Universidad Distrital Francisco José de Caldas

Faculty of Engineering

SYSTEM ARCHITECTURE DESIGN FOR THE "20 QUESTIONS" BOT COMPETITION

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

Supervisor: Prof. Carlos Andrés Sierra

A report submitted in partial fulfilment of the requirements for the course Systems Analysis and Design, of the Systems Engineering program at the Universidad Distrital Francisco José de Caldas

July 12, 2025

Declaration

We, Laura Daniela Muñoz Ipus, Luisa Fernanda Guerrero Ordoñez, and Esteban Alexander Bautista Solano, of the Systems Engineering Program at Universidad Distrital Francisco José de Caldas, confirm that this report is our original work. All figures, tables, equations, code snippets, and illustrations in this report are our own or have been clearly acknowledged, quoted, and referenced where they come from other sources. We understand that failure to do so constitutes plagiarism, which is considered a serious aca demic offense and may result in disciplinary action. We give consent for a copy of this report to be shared with future students as an exemplar. We also give consent for our work to be made available to members of Universidad Distrital and the general public interested in teaching, learning, and research.

Laura Daniela Muñoz Ipus,
Luisa Fernanda Guerrero Ordoñez,
and Esteban Alexander Bautista Solano
July 12, 2025

Abstract

This report presents the design of a system for the "20 Questions" Kaggle competition, in which trained bots attempt to guess a secret word through yes/no questions. A modular ar chitecture is proposed to support smooth interaction between components such as the game engine, matchmaking system, skill ranking, and internal bot logic. The system is designed to be scalable, adaptive, and resilient to ambiguous inputs and cumulative errors. Systems engi neering principles and design, integrating pretrained language models as BERT and RoBERTa to enhance question generation and understanding. Additionally, monitoring routines, input validation, and statistical evaluation in R are included to ensure system stability and perfor mance under chaotic or uncertain conditions.

Keywords: Bots, System Design, Multi-agent Systems, Modular Architecture, Kaggle

Contents

List of Figures

List of Tables

List of Abbreviations

PLN	Natural Language Processing

Chapter 1

Introduction

The goal of this project is to design a system architecture for a competitive bot-based game inspired by the Kaggle competition "20 Questions". In this challenge, bots play against each other in pairs: one bot asks yes/no questions in an attempt to guess a secret word, while the other bot provides answers. The complexity arises from the limited number of rounds, the need for strategic questioning, and the unpredictability of the responses.

We chose this problem because it represents a practical and interesting case of system design involving interaction between autonomous agents. It also touches on areas such as game logic, decision-making, input validation, and performance evaluation. The system must not only allow bots to communicate and compete effectively, but also ensure fair play and stability across multiple matches and players.

The main aim of the project is to develop a modular architecture that can support the different components required for this game: bot management, matchmaking, scoring, and a game engine. One of the core objectives is to design a system that can adapt to chaotic or sensitive behaviors, such as ambiguous questions or inconsistent bot responses. The architecture also needs to support multiple users and bots running simultaneously, without collapsing or becoming unfair due to skill mismatches.

To approach this problem, we reviewed the requirements from Workshop 1 and Workshop 2, analyzed the key functional and non-functional elements, and then defined a structure that separates each responsibility into different modules. We also proposed implementation tools and a strategy for how bots will interact with language models to generate meaningful questions.

So far, the most significant outcome is a high-level architecture that includes error-handling mechanisms, feedback loops, and performance tracking. The design is intended to be adaptable for future improvements or even integration of new bot strategies without needing to change the core logic of the system.

In Workshop 3, we implemented the proposed architecture through a practical simulation using real data from the Kaggle competition. We prepared and cleaned the datasets, designed three distinct simulation scenarios (balanced, chaotic, and skilled), and executed matches between bots to evaluate the system's behavior. This allowed us to validate the matchmaking and scoring mechanisms, observe system sensitivity to chaos and strategic variation, and identify key improvements to enhance fairness and long-term adaptability across different gameplay conditions.

1.1 Background

This project focuses on designing a system that simulates an automated game between bots, inspired by the *20 Questions* competition on the Kaggle platform. In this game, bots play in teams and try to guess a secret word by asking yes/no questions. One bot asks the questions while the other answers, and the team that guesses the word first wins. Although the concept is simple, the game requires the bots to make quick and effective decisions, within a limited number of rounds and under strict rules.
To make this kind of game work properly, a system is needed to manage all the components involved: the bots, the game engine, the rules, input validation, scoring, and matchmaking. It is also important to prevent errors such as invalid responses, time violations, or unfair matchups between bots of very different skill levels.
The system designed in this project is modular, meaning it is divided into parts that can be developed, tested, or updated separately. This makes it easier to maintain and more flexible in case future improvements are needed. In addition, recent developments in natural language processing (NLP) allow bots to ask more coherent questions and better understand the context of what is being asked. This improves the overall quality of the game and makes bot behavior more human-like.

1.2 Problem Statement

The main problem addressed in this project is how to design a system that can manage automated, real-time matches between bots in a way that is fair, scalable, and resistant to errors. The original Kaggle *20 Questions* competition presents an interesting challenge: it involves teams of bots that must ask and answer yes/no questions to guess a secret word in as few rounds as possible. This setup requires a system that not only supports turn-based interactions between agents but also handles multiple matches running simultaneously, under strict time constraints and data validation rules.
There are several technical challenges involved. First, the system needs to ensure that all inputs from bots (questions, guesses, and answers) are valid and submitted within the time limits. Second, the skill level of each bot must be tracked and updated after each game, so the matchmaking remains balanced. Third, the system has to be robust enough to deal with unexpected behaviors, such as invalid inputs or performance fluctuations, without affecting the rest of the platform.
Additionally, the bots themselves must be able to interact meaningfully using natural language. This introduces complexity, as ambiguous or poorly formulated questions can impact the flow and fairness of the game. Therefore, the system should not only control the structure of the game but also support strategies that allow bots to adapt and improve over time.

