Doblin

Group Members:

up202208511 Tiago Teixeira - 50%

Contributed in: game.pl, board.pl, bot.pl, evaluate.pl, gameover.pl, score.pl, README.md

up202300600 Yuka Sakai - 50%

Contributed in: game.pl, board.pl, IO.pl, menu.pl, movement.pl, view.pl, README.md

Index

- **Installation and Execution**
    - How to Open the Game
- **Description of the Game**
    - Game Rules
- **Additional Information**
- **Considerations for Game Extensions**
- **Game Logic**
    - Game Configuration Representation
    - Internal Game State Representation
    - Move Representation
    - User Interaction
- **Conclusions**
- **Bibliography**

Installation and Execution

Our game was designed to work on **SICStus Prolog 4.9**. In order to execute it, you must have the program installed along with our game files.

How to Open the Game:

To run our game you must follow the steps below:

1. Start by downloading the project's zip file and extract it to a directory of your choice.
2. Open SICStus Prolog and select the **File** option in the header menu.
3. Select **Compile...** and navigate to the project's files in the directory you chosed in **1**.
4. Click on **game.pl** and see the software's terminal loading the contents.
5. From here, you can begin the game by typing **play.** on the SICStus terminal.
6. You will be presented with a Menu where you will be able to select different game modes and difficulties according to your preferences.

menu display

Description of the Game

Brief Explanation

Our group developed Doblin in Prolog, a Tic-Tac-Toe inspired board-game for two players.
This game uses two boards, one for each player, with a set of coordinates to determine row and collumn numbers.
Coordinates in the first player's board are sequential ([a, b, c, ..., h] and [1, 2, 3, ..., 8]), while randomized in the second player's board.
As the game progresses, the boards will be filled with **O**'s and **X**'s until no further plays can be done.
The game provides different playable modes, such as:

- Human vs. Human
- Human vs. Computer
- Computer vs. Computer

Game Rules:

Doblin is a strategic two-player game involving two quadratic boards. Each player must avoid creating a line or square of four matching symbols. The game ends when both boards are full.

Game Setup

Each player has a personal 6x6 or 8x8 grid:

- Player 1's grid has ordered coordinates.
- Player 2's grid has randomized characters for rows and columns.
- At the start, both boards are empty.

How to Play

1.**Turn-Styled Structure:**

- Players take turns.
- On their turn, a player chooses an empty space on their board and places two symbols (X and O).
- The same symbol will be placed on the other player's board at the matching coordinates.
  - Example: If Player 1 plays O at (a,1), Player 2 will have an O placed at the equivalent coordinates (a,1) on their grid.
- The symbols alternate between X and O.

2.**Objective:**

- Avoid creating any group of 4 or more identical symbols in any direction (horizontal, vertical, diagonal, or a square).

3.**End of the Game:**

- The game ends when:
  - Both boards are completely filled.
- The winner is the player with the fewest amount of points, determined by how many groups of 4 of the same symbol they have on their boards.

Forbidden Patterns:

Players must avoid creating:

- A square of 4 identical symbols.
- A line of 4 identical symbols (horizontal, vertical, or diagonal).

```
        x       o o square           o
         x      o o                  o vertical
diagonal  x             x x x x      o
           x      horizontal         o
```

Additional Information:

- Official Game Website
- Official Game Rules

Game Logic

In this section, we provide an in-depth explanation of our project's code, mentioning key components and code architecture.

File Separation and Logic

For better organization and understanding, the game was divided in some files:

- **board.pl** - Contains the `initial_state` to setup the first table as "empty";
- **bot.pl** - Main for computer-side of the project;
- **evaluate.pl** - Used to analyze the possible moves, if better or not, possible or not, etc;
- **game.pl** - Main file containing `play` function to start the game;
- **gameover.pl** - Called in every `game_cycle` to check if the board is full to end the game;
- **IO.pl** - General file to validate the user input;
- **menu.pl** - Display the initial `menu` to obtain informations such as board `Size` and type of `Players`;
- **movement.pl** - Receive and validate inputs with allowed movements;
- **score.pl** - After ending, the game counts the Player1 and Player2 scores to determine who was the winner (or, if same, draw);
- **view.pl** - Auxiliar file to display the board in more elegant way;

Game Configuration Representation:

For our Doblin implementation, we decided to represent the game's configuration using three parameters:

- Size (number) :: 6 for a **6x6** board or 8 for an **8x8** board;
- Player1 (word) :: player **type** => playerX or computerX (with *X* being 1 or 2);
- Player2 (word) :: player **type** => playerX or computerX (with *X* being 1 or 2);

We collect this information from the selected choices the user makes in the main menu (`menu(Mode, Size, Player1, Player2, Player)`). With this, we are able to send the parameters to the predicate `initial_state(Size, Player1, Player2, GameState)` .

Internal Game State Representation:

The GameState was used to auxiliate in passing dynamic arguments (such as Boards, current Player, Symbol), to functions and manage the flow. Specifically it's build as:

```
GameState = game_state(ListR1, ListC1, Board1, ListR2, ListC2, Board2, Player, Symbol)
```

Where:

- ListRX - List of PlayerX's row state (shuffled for player2)
- ListCX - List of PlayerX's column state (shuffled for player2)
- BoardX - PlayerX's board
- Player - Current player of the round in `1` or `2`

- Symbol - Current symbol of the play between `x` or `o`

The main usage could be exemplified in the main game function such as choose_move and move.

