game Details

The game has the following rules. Let $N = \{1, 2, 3, 4\}$ be the set of players, with some partnering $\{\{1, 3\}, \{2, 4\}\}$, and let S be the set of 28 unique dominoes, which is the double six set, e.g. each domino is a set of the form $S_{ij} = \{i, j\}, 0 \leq i, j \leq 6$ and i can be equal to j. The *total number of pips* of a domino is defined as $p(S_{ij}) = i + j$. Players sit in a circle with team members facing each other, and play counter-clockwise.

The game begins with setting a uniform permutation of the dominoes ($P_{ij} = \pi(S_{ij})$) and allowing player i to pick dominoes[2] $\{P_{8(i-1)+1}, P_{8(i-1)+2}, \ldots, P_{8i}\}$ from the permutation. In other words, players each start a round with 7 randomly allocated tiles. The first move of the first round must be made by the player with the $\{6, 6\}$ domino.[3] and the play is continued to the 'left,' in particular to $i \to i + 1$ and $4 \to 1$. Additionally, the first move of the next round is made by the person to the 'left' of the first player in the current round and

[1] The main example for a simple AI was found in https://github.com/carlosmccosta/Dominoes/ which was likely a class project and also seemed to be focused on OOP as a programming paradigm, rather than the player AI itself.

[2] Here we use the natural isomorphism $\{i, j\} \mapsto k$ where $1 \leq k \leq 28$ for simplicity.

[3] We use repetition of a number in sets for symmetry. This is not strictly necessary as $\{6, 6\} = \{6\}$.

this rule continues to be applied every round until the end of the game.

At their turn, a player can only perform three possible actions: (1) put down a piece with an endpoint matching one of the endpoints of the current board, (2) pass if the player has no moves they can make (e.g. that (1) cannot be completed), or (3) call that a player has passed even though they could have played. In the latter case, if the player calls the foul correctly, the round ends and 50 points are added to the score of the offending team. Each round can also end when one player has run out of dominoes or when no player can put down a legal domino. In the former case, the team of the player who has run out of dominoes wins the round and the opposing team has to add the current number of pips on their hand to their score. In the latter case, the team that has the highest number of pips in their hand loses and must add that number to their total score. The first team to get to 100 points loses and the game ends.

More mathematically, in a given round, we represent the board B as a an ordered tuple of the moves performed by each player and the player who began the round. PASS identifies the player has passed and CALL identifies that the player called. Otherwise, the move is just the domino that the player put down such that the board in any given round is given by a tuple of dominoes, P_{ij}. If a domino be placed at either end of P_{ij}, then there are two possible moves: one where the domino is placed at the i end and the other where it is placed at the j end.

B. Evaluation

Ideally, the performance of an AI agent should be evaluated based on how many rounds it wins against experienced human players. However, due to limited time, people, and resources, the bulk of our results rely on playing 100+ games of our advanced algorithms against our baseline with different parameters.

C. Infrastructure and Game Representation

We built game engines that allow humans to input moves and AIs to play against each other. These frameworks tracked moves made by players as well as outputting game statistics such as tile probabilities, wins, and final pip sums per team. [4]

The game's state is defined by the current open ends on the table, the plays that have been made so far by each player, and the tiles still not played. For it to intelligently assess the state of the game, the AI player also tracks and updates the probabilities of each player having a tile in the game. For example, at the beginning, all 21 tiles not in the AI's hands have a 1/3 probability of being in each of the other player's hands. When a player passes, we know that this player has zero probability of having any tile with a value equal to any of the open ends on the table. So, we can renormalize the probabilities pertaining to the other players for those tiles. When a player puts down a tile, we update the probability of that tile to be 1 for that player, and 0 for all other players. We additionally have to update the probabilities of all other tiles, because there is a higher probability of any tile belonging to a player with more unplayed tiles than one with fewer. In other words, a player's move not only changes the probability of the tile in question but also how remaining unknown tiles are likely allocated among players.

IV. APPROACH

The simplest strategy is to play in a greedy manner; e.g. if there are any moves that can be played, the current players plays the domino with the most pips in their hand. Many beginners play in this way and the optimal strategy coincides with the greedy one in later parts of the game, but this strategy often fails early[5] on as optimal moves in the early game aren't always the ones with highest value. We expect (and therefore assume from here on out) that players do no worse than playing greedy. The most optimal strategy in terms of pure actions[6] is to minimax/negamax the rest of the players that are not on your team, while keeping track of all possible sets of dominos that have been observed.

