Prime Strategies: Game Theoretic Insights into Primi Composti

Marco Giacomin
Department of Information Engineering
University of Padova
marco.giacomin.4@studenti.unipd.it

Federico Meneghetti
Department of Information Engineering
University of Padova
federico.meneghetti.2@studenti.unipd.it

Francesco Roncolato
Department of Information Engineering
University of Padova
francesco.roncolato@studenti.unipd.it

***Abstract*—This project is dedicated to the analysis of the board game "Primi Composti" It was designed and developed by Luciano Murrone in two versions which, despite being similar in concept, differ in scoring rules and mechanics. Each version has been modelled as a two-player game with complete information, to identify optimal strategies through simulation of numerous games. The game dynamics have therefore been studied by adopting different policies, which range from simple heuristic approaches to more complex strategies that require the execution of algorithms such as minimax and alpha-beta pruning. The results offer interesting considerations on the balance and competitiveness of the game, trying to investigate possible future improvements and optimizations to identify increasingly effective and convenient strategies for players.**

***Index Terms*—game theory, alpha–beta pruning, minimax algorithm**

I. Game description

"Primi Composti" [1] is a strategy card game that can be played by 2 to 6 people. The game is presented in two possible versions (denoted as 1 and 2), both requiring the use of a deck of cards with numbers from 2 to 73, of which only the smallest $12 \cdot \mathcal{N}$ will be used, where $\mathcal{N}$ is the number of players. For the sake of simplicity, we will consider only the two-player case, meaning that only cards 2 to 25 will be played.

The game board hosts two (initially empty) stacks of cards for each player, one for cards corresponding to prime numbers and one for composite ones. Only the topmost cards of each stack are considered to be in active play. The game starts after the appropriate number of cards has been shuffled and equally distributed among the players, at which point the first move is made by the player holding the 2 card (from now on $player1$ or $P1$). The two versions of the game are played with similar but non-identical rules, which we will describe separately.

A. Version 1

This version features a scoreboard on which players track the points they will accumulate during the course of the game with some markers. Each card is worth a certain amount of points $VALUE(Card)$ that is either 1 ($COMPOSITE\_SCORE$) or 2 ($PRIME\_SCORE$) depending on whether the card is composite or prime.

Each move consists of a player choosing a card C from their hand. If C is the result of a mathematical operation between exactly two cards A and B among the visible ones (that can be at most four) the player will earn $VALUE(A) + VALUE(B) + VALUE(C)$ points. C must strictly be the result, with A and B being the operands and the valid operations are addition, multiplication, subtraction and division. If C is not the result of a valid operation, then the player only gets $VALUE(C)$ points.

After having computed the points earned (and moved their marker on the scoreboard accordingly), the player places C on the corresponding stack (the one of their primes or of their composites). The game ends when all players have played their entire hand, at which point the one with the most points is declared the winner.

Of course the physical scoreboard with the markers is just a marketing ploy to make the tabletop version more visually competitive but in reality we can represent the exact same thing with two numerical values, one for the score of each player. Another useful and analogous representation of the score can be to compute (and update each time) the difference of the scores between the two players. This is particularly close to what the *minimax* algorithm does (as explained later).

The difference of the scores (the delta) is useful for the purposes of minimax calculation and alpha-beta pruning. The players are not interested in their absolute score per-se, but rather in the difference between their score and the opponent's, as any positive difference at the end of the game will result in their victory. This also allows us to select a single variable to optimize in our tree search.

B. Version 2

This version is played without a scoreboard, as the stacks themselves keep track of the score.

As for version 1, each move consists of a player choosing a card C and searching for a valid operation that has as operands exactly two visible cards A and B and result C but now when they spot this operation they have to remove A and B from the stack they're on and place them on their own side of the board, after which they will place C. Of course if A or B are already on the current player's decks, they just need to place C but if they are not then the player is possibly "stealing" some cards from the opponent's stacks.

The game still ends when all cards are finished, the winner being the player that has the highest sum $VALUE$ of the cards on their stacks.

A remarkable property of the second version is that games played with this ruleset cannot end in a tie, as opposed to the first formulation. The cards consist of 9 prime numbers and 15 composite numbers, adding up to a total of $9 \cdot 2 + 15 = 33$ points, which cannot be evenly divided.