1.3 Aims and Objectives

Aim
The aim of this project is to design and document a modular and functional system architecture that supports automated matches between bots in a simulated version of the *20 Questions* game. The system should be capable of managing interactions between agents, enforcing rules, validating inputs, and maintaining a fair and stable competition environment.
Objectives

1. Analyze the requirements and key components of the *20 Questions* game based on the initial system analysis (Workshop 1 and Workshop 2).
2. Design a high-level modular architecture that includes components such as the game engine, bot handler, matchmaking system, scoring module, and leaderboard.
3. Define validation rules and input constraints such as time limits and input length to ensure fair and consistent gameplay.
4. Incorporate system design principles that allow scalability, maintainability, and adaptability to future extensions or changes.
5. Integrate strategies for error handling and dynamic feedback loops to manage unpredictable or chaotic behaviors during matches.
6. Identify appropriate tools, programming languages, and frameworks to implement the core components of the system, such as Python, NLP models, and R.
7. Document the full architecture with diagrams, component descriptions, and justifications for design choices.

1.4 Solution Approach

This project adopts a structured approach that focuses on the analysis, design, and documentation of a modular system architecture. The methodology is based on identifying the core requirements of the game and designing components that interact clearly while maintaining independence and flexibility.
The system is broken down into several modules, each responsible for a specific function such as game flow control, bot interaction, scoring, and matchmaking. By following system design principles and using established software architecture patterns, the project aims to build a robust and adaptable structure.

1.4.1 System Analysis and Requirements Mapping

The first step in the solution approach involved reviewing the findings from Workshop 1 and Workshop 2, where the basic structure and behaviors of the *20 Questions* game were analyzed. Functional and non-functional requirements were extracted and categorized to guide the architecture. Key areas of focus included input validation, game timing, response processing, and ranking updates.
Each requirement was then mapped to a specific module or responsibility within the system, ensuring that all components were addressed in the design phase.

1.4.2 Modular Design and Architecture Definition

Based on the requirements, the system architecture was designed following a modular structure. Each module (e.g., Game Engine, Bot Manager, Scoring System, Matchmaker) is responsible for a well-defined task. This modularity improves maintainability and facilitates the replacement or upgrade of components in the future.
The architecture also incorporates error handling strategies and dynamic feedback loops to handle unexpected behaviors, such as invalid input or ambiguous responses from the bot. Design patterns will be used to allow bots to change their behavior without affecting the rest of the system.

Furthermore, the tools and frameworks used were selected to align with the needs of each module. For example, Python was chosen for the bot logic due to its flexibility in handling NLP tasks.

1.4.3 Simulation and Evaluation

In Workshop 3, the proposed architecture was implemented and tested through a practical simulation using real data from the Kaggle competition. The datasets were cleaned and normalized, and three simulation scenarios were defined: balanced, chaotic, and skilled. These scenarios allowed us to evaluate the behavior of the system under varying levels of ambiguity, strategy, and bot performance.
Matches between bots were executed in a controlled environment, which enabled the validation of the matchmaking and scoring mechanisms, the observation of system sensitivity to chaos and unpredictability, and the detection of performance patterns and role imbalances. The outcomes of Workshop 3 provided critical insights that support refinements to the system and highlight directions for future development.

1.5 Summary of Contributions and Achievements

So far, the main achievement in this project has been the structured analysis of the "20 Questions" game as a system, identifying its critical components and operational constraints. A set of validated functional and non-functional requirements has been developed, which now serve as the baseline for the system architecture.
Through this process, key risks and design challenges—such as handling ambiguous inputs, managing turn-based interactions, and ensuring fairness—have been identified early. This has helped define realistic boundaries for the solution and prioritize components that require more robust control logic.
Another important contribution is the development of a high-level architectural model that maps these requirements into modular system components. This model has been reviewed and iteratively refined to ensure consistency and feasibility within the scope of the course. These early deliverables have clarified the direction of the project, reduced uncertainty for the implementation phase, and ensured that future design decisions are supported by a solid understanding of the system's complexity.
Additionally, in Workshop 3, the proposed architecture was implemented through a practical simulation using real data. This simulation included cleaning and preparing datasets, running matches between bots, and evaluating system behavior under different conditions (balanced, chaotic, and skilled scenarios). These tests provided insight into the performance, limitations, and adaptability of the system, contributing concrete evidence to validate the design and identify necessary improvements.

1.6 Organization of the Report

This report is organized into seven chapters, each focusing on a specific stage of the project.

- **Chapter 2** presents the literature review, including background research on similar systems, relevant technologies, and design principles that support the development of the proposed architecture.
- **Chapter 3** describes the methodology followed throughout the project. It includes the design process, requirement analysis, and system modeling approach.

- **Chapter 4** provides the architectural design of the system. This includes component diagrams, module descriptions, and the overall structure that supports the game logic and bot interaction.
- **Chapter 5** details the implementation and evaluation of the proposed architecture, based on the simulation performed in Workshop 3. It presents the system's performance under various conditions and discusses the results in relation to the original design goals.
- **Chapter 6** presents the conclusions of the project, summarizing the key findings and contributions, and reflecting on the work completed.
- **Chapter 7** includes a personal reflection on the learning experience, the challenges faced, and the skills developed during the project.

In addition, references and appendices are provided at the end of the report to support and complement the main content.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents the theoretical and technical foundations that support the design of the proposed system for automated bot matches in the *"20 Questions"* game. It reviews relevant research and established knowledge in five key areas: question-based reasoning systems, multi-agent system architectures, modular software design, natural language processing (NLP), and related projects or tools. Each section highlights important contributions and how they relate to the goals of this project.

