Text Puzzle Game Development using LLMs

Objective: Design and develop a capstone project on "Text Puzzle Game Development using LLMs" that explores the use of large language models to create engaging, dynamic text-based puzzle games.

This guide details suitable open-source models, step-by-step game development, and evaluation methods for the project.

Required Models

- **GPT-Neo/GPT-J or smaller GPT-like models**: Open-source LLMs suitable for generating text-based puzzle narratives and user interaction responses.
- **Llama 2 (7B or smaller variants)**: Effective for nuanced story progression and puzzle hint generation.
- **LangChain**: Helps orchestrate multi-turn conversations and game state management with LLM prompts.
- **RAG (Retrieval-Augmented Generation) frameworks**: For incorporating external knowledge bases into puzzle hints or clues dynamically.

Step by Step Instructions

1. Project Setup

- Choose an open-source LLM capable of generating coherent narrative text and reasoning over puzzles.
- Setup the environment with necessary packages like Transformers, LangChain, and an interface library (e.g., Flask or Streamlit for UI).
- Define the game framework to handle user inputs, story states, and puzzle progression.

2. Puzzle Design and Content Generation

- Create initial puzzle templates and storyline arcs using prompt engineering to guide the LLM in producing puzzles, clues, and narrative elements.
- Use LLM prompts to generate puzzle descriptions, valid player actions, and potential outcomes.
- Implement constraints in prompt design to maintain game balance and avoid nonsensical or unsolvable puzzles.

3. Game Loop and Interaction Handling

- Develop the core game loop to:
  - Accept user textual inputs (commands, guesses).
  - Generate dynamic responses using the LLM, updating game states accordingly.
- Maintain context across turns either through token window management or retrieving relevant state summaries.
- Integrate mechanisms for hints or retries, powered by LLM-generated clues or explanations.

4. Evaluation and Refinement

- Conduct testing sessions to evaluate gameplay smoothness, puzzle difficulty, and narrative coherence.
- Use player feedback to refine prompt templates and adjust puzzle complexity.
- Explore automated evaluation by testing multiple paths or interactions to ensure logical consistency and completeness.

Evaluation Criteria

1. Report

- **Scope**: Overview of game design, model selection, prompt engineering techniques, and system architecture.
- **Analysis**: Discuss narrative richness, puzzle diversity, user engagement, and technical challenges.
- **Metrics**: Measure player success rates, dialogue coherence scores, and prompt efficiency.

2. Presentation

- **Slides**: Present background, LLM integration methodology, game demo overview, and key insights.
- **Demo**: Showcase live gameplay illustrating typical user interactions and puzzle solving steps.
- **Highlights**: Focus on LLM-driven dynamic content creation and adaptability across different player inputs.

3. Project Code Submission

- **Repository**: Complete source code for the game engine, LLM prompt modules, and UI components.
- **Documentation**: Setup instructions, gameplay guide, prompt examples, and troubleshooting notes.
- **Examples**: Include sample puzzle scripts, user interaction logs, and model configuration files.