Universidad Distrital Francisco José de Caldas
Department of Systems Engineering
Faculty of Engineering
Systems Analysis and Design

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Workshop No. 3 — Kaggle Systems Simulation

Laura Daniela Muñoz Ipus - Code: 20221020022
Luisa Fernanda Guerrero Ordoñez - Code: 20212020099
Esteban Alexander Bautista Solano – Code: 20221020089

Introduction

Following the analysis conducted in Workshop 1 on the "20 Questions" Kaggle competition, and the design of a modular system architecture in Workshop 2, the next step is to implement that proposal through a practical simulation. In the first stage, the key elements of the system were identified, along with sensitive factors that could affect its performance. Then, a structure focused on the interaction between bots was proposed, incorporating basic systems engineering principles.

This phase aims to test the proposed design using real data, simulating matches between bots, updating their skills, and observing the general behavior of the system. The simulation allows us to validate whether the architecture responds appropriately under different conditions, whether matchings are fair, and whether the results are properly recorded. Additionally, it offers the opportunity to identify errors, unexpected behaviors, or aspects that were not evident during the theoretical design.

Objective

The objectives of this work are to simulate matches of the "20 Questions" game using real data from the Kaggle competition, evaluate the behavior of the system designed in Workshop 2 through interactions between bots, validate the data flow between the different components of the system and their response to various conditions, and identify possible errors, sensitive variations, or unexpected behaviors that may arise during the simulation.

1. Data Preparation

In the first workshop, we analyzed the Kaggle competition called *"20 Questions"*, where two teams of bots (one asking and one answering) try to guess a secret word by using yes/no questions. The goal is to find the word before the other team or at the same time, which results in a tie.

To begin the simulation of the "20 Questions" competition process, two datasets available in the original Kaggle competition were selected. This ensures that the simulation is built on real data. The link to the dataset can be found in the bibliography.

To enable better data processing during the simulation, each dataset was cleaned according to best practices. The two datasets used are:

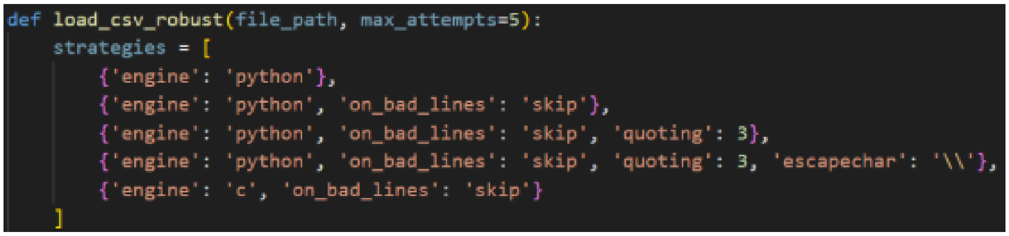
- **Games_data.csv**: Contains historical records of matches from the competition.

- **Keywords.csv**: A database of keywords along with their associated categories.

Data cleaning was necessary because the original Kaggle datasets contain inconsistent formats in questions and answers, missing or incomplete records, special characters, and non-standard text formats. In addition, outliers could distort the simulation results, and textual representations needed to be converted into numerical values for proper analysis.

The cleaning process included the following:

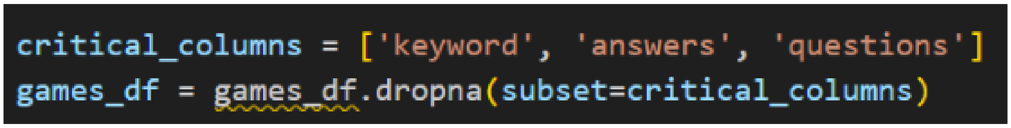
- **Handling corrupted or inconsistent CSV files**: This step was necessary because the original files might contain formatting errors, corrupted lines, unbalanced quotation marks, or special characters that prevent correct parsing and automated processing. By applying progressive loading strategies, the system was able to recover as much usable data as possible, preventing isolated issues from blocking the analysis of the entire dataset.



```
def load_csv_robust(file_path, max_attempts=5):
    strategies = [
        {'engine': 'python'},
        {'engine': 'python', 'on_bad_lines': 'skip'},
        {'engine': 'python', 'on_bad_lines': 'skip', 'quoting': 3},
        {'engine': 'python', 'on_bad_lines': 'skip', 'quoting': 3, 'escapechar': '\\'},
        {'engine': 'c', 'on_bad_lines': 'skip'}
    ]
```

Figura 1: Robust loading strategies for CSV files with formatting errors.