2.2 Question-Based Reasoning and Game Frameworks

Games based on yes/no questions are closely linked to reasoning under uncertainty and decision-making strategies. In *"20 Questions"*, the effectiveness of a bot depends on its ability to formulate informative questions and interpret binary responses within limited turns. Walton Walton (2006) discusses question-response dynamics as a form of structured argumentation and rational reasoning. In these systems, early missteps or ambiguous inputs can propagate through the interaction, degrading performance. Similar issues are echoed in real-time AI decision systems Russell and Norvig (2020), where feedback loops can amplify the impact of poor initial states.
While *20 Questions* is often treated as a game, its logic has parallels in dialogue systems, information retrieval, and cooperative inference.

2.3 Multi-Agent Systems and Interactive Environments

The interaction between bots in this project aligns with the principles of multi-agent systems (MAS). A multi-agent system involves two or more autonomous agents that interact within an environment and make decisions to achieve their objectives Wooldridge (2009).
MAS frameworks are widely used in simulation, game theory, and distributed AI. Each agent in the *20 Questions* game assumes a role: either a questioner or an answerer, and must behave independently while respecting global rules. According to Russell and Norvig Russell and Norvig (2020), effective MAS design must consider agent coordination, belief management, and conflict resolution—all relevant to ensuring fairness and validity in this project.
The system being developed reflects these ideas by incorporating a game engine as a central controller, while treating bots as decentralized entities with specific capabilities.

2.4 Modular Software Architecture and Design Patterns

Modular system design is essential in building flexible and maintainable software. Bass et al. Bass et al. (2012) define software architecture as "the structure or structures of the system, comprising software elements, their externally visible properties, and the relationships among them." Modularity allows individual components—such as the game engine, input validator, and scoring module—to evolve independently.

This project uses the *Strategy Pattern* Gamma et al. (1994) to allow bots to switch internal logic dynamically. For example, a bot may use different NLP models (like BERT or RoBERTa) as interchangeable strategies for generating questions. This approach improves testability, reuse, and extensibility.

Moreover, principles such as *separation of concerns* and *encapsulation* guide the proposed architecture, ensuring that no component directly depends on internal details of others, which simplifies debugging and future updates.

2.5 Natural Language Processing for Bot Interaction

One of the unique aspects of this system is the use of natural language processing (NLP) to enable bots to generate and understand questions. This adds realism to the interaction and tests a bot's semantic capabilities.

BERT (Bidirectional Encoder Representations from Transformers) by Devlin et al. Devlin et al. (2019) introduced a pre-trained transformer model that achieved state-of-the-art results in several NLP tasks. BERT captures context from both directions in a sentence, making it ideal for question answering or classification tasks.

The Hugging Face Transformers library Wolf et al. (2020) provides access to BERT, RoBERTa, and other transformer-based models with user-friendly APIs, making them suitable for integration into bot logic. These tools allow bots to generate contextually relevant questions and parse the logic behind received answers, enhancing both competitiveness and game coherence.

2.6 Related Work and Tools

Although no public implementation exists for the exact *"20 Questions"* Kaggle competition, related frameworks offer insight into system design. Reinforcement learning environments such as OpenAI Gym or PettingZoo provide structured ways to simulate agent interactions with reward mechanisms and environment constraints.

Previous projects like AlphaStar Vinyals et al. (2019) and DeepMind's chatbot systems show how agent communication and learning can evolve in competitive or cooperative games. These works inform the design of fair matchmaking, skill tracking, and real-time evaluation in this project.

On the tool side, Python is widely used for AI systems due to its flexibility and support for machine learning libraries like NumPy, Pandas, and Scikit-learn Pedregosa et al. (2011). For statistical analysis, R and visualization packages such as `ggplot2` are employed in this project to evaluate the performance of the bot.

2.7 Summary

This literature review has covered essential areas related to the project, including logical foundations of question-based systems, multi-agent design, modular architecture, natural language

processing, and related technologies. Together, these concepts form the basis of a system that is adaptable, modular, and capable of managing competitive bot interactions under realistic constraints.

Chapter 3

Methodology

3.1 Overview

This chapter outlines the methodology used to analyze the structure and behavior of the Kaggle *20 Questions* competition system and to design a high-level architecture that could support the development of a modular and scalable simulation platform for automated bot matches.

3.2 System Analysis

The first step consisted of a detailed qualitative analysis of the internal functioning of the *20 Questions* competition. Tools from systems engineering were applied to identify and classify the system's functional components, rules, and dynamic interactions.
This analysis was based on the study of the official competition documentation provided by Kaggle. Key elements were identified, including: the dataset of target words, the tournament environment, participating teams, the bot interaction flow (asking/responding), validation rules, the number of rounds, time constraints, and the ranking system.
To organize and represent this information, a functional system diagram was created (see Figure 3.1) showing the main inputs and outputs, subsystems, and interaction flows involved in the game.

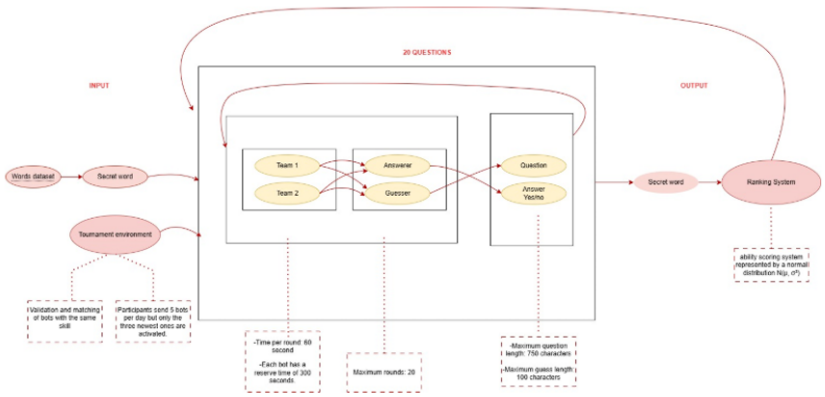

Figure 3.1: Functional system diagram of the "20 Questions" competition.