Move Representation:

The moves in the game are essentially handled by the choose_move, validate_move, and move functions. The first function receives the list of Rows and Columns, the Board, and the Player level to determine the type of movement and validation required. For example, the Human Player, designated as level 0, only requires input validation to ensure the user-provided coordinates are valid and the target position is empty. Meanwhile, the Level 1 Computer Player evaluates all possible moves on the board (specifically those involving z) and randomly selects one of the available coordinates.

To achieve this, the example input a1 is split into two characters, a and 1, and converted to ASCII. This allows the program to identify the corresponding index in the Row List and Column List, making it possible to locate the same coordinate on the board. After validation, the indices are passed to the move function, which updates the current content (z) with the given symbol (x or o) and refreshes the Board.

User Interaction:

To prepare our game's matches, the user is able to navigate through the Main Menu and define the following game parameters:

1. **Game Mode**

   ```
   Select a game mode:
   [1] Player vs Player
   [2] Player vs Computer
   [3] Computer vs Computer
   ```

2. **Computer Level**

   Works for Computer Players. e.g.:

   ```
   Choose level for computer:
   [1] level 1 - Random moves
   [2] level 2 - Greedy moves  # best turn move available
   ```

3. **Board Size**

   ```
   Choose Board Size:
   [1] 6x6
   ```

```
[2] 8x8
```

4. **Player to have the First Move**

   Responsible for assigning the turn order when playing. e.g.:

   ```
   Choose who starts:
   [1] Player (Player 1)
   [2] Computer (Player 2)
   ```

To fulfill the options above, along with the player moves ingame, we designed `read_option(-Opt, +Max)` in **IO.pl**, which is used during Main Menu to select each one of the presented options.

After the setup of the Board and Players, the user interaction will be done in the input of movements in `choose_move` by `read_move(+Size, (-Row)-(-Col))` during gameplay to choose a valid move on the board.

Game Flow:

Menu - Start

In the menu states, we declare the type of play (Human or Computer), the level of players (if computer), the size of the board and who will be the first player. Menu Display

Initial states (6x6)

When the game starts, the tables and lists go by:

```
ListR1 = [a, b, c, d, e, f]       ListR2 = [e, a, d, b, f, c]
ListC1 = [1, 2, 3, 4, 5, 6]       ListC2 = [2, 5, 4, 3, 1, 6]
Boad1 =                           Boad2 =
[[z,z,z,z,z,z],                   [[z,z,z,z,z,z],
 [z,z,z,z,z,z],                    [z,z,z,z,z,z],
 [z,z,z,z,z,z],                    [z,z,z,z,z,z],
 [z,z,z,z,z,z],                    [z,z,z,z,z,z],
 [z,z,z,z,z,z],                    [z,z,z,z,z,z],
 [z,z,z,z,z,z]],                   [z,z,z,z,z,z]],
```

Initial State

Intermediate states (6x6)

```
ListR1 = [a, b, c, d, e, f]       ListR2 = [e, a, d, b, f, c]
ListC1 = [1, 2, 3, 4, 5, 6]       ListC2 = [2, 5, 4, 3, 1, 6]
Boad1 =                           Boad2 =
```

```
[[x,o,x,z,z,x],                  [[x,z,o,z,o,z],
 [z,z,o,x,z,z],                   [z,z,o,x,z,z],
 [z,x,z,z,z,o],                   [z,x,z,z,z,o],
 [z,z,o,x,o,z],                   [z,z,o,x,o,z],
 [o,x,z,o,z,z],                   [o,x,z,o,z,z],
 [z,z,o,z,x,z]],                  [z,z,o,z,x,z]],
```

Intermediate State

Final states (6x6)

And the final state of the boards would look like:

```
ListR1 = [a, b, c, d, e, f]      ListR2 = [e, a, d, b, f, c]
ListC1 = [1, 2, 3, 4, 5, 6]      ListC2 = [2, 5, 4, 3, 1, 6]
Boad1 =                          Boad2 =
[[x,o,x,o,o,x],                  [[x,x,o,o,o,o],
 [x,x,o,x,x,x],                   [o,o,o,x,x,x],
 [o,x,o,o,x,o],                   [x,o,x,o,o,x],
 [o,x,o,x,o,x],                   [x,x,x,o,x,x],
 [o,x,o,o,x,o],                   [o,x,x,o,o,x],
 [o,o,o,x,x,x]],                  [x,x,o,o,o,o]],
```

Final State

Game Over - End

After all the board is filled with `x` and `o` (there is no `z` in the boards anymore) it indicates the game is now over. The score will count the points for each player and, the one with less points, wins. e.g:

Game Over

Conclusions

Throughout the development of this project, we were able to improve our knowledge on this programming language and understand how to build a game from rules and predicates. However, we would like to mention how we encountered some, if not many obstacles as we built our program. From trying to translate game rules into code with some established restrictions, managing time while working on other courses, or even achieving a working computer player of a higher difficulty.

Due to this, we list the following:

Program Limitations:

- Computer Player only has one difficulty;
- Rules only explained through README;
- (Although true to the original game), only two board sizes are given to choose from;

Possible Improvements:

- Accessible Game Rules through the Main Menu - Providing a simple explanation for novice players and a more advanced one for expert players;
- Additional Game Rules for matches against computer players - Expanding the game to give our players a more unique experience;

Bibliography

- **Official Game Website:** https://boardgamegeek.com/boardgame/308153/doblin
- **Official Game Rules:** https://boardgamegeek.com/filepage/200477/doblin-rules-11
- **ChatGPT** was used with the following prompt and Official Game Rules image to generate the text for Game Rules:

  > "Based on this image, help me write the rules for the Game I'm implementing in Prolog. Note that the game will be playable by two players only."