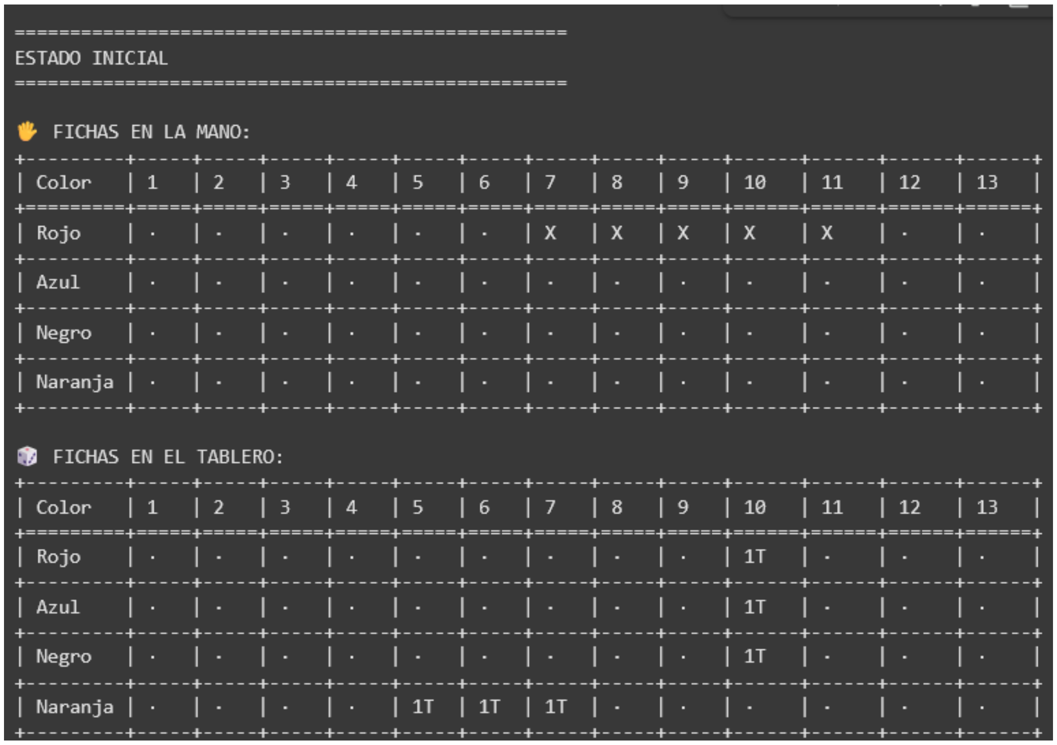
PIA: Week 3 Deliverable

Two Woman four men

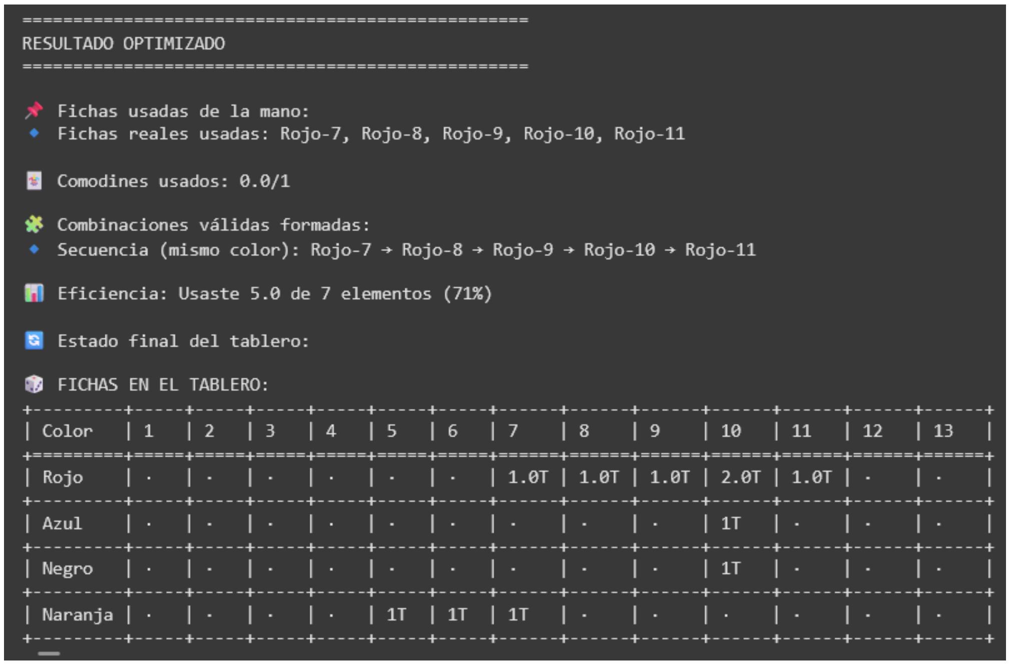
May 2025

1. Model testings

The model gave some results that are favorable to are objective. First, we got all the combinations needed to start considering what move we should do with whichever tiles we had. Second, the considerations of the moves weren't completed at all. But, with all the considerations of the combinations, we expect to get the results we want.



```
==================================================
ESTADO INICIAL
==================================================

🖐 FICHAS EN LA MANO:
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Color   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10   | 11   | 12   | 13   |
+=========+=====+=====+=====+=====+=====+=====+=====+=====+=====+======+======+======+======+
| Rojo    | ·   | ·   | ·   | ·   | ·   | ·   | X   | X   | X   | X    | X    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Azul    | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·    | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Negro   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·    | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Naranja | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·    | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+

🎲 FICHAS EN EL TABLERO:
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Color   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10   | 11   | 12   | 13   |
+=========+=====+=====+=====+=====+=====+=====+=====+=====+=====+======+======+======+======+
| Rojo    | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | 1T   | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Azul    | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | 1T   | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Negro   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | ·   | 1T   | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
| Naranja | ·   | ·   | ·   | ·   | 1T  | 1T  | 1T  | ·   | ·   | ·    | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+-----+-----+-----+------+------+------+------+