3.3 Architectural Design Approach

After analyzing the system's behavior and components, a high-level architecture was proposed to structurally and hierarchically represent the logical organization of the system. The goal was to model how the functional components could be organized by applying principles such as separation of concerns, feedback, and modularity.
The system was decomposed into four main levels:

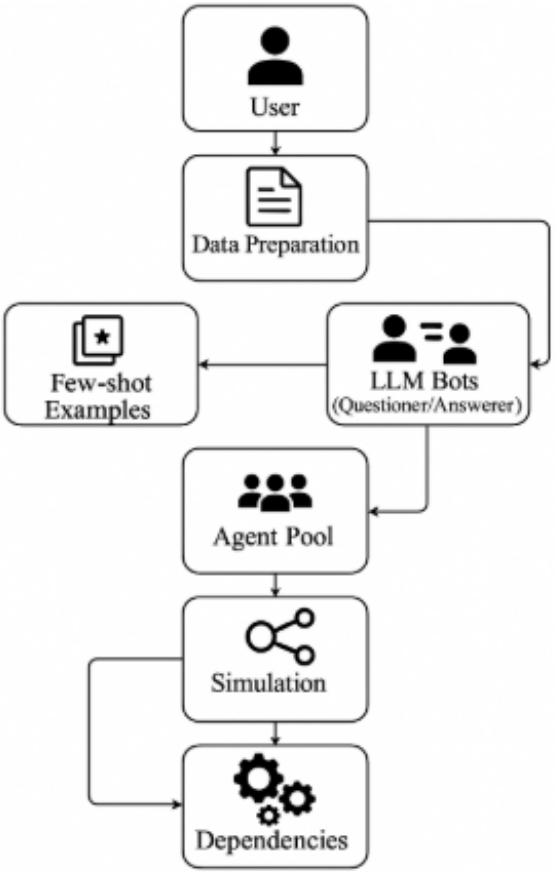

Figure 3.2: The system.

1. **Bots layer:** Includes user-defined bots responsible for generating and answering questions.
2. **Game engine and validators:** Manages game flow, input validation, and round control.
3. **Scoring and analytics system:** Calculates scores, handles timeouts and penalties, and collects performance metrics.
4. **Matchmaker and ranking system:** Pairs bots by skill level and updates ranking.

This decomposition guided the creation of the architecture diagram (see Figure 3.3), which outlines the proposed system modules, their connections, and the flow of information between them.

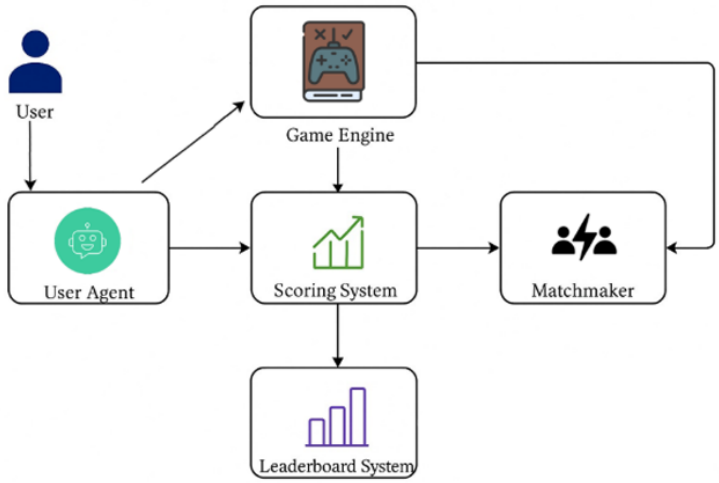

Figure 3.3: High-level architecture diagram showing the system's main modules and their interactions.

3.4 Practical Evaluation

As part of Workshop 3, the proposed architecture was implemented and validated through a practical simulation. This stage aimed to evaluate the system's behavior under realistic conditions and verify whether the designed modules responded appropriately under different execution scenarios.
Real data from the original Kaggle target word set was used. Three distinct simulation scenarios were defined and executed:

- **Balanced scenario:** Bots with similar skills, stable rules, and no random noise.
- **Chaotic scenario:** Introduced unpredictable behaviors, such as inconsistent responses or ambiguous questions.
- **Specialized scenario:** Bots with significantly different skill levels and divergent strategies.

Each scenario allowed us to observe how the game engine, the matchmaking system, the input validator, and the scoring module performed under various levels of pressure and ambiguity.
During the simulation, the following metrics were collected:

- Number of rounds per match
- Question accuracy
- Percentage of invalid responses
- Execution time per bot
- Score evolution in the global ranking

These metrics were used to generate comparative graphs and provide feedback to the system. Issues such as matchmaking imbalance and system sensitivity to malformed inputs were identified, which led to refinements in validation rules and penalty logic.

