Documentation for Bulls and Cows Game with Entropy and Information Gain

Project Overview

This project implements the **Bulls and Cows** game using **Python** with an added layer of **entropy** and **information gain** calculations. The player guesses a secret 4-digit number, and after each guess, the game provides feedback on how many digits are correct and in the correct position (bulls) and how many digits are correct but in the wrong position (cows). The game also calculates and displays the **entropy** and **information gain** with each guess, providing players with real-time insights into how much uncertainty remains about the secret number.

Features

- **Game Logic**: Standard Bulls and Cows game mechanics, with feedback on bulls and cows.
- **Entropy Calculation**: Measures the uncertainty of remaining possibilities after each guess.
- **Information Gain**: Quantifies the reduction in uncertainty after each guess.
- **Interactive User Interface**: Built using **Streamlit**, where players input their guesses and receive immediate feedback.
- **Visualization**: Entropy and information gain are visualized on line graphs, and bulls and cows are displayed using LiquidFill charts.

File paths:
for presentation→COWS_BULLS_ENTROPY.pptx

for presentation_video→Video_Presentation.mp4
for video_demonstration of game->GAME_PLAY.mp4
for code->WORKING_GAME.py

For git->https://github.com/SainathChettupally/MFDS_EXAM_3.git

Note:

Install all the packages mentioned in requirements.txt found in this folder
link to view

- **streamlit==1.19.0**
- **plotly==5.0.0**
- **pandas==1.5.3**
- **numpy==1.23.5**
- **sympy==1.11.1**
- **streamlit-echarts==0.2.0**

In order to run the code you need to use the following command in your terminal(open terminal in the same file path where you have all the files as they have dependencies)
streamlit run WORKING_GAME.py

Code Overview

Imports and Libraries

```
import streamlit as st
import random
import itertools
import math
import pandas as pd
import plotly.express as px
from streamlit_echarts import st_echarts
import numpy as np
from sympy.stats.rv import probability
```

- **Streamlit**: Used for creating the interactive user interface.
- **random**: For generating random secret numbers.
- **itertools**: To generate all possible 4-digit combinations.
- **math**: For logarithmic calculations required in entropy.
- **pandas**: Used for data manipulation.
- **plotly.express & streamlit_echarts**: For visualization of entropy and information gain.
- **numpy**: For mathematical operations and calculations.

Functions

1. start_new_game()

- **Purpose**: Initializes the game by generating a random 4-digit secret number, setting up history, and defining possible guesses.

- **Parameters**: None
- **Returns**: None

```
# Function to start a new game
def start_new_game():  2 usages
    st.session_state.secret_number = ''.join(random.sample( population: '0123456789',  k: 4))  #Generating a random 4 digit number
    st.session_state.history = []  # Resetting  history
    st.session_state.attempts = 0  # Attempts Counter being reset
    st.session_state.possibilities = [''.join(p) for p in
                                      itertools.permutations( iterable: '0123456789',  r: 4)]  # All 4-digit possibilities
    st.session_state.game_over = False  # Game status flag
    st.session_state.entropy_values = []  # Track entropy values for plotting
```

2. calculate_bulls_and_cows(secret, guess)

- **Purpose**: Compares the player's guess with the secret number and returns the number of bulls and cows.
- **Parameters**:
    - secret: The secret 4-digit number.
    - guess: The player's 4-digit guess.
- **Returns**: A tuple (bulls, cows) representing the number of bulls and cows.

```
# Function to calculate bulls and cows
def calculate_bulls_and_cows(secret, guess):  2 usages
    bulls = sum(1 for i in range(4) if secret[i] == guess[i])
    cows = sum(1 for i in range(4) if secret[i] != guess[i] and guess[i] in secret)
    return bulls, cows
```

3. filter_possibilities(possibilities, guess, bulls, cows)

- **Purpose**: Filters the list of remaining possible secret numbers based on the bulls and cows feedback from the player's current guess.
- **Parameters**:
    - possibilities: A list of remaining possible secret numbers.
    - guess: The current guess.
    - bulls: The number of bulls in the current guess.
    - cows: The number of cows in the current guess.
- **Returns**: A list of remaining valid possibilities.

```python
def filter_possibilities(possibilities, guess, bulls, cows):  1 usage

    valid_possibilities = []

    # Loop through each possibility
    for p in possibilities:
        calculated_bulls, calculated_cows = calculate_bulls_and_cows(p, guess)

        # Check if the possibility gives the same bulls and cows as the current guess
        if calculated_bulls == bulls and calculated_cows == cows:
            valid_possibilities.append(p)

    return valid_possibilities
```

4. calculate_entropy(possibilities, guess, bulls, cows)