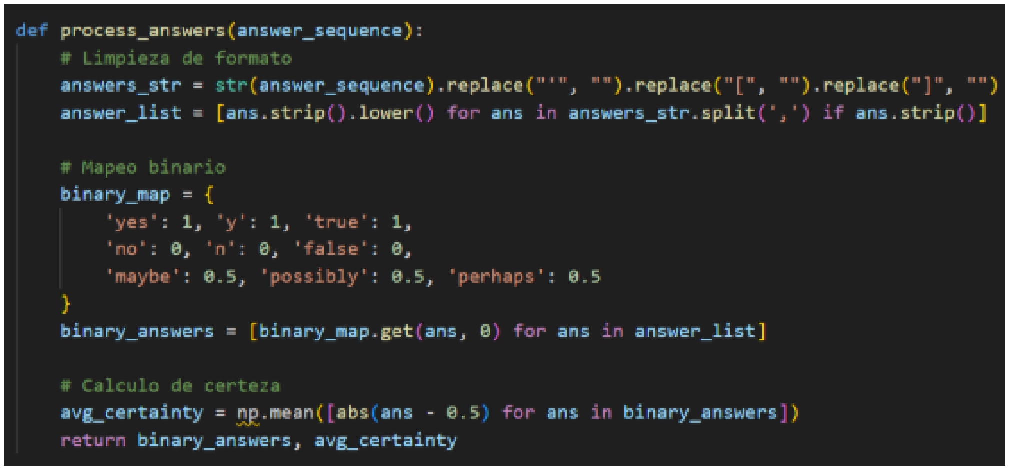
- **Removal of rows with missing essential data**: Rows missing critical information required to simulate a valid game were removed. This step reduces noise and prevents incomplete or inconsistent matches that could affect the simulation results.



```
critical_columns = ['keyword', 'answers', 'questions']
games_df = games_df.dropna(subset=critical_columns)
```

Figura 2: Removal of rows with missing essential data.

- **Text normalization of answers**: The responses “Yes” and “No” were very frequent, so they were converted into binary representation — 1 and 0 respectively. Ambiguous answers were represented with a value of 0.5. Additionally, a certainty score was calculated to quantify how definitive each response was, allowing for better analysis during the simulation.



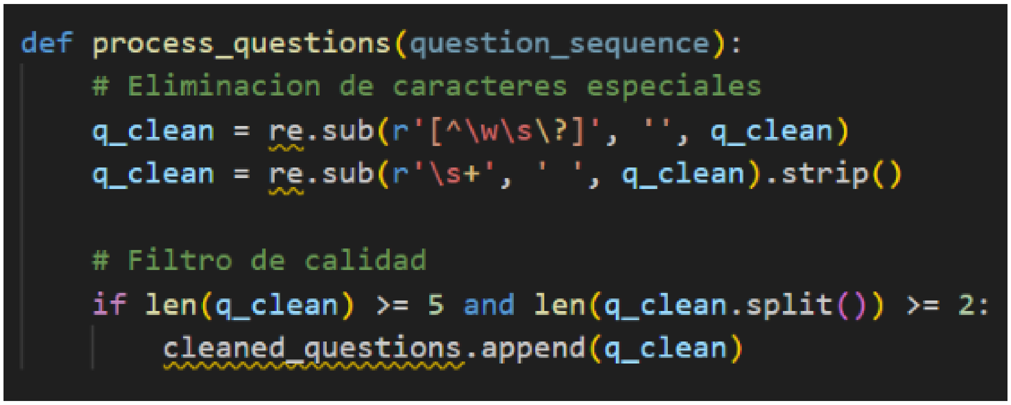
```
def process_answers(answer_sequence):
    # Limpieza de formato
    answers_str = str(answer_sequence).replace("'", "").replace("[", "").replace("]", "")
    answer_list = [ans.strip().lower() for ans in answers_str.split(',') if ans.strip()]

    # Mapeo binario
    binary_map = {
        'yes': 1, 'y': 1, 'true': 1,
        'no': 0, 'n': 0, 'false': 0,
        'maybe': 0.5, 'possibly': 0.5, 'perhaps': 0.5
    }
    binary_answers = [binary_map.get(ans, 0) for ans in answer_list]

    # Calculo de certeza
    avg_certainty = np.mean([abs(ans - 0.5) for ans in binary_answers])
    return binary_answers, avg_certainty
```

Figura 3: Normalization of answers.

