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

Workshop No. 2 — Kaggle Systems Design

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

Objective

The main objective of developing this workshop is to design a system architecture for the Kaggle competition that we decided to analyze, which was the "20 Questions" competition, based on the functional, structural and sensitivity analysis performed in the previous Workshop 1. This design should integrate systems engineering principles, ensure modularity, scalability, and maintainability, and propose strategies to handle chaotic behaviors and sensitive elements of the system. In addition to the above, all design decisions should be clearly and technically documented, incorporating structural diagrams and interfaces that allow a complete understanding of the data flow and interaction between components.

1. Review of Workshop #1 Findings

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.

1.1. Key Points from the Analysis

The key components identified in the system include the secret word, which serves as the central goal of each match; the bots, divided into two functional roles—questioners, who formulate strategic yes/no questions and make guesses, and answerers, who respond with binary answers; the game history, which stores the sequence of questions and responses that guide the reasoning process; the matchmaking system, which pairs bots with similar skill levels to ensure balanced gameplay; and the ranking mechanism, which updates each bot's estimated skill (μ) and uncertainty (σ) after every match, enabling continuous adaptation and fair competition.

1.2. Main Elements of the System

Some of the core components we identified are:

- The secret word to be guessed.
- The history of questions and answers.
- The bots acting as questioners and answerers.
- The matchmaking and ranking system.

1.3. Sensitive and Chaotic Factors

When designing the system, it's essential to consider several sensitive and chaotic factors that can impact its performance. For instance, a poorly worded or ambiguous question can easily steer the team in the wrong direction from the start, and early mistakes tend to accumulate and influence future decisions. As we observed in Workshop 1, starting with a wrong assumption can lead to a flawed line of reasoning that moves further away from the real objective. In addition, after every game, the system updates each bot's skill (μ) and uncertainty (σ), creating a constant feedback loop. This makes the system highly dynamic and unpredictable, as there are many possible paths that can lead to the correct answer. That's why it's important to design an architecture that is clear, stable, and adaptable to unexpected situations. To achieve this, the system should include well-defined modules, error-handling mechanisms, and real-time adjustment strategies that help reduce sensitivity to errors and maintain consistent performance, even in complex scenarios.

2. System Requirements

Based on the analysis from Workshop #1, we identified several requirements that the system design must meet to ensure good performance and a suitable experience for participants. These requirements include both technical aspects and user-centered considerations.

2.1. Functional Requirements

- The system must allow 2-vs-2 matches, where each team has one questioner and one answerer.
- The game must end after a maximum of 20 rounds.
- Each agent must be able to submit a question (up to 750 characters) or a guess (up to 100 characters) within 60 seconds.
- Each bot must have 300 seconds of additional reserve time per game.
- The system must validate the input data (questions, guesses, and answers) and enforce the time limits.
- Each bot's skill must be updated using a ranking system based on match results.
- Each user may only have three active bots at a time.
- The system must match bots with similar skill levels to ensure fairness.

2.2. Non-Functional Requirements

- The system must be modular, allowing components (such as game logic, validation, or ranking) to be updated or replaced independently.
- It must be scalable and capable of handling multiple users and matches simultaneously.
- It must provide clear and quick feedback to participants (e.g., validation errors or match results).
- The system must ensure fair play through balanced matchmaking and transparent rules.
- It must include security measures to prevent users from manipulating results or breaking the rules.

2.3. User-Centered Considerations

- Participants must have an easy way to upload their bots and check their performance.
- Match results and leaderboard positions should be displayed clearly and understandably.
- The system should provide reports or feedback after each match to help users improve their models.

These requirements are the foundation for the system architecture and implementation strategy. The design will aim to meet these needs while remaining flexible enough to adapt to future changes.

3. High-Level Architecture:

For the development of this section, an architecture is proposed to describe the data flow and the interaction between the different components of the system. In addition, each of the modules was labeled and a description of their responsibility was made. Finally, the principles of systems engineering were mentioned, which allowed to give these structural solutions.

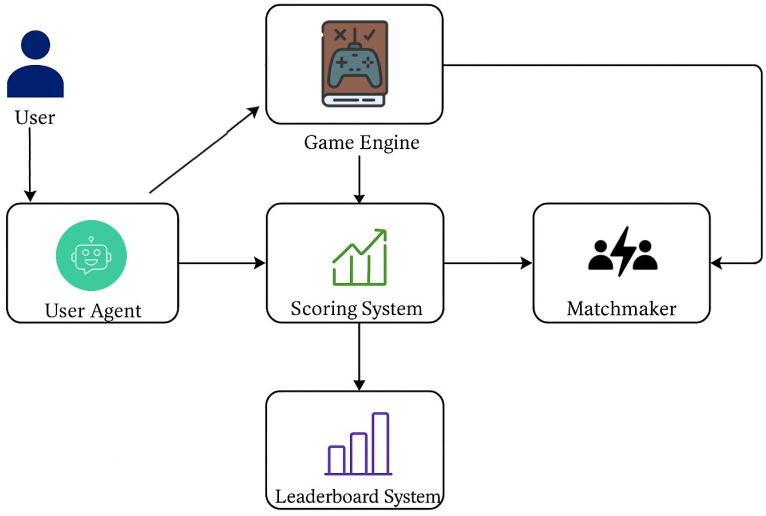

Figura 1: Architectural diagram describing the flow of data and interaction between components

The system follows a modular architecture that allows to organize and process data efficiently, ensuring a correct interaction between the different components. Everything starts when the user creates and trains a customized bot (User Agent) and uploads it to the platform. This agent contains the logic in charge of formulating questions and making guesses with the objective of guessing a secret word in as few rounds as possible.
The User Agent interacts directly with the Game Engine, which is in charge of controlling the development of the game: starting and managing rounds, validating questions and answers, making sure that time limits are met, and ensuring that rules are respected. Each interaction between the agent and the game engine generates valuable information about the bot's performance.
The Scoring System is in charge of evaluating the bot's performance after each game. To do so, it uses data such as the bot's initial score, the result of the game and the new updated score. It also considers how much confidence the system has in that score, based on how many times the bot has played. With this information, the system automatically adjusts the bot's skill.
These updated values allow the system to match the bot with others of a similar level, ensuring fairer and more balanced games.

