CS 511: Artificial Intelligence II
Project 5: LLM-Based Agent

`Agent Theseus511`

Spring 2025

1 Concept

This project implements an **LLM-based agent (LBA)** that operates in the wumpus world, leveraging the reasoning capabilities of large language models to drive decision-making. Specifically, it implements the agent function:

$$f_{\text{LBA}} : \Omega^* \to A$$

that maps the observed sequence of percepts to an action. Unlike traditional rule-based or planning agents, the LBA defers the entire decision-making process to an LLM: at each step, the accumulated percept history is encoded into a natural-language prompt, which is then submitted to the model along with a JSON specification defining a rough layout for the response. This capability is provided by the API used in the implementation. The large language model's JSON response is parsed to select the next action. The prompt engineering startegy and the analyses of the case studies are discussed in the subsequent sections.

2 Getting Started

The LLM-based agent is implemented in Scala. The source code is contained in the files `src/scala/AgentFunctionImpl.scala` and `src/scala/LLMBasedAgent.scala`. The project directory contains a Makefile that automates building and running the LBA. The Makefile runs the project with the options `forwardProbability (-n)` set to 1 and `randomAgentLoc (-r)` set to `false`.

Successfully running the LBA requires the environment variable `GOOGLE_API_KEY` to be set to a valid API key.

The Makefile contains a `check` target that checks the system for the necessary tools (`scala`, `java`). It is recommended that the project is run after checking for the necessary tools as:-

```
$ make check
$ make #or "make run"
```

The above commands are for a single run by default. Of course, the `run` recipe can be updated with the `-t` option for multiple trials. A separate `make` target called `tenk` is provided for evaluating the LBA that runs 10,000 (or how many ever) trials. It can be run using:-

```
$ make tenk
```

The score for each trial and the average score is written to `wumpus_out.txt` or to the output file you specify using the `-f` option in the recipe.

The project was tested using:-

- **Scala Version:** 3.6.4
- **Java Version:** OpenJDK 22.0.1

3 Prompt Engineering

The prompting strategy used by the LBA is simple, although it can be made more sophisticated for fine-tuning the LLM. The LBA defines a *priming message*, M_0, that contains information about the environment and its rules. This message reads:

> ‘You are an intelligent agent in the Wumpus World who navigates it to maximize the score. The Wumpus World is a 4x4 grid world indexed by (x, y) from (1, 1) to (4, 4). You are currently in (1, 1), facing East, and your aim is to navigate and find the square with the gold. It’s not that easy because there is a wumpus in one of the squares, and pits in exactly two of the squares. So, basically, 4 out of the 16 squares are “special” since they contains the gold, wumpus, and the pits. The pits are necessarily in different squares. A wumpus and pit can be colocated, or a gold and a pit, or the wumpus and a pit, or the wumpus, a pit, and the gold can all be colocated. You start exploring from (1, 1) so that square is necessarily safe and does not contain the wumpus or the pit. The difficulty is that you don’t know which square contains what (wumpus, gold, pit, any two, all three, or nothing). The only information you have is an “observation” at every time step that given you information about the environment. If you observe a stench in the square you are in, the wumpus is in a square adjacent to yours. Similarly, if you observe a breeze, then at least one of the adjacent squares contains a pit. If you observe a glitter, then you are colocated with the gold. If you enter a square with a wumpus or a pit, you DIE. You win if you “find” the gold i.e. execute the GRAB action when you are colocated with it. To make things easier, you are equipped with a bow and arrow and you have only one arrow in your quiver. When you shoot, the arrow spans the entire row or column in front of you depending on where you’re facing. If the wumpus happens to be in the line of fire, you will hear a scream which will give you information if the wumpus is dead. Try and find the gold based on the information you get, and please don’t die! At each time step, choose one action to execute out of go forward, turn left, turn right, shoot, grab, or do absolutely nothing. Each action gives you a score. The actions go forward, turn left, turn right, and grab award you -1. The shoot action awards you -10, and doing nothing awards you 0. Dying awards you -1000, and grabbing the gold awards you +1000. Your job as an intelligent agent is to maximize the score. Ask me what you observe after each action.’

At every time step, t, the LLM is primed with a prompt:

$$M_t := M_0 + m_o^{(1)} + m_a^{(1)} + m_o^{(2)} + m_a^{(2)} + \ldots + m_o^{(t)},$$

where the string $m_o^{(i)}$ summarizes the agent’s observations at time step i in natural language, the string $m_a^{(j)}$ reminds the LLM of its action response at time step j, and “+” represents the concatenation of all these strings. In effect, at every time step, t, the LBA primes the LLM with a description of the environment and its rules prepended to a natural language rendition of the **history**, h_t, until that time step. The history is an alternating sequence of observations and actions observed and executed by the agent.

$$h_t = \langle o_1, a_1, o_2, a_2, \ldots, o_t \rangle$$

The priming message is prepended to a prompt, p, that enumerates the different possible actions, and asks the LLM to choose one. This prompt reads:

> “Choose one action to execute out of go forward, turn left, turn right, shoot, grab, or do nothing.”