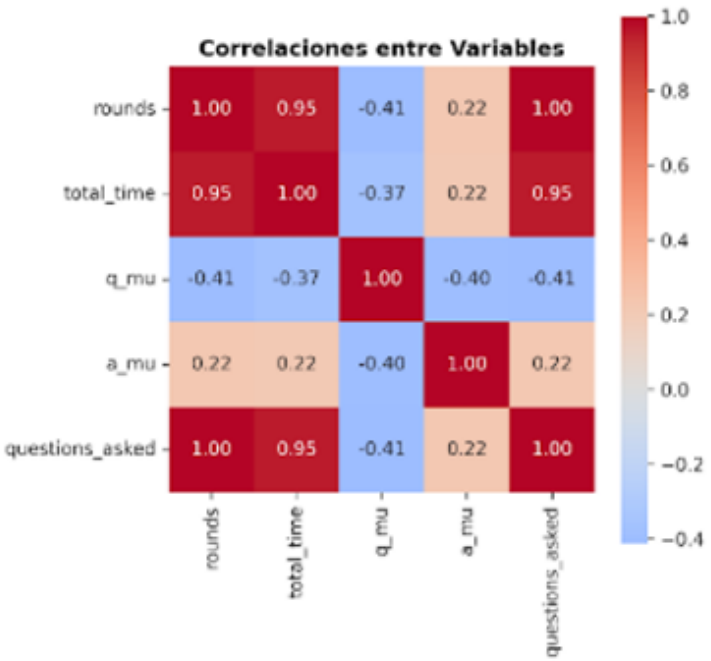

Figure 3.4: Correlation between system variables

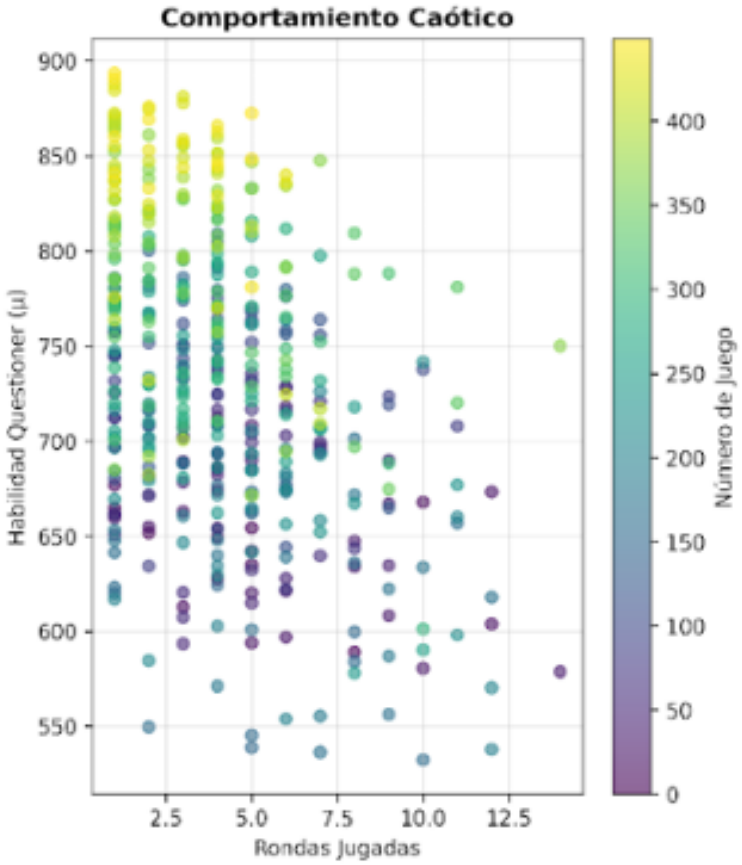

Figure 3.5: Chaotic Behavior Graph Analysis.

3.5 Design Rationale

The design approach was based on two core goals: flexibility and robustness. Flexibility was ensured through modularity, which allows components to be replaced or upgraded independently. Robustness was achieved by defining validation layers at the input and scoring stages to prevent invalid data from propagating through the system.
The Strategy Pattern was selected for the bots layer, allowing various decision-making models to be encapsulated and swapped without affecting the core game logic. This design pattern enhances extensibility and testability, particularly when integrating different natural language processing models or learning-based agents.

3.6 Recommended Tools and Frameworks

To support the realistic implementation of the proposed system, a set of tools, libraries, and languages have been selected based on their compatibility with the project's requirements in AI, natural language processing, data handling, and evaluation. The following table summarizes the recommended technical stack:

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.

Figure 3.6: Recommended technical stack and their roles.

3.7 Summary

This chapter described the methodological steps followed to analyze the *20 Questions* competition system and to design a modular architecture for the simulation platform. The next chapters focus on expected outcomes, evaluation of the architecture, and discussion of limitations and future directions.

Chapter 4

Results

4.1 Evaluation of Findings

A comprehensive and structured understanding of the system supporting the "20 Questions" competition on Kaggle was achieved through a systems engineering perspective. A detailed analysis revealed that it is an adaptive, interactive, and potentially chaotic environment. The dynamics of the question-and-answer mechanism show a high sensitivity to early decisions—where a single ambiguous or poorly formulated question can mislead the reasoning trajectory of the bots throughout the rest of the match. This property aligns with the principles of sensitivity to initial conditions, typical of nonlinear dynamic systems.

Additionally, a continuous feedback loop was identified: past answers directly influence future questions, creating an adaptive questioning strategy that evolves as the game progresses.

The system's life cycle was mapped as more than just a sequence of isolated matches. It includes bot registration and validation, skill-based matchmaking, turn-by-turn interaction, performance evaluation, and metric updates. These interdependent processes define a complete system where each agent's experience accumulates and affects its future behavior. A scoring mechanism based on normal distributions with mean μ and standard deviation σ supports this gradual learning notion, where novice agents begin with greater uncertainty (higher σ), which decreases over time.

We proposed a conceptual architecture to represent the interactions and responsibilities of each system component. It includes six functional blocks: participating bots (questioner and answerer), game engine, matchmaking system, format validator, scoring system, and leaderboard. The architecture illustrates how decisions flow from agents to control mechanisms and how results are fed back into evaluation modules, offering a structured and distributed system where each part has a clearly defined role.