- **Purpose**: Calculates the entropy based on the remaining possible secret numbers. Our Entropy calculation does not determine how close our guess to the secret number but based on the remaining possibilities which can potentially have same or more number of cows and bulls sent from the filter_possibilities function.
- **Parameters**:
    - possibilities: A list of remaining possible secret numbers.
    - guess: The current guess.
    - bulls: The number of bulls in the current guess.
    - cows: The number of cows in the current guess.
- **Returns**: A float representing the calculated entropy.

```python
#Function to calculate entropy
def calculate_entropy(possibilities):  2 usages
    total = len(possibilities)
    if total == 0:
        return 0

    # Probabilities for each possible outcome (assuming uniform distribution)
    probabilities = [1 / total] * total  # Every remaining possibility is equally likely

    # Calculate entropy using the formula H(X) = -Σ P(x) log2(P(x))
    entropy = -sum(p * math.log2(p) for p in probabilities if p > 0)

    return entropy
```

5. display_graph()

- **Purpose**: Displays the line graph for entropy and information gain over time using **Plotly** and **ECharts**.
- **Parameters**: None
- **Returns**: None

```python
def display_graph():  1 usage
    # Ensure entropy_values and information_gain are stored in session state
    if "entropy_values" not in st.session_state:
        st.session_state.entropy_values = []

    if "information_gain_values" not in st.session_state:
        st.session_state.information_gain_values = []

    # Check if the entropy and information gain values are valid
    if any(isnan(x) for x in st.session_state.entropy_values + st.session_state.information_gain_values):
        st.error("Error: NaN values found in the entropy or information gain data!")
        return

    # Create x-axis data for each guess attempt
    x_data = [f"Guess {i}" for i in range(1, len(st.session_state.entropy_values) + 1)]

    # Chart options for entropy and information gain
    options = {
        "title": {"text": "Entropy and Information Gain Over Time"},
        "tooltip": {"trigger": "axis"},
        "legend": {
            "data": ["Entropy", "Information Gain"],
            "top": "10%",
            # Adjust the top position of the legend (you can set it to a percentage of the height or a fixed value)
            "right": "5%",  # You can set the right position to move it further from the left side
            "left": "auto",  # This will allow the right positioning to work automatically
            "orient": "horizontal",  # If you want the legend to be horizontal, you can use "horizontal"
        },
        },
        "yAxis": {"type": "value"},
        "series": [
            {
                "name": "Entropy",
                "type": "line",
                "data": st.session_state.entropy_values,
                "smooth": True,
                "lineStyle": {"color": "blue"},
            },
            {
                "name": "Information Gain",
                "type": "line",
                "data": st.session_state.information_gain_values,
                "smooth": True,
                "lineStyle": {"color": "green"},
            },
        ],
    }

    # Render the graph
    st_echarts(options=options, height="400px")
```

6. display_attempt_history()

Purpose:
Displays the history of the player's previous guesses, showing the number of bulls and cows for each attempt.

Parameters:

- **None** (Uses session state to track guesses).

Returns:

- **None** (Directly displays the attempt history on the UI).

```python
# Function to display attempt history with images as headers
def display_attempt_history():  1 usage
    with st.sidebar:
        st.subheader("Attempt History:")

        # Display images as headers with counts below them
        bulls_image = r"C:\Users\csain\Downloads\bull.png"
        cows_image = r"C:\Users\csain\Downloads\cow.png"
        col1, col2 = st.columns(2)
        with col1:
            st.image(bulls_image, use_container_width=True)
        with col2:
            st.image(cows_image, use_container_width=True)
        # Loop through history to display each attempt
        for attempt, bulls, cows in st.session_state.history:
            col1, col2 = st.columns(2)
            with col1:
                #st.image(bulls_image, use_container_width=True)
                st.markdown(f"**{attempt}:** {bulls} Bulls")
            with col2:
                #st.image(cows_image, use_container_width=True)
                st.markdown(f"**{attempt}:** {cows} Cows")
```

7. display_bulls_and_cows()

Purpose:

Visualizes the bulls and cows feedback using **LiquidFill** charts, one for bulls (green) and one for cows (yellow).

Parameters:

- **bulls** (int): Number of bulls.
- **cows** (int): Number of cows.

Returns:

- **None** (Displays the visualizations on the UI).

```
# Function to display bulls and cows using LiquidFill chart (side by side)
def display_bulls_and_cows(bulls, cows):  1 usage
    bulls_water_level = bulls / 4  # Max bulls = 4
    cows_water_level = cows / 4  # Max cows = 4

    # ECharts LiquidFill chart for Bulls
    bulls_option = {
        "series": [{
            "type": "liquidFill",
            "data": [bulls_water_level],
            "color": ["#28a745"],  # Green color for bulls
            "label": {
                "show": True,
                "position": "inside",
                "fontSize": 24,
                "color": "#fff",
                "formatter": f"Bulls: {bulls}",
            }
        }]
    }

    # ECharts LiquidFill chart for Cows
```

```
# ECharts LiquidFill chart for Cows
cows_option = {
    "series": [{
        "type": "liquidFill",
        "data": [cows_water_level],
        "color": ["#ffc107"],  # Yellow color for cows
        "label": {
            "show": True,
            "position": "inside",
            "fontSize": 24,
            "color": "#fff",
            "formatter": f"Cows: {cows}",
        }
    }]
}

# Display the liquid fill charts for bulls and cows side by side
col1, col2 = st.columns(2)
with col1:
    st_echarts(options=bulls_option, height="300px")
with col2:
    st_echarts(options=cows_option, height="300px")
```

