

## Practical Assignment 2

### Implementation of a Board Game in Prolog

#### Description

**Objective:** To implement a two-player board game in the Prolog language. A board game is characterized by the type of board and pieces, the rules for moving the pieces (possible moves) and the conditions for ending the game with defeat, victory or a draw. The game must allow three modes of use: Human/Human, Human/Computer and Computer/Computer. At least two game levels must be included for the computer. The implementation should also include a suitable user interface in text mode.

#### Conditions for Assignment Completion

**Group Formation:** Groups of two (2) students enrolled in the same practical class. Exceptionally, and only if strictly necessary, one group of three per practical class may be accepted.

**Group and Topic Choice:** The groups are chosen in the activity to be made available for this purpose on Moodle from 16:00 on November 22, 2024. In the first phase, one of the group members must choose the group/topic; in the second phase, the second member will join the group. Each topic (game) can only be chosen by a maximum of 9 groups, to ensure that all topics are equally selected. The list of available games can be found at the end of this document.

**Deadlines:** The assignment (source code and readme file) must be delivered by 20:00 on January 5, 2025, in the activity to be made available for this purpose on Moodle, with the corresponding demonstrations to be carried out during the week of January 6-10, 2025. The demonstrations should be scheduled with the instructor of each practical class.

**Assessment Weight:** This assignment counts for 25% of the final grade. The evaluation of the assignment focuses on the implemented features, the quality of the code and respective comments, the readme file and participation in the presentation. Grades may differ between group members.

**Languages and Tools:** The game must run under SICStus Prolog version 4.9 and should work on Windows and Linux. If any configuration is required (beyond the standard installation of the software), or a font other than the default one is used, this must be expressed in the README file, which must also include the steps required to configure and/or install the necessary components (on Windows and Linux). Inability to test the developed code will result in penalties in the evaluation. Ensure that predicate names and functionality are as requested below and that all code is properly commented.

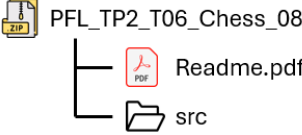
#### Deliverables and Evaluation

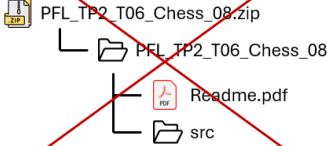
Each group must submit a **zip archive** containing a **README** file in **PDF format** as well as the developed **source code**. The submission should be made via Moodle, in a ZIP file named:

**PFL\_TP2\_TXX\_#GROUP.zip**

Where **TXX** specifies the practical class (e.g. T06 for class 3LEIC06), and **#GROUP** is the group designation. Example: PFL\_TP2\_T06\_Chess\_08.zip

The ZIP file should contain the **README** file in **PDF** format at the root of the archive and a folder named **src** containing all the PROLOG **source-code**. Do not include additional directories in the archive.

Correct: 

Wrong: 

**Source code:** Consider the following guidelines for your source code:

- The main code file must be named **game.pl** and should be in the **src** folder.
- You can separate your code in different files for increased organization; however, all files should be in the **src** folder (i.e., do not include additional folders).
- You can use any available SICStus libraries in your code.
- The following predicates should have the specified signatures:
  - The main predicate, **play/0**, must be in the *game.pl* file and must give access to the game menu, which allows configuring the game type (H/H, H/PC, PC/H, or PC/PC), difficulty level(s) to be used by the artificial player(s), among other possible parameters, and start the game cycle.
  - **initial\_state(+GameConfig, -GameState)**. This predicate receives a desired game configuration and returns the corresponding initial game state. Game configuration includes the type of each player and other parameters such as board size, use of optional rules, player names, or other information to provide more flexibility to the game. The game state describes a snapshot of the current game state, including board configuration (typically using list of lists with different atoms for the different pieces), identifies the current player (the one playing next), and possibly captured pieces and/or pieces yet to be played, or any other information that may be required, depending on the game.
  - **display\_game(+GameState)**. This predicate receives the current game state (including the player who will make the next move) and prints the game state to the terminal. Appealing and intuitive visualizations will be valued. Flexible game state representations and visualization predicates will also be valued, for instance those that work with any board size. For uniformization purposes, coordinates should start at (1,1) at the lower left corner.
  - **move(+GameState, +Move, -NewGameState)**. This predicate is responsible for move validation and execution, receiving the current game state and the move to be executed, and (if the move is valid) returns the new game state after the move is executed.
  - **valid\_moves(+GameState, -ListOfMoves)**. This predicate receives the current game state, and returns a list of all possible valid moves.
  - **game\_over(+GameState, -Winner)**. This predicate receives the current game state, and verifies whether the game is over, in which case it also identifies the winner (or draw). Note that this predicate should not print anything to the terminal.
  - **value(+GameState, +Player, -Value)**. This predicate receives the current game state and returns a value measuring how good/bad the current game state is to the given **Player**.
  - **choose\_move(+GameState, +Level, -Move)**. This predicate receives the current game state and returns the move chosen by the computer player. Level 1 should return a random valid move. Level 2 should return the best play at the time (using a greedy algorithm), considering the evaluation of the game state as determined by the **value/3** predicate. For human players, it should interact with the user to read the move.
- You cannot use **assert** or **retract** to store / manipulate game configuration or game state information. If you need information from the game configuration, you can include it in the game state term.
- Source code should follow established conventions for code quality and organization. Use meaningful names for predicates and arguments. Try to write code that ‘looks declarative’ and avoid using ‘imperative-looking’ constructions (e.g., if-then-else clauses). Try to write efficient code (e.g., using tail recursion when possible).<sup>1, 2</sup>
- All source code must be **properly commented**. Each of the predicates listed above should have extended comments describing the strategy used to implement it.