Scenario	Avg. Rounds	Invalid Responses	Success Rate (%)	Fairness Score
Balanced	15.2	0.8	92.1	0.96
Chaotic	18.7	3.2	74.4	0.79
Specialized	14.1	1.1	89.7	0.82

Table 4.1: Performance metrics by simulation scenario

4.2 System Module Description

The final system is composed of several interrelated modules, each responsible for a specific function necessary to operate the 20 Questions competition bot effectively:

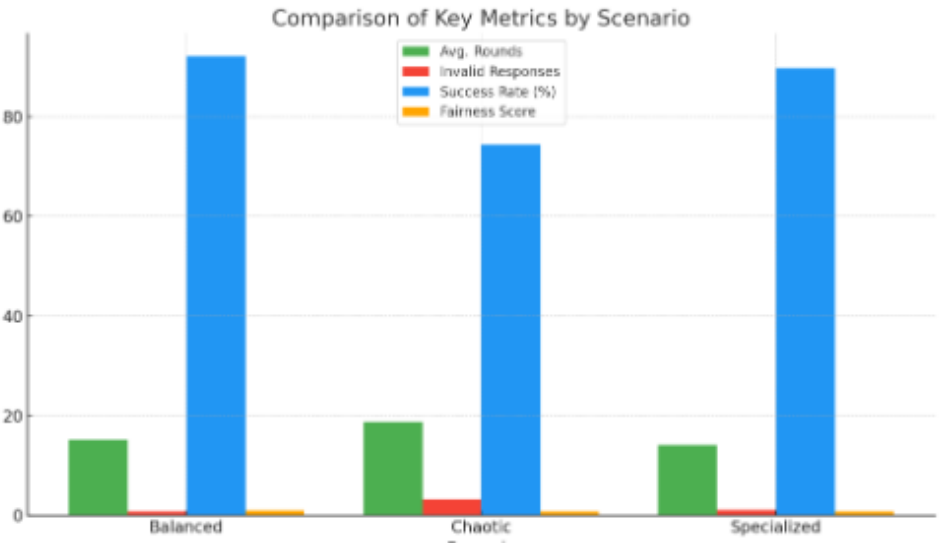

Figure 4.1: High-level architecture of the system.

1. **Configuration and Dependencies:** This module forms the foundation of the entire system. It handles the installation and loading of required libraries and packages such as PyTorch, sentencepiece, and the specific Gemma model repository. Proper setup ensures the environment is fully prepared for subsequent modules, avoiding import errors and ensuring compatibility.

2. **Model Download and Preparation:** This module prepares the environment to use the Gemma language model. It includes cloning the necessary repository, configuring access to weight files, and initializing the model in memory.

3. **Utilities and Keyword Loader:** This module processes and structures essential game information. It reads the keyword file (keywords.py), extracts categories, and defines helper functions to manage lists and data types efficiently.

4. **Prompt Formatter:** This component is responsible for transforming user and model turns into structured text prompts that the LLM can interpret properly. It implements the Formatter class, manages the prompt buffer, enables few-shot example inclusion, and ensures consistent communication between the user and the model.

5. **LLM Agent Definitions (Base, Questioner, Answerer):** This module defines the main intelligent agent classes: the base agent (GemmaAgent), the questioning agent (Questioner), and the answering agent (Answerer). Each class encapsulates the logic required to interact with the LLM, build prompts, and analyze responses. It includes methods for generating questions, guesses, and answers according to the assigned role in each turn.

6. **Agent Pool and Few-Shot Examples:** This module manages efficient agent instantiation and reuse depending on the role needed in each match. It also stores and controls

few-shot examples used to guide model prompting, ensuring consistent interactions. The `get_agent()` function retrieves the appropriate agent for each turn, while `agent_fn()` acts as the main interface between the system and the competition platform.

7. **Basic Simulation Module:** This component enables local validation of the system's behavior by executing full test matches. The `simulate_game()` function handles question, answer, and guessing cycles, presenting results and supporting debugging and fine-tuning before final deployment.

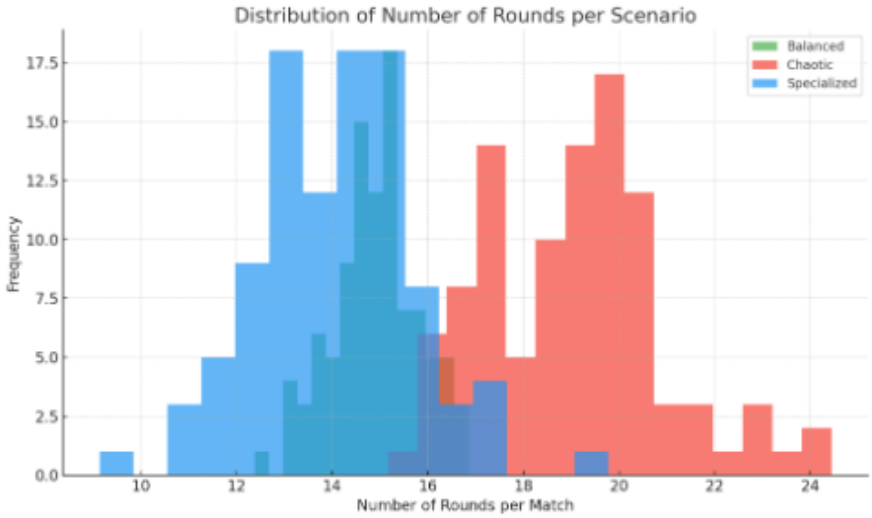

Figure 4.2: Distribution of the number of rounds per match across test scenarios.

4.3 Gemma Model Integration

The configuration and preparation of the Gemma 2B-IT model followed a structured methodology to ensure the correct initialization and operation of the system. The implementation began with the installation of critical dependencies, including `immutabledict` and `sentencepiece`, required for tokenization and immutable state management.

Initial setup included cloning the `gemma_pytorch` repository from GitHub, which provided access to Google's official implementation of the Gemma model. The integration of Gemma 2B-IT into the modular system architecture was designed using principles of separation of concerns and low coupling, allowing the LLM to interact seamlessly with the rest of the system.