- **Quality filtering of questions**: The goal of this step was to transform each question into a uniform and high-quality version for later analysis. A quality filter was applied to ensure that the questions used in the simulation were clean, comparable, and suitable for automated analysis techniques such as tokenization or semantic feature extraction.

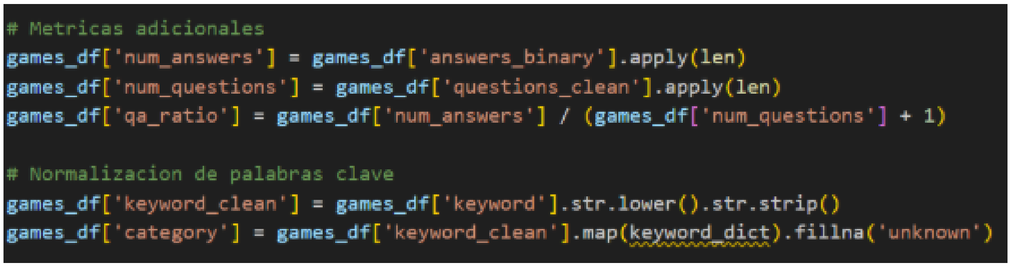


```
def process_questions(question_sequence):
    # Eliminacion de caracteres especiales
    q_clean = re.sub(r'[^\w\s\?]', '', q_clean)
    q_clean = re.sub(r'\s+', ' ', q_clean).strip()

    # Filtro de calidad
    if len(q_clean) >= 5 and len(q_clean.split()) >= 2:
        cleaned_questions.append(q_clean)
```

Figura 4: Quality filtering of questions.

- **Keyword normalization and categorization**: The `keyword_clean` field was normalized to ensure consistent formatting by removing differences due to capitalization or spacing. Each keyword was then assigned a category using a dictionary-based mapping (`keyword_dict`). This process is essential for grouping and analyzing matches based on word type or thematic category. Tokenization was also applied to clean and segment both questions and answers, allowing each text element to be treated as a semantic unit ready for further analysis, such as embedding extraction or processing by language models.



```python
# Metricas adicionales
games_df['num_answers'] = games_df['answers_binary'].apply(len)
games_df['num_questions'] = games_df['questions_clean'].apply(len)
games_df['qa_ratio'] = games_df['num_answers'] / (games_df['num_questions'] + 1)

# Normalizacion de palabras clave
games_df['keyword_clean'] = games_df['keyword'].str.lower().str.strip()
games_df['category'] = games_df['keyword_clean'].map(keyword_dict).fillna('unknown')
```

Figura 5: Keyword normalization and tokenization.

2. Simulation Planning

Three distinct simulation scenarios have been developed to exercise and validate different aspects of the “20 Questions” system. These scenarios are designed to explore system behavior under a variety of operational conditions, ranging from stable environments to high-uncertainty situations, providing a comprehensive evaluation of the system’s robustness, adaptability, and sensitivity.

Each scenario represents a specific set of operational conditions that reflect different potential states of the system in production. The selection of these three scenarios allows for a systematic exploration of possible behavior spaces, facilitating the identification of emergent patterns, design limitations, and optimization opportunities.

2.1 Balanced Scenario

The balanced scenario represents the system’s reference configuration, designed to simulate standard operating conditions that reflect expected behavior in a typical competition environment. In this scenario, QuestionerBots implement an adaptive strategy that follows a structured logical progression: they begin with general questions to establish broad categories, continue with intermediate-level questions to narrow the search space, and conclude with specific questions aimed at identifying the target word. This strategy mirrors natural deductive reasoning and allows for evaluation of a systematic problem-solving approach.

AnswererBots in this scenario operate with a consistency level of 80 %, meaning their responses are mostly accurate and reliable but include a controlled degree of ambiguity to simulate natural uncertainty in real communication systems.

The main purpose of this scenario is to establish a performance baseline that serves as a reference point for subsequent comparisons.

2.2 Chaotic Scenario

The chaotic scenario intentionally introduces high levels of randomness and ambiguity into the system, simulating adverse conditions that may arise from poorly optimized bots, communication interference, or operational uncertainty. QuestionerBots in this scenario follow a completely random strategy, abandoning the logical structure of deduction and generating questions without a coherent pattern.

