Prolog: Exploring the Implementation of Mancala

AREN MELKON, MAXIMUS GHOSH, AND YOUSEF HAZOYEN

Abstract: Mancala is a classic board game enjoyed by many around the world. Our Prolog program gets this multiplayer game in Prolog coding language using various techniques. In addition to this, AI was developed using MiniMax search with a depth of 5, and other algorithms to pose a challenge. Two AIs vary in difficulty, one which is an "easy" AI that incorporates random selection, and one that uses the aforementioned MiniMax search to find the best move.

Key terms **- mancala, artificial, intelligence, game, prolog.**

I. INTRODUCTION

Mancala is a board game that utilizes marbles and pits. This game consists of players selecting one of six pits on their side with marbles in them, moving them one by one across the board, until one side is empty. The twist is that there are two additional pits named "Mancalas" to keep track of both players' scores, and when the player passes their mancala, they must deposit one marble in there before continuing. There are also special rules as if a player ends their turn landing in the mancala, the player gets to go again, and if the player lands on an empty pit on their side, they collect that marble as well as any marbles on the opponent's pit parallel to it, only if there are marbles in that pit. Once the game ends, any marbles left on a player's side are sent to their mancala. The winner is then declared depending on which player collected the greatest amount of marbles in their mancala.

II. CODING FUNCTIONS

[For snippets of code, please view the provided google slides]. There were various functions needed to translate the physical game of mancala into Prolog. Our approach consisted of identifying the core mechanics of Prolog verbally, establishing an order of priority (low-level to high-level), and then attempting to code them

We began by identifying any variables that could potentially change throughout the game, such as the name of the player(s), whose turn, as well as the game winner. This was done using Prolog's "dynamic" definer that would be used for predicates that could change. In total, we created 4

dynamic predicates; player, player_name, cpu_name, winner_name.
We then needed to code the base-interactable parts of the code. We knew that we wanted the launching of the game to both look visually appealing and functional. To do this, we coded two predicates; start & rules. Both of these predicates consist of write statements as well as input prompts (when necessary), that are formatted in a way that a user would be used to when interacting with other game interfaces.

From here, we get into the fundamental parts of the game. We decided there should be two "gamemodes", which we defined as play and play_cpu. The first predicate is for a two-player match with humans, and the latter is for a match between the user and a bot. Both of these predicates are shown when the user runs "start.". We made "play" ask each player's name respectively, and then updated the name variable for each. To save on file length, and to avoid code duplication, we use the 2nd player's input in the established "CPUname" variable, but ensure that they retain all functions as the human player would. "play_cpu" is different than "play" in a few ways, as it prompts the user for a difficulty, and will auto-generate the CPU/opponents name. The difficulty selector, which asks the user for 0 or 1, is stored as DifficultyStr and then converted into an integer and stored in the variable Difficulty. The CPU random name generator functions by calling on a defined database of names we have provided in another predicate.

No matter whether the user calls on "play" or "play_cpu", immediately following either, play will run on outputs provided from them (1,0). While it runs, it calls on initialise, a predicate that generates the 6 pits with 4 marbles each, as well as each mancala. It then calls on itself again with the now generated board, and we are now ready to play mancala.

From here, we then broke down core functions that would handle game mechanics. We started with select_and_make_move, which is a predicate that handles both selecting and making a move. This predicate needed to first check if the game was over, and upon establishing it was not, would display the board to allow the user to select and make a move.

The two predicates included in select_and_make_move include select_pit & make_move. Select_pit is a crucial predicate for the game of Mancala, as it deals with the selection of a valid pit according to the rules. It begins by

identifying the current state of the game for the player's turn.
We then start a loop with prolog's repeat function in order to allow the player to have multiple attempts to select a Pit if they pick the wrong one. We prompt the player to select a pit (1-6). We then read their input with read_pit, but take their input - 1 to adjust for computer indexing which starts at 0. We then check the number of marbles in the selected pit, proceeding if its greater than 0 but than writing a statement to redo their input if its less.

Make_move does the calculation to perform the move. It takes the selected pit, and calculates the new game state based off the original known as game state 2
We deconstruct the current player's board, and then get the number of marbles in the selected pit. We then run a check for more_turns which sees if moving marbles from the selected pit would grant another turn (inline with the rules of mancala). If so, we notify the player they get an extra turn through a write statement, and swap the player and cpu (or player 2's name) to signify the change in turn within the game logic. Then we recursively call the select_and_make_move function to allow another move within the same turn. If there is no extra turn granted, it runs a more simplified make_move predicate.