```



```
==================================================
RESULTADO OPTIMIZADO
==================================================

📌 Fichas usadas de la mano:
🔹 Fichas reales usadas: Rojo-7, Rojo-8, Rojo-9, Rojo-10, Rojo-11

🃏 Comodines usados: 0.0/1

🧩 Combinaciones válidas formadas:
🔹 Secuencia (mismo color): Rojo-7 → Rojo-8 → Rojo-9 → Rojo-10 → Rojo-11

📊 Eficiencia: Usaste 5.0 de 7 elementos (71%)

🔄 Estado final del tablero:

🎲 FICHAS EN EL TABLERO:
+---------+-----+-----+-----+-----+-----+-----+------+------+------+------+------+------+------+
| Color   | 1   | 2   | 3   | 4   | 5   | 6   | 7    | 8    | 9    | 10   | 11   | 12   | 13   |
+=========+=====+=====+=====+=====+=====+=====+======+======+======+======+======+======+======+
| Rojo    | ·   | ·   | ·   | ·   | ·   | ·   | 1.0T | 1.0T | 1.0T | 2.0T | 1.0T | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+------+------+------+------+------+------+------+
| Azul    | ·   | ·   | ·   | ·   | ·   | ·   | ·    | ·    | ·    | 1T   | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+------+------+------+------+------+------+------+
| Negro   | ·   | ·   | ·   | ·   | ·   | ·   | ·    | ·    | ·    | 1T   | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+------+------+------+------+------+------+------+
| Naranja | ·   | ·   | ·   | ·   | 1T  | 1T  | 1T   | ·    | ·    | ·    | ·    | ·    | ·    |
+---------+-----+-----+-----+-----+-----+-----+------+------+------+------+------+------+------+
```

2. Adjustment and Optimization

Initially, the model was considered to only use runs and groups of 3. Now, the code needed to obtain all existing combinations from the entire rummikub game has been implemented for consideration in the model.

Among other things, the model is expected to analyze all possible combinations considering all available tiles, whether on the table or in the hand, including the joker. Currently, the model only considers throwing the largest number of tiles, regardless of whether there are any extra combinations.

