Alexander Santagata, Nora Cleary, Era Kalaja - Team **GaborLLM**

Contributions: As a team we did pair programming and used VSCode liveshare to actively work on the code together. All code was written together with the entire group participating fully.

To compile:
mvn compile assembly:single

To run:
cs4341-referee laskermorris -p1 "java -jar target/demo-1.0-SNAPSHOT-jar-with-dependencies.jar" -p2 "java -jar target/demo-1.0-SNAPSHOT-jar-with-dependencies.jar" --visual -t60

Note on compiling/running: When running from the terminal, make sure these commands are run inside of the ‘demo’ folder.

System description: Our system implements Gemini API with a developer key and the gemini library packages. We use maven to import the required dependencies into our java project. We did not change too much of our game code, our main methodology was replacing our minimax function and evaluation function with the LLM Gemini call. We prompted the LLM by telling Gemini the current state of the board, the count of pieces in each player's hand, and the rules of Lasker Morris. We parsed its responses using a regular expression which expected moves of the exact format we use (also with parentheses to distinguish them from context). We handled invalid moves by prompting it with why its invalid move was incorrect and that it needs to try a different position on the board. To prevent Gemini from going in a long loop of suggesting invalid moves and getting a new prompt, we use a fallback plan for when it has 3 invalid move suggestions in a row.

Prompt: Our prompt consists of three sections.

-Prompt section 1: Explanation
You will be given the state of a game of Lasker Morris, following standard Lasker Morris rules. In Lasker Morris, Player 1 and Player 2 both start with 10 pieces in their hand. On alternating turns, they perform one of two actions:
- Placing a new piece from their hand on a point on the board
- Moving one of their existing pieces to an adjacent empty point on the board
Both players want to create 'mills', or three of THEIR pieces on legal board points on the same row or column (not diagonals). Mills are specifically three of the same player's pieces. If you create a mill, you are allowed to remove an opponent's piece on that turn. (You may only remove an opponent's piece that is NOT part of a mill, unless all opponent pieces are part of a mill, in which case any enemy piece can be removed.)
As such, both players want to block their opponent from forming mills. For instance, if one player has claimed g1 and g7, the opponent should claim g4 in order to block that mill.

When a player's hand is empty and they have only three pieces left on the board, they are now flying! This gives them the ability to move their pieces to any empty board point, not just adjacent ones.
A player loses when their hand is empty and they have only two pieces left on the board. Thus a player wants to create mills in order to remove their opponent's pieces from the board.

The board is given as a seven-by-seven grid where each valid point in the game is represented as '.' if empty, '1' if holding one of Player 1's pieces, and '2' if holding one of Player 2's pieces. The complete list of valid board points is: a1, a4, a7, b2, b4, b6, c3, c4, c5, d1, d2, d3, d5, d6, d7, e3, e4, e5, f2, f4, f6, g1, g4, g7. On this board, coordinates are given in the format 'a4' where 'a' refers to the row and '4' refers to the column. Here is an example board:

```
.          1 2 3 4 5 6 7
row a: 2          .          1
row b:   .   .   .
row c:  . . .
row d: 2 . .   . . .
row e:  . . .
row f:   1   .   .
row g: .          .          .
```

Here, Player 1 has pieces at a7 and f2, while Player 2 has pieces at a1 and d1.

In this game, moves are stored in the form: (SOURCE DESTINATION REMOVAL). A piece is moved from the SOURCE to the DESTINATION, and the piece marked REMOVAL is removed.
- The SOURCE is either the coordinate of a point on the board, or 'h1' (referring to Player 1's hand) or 'h2' (referring to Player 2's hand.)
- The DESTINATION is the coordinate of a point on the board.
- The REMOVAL is either the coordinate of a point on the board (if the point on the board is occupied by an opponent's piece) or 'r0' if no piece will be removed.

Here are some example moves:
- (h1 a4 r0): a piece is moved from h1 (Player 1's hand) to the board point a4, and no piece is removed.
- (h2 f2 c3): a piece is moved from h2 (Player 2's hand) to the board point f2, and the piece at board point c3 is removed.
- (d3 d2 r0): a piece is moved from the board point d3 to the board point d2, and no piece is removed.

Remember to always include your move in your response using the format: (b4 b6 r0)

-Prompt section 2: **State of game** (example)

```
.          1 2 3 4 5 6 7
Row a: 2          .          1
Row b:   .   .   .
```

Row c: . . .
Row d: . . . . . .
Row e: . . .
Row f: . . .
Row g: . . .

Player 1 pieces in hand: 9
Player 2 pieces in hand: 9
Player 1 pieces on the board: 1
Player 2 pieces on the board: 1
Player 1 pieces discarded: 0
Player 2 pieces discarded: 0

Please provide the optimal legal move for Player 1 given these conditions.

-Prompt section 3: Corrections (example)

A previous response to this prompt, (a4 a1 r0), was invalid for the following reason: a4 has a blue piece, which you are not permitted to move!

A previous response to this prompt, (d3 d2 e5), was invalid for the following reason: You cannot remove a piece if you have not created a mill!

-Prompt explanation:

Initially, we tried to just give Gemini an explanation of our syntax, the **state of game** section and a basic prompt like "Using standard Lasker Morris rules, please choose the best move for Player 2." However we found that he would forget the explanation as soon as he received the state of the game section, and his response would be somewhat incoherent. Once we realized Gemini has no memory, we changed our prompt to include the entire **explanation** section. This sorted out the larger issues, but sometimes Gemini would return the same incorrect move over and over, so we added the **corrections** section which was based on error messages we had written in our existing isMoveValid function. All three of these components are used in our prompts.

Results: To test our program we ran it against us and a copy of itself with the referee. Most games resulted in ties, which makes sense since GaborLLM is at the same skill level as itself. GaborLLM is not as skilled as human players or our Minimax player Gabor. One of the main skills that GaborLLM fails to do is to accurately notice and block mills from its opponent. GaborLLM does not play very competitively either, and often comes very close to making a mill and then does not do it. While disappointing, these results are generally expected, as Gemini API was made to parse and reproduce linguistic content, and a text-based representation of Lasker Morris is a pretty insignificant edge case for that goal. This shows that a model trained to perform a specific task will generally outperform a general "jack-of-all-trades"-type model such

as Gemini. Since optimal play at Lasker Morris is not the task Gemini is designed to do, it is unsurprisingly not very good at this.