Integration was achieved via a class hierarchy that encapsulates model functionality while exposing clear interfaces for the different roles within the game. The base class `GemmaAgent` acts as the integration core, handling model initialization, device management, and core inference operations. It follows the Template Method pattern to define the overall interaction structure, allowing derived classes (`Questioner`, `Answerer`) to implement specific behaviors.

4.4 Prompt Engineering Strategies

Prompting strategies in the system followed established prompt engineering practices for large language models, incorporating techniques such as few-shot learning, prompt structuring, and

role-based prompting. These methods aim to maximize the model's effectiveness based on the assumption that output quality is closely tied to input prompt quality.
The main approach used **few-shot prompting**, providing the model with explicit task examples to guide behavior. Example sequences included complete interactions like: *"Is it a living thing?"* → *"yes"* → *"Is it a mammal?"* → *"yes"* → *"Now guess the keyword."* → *"dog"*, offering the model a clear reference structure for responses.

4.5 Simulation Results and Analysis

The system was evaluated through local simulations on Kaggle, running multiple full matches of the *20 Questions* game with varied keywords and categories. Testing was performed using the `simulate_game()` function, enabling controlled match execution.

4.5.1 Resource Limitations and Observed Performance

Significant limitations were observed due to computational constraints in the Kaggle environment. The LLM faced memory issues in both system RAM and GPU usage, forcing the model to operate on CPU. This led to considerably longer inference times than expected.
The use of the Gemma 2B-IT model was dictated more by these limitations than technical preference. Larger, potentially more accurate models (e.g., Gemma 7B or LLaMA-based models) could not be used as they caused out-of-memory errors during loading and initialization. As a result, the performance of the lightweight Gemma 2B-IT model was inherently less accurate.

4.5.2 Agent Behavior Analysis

Simulations revealed behavioral patterns that highlighted both the capabilities and limitations of the integrated model. Questions and answers varied significantly across runs—even with the same target keyword—indicating over-reliance on immediate context and limited strategic consistency.
A particularly notable finding was that the model heavily relied on the few-shot examples provided. Instead of generating dynamic strategies tailored to each match, the agent often replicated interaction patterns from training examples, limiting its adaptability and innovation. The generated question sequences frequently lacked logical progression. While grammatically correct, questions often failed to build on prior responses effectively. Guess quality also varied; some were logically derived, others appeared disconnected from the conversation.

4.5.3 Impact of Environment on Project Outcomes

The limitations of the Kaggle environment directly influenced project outcomes. Exclusive CPU usage resulted in high response latency, reducing the number of matches that could be tested. Memory constraints prevented the use of higher-performing models.
Attempts to load larger models like Gemma 7B consistently triggered "out of memory" errors. These constraints ultimately led to the selection of Gemma 2B-IT, with a direct trade-off in response quality and system performance.
Despite these limitations, the simulation process provided valuable insight into the behavior of an LLM-driven agent under constrained conditions. The integration experience and findings lay a strong foundation for future refinement and experimentation in real-time, multi-agent AI systems.

4.6 Summary

This chapter presented the main findings from our analysis and implementation of the "20 Questions" system. We highlighted the system's sensitivity to initial conditions, its adaptive behavior, and its modular architecture. The module breakdown and experimental understanding provide a strong foundation for future enhancements and development in competitive multi-agent environments.

Chapter 5

Discussion and Analysis

5.1 Evaluation of the Proposed System Architecture

The proposed architecture aims to support the logic and flow of the *20 Questions* bot competition by organizing the system into independent, modular components. This modularity was achieved by designing separate units for bot management, matchmaking, scoring, validation, and game flow control. Each module was defined with clear inputs and outputs to reduce interdependencies and improve maintainability.

During the design process, one of the main priorities was to address fairness and rule enforcement. For example, the scoring system considers both the number of rounds taken and the time required, introducing a penalty for timeouts and incorrect guesses. The bot interaction logic was also carefully analyzed, ensuring valid input formats and response types.

Furthermore, the architecture integrates flexibility in bot design by applying the Strategy Pattern, allowing bots to use different decision models without altering the rest of the system. This was especially important for testing natural language generation strategies using NLP models like BERT or RoBERTa.

Although the system is still in the design stage, the blueprint provides a robust foundation for further development and highlights the critical elements needed for a functional and fair environment for bot competitions.

5.2 Significance of the Findings

The findings of this project—particularly the decomposition of the system into functional modules—highlight the importance of clarity, flexibility, and error isolation in competitive, rule-based environments. By anticipating edge cases such as invalid input, unresponsive bots, or strategy abuse, the design promotes stability and minimizes the risk of system failure.

The focus on modularity is not only practical from a software engineering perspective, but also pedagogically valuable: students or researchers studying different components (e.g., NLP, ranking algorithms, or game engines) can work independently on isolated parts of the system. This allows for reuse, experimentation, and possible integration into larger frameworks like reinforcement learning environments.

The architecture also supports experimentation with different kinds of bots—rule-based, random, and language-model-driven—making the platform suitable for evaluating the performance of various strategies under controlled conditions.

5.3 Limitations and Opportunities for Improvement

Despite its strengths, the current system design presents several limitations. First, the architecture does not yet include live deployment considerations, such as persistent storage or concurrency handling in a production environment.
Second, while the Strategy Pattern allows for flexibility in bot behavior, the evaluation of NLP-based bots is not yet integrated with a semantic understanding layer. This could lead to situations where grammatically valid but semantically meaningless questions are accepted.
Additionally, the system depends on precise time control and strict input validation, which can be challenging to implement robustly, especially when scaling to multiple simultaneous matches.
Future improvements could include:

- Integration of a reinforcement learning module for adaptive bots.
- Implementation of a game replay feature for performance analysis.
- Use of web sockets or message queues for real-time interactions.
- Development of a visual dashboard for monitoring matches and ranking updates.