8. play_game()

- **Purpose**: Runs the main game loop, allowing the player to input guesses and receive feedback.
- **Parameters**: None
- **Returns**: None

```python
def play_game():  1 usage
    # If the game is not started, initialize the session state
    if "secret_number" not in st.session_state:
        start_new_game()

    # Initialize session state variables if not already initialized
    if "entropy_values" not in st.session_state:
        st.session_state.entropy_values = []

    if "information_gain_values" not in st.session_state:
        st.session_state.information_gain_values = []

    st.title('BULLS AND COWS GAME')

    # Main Game UI
    guess = st.text_input("Enter your guess: ", max_chars=4)

    if guess:
        # Validate the guess
        if len(guess) == 4 and guess.isdigit() and len(set(guess)) == 4:
            print(guess)
            bulls, cows = calculate_bulls_and_cows(st.session_state.secret_number, guess)
            st.session_state.attempts += 1

            # Check if the guess already exists in history
            guess_exists = any(existing_attempt[0] == guess for existing_attempt in st.session_state.history)

            # Add the guess to the history only if it doesn't exist already
            if not guess_exists:
                st.session_state.history.append((guess, bulls, cows))  # Store the guess and the results
            display_attempt_history()
            display_bulls_and_cows(bulls, cows)

            # Filter possibilities and calculate entropy
            # Filter possibilities and calculate entropy
            st.session_state.possibilities = filter_possibilities(st.session_state.possibilities, guess, bulls, cows)

            # Calculate entropy with proximity adjustment
            entropy = calculate_entropy(st.session_state.possibilities)

            # Track entropy values for plotting and entropy change
            if len(st.session_state.entropy_values) > 0:
                previous_entropy = st.session_state.entropy_values[-1]
                entropy_change = entropy - previous_entropy  # Change from the last guess
            else:
                entropy_change = 0  # For the first guess, there's no change

            # Store current entropy value
            st.session_state.entropy_values.append(entropy)

            initial_entropy = calculate_entropy([''.join(p) for p in itertools.permutations( iterable: '0123456789',  r: 4)])
            information_gain = initial_entropy - entropy
            st.session_state.information_gain_values.append(information_gain)

            # Display the graph with entropy and information gain
            display_graph()
```

```python
# Display additional calculations
initial_possibilities_count = 5040  # Total number of 4-digit permutations without repetitions (10P4)

# Check if possibilities are available (non-empty) before calculating the probability
if len(st.session_state.possibilities) > 0:
    # Calculate probability of finding the secret number
    probability_of_finding = 1 / len(st.session_state.possibilities)
    probability_percentage = probability_of_finding
else:
    # If no possibilities left, it's the correct guess
    probability_percentage = 1.00 if bulls == 4 else 0.0
intial_probability=0
st.write(f"Probability of Finding the Secret Number: {probability_percentage:.6f}")
current_probability=probability_percentage
probability_change=current_probability-intial_probability
# Use columns for displaying metrics
col1, col2, col3 = st.columns(3)

with col2:
    # Show entropy change in delta format (Green for Gain, Red for Loss)
    if entropy_change < 0:  # Entropy has decreased (information gained)
        st.metric(label="Entropy", value=f"{abs(entropy):.2f}",
                  delta=f"-{abs(entropy_change):.2f}", delta_color="normal")
    elif entropy_change > 0:  # Entropy has increased (information lost)
        st.metric(label="Entropy", value=f"{abs(entropy):.2f}",
                  delta=f"+{abs(entropy_change):.2f}", delta_color="normal")
    else:  # No change in entropy
        st.metric(label="Entropy", value=f"{abs(entropy):.2f}", delta="0.00", delta_color="off")
```

```python
        with col3:
            # Show probability of finding the secret number as a metric
            st.metric(label="Probability of Finding Secret", value=f"{probability_percentage:.6f}", delta=f"+{abs(probability_change):.2f}",
                      delta_color="normal")

        # Additional metrics as before
        st.write(f"Information Gain: {information_gain:.2f}")

        avg_bulls = sum(bulls for _, bulls, _ in st.session_state.history) / len(st.session_state.history) if len(
            st.session_state.history) > 0 else 0
        avg_cows = sum(cows for _, _, cows in st.session_state.history) / len(st.session_state.history) if len(
            st.session_state.history) > 0 else 0
        st.write(f"Average Bulls per Attempt: {avg_bulls:.2f}")
        st.write(f"Average Cows per Attempt: {avg_cows:.2f}")

        # Check if the user guessed the correct number
        if bulls == 4:
            st.balloons()
            st.success(f"Congratulations! You guessed the number in {st.session_state.attempts} attempts.")
            st.session_state.game_over = True
    else:
        st.error("Invalid guess. Please enter a 4-digit number with no duplicate digits.")
```