C. Note on the rationality of the players

As a further assumption to simplify the game, we consider an action as just the selection of a card. In reality, the rules of the game introduce more uncertainty, as humans are not perfectly rational players.

Let's express a player's current state (that is, their current hand) as an ordered array of numbers corresponding to their value. For example, if a player's hand consists of the cards 2, 5 and 10 we will say that their current hand is $[2, 5, 10]$. A possible mistake that one player could do without breaking the rules is, in version 1, to consider some valid operands that are not the best ones. For example if the visible cards are $[2, 10, 3, 15]$ and the player chooses $C = 5$ then there are four different valid operations but only one of them $(2 + 3 = 5)$ gives the highest total score (6 points). An human player could not see that operation and thus getting less points (e.g. they only see $15 - 10 = 5$ that leads to 4 points).

In version 2 the same thing could happen with a small but not irrelevant catch: let's assume we are in the same position as the previous example, now a player could choose to consider different operations with different trade-off of the points made on the current hand and the cards revealed by the theft. For example if they choose the operation $15/3 = 5$ then they get the best outcome for the current hand (because they steal both the opponent's topmost cards) but they could uncover something that is useful for the opponent (note that a rational player has perfect recall and thus knows the cards in all the decks). In other words, now it could be a smart choice not to steal some cards right away, sacrificing the short term gain to obtain a better payoff later on.

To simplify all these cases, we only consider that for each version of the game, for each hand played, each player gets the immediate maximum amount of points. Once the card is chosen, we deterministically find the score made for version 1 and the card/cards stolen for version 2. This is to say that for version 1 we do not allow "mistakes" that a human could do (since there is certainly no advantage in getting less points than the maximum possible), and for version 2 we slightly reduce the possible strategic actions, which greatly prunes the total tree and improves the computing time of the algorithms.

Had we allowed the players the option to steal cards other than the maximum number of points allowed each move would've consisted of 1 out of a maximum of 12 chosen cards and between 1 and 4 combinations of cards to steal: steal only opponent's prime, only opponent's composite, both prime and composite or nothing. If we assume the average number of options is 2 (since at all times the 'nothing' option is available and often you have one stealing option), every player will have 12*2 options in the first turn, then 11*2 at the second, and so on. This brings the game tree to $(12! \cdot 2^{12})^2 \approx 3.8 \cdot 10^{24}$ leaves, seven orders of magnitude greater than the version we actually used (that is $12!^2 \approx 2.3 \cdot 10^{17}$).

D. Game typology

As for most real games, every player knows the rules, the payoff system, their cards but in general not the other players' cards. However, in the two player case, knowing your cards you also know your opponent's cards. This is why we can treat the game as a dynamic game of *complete information*.

This means that two rational players could theoretically employ backward induction with a minimax criterion to solve the game finding the Subgame Perfect Equilibrium.

The version 2 of the game can also be seen as a *zero-sum* game since the sum of the scores at the end of a match is constant (33 as we previously said). This means that one match of the game could theoretically be solved via linear programming, although the enumeration of the variables (the pure strategies of the two players) is by itself a challenging task. Each player, to design their strategy, has to choose the first played card out of 12, then choose the next card out of 11 for each combination of cards played in the first round (12^2), then a card out of 10 for the two cards played so far $(12^2 \cdot 11^2)$ and so on. In other words:

- the first action has 12 possibilities
- the second $11 \cdot 12 \cdot 12 = 11 \cdot 12^2$
- the third $10 \cdot 11 \cdot 11 \cdot 12 \cdot 12 = 10 \cdot (11 \cdot 12)^2$
- the fourth $9 \cdot 10 \cdot 10 \cdot 11 \cdot 11 \cdot 12 \cdot 12 = 9 \cdot (10 \cdot 11 \cdot 12)^2$

and so the total number of pure strategies would be $(12) \cdot (11 \cdot 12^2) \cdot (10 \cdot (12 \cdot 11)^2) \cdot \dots = 12! \cdot \prod_{i=1}^{11} (\frac{12!}{i!})^2 \approx 6.3 \cdot 10^{128}$. Without a more efficient way of encoding the strategies, this number of variables cannot conceivably be solved by any existing optimization software with any number of constraints.