AnswererBots operate with a reduced consistency level of 50 %, introducing frequent incorrect or ambiguous responses that mimic poor communication conditions or faulty information processing.

This configuration creates a high-entropy informational environment, allowing for exploration of the system’s tolerance thresholds and observation of how feedback mechanisms respond to low-quality input.

2.3 Skilled Scenario

The skilled scenario represents the upper end of the performance spectrum, simulating conditions in which highly optimized bots with advanced strategies interact in an elite competitive setting. QuestionerBots retain the adaptive strategy but begin with significantly elevated skill levels, with random increments ranging from 0 to 200 points above the base level. This simulates agents that have achieved high specialization through extensive training or optimized algorithms.

AnswererBots operate with a consistency level of 90 %, producing near-perfect responses that represent high-precision systems in which ambiguity and error have been minimized through advanced technical refinement.

2.4 Integration with System Architecture

Each component of the proposed architecture is activated during the simulation. The following table outlines the role and implementation of each module:

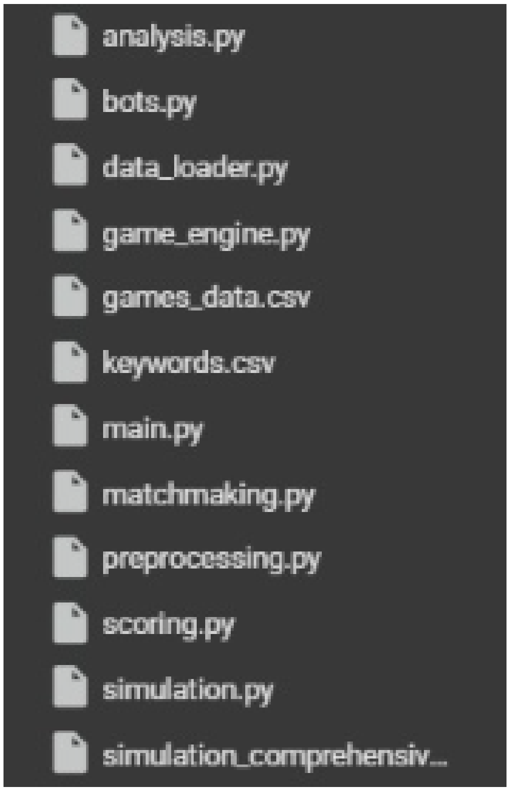
System Component	Role in the Simulation	Application in the Simulation
User Agent (bot)	Generate questions or guesses in each round to identify the secret word.	Different bots with various strategies are simulated to observe how their performance varies under different conditions.
Game Engine	Control game progression, turns, validations, and enforcement of rules and time constraints.	Simulates matches, manages rounds, validates inputs, logs actions, and ensures time and length limits are respected.
Scoring System	Evaluate bot performance and update its skill (μ) and uncertainty (σ) based on results.	Calculates and adjusts each bot's metrics after every match, simulating learning or regression based on performance.
Matchmaker	Match bots with similar skill levels to ensure fairness.	Determines which bots compete based on updated μ and σ values, enabling realistic and adaptive matchups.
Leaderboard System	Record and display each bot's position in the system's ranking.	Simulates a ranking system updated after each match, showing the progression and performance of the bots.
Feedback Loop	Continuous update mechanism that influences future matchups.	Allows performance in one match to affect future pairings, generating adaptive or even chaotic dynamics.

Cuadro 1: Integration of architecture components within the simulation.

3. Simulation Implementation

The simulation was implemented using Python, executed in Jupyter Notebooks through Google Colab. The code was developed in a modular fashion using stubs to represent the main system components.

The modular division and explanation of the main stubs are presented below:



analysis.py
bots.py
data_loader.py
game_engine.py
games_data.csv
keywords.csv
main.py
matchmaking.py
preprocessing.py
scoring.py
simulation.py
simulation_comprehensiv...

Figura 6: Data$_l$*oader.*

The `data_loader.py` module is responsible for robust dataset loading, using multiple strategies to handle formatting errors. It aligns with Workshop 2 by implementing the ”Data Source”module from the architectural design, ensuring data integrity for the simulation and managing variability in file formats—a potential source of chaos.