After coding select_and_make_move, we were ready to establish the main game loop. Through a combination of Ch 6. Of the textbook and various online resources, our main game loop utilizes recursion.
It calls the play predicate with the game state and weights. It then calls on the select_and_make_move predicate, transitioning from GameState0 to GameState1 (which represent moves). It then calls the switch_player predicate with the new game state (1) to then calculate game state 2). We include the "!" to ignore any past move once the move has been made as to not confuse it. It then recursively calls on itself with the new gamestates at the end to keep the game loop going.

The display board itself was quite simple after establishing all the variables for names and values. It Displays a two-player game board with pits and stores for both players, adjusting the view to simulate sitting across from each other.Shows player and CPU names at the top and bottom, indicating who's turn it is.
Uses recursion to list the game pieces in each pit and swaps player names for the next turn. We use a combination of write statements and formatting characters to ensure alignment and consistency when swapping turns.

III. AI OPPONENTS

We've developed two AI opponents, designed for different skill levels:

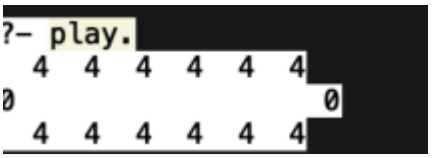
- **Easy AI:** Initiates the game with a easy approach, selecting pits randomly. This mode is perfectly suited for newcomers or those looking to play for fun.
- **Hard AI:** The well developed AI, uses a MiniMax strategy that anticipates up to five moves ahead. This approach is about carefully creating strategies and avoiding potential setbacks, offers a significant challenge to experienced players .

Our Hard AI is very complex, and this is because the implementation of a MiniMax algorithm with a search depth of five. This depth enables the AI to see potential game outcomes several moves in advance, providing players with a strong challenge. It doesn't only react to the current board state but looks ahead for future developments, requiring players to think strategically and plan their moves with care. This approach ensures that the AI can adapt to the ongoing gameplay, offering a an engaging experience that pushes even experienced Mancala fans to refine their strategies and adapt to the AI's advanced level of play.

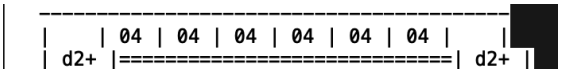
IV. ISSUES AND PROBLEMS

While there were various issues and problems when coding mancala in prolog, there are two key areas that stick out the most.

When we first began coding the display board, getting the alignment to properly display the board was quite challenging.



Our initial board was very barren, and did not visually show which player was on which side, or signify whose turn it was. Our goal was to replicate the board as close as we could within the bounds of a console.



A big issue was getting the “|” to stay consistent. Eventually, after constant iterations, the key was the utilization of the

following formatting:

```
~|~`0t~d~2+
```

Another issue we face with our code is resolving the following warnings at launch.

```
?- ['main'].
Warning: /Users/max/Desktop/PrologCP468Final/main.pl:143:
Warning:    Singleton variables: [Winner]
Warning: /Users/max/Desktop/PrologCP468Final/main.pl:148:
Warning:    Singleton variables: [P]
Warning: /Users/max/Desktop/PrologCP468Final/main.pl:148:
Warning:    Singleton variable in branch: P
Warning: /Users/max/Desktop/PrologCP468Final/main.pl:524:
Warning:    Singleton variables: [MancalasWeight,BoardsWeight]
true.
```

Since these are singleton errors, they most likely mean that where these predicates and variables are listed are unnecessary or being used incorrectly.

In a scenario where we had an extended timeframe, we should navigate to the lines specified with these warnings and ensure that we are following the logic we applied to our code. If it that is true, then we need to ensure that the context we are using the features including in the warnings is applicable.

V. OUTPUTS

The following section will be dedicated to showing the outputs of the code and the results from testing this on a group of 10 people.