II. Game state exploration

Both versions of the (two-player) game have $\binom{24}{12} \approx 2 \cdot 10^6$ possible initial states and each of them can evolve in $(12!)^2 \approx 2.3 \times 10^{17}$ different games (because player 1 can choose the first card among 12 and the same goes for player 2, then the next card is chosen among 11 and so on). This is the number of leaves of the full tree that can be generated given a particular distribution of the cards. The size of this tree makes a full backward induction analysis of the game infeasible with normal hardware. We describe some approaches we took to optimize the computation or obtain an approximate solution.

A. Alpha-beta pruning

This is a technique used to prune search-space trees for multiplayer adversarial games by keeping track of the minimum guaranteed payoff for each player and discarding subtrees that will reward them with less than those values (which are called α and β for the first and second player respectively. The algorithms (whose pseudocode is shown below) is usually implemented in a recursive form, with α and β being initialized as $-\infty$ and $+\infty$ respectively, representing the "worst" possible guaranteed rewards each player can get (or, equivalently, that no guarantee at all can be made at the start of the game). The

two estimates are then updated with each recursive call into the branches, getting closer to each other and increasing the likelihood that a branch can be safely cut. Unfortunately, while this search algorithm still guarantees completeness, it doesn't provide a sufficient performance improvement to allow for an exhaustive search.

For a tree with an average branching factor b and depth d the search algorithm is $O(b^d)$ and the alpha-beta pruning can, in the best case, reduce it to $O(b^{d/2})$. In our case, $d = 24$ and the branching factors are $12, 12, 11, 11, ..., 1, 1$ so on average $b = 6.5$.

Alpha-beta pruning results in a significant reduction in branching but it's still insufficient to allow us to fully explore the tree with our hardware as the number of computational steps would be approximately proportional to $6.5^{24/2} \approx 5.7 \cdot 10^9$ in the best case and to $6.5^{24} \approx 3.2 \cdot 10^{19}$ on average.

alphabeta fail-soft alpha-beta pruning algorithm

Input: node, depth, α, β, maximizingPlayer
Output: value
  if depth==0 or node is terminal **then**
    return value of the node
  end if
  if maximizingPlayer **then**
    value:=$-\infty$
    for all child of node do **do**
      value:= max(value, alphabeta(child, depth-1, α, β, FALSE))
      α:=max(α, value)
      if value $\geq \beta$ **then**
        break
      end if
    end for
    return value
  else
    value:=$+\infty$
    for all child of node do **do**
      value:= min(value, alphabeta(child, depth-1, α, β, TRUE))
      β:=max(β, value)
      if value $\leq \beta$ **then**
        break
      end if
    end for
    return value
  end if=0

In our implementation of the alpha-beta pruning algorithm [2], `node` is an object of the class `Node` and has among its variables the value of that node. For the ease of use our implementation also returns the best node found (that is the one that has the best value).

B. Nash Equilibrium and limited-depth runs

The goal of the research was to find the equilibrium path and thus the Nash Equilibrium of a potential match. However, despite the use of alpha-beta pruning, the execution of backward induction remains infeasible due to the size of the search tree. So, initially, an attempt was made to simulate a sub-optimal but believable player behavior by executing the alpha-beta pruning algorithm down to a limited depth and then starting another exploration from the best terminal node found. If, for example, we pass from the original tree of height 24 (equal to the number of cards played) to a simpler one of height 6 (meaning 3 cards per player) we will reduce the number of nodes from $(12!)^2 \approx 2.3 \cdot 10^{17}$ to $(12 \cdot 11 \cdot 10)^2 = 1742400$ (representing every combination of three initial moves from the players). Additionally, as the players deplete their available cards, a full search of the residual game space becomes more feasible. When only 12 cards are left (6 per player) there are only $(6!)^2 = 518400$ nodes to consider. This means that as the search progresses we can allow the depth to increase, with several possible combinations of increasing depths that total up to 24 (such as $[6, 6, 12], [4, 5, 7, 8], \dots$).