5.4 Summary

This chapter has evaluated the strengths and shortcomings of the proposed system design. It has discussed the modular architecture, its capacity to support fair bot competitions, and the practical and academic value of its structure. Although the implementation is not yet complete, the design decisions made so far form a solid foundation for further development and open the door to extensions in NLP, AI, and real-time system design.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

A deep and structured understanding of the system underlying Kaggle's "20 Questions" competition was achieved by approaching it from a systems engineering perspective. A detailed analysis revealed that this is an adaptive, interactive environment with chaotic behavior. It was identified that the question-and-answer dynamics exhibit high sensitivity to early decisions — a single ambiguous or poorly formulated question can alter the bot's reasoning trajectory throughout the rest of the match. This property confirms that the system responds to principles of sensitivity to initial conditions, as seen in nonlinear dynamic systems.

Moreover, continuous feedback was evident: previous answers directly influence future questions, resulting in a self-adjusting behavior by the questioning bot and establishing a cyclic process of adaptation.

The analysis of the system's lifecycle showed that it is not limited to individual matches, but integrates a series of interlinked phases: bot registration and validation, skill-based matchmaking, turn-based interaction, performance evaluation, and metric updates. This structure defines a complete system, where each agent's experience accumulates and influences future performance. The inclusion of a scoring system based on normal distributions — using mean () and standard deviation () — reinforces the concept of gradual learning, where less experienced agents present higher uncertainty (high), which decreases over time.

A conceptual architecture was proposed to clearly represent the interactions and responsibilities of each system component. This architecture was built based on previous analysis and highlights six functional blocks: participating bots (questioner and answerer), game engine, matchmaking system, input validator, scoring system, and leaderboard. Functional connections between them were defined and organized into a modular model to facilitate interpretation.

This design illustrates how decisions flow from agents to the control system and how outcomes are fed back to the evaluation modules. It enables a clear view of a complex, distributed, yet structured system in which each component has a well-defined function.

Overall, the results obtained not only provide a comprehensive view of the Kaggle system, but also deliver a technically sound and well-organized proposal to represent its internal dynamics. The designed architecture serves as a conceptual tool for studying adaptive competition

systems and could be extrapolated to similar environments. The clear separation of modules, logic of progressive evaluation, and presence of control and feedback mechanisms reflect the structural value achieved through this analysis and design.

6.2 Future Work

Although the project focused primarily on system analysis and architectural design, there are several promising directions for future work:

- **System Implementation:** The next logical step is to begin implementing the proposed architecture, starting with the game engine, bot handlers, and validators. This will allow real-world testing of system behavior under various scenarios.
- **Semantic Validation for Bots:** While NLP models like BERT and RoBERTa were discussed, future work should include the development of semantic validation tools to ensure that generated questions are not only syntactically valid but also contextually meaningful.
- **Reinforcement Learning Integration:** Enabling bots to improve based on performance history through reinforcement learning would increase competitiveness and realism.
- **Visualization and Replay Tools:** A dashboard for monitoring matches and analyzing results, along with the ability to replay games, would improve the usability and pedagogical value of the platform.
- **Scalability and Concurrency:** Adapting the system for concurrent matches and persistent user data is crucial for deployment at scale.
- **Applicability to Other Domains:** The architecture can be adapted to other multi-agent environments such as educational games, interactive simulations, or negotiation platforms. The modular structure remains valid even if system rules change.

In summary, the foundation established in this project offers both theoretical and practical value, serving as a launchpad for continued development and experimentation in agent-based AI environments.

Chapter 7

Reflection

During this project, we learned how to analyze and design a system in a more structured way, starting with a deep understanding of the problem before thinking about any technical solutions. One of the things that stood out to us was realizing how important it is to understand how a system works, how its parts interact, and how rules are maintained—sometimes even more than implementing it right away.

As a group, we developed important skills such as organizing ideas, analyzing requirements, and working collaboratively. We didn't always agree on every decision, but we learned how to listen to each other, explain our reasoning, and make decisions that considered everyone's input. We also discovered the value of using a modular architecture because it helped us better visualize how the system works and spot potential issues early on.

One of the biggest challenges we faced was setting clear boundaries for the project. There were moments when we wanted to go beyond what was realistic in terms of time and scope. However, this helped us learn to prioritize what was essential and focus on what was feasible. If we were to start this project again, we would probably spend more time upfront organizing our tasks and defining how each part of the system connects with the others.

This experience also made us appreciate the importance of documenting our decisions—not just to meet the requirements of the report, but because it helped us reflect on what we were doing and make changes when something wasn't working as expected.

In summary, this project helped us better understand what it means to design a functional system and how good analysis can make a big difference in any future development. We're taking away valuable lessons that we know will be useful in both academic and professional projects going forward.

References

Bass, L., Clements, P. and Kazman, R. (2012), *Software Architecture in Practice*, 3 edn, Addison-Wesley.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Édouard Duchesnay (2011), 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research* **12**, 2825–2830.

Russell, S. and Norvig, P. (2020), *Artificial Intelligence: A Modern Approach*, 4 edn, Pearson.

Vinyals, O., Babuschkin, I., Chung, J., Denil, M., Apps, C. and et al. (2019), 'Alphastar: Mastering the real-time strategy game starcraft ii', *DeepMind Blog* . https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

Walton, D. (2006), *Fundamentals of Critical Argumentation*, Cambridge University Press.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q. and Rush, A. M. (2020), 'Transformers: State-of-the-art natural language processing', *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* pp. 38–45.

Wooldridge, M. (2009), *An Introduction to MultiAgent Systems*, John Wiley & Sons.