The `game_engine.py` module serves as the core layer, implementing the ”Game Logic Controller,”which manages flows and rules according to the analysis conducted in the previous workshops. The `scoring.py` module corresponds to the ”Skill Evaluation Systemïn the design, while `matchmaking.py` implements the ”Matchmaking System”to ensure competitive fairness and control potential imbalances.

The `analysis.py` stub generates metrics and visualizations to detect chaotic patterns and implements the ”Performance Analyzer”from the design. The `simulation.py` script acts as the ”Simulation Controller,.^{a}ctivating all components and exploring chaotic behavior by testing scenarios with different levels of ambiguity and strategy. Finally, `main.py` coordinates the entire simulation pipeline.

4. Implementation Analysis

After completing the dataset cleaning process, the system implementation was carried out. Although the original “20 Questions” Kaggle competition is designed for teams of two bots (one questioner and one answerer per team, competing in a 2 vs 2 format), this Workshop 3 simulation opted for a simplified approach by modeling only the interaction between one questioner and one answerer (1 vs 1). Despite this adjustment, the original conditions and rules of the competition were maintained.

Each simulated match consists of a questioner bot and an answerer bot, both selected from independent pools, competing to guess the secret word.

The flow of information, the skill updates (μ, σ), and the matchmaking system remain faithful to the competition’s architecture, though applied on an individual level.

The interpretation and analysis of results — such as success rate, sensitivity to chaos, and skill evolution — remain valid and can be extrapolated to the original 2 vs 2 environment, as the internal logic of each interaction is preserved.

4.1 Simulation Results Analysis

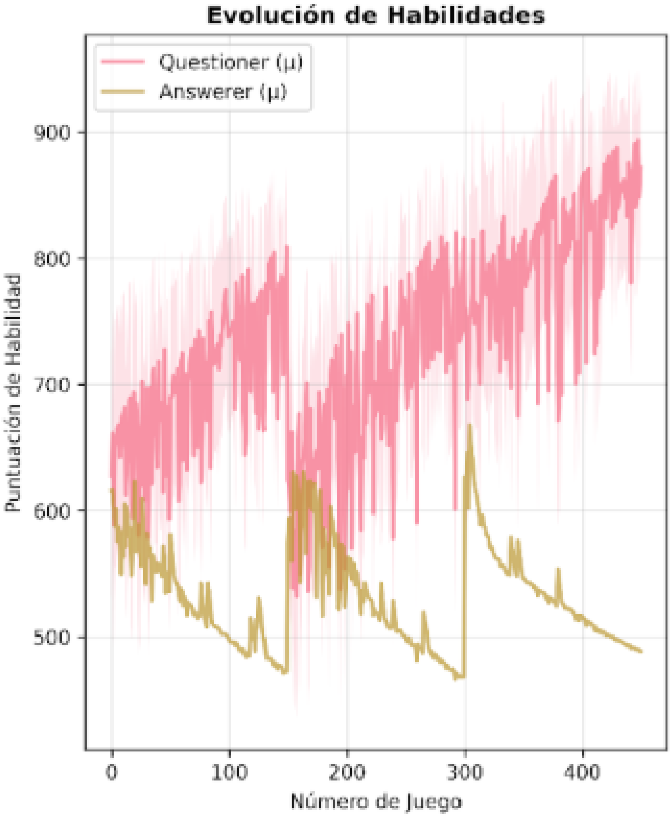


Figura 7: Skill evolution graph.

The skill evolution graph reveals a systematic and marked divergence between the roles of questioner and answerer. The pink line, corresponding to the questioner, exhibits a consistent and pronounced upward trend, evolving from approximately 650 skill points to nearly 900 over the course of 450 simulated games. This ascending trajectory is characterized by high initial variability, as evidenced by the wide uncertainty bands at the beginning of the simulation, which gradually narrow as the system reduces the uncertainty (σ) associated with the agent's skill estimation.

In contrast, the brown line representing the answerer shows a steady downward behavior, starting near 600 points and gradually declining until stabilizing within the 450–500 point range. This systematic divergence between the two roles suggests a fundamental imbalance in the inherent difficulty of each position within the game, where the questioner role appears to have structural advantages that allow it to accumulate success more consistently than the answerer.