However, this approach of dividing the game tree vertically into several discrete runs is possibly very distant to the real SPE because it is founded in a strong assumption on the players' behavior: the second player should cooperate with the first and play along the initially calculated security strategy instead of re-computing a run of identical depth when their turn comes. In fact, a rational player would be encouraged to individually simulate future moves as deeply into the future as possible, which would make simulating them a much harder task.

```
P1 has [ 2 6 23 9 22 12 13 14 24 15 8 4]
P2 has [11 25 20 17 3 5 21 19 16 10 18 7]
Player   Played card   Gameboard        Delta
1        2             [2 _ _ _]        2
2        3             [2 _ 3 _]        0
1        6             [2 6 3 _]        5
2        5             [2 6 5 _]        -1
1        12            [2 12 5 _]       3
2        7             [2 12 7 _]       -3

1        9             [2 9 7 _]        5
2        18            [2 9 7 18]       1
1        14            [2 14 7 18]      6
2        25            [2 14 7 25]      2
1        23            [23 14 7 25]     7
2        16            [23 14 7 16]     2

1        8             [23 8 7 16]      1
2        10            [23 8 7 10]      0
1        15            [23 15 7 10]     4
2        17            [23 15 17 10]    -1
1        13            [13 15 17 10]    4
2        20            [13 15 17 20]    3
1        22            [13 22 17 20]    4
2        19            [13 22 19 20]    2
1        24            [13 24 19 20]    3
2        11            [13 24 11 20]    -2
1        4             [13 4 11 20]     1
2        21            [13 4 11 21]     0
```

TABLE I
GAME VERSION 1: WRONG APPROXIMATION OF SPE WITH 3 ITERATIONS OF MINIMAX ALGORITHM (RESPECTIVELY OF DEPTHS 6, 6 AND 12)

C. Limiting card number

A modified game of *primi composti* where the players are only dealt 6 cards each would lead to a total game tree with $(6!)^2 \approx 518000$ leaves, which can be computed in its entirety with low computational effort. Therefore, our initial goal of finding the Nash equilibrium of a single game was unfortunately only achieved by modifying the game, which represents a failure. Due to these difficulties we will now concentrate on studying discrete runs, focusing on statistically identifying the best strategies by simulating many games. We believe that certain features or patterns observed in these simulated runs can serve as meaningful indicators of objective properties inherent to the game. However, this is contingent on these observations being consistently validated across a sufficiently large and diverse set of simulations, ensuring that the evidence is robust and reliable.

```
Deck for Player 1: [15 5 2 10 4 11]
Deck for Player 2: [3 6 13 7 12 9]
Player   Played   Gameboard      Delta   Deck P1          Deck P2
1        10       [_ 10 _ _]     1       [15 5 2 4 11]    [3 6 13 7 12 9]
2        3        [_ 10 3 _]     -1      [15 5 2 4 11]    [6 13 7 12 9]
1        5        [5 10 3 _]     1       [15 2 4 11]      [6 13 7 12 9]
2        7        [5 10 7 _]     -4      [15 2 4 11]      [6 13 12 9]
1        11       [11 10 7 _]    -2      [15 2 4]         [6 13 12 9]
2        13       [11 10 13 _]   -4      [15 2 4]         [6 12 9]
1        2        [2 10 13 _]    2       [15 4]           [6 12 9]
2        12       [2 10 13 12]   -2      [15 4]           [6 9]
1        15       [2 15 13 12]   3       [4]              [6 9]
2        6        [2 15 13 6]    -1      [4]              [9]
1        4        [2 4 13 6]     3       []               [9]
2        9        [2 4 13 9]     -1      []               []
```

TABLE II
GAME VERSION 1: EQUILIBRIUM PATH OF ONE GAME WHERE EACH PLAYER HAS 6 CARDS WHERE THE SECOND PLAYER WINS

```
Deck for Player 1: [5 13 10 2 3 4]
Deck for Player 2: [6 9 15 12 7 11]
Player   Played   Gameboard      Delta   Deck P1          Deck P2
1        5        [5 _ _ _]      2       [13 10 2 3 4]    [6 9 15 12 7 11]
2        6        [5 _ _ 6]      1       [13 10 2 3 4]    [9 15 12 7 11]
1        2        [2 _ _ 6]      3       [13 10 3 4]      [9 15 12 7 11]
2        9        [2 _ _ 9]      2       [13 10 3 4]      [15 12 7 11]
1        3        [3 _ _ 9]      4       [13 10 4]        [15 12 7 11]
2        12       [2 _ 3 12]     -1      [13 10 4]        [15 7 11]
1        4        [3 4 _ 9]      6       [13 10]          [15 7 11]
2        7        [2 12 7 4]     -2      [13 10]          [15 11]
1        13       [13 12 7 4]    0       [10]             [15 11]
2        15       [13 12 7 15]   -1      [10]             [11]
1        10       [13 10 7 15]   0       []               [11]
2        11       [13 10 11 15]  -2      []               []
```

TABLE III
GAME VERSION 2: EQUILIBRIUM PATH OF ONE GAME WHERE EACH PLAYER HAS 6 CARDS WHERE THE SECOND PLAYER WINS

III. STATISTICS

Here we will describe our approach to simulating matches of the game, the statistics we derived from them, and our insight into the characteristics of the game stemming from them.