Hence, at every time step, t, the LBA prompts the LLM with the message $M_t + p$ and defers the decision-making *completely* to the LLM. The LBA itself stores no internal state, or does no planning. It acts as a conduit between the environment and the LLM. The LBA expects a JSON response from the LLM with the keys `best_action`, `agent_position_after_action`, `agent_orientation_after_action`, `belief_state_after_action`, and `score_after_action`. The capability to implicitly provide a JSON specification for the response is provided by the **openai-scala-client** API, which is used in the implementation of the LBA. Even at the point when the response is received from the LLM, the LBA does no kind of validation on the different values, but simply extracts the `best_action`, executes it, and prints out what the LLM considers to be the current belief state.

4 Case Studies

The two LLMs I chose to test are Google's **Gemini 2.0 Flash** model and OpenAI's **o4-mini-high**.

4.1 Gemini 2.0 Flash

The LBA implementation queries the Gemini 2.0 Flash model using Google's API. The widely observed result was that this LLM often hallucinates with incorrect values even for simple inferences such as `agent_position_after_action` and `agent_orientation_after_action`. Although the values themselves are incorrect, the LLM generally gets the format right; for instance, it will respond with a 2-tuple of integers between 1 through 4 for `agent_position_after_action`, a value out of North, South, East, or West for `agent_orientation_after_action`, and a string in natural language that resembles a belief state but is grossly logically incorrect. These findings suggest that planning is an unrealistic expectation for the LLM. The LLM seems to have no internal state whatsoever, and an incoherence in its own reasoning and choice of action is often observed. For instance, the LLM will provide some reasoning for going forward, but the action it would give is a turn. This behavior is in favor of the argument that LLMs only piece together language and are incapable of planning with a coherent understanding of the world. However, every time that the LLM encountered a glitter, it chose to `GRAB` with a reasoning that grabbing the gold would award 1000 points.

The LLM-based agent with a Gemini 2.0 Flash model was run 32 times and an average score of **-485.03** was achieved. The following are the summary statistics:-

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. dev.	Mode
-1046	-1010	-1002.5	-485.03	-49.75	1000	801.5975	-1000

The LLM got lucky whenever it found the gold in a square other than $(1, 1)$ because the reasoning until then was evidently hallucinated. The only consistent behavior, which could be characterized as *reasoned* or *rational* that the LLM showed was to execute `GRAB` when a `glitter` is observed.

4.2 o4-mini-high

OpenAI claims its o4-mini-high model to be the best at reasoning and coding, and this project explored this aspect of the LLM through a manual back-and-forth with it instead of code integration and simulation because of API restrictions. The exchange can be viewed here. The o4-mini-high model is posed with the problem of exploring the wumpus world for 4-5 different configurations, each of which are deliberately chosen because they offer a trickiness in inferring dangers and require the agent to find a roundabout path to the gold. The o4-mini-high model performs extremely well. It strongly suggests that it stores an internal state of the world, and it uses precise logical inference based on the observations to locate the dangers and avoid them—much like a model-based reflex agent. Its abilities go as far as precisely shooting the wumpus once its exact position has been inferred, giving up and concluding that the gold is colocated with the pit after the LLM has explored all possible safe squares, and back-tracking from `breeze`-squares if the position of the pit cannot be precisely inferred. This truly suggests that the LLM is doing more than just piecing language together such as computing a logical calculus under the hood. When queried with this question, the o4-mini-high model answers that it encodes the environment within its neural network, which enables it to perform precise logical inference. However, beyond these extremely impressive abilities for an LLM, the o4-mini-high model still lacks in constructing belief states and planning using probabilities. At some points during the conversation, the LLM is explicitly nudged to calculate probabilities and keep track of a belief state, which it evidently tries to do for the next 2-3 time steps, although somewhat inaccurately, before falling back on an approach based on rules and logical inference. The LLM is good at doing what it is told to do and its logical inference improves if it is given hints or feedback with every prompt about what a better strategy would be or why what it's doing is suboptimal. Of course, formulating hints with the prompt would take away from complete reliance on the LLM, although incorporating feedbacks that guide the LLM's reasoning in the right direction would be the way to go. The o4-mini-high model also shows capabilities to "learn" from mistakes and not repeat the same mistakes if the same configuration is given. Overall, the model shows outstanding capabilities of navigating the wumpus world. It is much closer to a model-based reflex agent than a planning agent, but with carefully engineered prompts that incorporate some kind of reinforcement, the sky is the limit for the o4-mini-high in exploring the wumpus world.