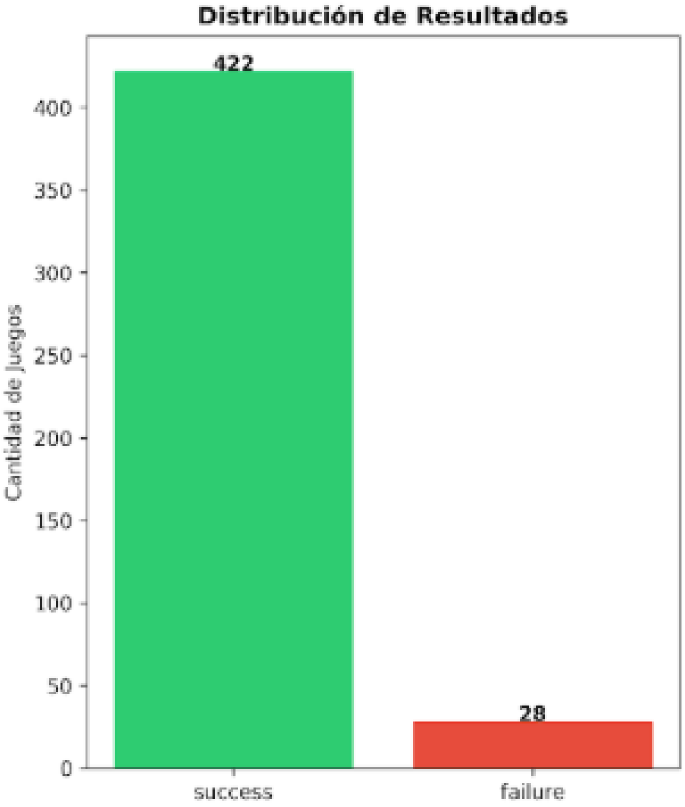


Figura 8: Distribution of results.

The distribution of results reveals a disagreement with the design hypotheses established in the previous workshops, showing a high success rate of 93.8 % (422 successes versus 28 failures). This proportion significantly contradicts the expectations based on the original Kaggle competition, where a more balanced success rate—around 70–80 %—would be expected, reflecting the inherent complexity of the “20 Questions” domain and the natural difficulty of guessing secret words through binary questioning.

This result pattern indicates that the simulated system is overly favorable to the questioner role, suggesting that the implemented questions may be too effective in their deductive capacity, or that the answerer’s responses are overly predictable and lack the necessary ambiguity to generate a genuine challenge.

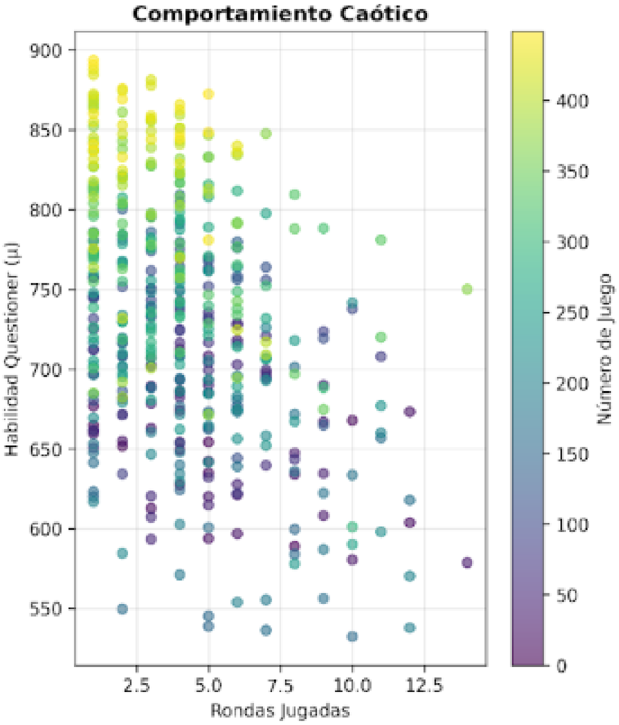


Figura 9: Chaotic Behavior Graph Analysis.

Chaotic Behavior Graph Analysis

This graph compares the skill level of the questioner with the number of rounds played in each match, using a color gradient to indicate temporal progression. It can be observed that most games are resolved in just a few rounds, especially when the questioner has higher skill.

The system responds coherently: more skilled bots tend to solve the games faster, reflecting an effective learning process and sensitivity to individual ability.

The concentration of most games within the 2–8 round range, with very few extending beyond 10 rounds, reveals a distribution that lacks the variability and dispersion expected in a system truly sensitive to chaos. The analysis of this graph suggests limited system sensitivity, as no clearly chaotic patterns, bifurcations, or nonlinear emergent behaviors—characteristic of complex systems according to chaos theory—are observed.