A. Types of agents

We designed three broad categories of player agents for our simulations:

- **Simple policy** players that will either play all their cards in deterministic ascending/descending order or at random, without any strategy, without looking at the visible cards and with no thought about the opponent's cards (`asc`, `desc`, `rand`)
- **Greedy** players that will play the card that results in the immediate best payoff (looking at the visible cards). Given the possibility of a tie in immediate payoff between moves each greedy player also implements a tie-breaker criterion of playing the lowest, highest or a random card among the best-scoring ones (`greedy_asc`, `greedy_desc`, `greedy_rand`)
- **Minimax** players capable of expanding the game tree up to a certain depth and applying the security strategy. At each turn, the player using this strategy generates a tree of limited depth and applies backward induction to solve it (in the implementation the tree is solved with the alpha-beta pruning algorithm). The player then makes the first move that corresponds to the equilibrium path identified by the algorithm. The strategies are differentiated by the depth up to which they will explore as a performance/computational resource tradeoff (`minimax_2`, `minimax_3`, ..., `minimax_6`).

The simulations consist of running several games with every combination of opponents and evaluating the best-performing ones, with the expectation that the greedy and minimax agents will significantly surpass the simple policy ones.

B. Simple policy vs greedy

Here we present the comparative performance of simple players versus greedy ones. For ease of reading only the random and greedy-random versions are shown, but the full data (that can be found in our repository [2]) is generally uniform within the same category, with greedy players consistently beating those who play simple strategies and players within the same category are considered to be on equal footing with one another. There is also a slight but noticeable statistical advantage in being the first player to move.

P1/P2	rand	greedy_rand
rand	51.02%, 5.33%, 43.64%	1.01%, 0.39%, 98.61%
greedy_rand	98.78%, 0.37%, 0.85%	50.9%, 5.38%, 43.72%

TABLE IV
GAME VERSION 1, 200000 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - TIE PERCENTAGES - WIN PERCENTAGES OF PLAYER 2

P1/P2	rand	greedy_rand
rand	27.64, 26.98, 5.87	26.84, 43.20, 16.42
greedy_rand	42.84, 25.95, 16.94	42.53, 41.86, 5.88

TABLE V
GAME VERSION 1, 200000 SIMULATED GAMES: AVERAGE SCORE P1 - AVERAGE SCORE P2 - ABS AVERAGE SCORE DIFFERENCE

P1/P2	rand	greedy_rand
rand	49.39%, 50.61%	1.38%, 98.62%
greedy_rand	97.34%, 02.66%	56.59%, 43.41%

TABLE VI
GAME VERSION 2, 200000 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - WIN PERCENTAGES OF PLAYER 2

We can notice that in both game versions, the average score difference is particularly high in matches where a

P1/P2	rand	greedy_rand
rand	16.38, 16.62, 7.00	07.69, 25.31, 17.71
greedy_rand	23.74, 9.26, 14.67	17.33, 15.67, 8.41

TABLE VII
GAME VERSION 2, 200000 SIMULATED GAMES: AVERAGE SCORE P1 - AVERAGE SCORE P2 - ABS AVERAGE SCORE DIFFERENCE

P1/P2	greedy_rand	minimax_4	minimax_5
greedy_rand	8.43	13.65	14.49
minimax_4	12.93	5.32	6.57
minimax_5	13.25	4.91	5.68

TABLE XI
GAME VERSION 2, 1000 SIMULATED GAMES: ABS AVERAGE SCORE DIFFERENCE

greedy strategy clashes with the simpler one, and this suggests landslide victories of strategic (greedy) players, meaning the strategy not only leads with high probability to a victory but also gives on average significantly higher scores.