The idea for approximating solutions to the domino game problem is to convert the original, partial information problem into a decomposition of many deterministic, complete information games. Note that this case is strictly a lower bound to the best possible strategy if there is no randomization on the part of the opposing players.[7]

[4] See https://github.com/guillean/DominAI.

[5] A simple construction where this is the case is found in the book [7] or in https://gists.github.com/guillean under 'domino_counter_example.txt'.

[6] In general, Nash equilibria in imperfect information games are mixed—that is, they specify a probability distribution over some support with more than one element.

[7] This is because negamax opponents always know the correct hand of the player, but the player is taking expectations by sampling from the distribution.

A. Negamax

As mentioned earlier, the most optimal pure strategy is minimax: assume that the opponent is playing optimally and alternate between minimizing the value over the opponent's actions and maximizing the value over our own. Using the negamax variant of this recurrence simplifies the problem and improves runtime over standard minimax [8].

Negamax is used for zero-sum, two-player games. Dominoes is a zero-sum game because the value of one player's team's score is the negation of that of the opponent. Though our version involves four players, pairs of players cooperate in teams and due to the final score of a given player being symmetric over the total team's score, each team can be seen as one large, colluding player playing optimally. The latter holds because we are assuming that we picked a particular set of dominoes which is common knowledge (a worst-case assumption, since every sample assumes the opponent knows the exact hand of the computer on every round).

Our evaluation function for the current round (assuming early termination of the search) is the expected difference in total dominoes between the current player's team and the opposing player's team; in general, since we seek to maximize the difference between the margin of player. We also considered using a modified evaluation function, which included a feature corresponding to the difference in number of played pieces for each team, but this did not seem to improve performance.

B. Imperfect Minimax Search

Minimax and its negamax variant are applicable to perfect information games. Games of imperfect information such as dominoes are games in which some players have information that other players are unable to see, though the game's structure and payoffs are common knowledge. In our specific case, we do not know the tiles that other players have, and that uncertainty must be handled in some way.

Using the usual idea of minimax and allowing possible moves to be discounted by their current probabilities, we arrive at a simple heuristic for approximating the order of some given dominos play using techniques for perfect information games. We call this approximation the Imperfect Minimax Search (IMS) (Algorithm 1). Note that, here $G =$ current game, $p =$ current player, $d =$ depth, $P_G(q, p)$ is a probability distribution over dominoes q and players p given information known by the current player[8]. We define $\text{supp}(p) \equiv \{e \in S \,|\, p(e) > 0\}$ as the support of probability distribution p over some set S.

Algorithm 1 Imperfect Minimax Search

```
1:  procedure IMS(G, p, d)
2:      if G is finished or d = 0 then
3:          return Evaluate(G, p)
4:      end if
5:      s_max ← −∞
6:      for m ∈ supp(P_G(·|p)) s.t. m is valid in G do
7:          q ← P_G(m|p)
8:          G′ ← G updated with m played by p
9:          p′ ← next player after p plays m in G
10:         s_max ← max{s_max, −q · IMS(G′, p′, d − 1)}
11:     end for
12:     return s_max
13: end procedure
```

IMS is motivated by a few notions:

1) Each move's score should be discounted by the probability of being possible—leading to a notion of 'most likely moves'
2) This reduction allows the use of alpha-beta pruning and also allows approximation techniques of complete-information games without the need to resort to expensive sampling algorithms.

We iteratively deepen the search every $n/4$ plays, where $n =$ # of turns played so far in the game, in an exponential fashion (as the number of possible moves exponentially decreased) and leave its form as a set of tunable parameters. The depth's functional form is

$$D = \alpha 2^{\beta \lfloor \frac{n}{4} \rfloor}$$

C. Perfect Information Monte Carlo (PIMC)

As described in [9], PIMC is an approximation technique for imperfect information games which reduces the game into a sampling over many perfect information games. In particular, PIMC samples over possible board states given the current information and solves the corresponding perfect-information game as if all players had complete information about each other's hands. By sampling and then taking the expectation over these samples, we get a lower bound of the expectation value of a given move.

[8]In other words, the game G is defined through the 'player whose eyes we're looking through.'