Once the bot's score is updated, the Scoring System sends these results to the Matchmaker, which is responsible for analyzing the skill and uncertainty values to determine which other bots it should compete with in the future. This ensures that the matches are with other bots of similar level. Finally, the calculated results are sent to the Leaderboard System, which stores and displays the positions of the best bots for each user.
The system incorporates continuous feedback, by automatically updating the bot's skill and uncertainty after each game. The architecture also considers hierarchy, where the Game Engine acts as the operational core, while the other modules play specialized roles in processing, control and evaluation. The principle of adaptability is also considered, since the system adjusts to new data and changing behaviors of the bots, due to the use of dynamic metrics from the dataset. Finally, the principle of equifinality is integrated, allowing different bot strategies to achieve the same goal: guessing the secret word.

4. Addressing Sensitivity and Chaos

The *"20 Questions"* system exhibits high sensitivity to variations in input, ambiguous interpretations, and dynamic environmental conditions. To manage these chaotic elements and ensure system stability, the following design strategies are implemented:

4.1. Feedback Loops

After each game, the system updates each bot's estimated skill (μ) and uncertainty (σ) based on performance. This continuous feedback mechanism ensures that future pairings are based on up-to-date performance, maintaining competitive balance and preventing skill mismatches.

4.2. Confidence-Based Adaptation

New bots start with high uncertainty values, and as they accumulate game data, their confidence increases. This dynamic adjustment allows the system to avoid unpredictable pairings and better manage agent performance under limited information.

4.3. Monitoring Routines

The system includes strict constraints on time (60 seconds per action, 300 seconds total reserve) and input length (750 characters for questions, 100 for guesses). These limits are continuously monitored to detect violations. Bots exceeding these thresholds are automatically penalized or disqualified, reducing the risk of cascading system failures.

4.4. Error Handling Mechanisms

All user inputs are validated for structure and format to prevent malformed queries. Unexpected behaviors—such as invalid characters, unanswered rounds, or inconsistent logic—are flagged by a monitoring service and logged for review, ensuring robustness under unpredictable conditions.

This layered approach integrates sensitivity management and chaos mitigation directly into the system architecture, allowing the design to remain resilient, adaptive, and reliable even under uncertain or volatile inputs.

5. Technical Stack and Implementation Sketch

5.1 Recommended Tools and Frameworks

The following table presents the recommended tools, programming languages, and frameworks for the realistic implementation of the bot system:

Tool / Language	Main Role	Justification
Python	Main bot logic (question and answer)	Flexible and widely used in AI/NLP tasks. Supports libraries such as Transformers, NumPy, and Pandas. Easy to debug and scale.
Hugging Face Transformers	Language model	Access to pretrained models (e.g., BERT, RoBERTa, GPT), enabling question generation and context interpretation without training from scratch.
Pandas / NumPy	Internal bot analysis	Useful for manipulating data and analyzing patterns in bot interactions (e.g., question/answer quality, timing).
Jupyter Notebooks / `.py` scripts	Development environment	Allow for rapid prototyping, interactive testing, and result visualization.
R	Statistical evaluation of performance	Excellent for quantitative analysis. Enables computation of performance metrics (mean, standard deviation), simulations, and comparisons.
ggplot2 / dplyr (in R)	Data visualization and transformation	Useful for generating descriptive graphics showing bot behavior across sessions and configurations.

Cuadro 1: Recommended technical stack and their roles.

5.2 Implementation and Integration Plan

- **Modular Bot Design (in Python):** A base bot interface is defined (e.g., `respond(question)`), and each strategy (e.g., using different Transformer models) is implemented as a subclass. This follows the **Strategy Pattern**, allowing easy swapping of behaviors without changing core logic.
- **Integration of Transformer Models:** Hugging Face's `pipeline` or fine-tuned models are encapsulated in a class. Each bot uses a different model as its internal engine to process input and generate responses.
- **Internal Analysis with Pandas / NumPy:** Interaction logs are stored in `DataFrames` including timestamps, questions, responses, response time, and scores. Basic metrics such as average response length or keyword frequency can be computed.
- **Performance Evaluation with R:** Logs exported as `.csv` files are analyzed in R using `dplyr` for data manipulation and `ggplot2` for visualization. Metrics such as mean score, standard deviation, and distribution plots can be generated for comparative studies.

5.3 Design Summary

- **Design Pattern:** Strategy Pattern for switching bot behavior dynamically.
- **Separation of Concerns:**
  - Python: Bot logic and execution.
  - R: Statistical evaluation and visualization.

- **Integration Method:** CSV files are used to transfer results from Python to R.
- **Deployment Scope:** Fully local, no external infrastructure (no APIs, databases, or servers).

GitHub repository

All materials related to Workshop 1 and Workshop 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