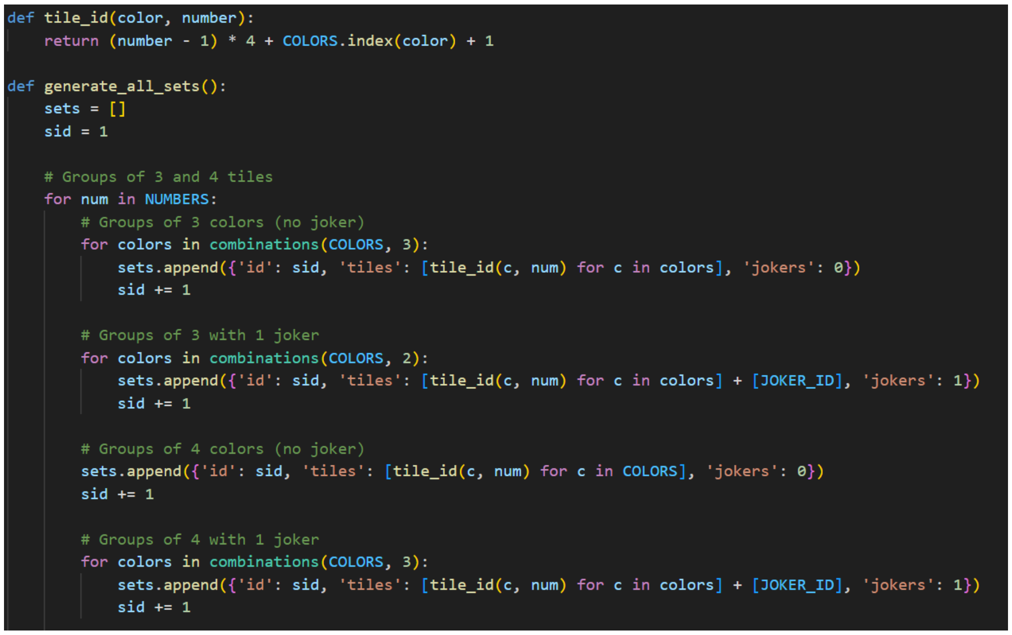
3. Analysis of the Initial Results

The model correctly analyzes each combination that can be held at any given time in the hand. However, the game of rummikub is based on the table layout, so the model is currently incomplete. These restrictions need to be implemented to achieve the model's objective.

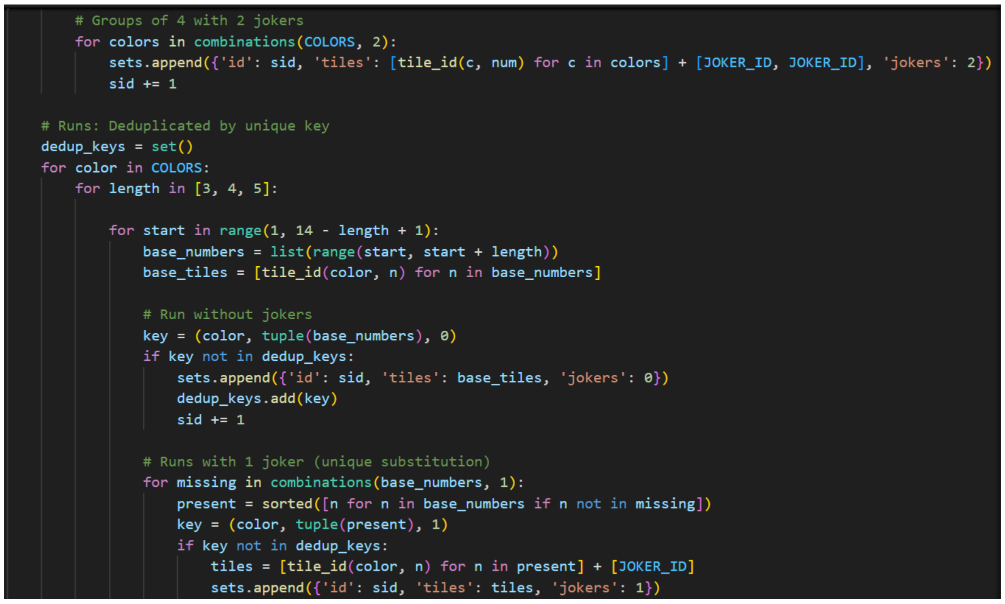
For now, the model considers that it is best to roll by considering the tossing of the tiles to minimize the number of tiles in the hand.