In general, this technique is quite paranoid: it assumes that every player over whose hands we're sampling actually knows the exact hand of the player who is making the plays. This is often good enough in individual games, but in team-based games, this assumption can fluctuate wildly—this can be intuitively observed by noting that, in the 'complete information' game, your partner also knows what you have, thus is able to make moves that the true partner won't be able to infer. This assumption is therefore broken throughout.

In order to compare PIMC with IMS, we implement the simple sampling scheme presented in Algorithm 2, where we allow N be the set of players for the game (in this case, we take $N = \{1, 2, 3, 4\}$).

Algorithm 2 Sampling Domino Hands

```
procedure SAMPLING(G, N)
    D ← unplayed dominoes in game G
    Q ← empty map N → 2^D
    for d ∈ D do
        p ← sampling from P_G(d, ·)
        Q[p] appends d
    end for
end procedure
```

D. Oracle & Baseline

As our baseline, we implemented the simplest strategy, which is a greedy one. When partnered with another human and playing against two humans, the AI's team won 5 out of 10 rounds. Many of the participants in this short experiment were beginners, so we would expect greedy to have a higher losing rate against more experienced humans. We use this baseline later on as performance comparison, so that we can run many rounds of the game in a short period of time.

Our oracle is a skilled player who knows everyone's pieces during the game, essentially cheating. In this case, as long as the player uses the optimal strategy we described earlier, they can guarantee that the possible loss in each play is minimized. The only way for this strategy to lose is if the player is dealt a very bad hand at the beginning of the game.

We can implement a pseudo-oracle using an all-knowing negamax, where the probabilities of each player having a domino would be set to either zero or one, since we would then have perfect certainty with who has what tiles. A team of negamax "oracle" players wins 91 and loses 9 games out of 100 against greedy players with $\alpha = 5$ and $\beta = 3$ for depth. In losing games, no matter what move negamax players make, the evaluation score is negative.

wins	losses	ties	α	β	
128	68	4	5	1/3	*
66	31	3	4	1/3	
63	37	0	6	1/2	

Fig. 1. IMS's record when playing against a greedy algorithm. The first row is out of 200 games, and the last two out of 100 games. Asterisk indicates a modified evaluation function, as discussed in the Negamax section.

The gap between our oracle and baseline is straightforward: an agent doesn't know what pieces any of the other players have, but should try to gain and use that information to its advantage.

None of our AI players, greedy or advanced, pass when they can make a legal move as the penalty for unnecessary passing is so high.

V. Analysis & Discussion

A. Imperfect Minimax Search

A team of IMS players whose first moves were greedy and had depth parameters $\alpha = 6$ and $\beta = 1/2$ won 40 (70.1%), lost 16 (28.1%), and tied 1 (1.8%) games against a team of greedy players. Without a greedy first move and playing a total of 100 games for each parameter setting, we have the results seen in Figure 1.[9]

The algorithm, for all of its simplicity, performs well against non-trivial opponents. It is somewhat insensitive to the parameters of the depth, leading us to believe that the evaluation function (in our case, the difference of the expectation of the sum of pips between both players) could be further improved. However, neither of the simple evaluation functions we came up with seemed to give a significant boost relative to one another.

B. Perfect Information Monte Carlo (PIMC)

The win percentage achieved by PIMC rivaled that of IMS, though tended to be a few percentage points higher (Figure 2).

To get a sense of how the algorithm behaves with humans, PIMC played a few games with different

[9]Game logs for the games shown can be found at https://github.com/guillean/DominAI/tree/master/results_texts with the prefixes "ims_" for IMS, sample # for PIMC, "oracle" for the negamax oracle, and "smart_smart" for PIMC and IMS playing against each other.

wins	losses	ties	α	β	# of samples
73	26	1	4	1/3	100
68	32	0	4	1/3	50
69	27	4	8	1/2	20
70	29	1	6	1/3	20

Fig. 2. PIMC's record when playing against a greedy algorithm, out of 100 games with different parameters.

pips left	α	β	# of samples
14	8	1/2	20
6	6	1/3	20
6	6	1/3	40
14	6	1/3	40
15	6	1/3	40

Fig. 3. PIMC's games against human players. Each row is a single game.

parameters against a team consisting of a beginner and an advanced player. The AI team lost every game, and approached some turns in unusual ways (Figure 3). For example, in one game, the AI player played the (6, 6) domino near the end of the game, and in earlier chances when it could have played it, it played other pieces containing 6 instead. It is generally wiser to play the (6, 6) early since its pip sum is so high and can only be put down on a 6 end, unlike any other non-double piece. It may be that the PIMC player thought that placing the (6, 6) would give its opponents the opportunity to place their highest pieces containing a 6.

