PFL Project 2

Class 11 Group Ayu_05

L.EIC 24/25

Leonor Couto up202205796@fe.up.pt

Rodrigo de Sousa up202205751@fe.up.pt

Introduction

This project focuses on the design and implementation of a board game called Ayu, using the text-mode interface to display the board and user input to move the pieces, using SICStus Prolog 4.9.

The prolog code follows the guidelines given to us, of having the predicates follow the specific signatures

The predicates were done by Leonor and Rodrigo, with a work distribution of 60/40 respectively.

Installation and Execution

On Linux and macOS, after the installation of the SICStus Prolog 4.9 and the download of the src folder, we open the terminal inside it and run this command:
“sicstus -l game.pl”. When the game loads, write the command “play.” to start.

On Windows, we open SICStus Prolog 4.9 in the working directory of the project and run:
“['src/game.pl'].” and when the file game loads, write “play.” to start.

Description of the Game

Ayu, short for "Attach Your Units", is a strategy game focused on connecting units through careful moves.

Players decide who goes first and then alternate turns. On each turn, a player must move a piece to an adjacent empty space, or relocate a piece from an existing group to an adjacent empty spot, ensuring the group remains connected after the move.

Each move must reduce the distance between the moved piece and the nearest friendly piece. The distance is measured as the shortest path of adjacent empty spaces between the two.

The game ends when a player cannot make a valid move, which usually happens when all their pieces form a single connected group. In this case, the player wins.

These rules are based on the rules on the website
https://www.mindsports.nl/index.php/arena/ayu/724-ayu-rules

Considerations for Game Extensions

The board sizes are selected at the start of the game, which can be 5, 11, 13 and 15, with 2 styles also being available. The game can also be configured to be Human vs Human, Human vs Computer or Computer vs Computer. The difficulty can also be set to either Easy or Hard. Easy Computer selects a random possible move, while Hard uses a Greedy Approach to calculate the optimal move each round. It does this by getting all the possible moves, testing one to see how many moves it allows the opponent to make, choosing the ones that maximise the amount of moves the opponent can pick from. The starting player can also be chosen at the start. Player One always plays with Crosses (X) and Player Two with Circles (O). Players can also pick the name they want to be called.

Game Logic

Game Configuration Representation

The game configuration is encapsulated in the GameState variable, which contains the board layout, current player, game type (e.g., human vs. human, human vs. computer, computer vs. computer), player names, and board style. The initial_state/2 predicate initializes this state based on values from set_up/3, creating a structured representation of the game that supports various modes and allows the game_loop/1 predicate to manage gameplay consistently.

Internal Game State Representation

The GameState variable dynamically represents the board as a list of lists, where each element denotes a cell that may be empty or occupied by a player's piece. It also tracks the current player and the game mode. This structure provides a snapshot of the game at any point, facilitating easy transitions between states. For example, the initial state has all pieces in their starting positions, intermediate states show moves in progress, and the final state represents the end of the game.

Move Representation

Moves are defined by the starting and ending coordinates of a piece on the board, ensuring that each move reflects a valid transition within the game rules. These are derived from a pre-validated list of possible moves and are processed by the move/3 predicate to update the game state by modifying the board layout to reflect the new positions of the pieces.

User Interaction

The game includes a menu system where players select the mode and other configurations through numerical inputs. Interaction is managed with robust input validation to ensure entries are within acceptable ranges, guiding players with clear prompts and feedback. For gameplay, players input moves by specifying coordinates, which are validated to ensure

legality before being applied to the game state. This design ensures smooth interaction and minimizes input errors.

Conclusions

Making a game in a logical language such as Prolog had its advantages and disadvantages. On one hand, it made the Computers and the logic of the game much easier to implement. On the other hand, we had trouble avoiding the use of the traditional If-Else Paradigm. The Hard Bot is very slow, taking a very long time to finish a game.

Bibliography

The PFL Theory Slides were consulted, as well as chatGPT, for debugging and other doubts we had.

Here are the queries:

- How to get a random element of a list in SICStus Prolog 4.9
- What does include/3 do in SICStus Prolog 4.9
- Create a get_player_names that stores text as a string in SICStus Prolog 4.9
- Logic of a BFS algorithm adapted for SICStus Prolog 4.9