C. Greedy vs minimax

As previously demonstrated, the computation required for minimax agents is considerably more resource-intensive than that of the two earlier categories, so the number of simulations run is significantly smaller and could impact on the quality of the statistics. Indeed, for a single cell in the following tables involving a player using a minimax strategy, the number of operations required is approximately proportional to the number of nodes in the tree that is generated times the number of steps (that is the 24 moves, 12 per player). The size of the tree depends on the number of remaining cards and exponentially on the depth d. On the other hand, in the cases of simple and greedy policies, each choice is almost instantaneous since it relies only on the sorting of a small array (that can also be computed once for all at the beginning of the match) and on a constant and small number of operations to evaluate the best move and best score.

P1/P2	greedy_rand	minimax_4	minimax_5
greedy_rand	50.9%, 5.6%	8.4%, 2.5%	07.4%, 2.1%
minimax_4	83.50%, 2.4%	30.6%, 5.8%	21.0%, 4.8%
minimax_5	85.50%, 3.6%	44.40%, 6.1%	34.50%, 7.3%

TABLE VIII
GAME VERSION 1, 1000 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - TIE PERCENTAGES

P1/P2	greedy_rand	minimax_4	minimax_5
greedy_rand	5.71	9.20	9.40
minimax_4	8.38	5.13	5.81
minimax_5	8.70	4.63	4.27

TABLE IX
GAME VERSION 1, 1000 SIMULATED GAMES: ABS AVERAGE SCORE DIFFERENCE

P1/P2	greedy_rand	minimax_4	minimax_5
greedy_rand	58.1%, 41.90%	3.0%, 97.00%	2.0%, 98.00%
minimax_4	94.0%, 6.00%	29.2%, 70.80%	17.5%, 82.50%
minimax_5	95.0%, 5.00%	39.6%, 60.40%	27.7%, 72.30%

TABLE X
GAME VERSION 2, 1000 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - WIN PERCENTAGES OF PLAYER 2

As predicted, the greedy algorithm fares much worse when compared to agents capable of some foresight. Furthermore, as expected, table III-C clearly shows that minimax_4 is also less effective than minimax_5. However, unlike what happens in the simpler strategies used previously, it is the second player who has a statistical advantage, which helps minimax_4, when playing second, to mitigate the strategic gap and win in most cases.

Another difference with the previous cases is that even though the minimax strategies win in most games, the score gap is smaller, making the game between the minimax players and the greedy_rand player more balanced than the one between greedy_rand and rand. This means if the games are played with sufficiently good strategies (like greedy), they remain interesting and engaging despite a little gap in effectiveness. For a card game, this is a great feature because the better player has the advantage, but even the less skilled player can still enjoy the game.

The results show that the first and second player has an advantage when the games are played with greedy vs greedy and minimax vs minimax strategies, respectively. However, this feature is not necessarily a problem; rather, it can add more dynamism to the game. Indeed, the decision of who goes first is determined randomly, so if five games were played, the less skilled player would, with a bit of luck, have a better chance of winning a few matches. Meanwhile, the more experienced player would still be able to leverage their skills to win the overall set. In the case where both players are equally skilled and play the same strategy, for example, minimax_4 vs minimax_4, the fact that one of them has a slight advantage can always serve as a reason to play more matches.

D. Minimax vs minimax

For what concerns the comparison between minimax_5 and minimax_6, all previous considerations are still valid. However we can see that the second player advantage is much greater in the case of minimax_6 vs minimax_6. This could be explained by the fact that the accuracy of the moves is so high that a small initial advantage is enough to win the majority of the games, even if with a small score gap.

P1/P2	minimax_5	minimax_6
minimax_5	35.00%, 5.56%	22.78%, 4.44%
minimax_6	43.33%, 8.89%	28.89%, 8.89%

TABLE XII
RIFARE: GAME VERSION 1, 180 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - TIE PERCENTAGES

IV. HUMAN VS CPU

We implemented a program that allows users to play both versions of the game against the computer, allowing them to choose which strategy to face (respectively

P1/P2	minimax_5	minimax_6
minimax_5	38.40, 40.04, 4.35	35.79, 38.82, 4.59
minimax_6	37.49, 37.92, 4.17	33.76, 35.87, 4.41

TABLE XIII
GAME VERSION 1, 180 SIMULATED GAMES: AVERAGE SCORE P1 - AVERAGE SCORE P2 - ABS AVERAGE SCORE DIFFERENCE

P1/P2	minimax_5	minimax_6
minimax_5	41.25%, 58.75%	16.25%, 83.75%
minimax_6	40.0%, 60.00%	25.00%, 75.00%