4. Technical documentation

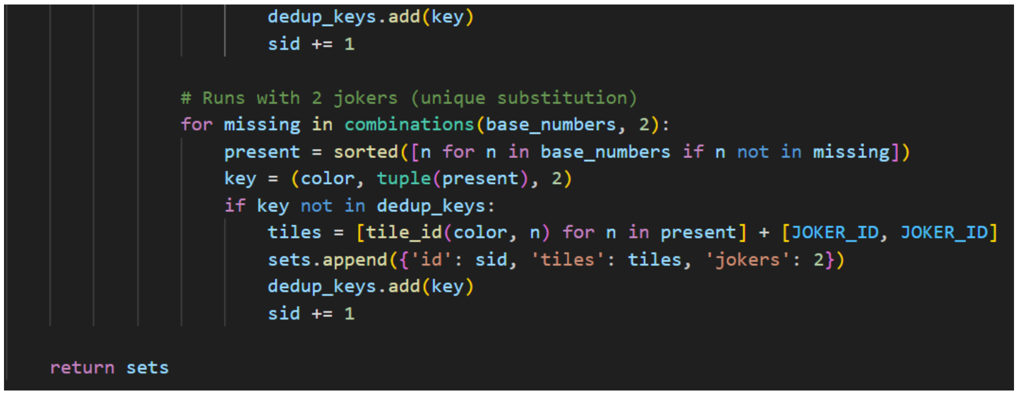
The code now has all the combinations that could be made in rummikub, being approximately 1300, this includes the combinations with groups and runs. The code counts all runs that have 3, 4 or 5 tiles in it, this is because every run after 5 tiles could be changed into two runs of 3 files each. Counting on this, the combinations are reduced to the best outcome possible.



```
def tile_id(color, number):
    return (number - 1) * 4 + COLORS.index(color) + 1

def generate_all_sets():
    sets = []
    sid = 1

    # Groups of 3 and 4 tiles
    for num in NUMBERS:
        # Groups of 3 colors (no joker)
        for colors in combinations(COLORS, 3):
            sets.append({'id': sid, 'tiles': [tile_id(c, num) for c in colors], 'jokers': 0})
            sid += 1

        # Groups of 3 with 1 joker
        for colors in combinations(COLORS, 2):
            sets.append({'id': sid, 'tiles': [tile_id(c, num) for c in colors] + [JOKER_ID], 'jokers': 1})
            sid += 1

        # Groups of 4 colors (no joker)
        sets.append({'id': sid, 'tiles': [tile_id(c, num) for c in COLORS], 'jokers': 0})
        sid += 1

        # Groups of 4 with 1 joker
        for colors in combinations(COLORS, 3):
            sets.append({'id': sid, 'tiles': [tile_id(c, num) for c in colors] + [JOKER_ID], 'jokers': 1})
            sid += 1
```



```python
        # Groups of 4 with 2 jokers
        for colors in combinations(COLORS, 2):
            sets.append({'id': sid, 'tiles': [tile_id(c, num) for c in colors] + [JOKER_ID, JOKER_ID], 'jokers': 2})
            sid += 1

    # Runs: Deduplicated by unique key
    dedup_keys = set()
    for color in COLORS:
        for length in [3, 4, 5]:

            for start in range(1, 14 - length + 1):
                base_numbers = list(range(start, start + length))
                base_tiles = [tile_id(color, n) for n in base_numbers]

                # Run without jokers
                key = (color, tuple(base_numbers), 0)
                if key not in dedup_keys:
                    sets.append({'id': sid, 'tiles': base_tiles, 'jokers': 0})
                    dedup_keys.add(key)
                    sid += 1

                # Runs with 1 joker (unique substitution)
                for missing in combinations(base_numbers, 1):
                    present = sorted([n for n in base_numbers if n not in missing])
                    key = (color, tuple(present), 1)
                    if key not in dedup_keys:
                        tiles = [tile_id(color, n) for n in present] + [JOKER_ID]
                        sets.append({'id': sid, 'tiles': tiles, 'jokers': 1})
```



```python
                        dedup_keys.add(key)
                        sid += 1

                # Runs with 2 jokers (unique substitution)
                for missing in combinations(base_numbers, 2):
                    present = sorted([n for n in base_numbers if n not in missing])
                    key = (color, tuple(present), 2)
                    if key not in dedup_keys:
                        tiles = [tile_id(color, n) for n in present] + [JOKER_ID, JOKER_ID]
                        sets.append({'id': sid, 'tiles': tiles, 'jokers': 2})
                        dedup_keys.add(key)
                        sid += 1

    return sets
```