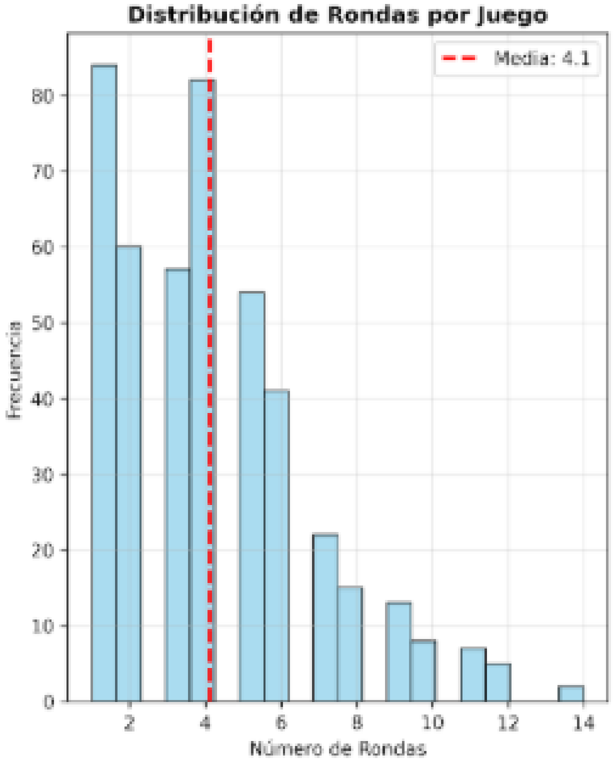


Figura 10: Distribution of rounds per game.

With an average of 4.1 rounds per game, indicated by the red dashed line, the system resolves matches in significantly fewer rounds than the theoretical limit of 20 rounds established by the competition rules. The distribution exhibits a sharp peak in the 2–4 round range, followed by a long tail extending to approximately 14 rounds, but it never reaches the maximum allowed limit.

This concentration in the lower round counts indicates that questioners are resolving approximately 80 % of the matches in 6 rounds or fewer, suggesting a high level of deductive efficiency.

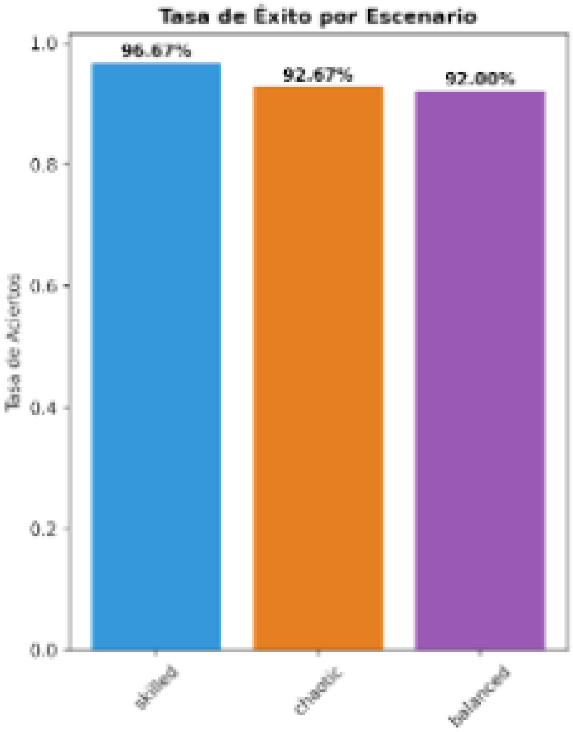


Figura 11: Success rate by simulated scenario.

The ”Skilled”scenario achieved a 96.67 % success rate, the Çhaotic”scenario reached 92.67 %, and the ”Balanced”scenario obtained 92.00 %, resulting in a variation of only 4.67 % between the best and worst performing cases. This convergence is highly problematic, as it contradicts the design expectations in which a significant difference was anticipated following the pattern *Skilled Balanced Chaotic*, with substantial variations reflecting the different configurations of ambiguity, consistency, and strategy implemented in each scenario.

The fact that all success rates exceed 90 %, including the scenario specifically designed to be çhaotic”with higher ambiguity and inconsistency, suggests that the implemented differentiation parameters are not having the intended impact on system behavior.