TABLE XIV
GAME VERSION 2, 80 SIMULATED GAMES: WIN PERCENTAGES OF PLAYER 1 - WIN PERCENTAGES OF PLAYER 2

P1/P2	minimax_5	minimax_6
minimax_5	15.08, 17.92, 5.88	13.42, 19.57, 7.07
minimax_6	15.60, 17.40, 4.30	14.59, 18.41, 4.97

TABLE XV
GAME VERSION 2, 80 SIMULATED GAMES: AVERAGE SCORE P1 - AVERAGE SCORE P2 - ABS AVERAGE SCORE DIFFERENCE

`human_vs_cpu_1.py` and `human_vs_cpu_2.py` in our repository [2]). Specifically, the policies can be classified as follows according to the game difficulty they constitute:

- very easy mode: simple policies
- easy mode: greedy policies
- normal mode: `minimax_4`
- hard mode: `minimax_5`
- impossible mode: `minimax_6`

These difficulty levels were chosen somewhat arbitrarily to distinguish between the agent classes, as the supposedly "easy" mode where the opponent employs a greedy policy poses a non-indifferent challenge. In fact, such a strategy is a good approximation for how a human player could approach this game, especially without any particular prior. As opposed to most established strategy games like chess, "Primi Composti" presents little opportunity for spatial reasoning (something humans are evolutionarily predisposed towards), rewarding instead a capacity for quick mathematical thinking to evaluate possible future moves. Therefore, a human player might find themselves unable to predict a significant number of possible moves and settle for a simple policy of picking the immediate best move.

V. POSSIBLE FUTURE WORK

A possible approach to designing a better-performing agent for this game could be **reinforcement learning**. Since the intermediate performance of the agent is measurable during the game by way of counting the cards on its stacks and the environment is fully observable, we can design an agent capable of learning from experience and obtaining feedback on the quality of its move. We predict that version 1 of this game is better suited for allowing this task, as the intermediate reward proceeds in a monotonic pattern during the game progression, preventing situations where the agent makes a move that sets its progress back after some moves in the future in a way that is causally difficult to ascertain and therefore hard to learn from.

A. Additional behavioral strategies

Though we only considered strategies we considered to be *canonical* (that is, ones that every reasonable person approaching this problem would conceive of) there are many other possible designs for strategies that could achieve strong results against computationally more expensive players. Concisely-expressed behavioral strategies where a move is defined as the result of a simple function of the previous game states could be constructed using search techniques to iteratively search for better-performing ones. A sufficient simplification of the behavior, and therefore of the number of pure strategies, in the space we considered for the linear programming solution could bring the number of variables down enough to be solved by a commercially available linear programming software.

B. Hyper-parameter exploration

As previously suggested, the definition of the game depends on a few implicit hyper-parameters, namely the number of cards dealt to each player, the reward granted by each card type and possibly the mathematical operations that are allowed. Our simulation software can be adapted to play many variants of the game depending on the provided settings, some of which completely explorable in feasible time. A grid exploration of the settings combinations, starting from the ones that make the game easiest to compute, could reveal balancing patterns; we expect the win percentage of similar player agents facing off against each other to be a convex function in the domain of the hyper-parameters, although there could be some irregularities stemming from some emergent property of the game.

VI. CONCLUSIONS

For what we've managed to analyze of it, "Primi Composti" appears to be a relatively well-balanced game that rewards strategic play. In both versions the results clearly show a clear advantage of players that play with some strategic depth compared to those that play randomly or even greedily, while the results are less clear about how advantageous it is for a player to increase their depth of game tree analysis after a certain point. Both versions seem to have a slight-to-moderate imbalance towards the second player if they both employ a similar-depth minimax strategy, with the second version presenting the highest difference in victories in our simulations. From a game design standpoint, version 2's property of disallowing draws arguably makes it a better designed game, though the aforementioned balancing issue could require some additional design iterations.

REFERENCES

[1] L. Murrone, "Primi composti. una gara strategica e aritmetica," 2024, © 2024 L. Murrone.

[2] "gtproject," https://github.com/Roncooo/gtproject/, gitHub Repository. [Online]. Available: https://github.com/Roncooo/gtproject/