As a team, we decided to pivot from our initial programming approach due to its high computational complexity, which made it impractical for solving larger instances efficiently. After reassessing the problem space, we recognized that by applying constraints and intelligently reducing the search space, we could limit the total number of cases to just 1,174. This discovery made it feasible to revisit integer programming and reexplore the Branch and Cut method. With a significantly smaller and more tractable search space, we adjusted our model accordingly to leverage the strengths of Branch and Cut for efficient solution finding.

The model defines the following sets, parameters, and variables:

Indices:

- $i \in I = \{1, \ldots, 53\}$: index for tile types (including the joker).
- $j \in J = \{1, \ldots, 1174\}$: index for all possible valid sets (runs and groups, with or without jokers).

Parameters:

- s_{ij}: equals 1 if tile i is part of set j; 0 otherwise.
- t_i: number of times tile i is currently on the table.
- r_i: number of times tile i is available in the player's rack.
- v_i: value of tile i (equal to its number; joker can have custom value).
- w_j: number of times set j is currently on the table (for change minimization).
- M: large constant used to scale the influence of secondary objectives (e.g., 40).

Variables:

- $x_j \in \{0, 1, 2\}$: number of times set j is formed in the new solution.
- $y_i \in \{0, 1, 2\}$: number of times tile i is played from the player's rack.
- $z_j \in \{0, 1, 2\}$: number of times set j appears both in the original table and in the new solution.

4.3 Objective Function

Option 1: Maximize number of tiles placed

$$\max \sum_{i \in I} y_i$$

Option 2: Maximize total value of tiles placed

$$\max \sum_{i \in I} v_i \cdot y_i$$

Option 3: Maximize value and minimize changes (model used)

$$\max \left(\sum_{i \in I} v_i \cdot y_i + \frac{1}{M} \sum_{j \in J} z_j \right)$$

The second term rewards preserving existing sets from the original configuration, weighted to be less important than the main objective.

4.4 Constraints

Tile Conservation Constraint:

$$\sum_{j \in J} s_{ij} x_j = t_i + y_i \quad \forall i \in I$$

Each tile used in the final solution must either:

- Already exist on the table, or
- Be added from the player's rack.

Rack Availability Constraint:

$$y_i \leq r_i \quad \forall i \in I$$

A tile cannot be played more times than it is available in the player's rack.

Set Preservation Constraints:

$$z_j \leq x_j \quad \forall j \in J$$

$$z_j \leq w_j \quad \forall j \in J$$

These ensure that $z_j = \min(x_j, w_j)$ — a set is only considered preserved if it appears both in the original and new table states.

Joker Usage Constraint:

$$\sum_{j \in J} s_{\text{joker},j} \cdot x_j \leq r_{\text{joker}}$$

This limits the number of jokers used in selected sets to the number available on the player's rack.

Variable Domains:

$$x_j \in \{0, 1, 2\}, \quad y_i \in \{0, 1, 2\}, \quad z_j \in \{0, 1, 2\}$$

4.5 Full Constraint Summary

$$\begin{aligned}
\sum_{j\in J} s_{ij}x_j &= t_i + y_i && \forall i \in I \\
y_i &\le r_i && \forall i \in I \\
z_j &\le x_j && \forall j \in J \\
z_j &\le w_j && \forall j \in J \\
\sum_{j\in J} s_{\text{joker},j} \cdot x_j &\le r_{\text{joker}} && \text{(joker limit)} \\
x_j, y_i, z_j &\in \{0,1,2\} && \forall j \in J, i \in I
\end{aligned}$$