A team of PIMC players won 55, lost 43, and tied 2 games out of 100 against a team of IMS players, with $\alpha = 5$, $\beta = 1/3$, and # of samples $= 50$. This confirms that the two algorithms seem to be on par with one another, with PIMC being slightly better.

VI. FUTURE STEPS

We first noted that both IMS and PIMC are quite insensitive to the α, β parameters of the search. We interpreted this to mean that, though this should not have a great impact especially so early on in the game (in which most pieces are uncertain), it is likely that our evaluation function leaves quite a bit to be desired.[10] In particular, a function incorporating more expert knowledge might do a bit better than the simple maximize (opponent pips - team pips) at every point in time. Additionally, it might be worth investigating a functional estimator for the expected value of a board position; e.g. is it possible to train any functional estimator in order to evaluate the expected value of a board for a player given her dominoes? Probabilistically speaking, this is of course the case if we assume uniform drawing from the possible distribution of dominoes (which we do in the current form of the game), but if we assume that players are playing carefully, then this assumption is wildly wrong, leading us to a weaker probabilistic update than should actually be the case.[11] This implies, then, that the expectation that we are computing for a given move is not as accurate as one would think.

In the same vein, it might also be worth implementing a parameter (as with the previous cases, $\gamma \in [0, 1]$) stating how likely it is for a player to draw from a uniform distribution vs. a mini-max approach and adjusting the parameter accordingly as the game continues. Thus, our Bayesian update would then be weighted by such a factor and would allow a potentially better deduction of what a player's tiles are based on their current set of moves, making the expectation taken over tiles and observed information more accurate.

Here γ serves as a 'playing level' for a given player; the closer to 1, the more minimax-optimal this player's decisions are. Note that assuming that $\gamma = 0$ (i.e. our update is uniform over all tiles) for all opposing players is equivalent to the algorithm presented in this paper.

Additionally, another simple improvement would be to improve the speed of minimax search using Monte-Carlo tree search in order to be able to explore the tree more deeply than is the case here. Since the results do not need to be that accurate, we expect that this might improve the algorithm by a decent amount if implemented correctly, but quick experiments increasing the depth of both IMS and PIMC show little improvement in the overall game playing, so this change may only be apparent at really large depths which are infeasible in the current implementation.

[10] In fact, we noted that our oracle performed marginally *worse* when increasing search depth, which should certainly not be the case. This could be attributed to the previous, or it could also be attributed to the machine over-estimating greedy players' abilities.

[11] We take 'weaker' to mean more uniform over some support. A more mathematical definition could be, say, with higher probabilistic entropy though we use the phrase loosely, here.

REFERENCES

[1] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search," *Proc. Assoc. Adv. Artif. Intell.*, pp. 134–140, 2010.

[2] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," *Journal of Artificial Intelligence Research*, vol. 14, pp. 313–368, 2001.

[3] D. Koller and A. Pfeffer, "Generating and Solving Imperfect Information Games," *Proc. 14th Int. Joint Conf. Artif. Intell.*, vol. 14, pp. 1185–1193, 1995.

[4] N. Burch and M. Bowling, "CFR-D: Solving Imperfect Information Games Using Decomposition," *arXiv preprint arXiv:1303.4441*, pp. 1–15, 2013.

[5] M. H. Smith, "A learning program which plays partnership dominoes," *Communications of the ACM*, vol. 16, no. 8, pp. 462–467, 1973.

[6] A. R. Da Cruz, F. G. Guimaraes, and R. H. C. Takahashi, "Comparing strategies to play a 2-sided dominoes game," *Proceedings - 1st BRICS Countries Congress on Computational Intelligence, BRICS-CCI 2013*, pp. 310–316, 2013.

[7] J. Anderson and J. Varuzza, *International Dominos*. Seven Hills Books, 1991.

[8] G. T. Heineman, G. Pollice, and S. Selkow, *Algorithms in a Nutshell*. O'Reilly Media, Inc, 1 ed., 2009.

[9] T. Furtak and M. Buro, "Recursive Monte Carlo search for imperfect information games," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2013.