Bulls and Cows Game - Technical Documentation

Overview

This document provides technical documentation for two implementations of the Bulls and Cows game using information theory principles. The project consists of two main Python files: Bulls_and_Cows_User_Guessing.py and Bulls__and_Cows_Computer_Guessing.py.

1. User Game Implementation (Bulls_and_Cows_User_Guessing.py)

Purpose

Implements a game where the computer generates a secret code and the user attempts to guess it, with entropy tracking to measure remaining uncertainty.

Module Structure

1.1 Functions

compute_feedback(secret: str, guess: str) -> Tuple[int, int]

Calculates the feedback for a user's guess.

• Parameters:

  - secret: The secret 4-digit code

  - guess: The user's guess

• Returns: Tuple containing (bulls, cows)

• Algorithm:

  - Counts exact matches for bulls

  - Counts matching digits in wrong positions for cows

  - Uses set operations to avoid double counting

calculate_current_entropy(bulls: int, cows: int) -> float

Calculates remaining uncertainty after a guess.

• Parameters:

  - bulls: Number of correct digits in correct positions

- cows: Number of correct digits in wrong positions

• Returns: Current entropy in bits

• Algorithm:

- Base entropy = 4 * log2(10) ≈ 13.29 bits
- Position certainty = bulls/4.0
- Digit certainty = cows/8.0
- Remaining uncertainty = 1 - (position_certainty + digit_certainty)
- Current entropy = base_entropy * remaining_uncertainty

is_valid_guess(guess: str) -> bool

Validates user input.

• Parameters:

- guess: User's input string

• Returns: Boolean indicating if guess is valid

• Validation Rules:

- Must be 4 characters long
- Must contain only digits
- Must have unique digits

Game Flow

1. Computer generates random 4-digit code
2. User makes guesses with feedback after each attempt
3. Entropy is calculated and displayed after each guess
4. Game ends on correct guess or after 20 attempts

2. Computer Solver Implementation (Bulls_and_Cows_Computer_Guessing.py)

Purpose

Implements an optimal solver where the computer guesses the user's secret code using information theory for decision making.

Module Structure

2.1 Functions

compute_feedback(secret: str, guess: str) -> Tuple[int, int]

Calculates feedback for a guess.

• Parameters:

- secret: Current code being evaluated
- guess: Computer's guess

• Returns: Tuple containing (bulls, cows)

• Algorithm:

- Uses marking strategy to handle digit matching
- Marks matched positions to avoid double counting
- Separates bulls and cows calculation

filter_codes(possible_codes: List[str], guess: str, bulls: int, cows: int) -> List[str]

Filters possible codes based on feedback.

• Parameters:

- possible_codes: List of remaining possible codes
- guess: Computer's last guess
- bulls, cows: Feedback received

• Returns: Filtered list of possible codes

• Process:

- Eliminates codes that wouldn't give same feedback
- Maintains consistency with all previous feedback

calculate_entropy(possible_codes: List[str], guess: str) -> float

Calculates expected information gain for a guess.

• Parameters:

- possible_codes: Current list of possible codes
- guess: Potential guess to evaluate

• Returns: Expected information gain in bits

• Algorithm:

- Uses Shannon entropy formula: $-\Sigma P(x) * \log_2(P(x))$
- Calculates probability distribution of possible feedback
- Higher entropy indicates better guesses

find_best_guess(possible_codes: List[str], all_codes: List[str]) -> Tuple[str, float]

Determines optimal next guess.

• Parameters:
  - possible_codes: Current list of possible codes
  - all_codes: List of all valid guesses

• Returns: Tuple of (best_guess, expected_entropy)

• Optimization:
  - Evaluates entropy for each possible guess
  - Includes early stopping when maximum entropy is found
  - Considers all valid codes as potential guesses

User solving Game Strategy

1. Maintains set of possible codes
2. For each guess:
   - Calculates entropy for potential guesses
   - Selects guess with maximum expected information gain
   - Updates possible codes based on feedback
3. Continues until unique solution is found

Technical Details

Information Theory Application

User Game

• Initial entropy = $4 * \log2(10) \approx 13.29$ bits

• Entropy reduction based on feedback

• Linear uncertainty reduction model

Computer Solving Game Strategy

• Uses Shannon entropy for decision making

• Probability-based information gain calculation

• Optimal guess selection through entropy maximization

Performance Characteristics

• User game: Fixed memory usage, O(1) feedback calculation

• Computer solver:

  - Space complexity: O(n) where n is number of possible codes
  - Time complexity per guess: O(n * m) where m is number of all possible guesses
  - Typically solves in 5-6 guesses

Error Handling

• Input validation for user guesses

• Feedback validation

• Detection of contradictory feedback

• Graceful handling of game termination