How the game displays when two human players:

```
Player 1, enter your name: Aren

Player 2, enter your name: Max

   Max
   ---------------------------------------
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   | 00 |============================| 00 |
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   ---------------------------------------
   Aren <----- CURRENT PLAYER

[Select pit (1 - 6)]
|:
```

How the game displays when human vs easy bot:

```
?- play_cpu.

Enter CPU difficulty (0 for Easy, 1 for Hard): 0

Enter your name: Aren

   Mary-Beth
   ---------------------------------------
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   | 00 |============================| 00 |
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   ---------------------------------------
   Aren <----- CURRENT PLAYER

[Select pit (1 - 6)]
|:
```

Out of 10 human players, 7 of them were able to defeat the easy difficulty bot.

How the game displays when human vs hard bot:

```
?- play_cpu.

Enter CPU difficulty (0 for Easy, 1 for Hard): 1

Enter your name: Yousef

   Ian
   ---------------------------------------
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   | 00 |============================| 00 |
   |    | 04 | 04 | 04 | 04 | 04 | 04 |    |
   ---------------------------------------
   Yousef <----- CURRENT PLAYER

[Select pit (1 - 6)]
|:
```

Out of 10 human players, only 2 were able to defeat the hard difficulty bot.

How the game displays a CPU doing a turn:

```
[Select pit (1 - 6)]
|: 1.

   Yousef
   ---------------------------------------
   |     | 04 | 05 | 05 | 05 | 05 | 00 |     |
   | 00  |============================| 00  |
   |     | 04 | 04 | 04 | 04 | 04 | 04 |     |
   ---------------------------------------
   Ian <----- CURRENT PLAYER
[Opponent's turn]
Opponent selected: 5

   Ian
   ---------------------------------------
   |     | 05 | 00 | 04 | 04 | 04 | 04 |     |
   | 01  |============================| 00  |
   |     | 01 | 06 | 05 | 05 | 05 | 04 |     |
   ---------------------------------------
   Yousef <----- CURRENT PLAYER

[Select pit (1 - 6)]
|
```

When a player gets an extra turn:

```
   Max
   ---------------------------------------
   |     | 04 | 04 | 04 | 04 | 04 | 04 |     |
   | 00  |============================| 00  |
   |     | 04 | 04 | 04 | 04 | 04 | 04 |     |
   ---------------------------------------
   Aren <----- CURRENT PLAYER

[Select pit (1 - 6)]
|: 3.
Extra turn!

   Max
   ---------------------------------------
   |     | 04 | 04 | 04 | 04 | 04 | 04 |     |
   | 00  |============================| 01  |
   |     | 04 | 04 | 00 | 05 | 05 | 05 |     |
   ---------------------------------------
   Aren <----- CURRENT PLAYER

[Select pit (1 - 6)]
|
```

When a player wins:

```
Game over!

   Ian
   ---------------------------------------
   |     | 00 | 00 | 00 | 00 | 00 | 00 |     |
   | 31  |============================| 17  |
   |     | 00 | 00 | 00 | 00 | 00 | 00 |     |
   ---------------------------------------
   Yousef <----- CURRENT PLAYER
And the winner is Ian
false.

?-
```

When the two players tie:

```
Game over!     0

   Maximus
   ---------------------------------------
   |     | 00 | 00 | 00 | 00 | 00 | 00 |     |
   | 24  |============================| 24  |
   |     | 00 | 00 | 00 | 00 | 00 | 00 |     |
   ---------------------------------------
   Ghosh <----- CURRENT PLAYER
Tie
false.

?-
```

VI. CONCLUSION

In this program, we programmed a game of mancala using various techniques so that it may support multiple game modes such as player vs player and player vs CPU, with two difficulties for the CPU. Using AI, we coded an easy “random” CPU for beginners, and a harder CPU using MinMax search with a depth of 5 to find the best move. Future replications could include a way to increase the size of the board, the number of pits each opponent has, as well as a way to decide which player starts first. Overall, the program was enjoyable to create and helped demonstrate the strengths of Prolog compared to other coding languages.

References

In order to complete this project, we utilized the class provided textbook, prolog’s documentation, and various online resources which provided insight into either coding mancala, prolog algorithms, or a combination of both.

Jesus, I. (n.d.). *Ivopt - Overview*. GitHub. https://github.com/ivopt

Luger, G. F. (2009). Artificial intelligence: Structures and strategies for complex problem solving (6th ed.). Pearson.

Newest "prolog" questions. Stack Overflow. (n.d.). https://stackoverflow.com/questions/tagged/prolog

Prolog documentation. SWI. (n.d.). https://www.swi-prolog.org/pldoc/index.html