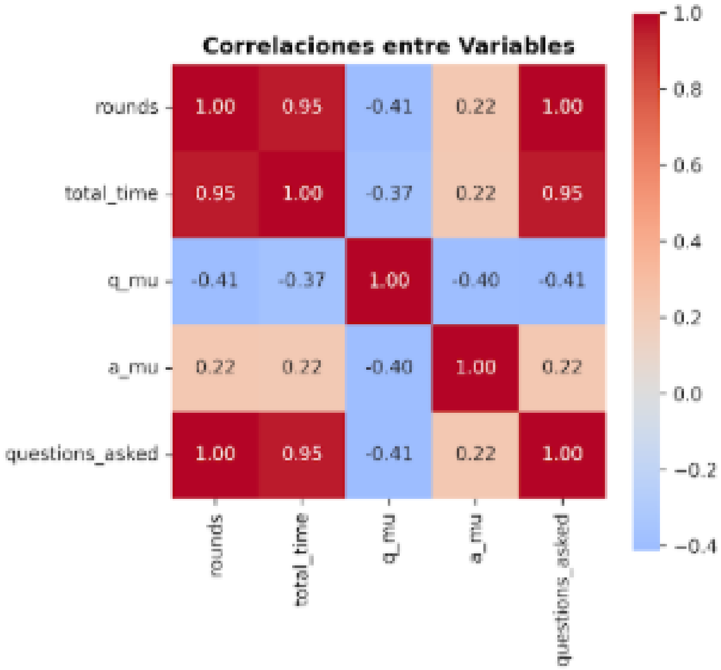


Figura 12: Correlation between system variables.

Perfect correlations (1.00) between `rounds-total_time` and `rounds-questions_asked` are expected by definition, as these variables are intrinsically linked by the mechanics of the game. However, the strong correlation (0.95) between `rounds` and `total_time`, along with the moderate negative correlations (-0.41) between `rounds` and both agents' skill levels (`q_mu`, `a_mu`), reveal highly predictable patterns.

The systemic interpretation of these correlations confirms logical relationships, such as the inverse relationship between skill and game duration (i.e., the higher the skill, the fewer rounds required).

The structure of these relationships confirms that the system is correctly modeling dependencies between variables, as expected in a well-designed and controlled environment.

5. Proposed Design Improvements for the "20 Questions" System

The skill analysis reveals an increasing disparity between the roles of questioner and answerer, suggesting the need for measures to ensure a more balanced competition. To address this issue, it is proposed to refine the matchmaking system by optimizing the algorithm so that matches occur between bots with truly comparable skill levels, thus avoiding imbalances that may distort skill development in each role.

Additionally, the inclusion of evaluation metrics beyond win/loss outcomes is recommended, such as the quality of the questions posed and the reasoning efficiency. These would promote more sophisticated strategies and foster a more equitable evolution between both roles.

Although the system already incorporates feedback through skill adjustment (μ, σ), it can still be strengthened by integrating more advanced learning mechanisms. One possible improvement is the implementation of contextual memory, enabling bots to adapt their strategies based on the history of previous games, thereby simulating long-term learning and adaptation.

Furthermore, the introduction of a multi-level feedback system is suggested, where the outcome of individual matches affects not only personal performance but also influences collective strategies or the overall system configuration. This could support self-organization processes and increase the system's resilience in the face of dynamic or unpredictable conditions.

6. Conclusion

The simulation carried out made it possible to put into practice the design of the "20 Questions" system proposed in the previous workshops. Based on the processed data and the defined scenarios, it was possible to observe the behavior of the bots and analyze how skill, strategy, and consistency influence the development of the matches. Although the difference between scenarios was not as noticeable as expected, certain performance patterns were identified, and the system was shown to operate in a stable manner under different conditions. The implementation, although simplified, served as a foundation for better understanding how the system works and identifying aspects that could be improved in future work.

GitHub repository

All materials related to Workshop 1Workshop 2 including reports and diagrams are available in the following GitHub repository:

- Systems design and analysis course repository link

References

[1] Waechter, T. (2024). *LLM 20 Questions Games Dataset*. Retrieved from `https://www.kaggle.com/datasets/waechter/llm-20-questions-games?select=EpisodeAgents.csv`

[2] Sierra, C. A. (2025). *Systems Analysis and Design – Course Materials*. Retrieved from `https://github.com/EngAndres/ud-public/tree/main/courses/systems-analysis`