<sup>1</sup> See <https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/Writing-Efficient-Programs.html>

<sup>2</sup> See, for instance, Michael A. Covington, Roberto Bagnara, Richard A. O'keefe, Jan Wielemaker, and Simon Price. 2012. Coding Guidelines for Prolog. Theory and Practice of Logic Programming, 12(6), pp. 889–927, November 2012. DOI: 10.1017/S1471068411000391 (preprint available at <https://arxiv.org/pdf/0911.2899>).

**Readme:** The README file (to submit in .pdf format) should be structured as follows:

- **Identification of the topic (game) and group** (group designation, student number and full name of each member of the group), as well as an indication of the **contribution** (in percentages, adding up to 100%, and a brief description of tasks performed) **of each member** of the group to the assignment.
- **Installation and Execution:** include all the necessary steps for the correct execution of the game in both Linux and Windows environments (in addition to the installation of SICStus Prolog 4.9).
- **Description of the game:** a brief description of the game and its rules; you should also include the links used to gather information (official game website, rule book, etc.).
- **Considerations for game extensions:** describe the considerations taken into account when extending the game design, namely when considering variable-sized boards, optional rules (e.g., simplified rules for novice players, additional rules for expert players), and other aspects.
- **Game Logic:** Describe the main design decisions regarding the implementation of the game logic in Prolog (do not copy the source code). This section should have information on the following topics, among others:
  - **Game Configuration Representation:** describe the information required to represent the game configuration, how it is represented internally and how it is used by the *initial\_state/2* predicate.
  - **Internal Game State Representation:** describe the information required to represent the game state, how it is represented internally, including an indication of the meaning of each atom (i.e. how different pieces are represented). Include examples of representations of initial, intermediate, and final game states.
  - **Move Representation:** describe the information required to represent a move, and how it is represented internally (e.g., the coordinates of a board location, and/or other information necessary to represent a move) and how it is used by the *move/3* predicate.
  - **User Interaction:** briefly describe the game menu system, as well as how interaction with the user is performed, focusing on input validation (e.g., when reading a move).
- **Conclusions:** Conclusions about the work carried out, including limitations of the program (and known issues), as well as possible improvements (future developments roadmap).
- **Bibliography:** List of books, papers, web pages and other resources used during the development of the assignment. If you used tools such as ChatGPT, list the queries used.

You can also include one or more **images** illustrating the execution of the game, showing initial, intermediate and final game states, and interaction with the game.

The entire document should not exceed four pages (including images and references).

**Demonstration:** Demonstrations must be scheduled with the teacher of your practical class for the week of January 6. All members of the group must be present and actively participate in the demonstration.

You should include some intermediate and near-final game states in your code to demonstrate all the rules of the game as well as the *game\_over/2* predicate more quickly (these game states can be hard-coded directly into the code file, using predicates similar to the *initial\_state/2* predicate).

**Groups of Three:** Groups of three elements are required to perform additional work:

- Implement a third level of AI, using the minimax algorithm. Include also a new section in the readme file with a description of the strategy for implementing this third level of AI;
- Implement at least one additional optional rule to the game. Explain this extension in the report.

**Late Deliveries and Plagiarism Detection:** Late deliveries will be subject to heavy penalties (cumulative and increasing with time). A plagiarism detection software will be used on all submitted assignments. All detected cases of academic misconduct will be handled with extreme prejudice.

## Proposed Topics (Games)

The games to be implemented are board games for (at least) two players in which there is no influence of the luck factor during the game. The games do not include dice, draws of any kind or hidden information.

Proposed games:

1. Anaash - [http://www.marksteeregames.com/Anaash\\_rules.pdf](http://www.marksteeregames.com/Anaash_rules.pdf)
2. Aqua Pipe - <https://boardgamegeek.com/boardgame/414235/aqua-pipe>
3. Ayu - <https://www.mindsports.nl/index.php/arena/ayu>
4. Blackstone - [http://www.marksteeregames.com/Blackstone\\_rules.pdf](http://www.marksteeregames.com/Blackstone_rules.pdf)
5. Blinq - [https://nestorgames.com/rulebooks/BLINQ\\_A5\\_EN.pdf](https://nestorgames.com/rulebooks/BLINQ_A5_EN.pdf)
6. Blütentanz - <https://boardgamegeek.com/boardgame/428363/blutentanz>
7. Byte - [http://www.marksteeregames.com/Byte\\_rules.pdf](http://www.marksteeregames.com/Byte_rules.pdf)
8. Collapse - <https://kanare-abstract.com/en/pages/collapse>
9. Crosswind - <https://boardgamegeek.com/boardgame/424939/crosswind>
10. Doblin - <https://boardgamegeek.com/boardgame/308153/doblin>
11. LOT - <https://boardgamegeek.com/boardgame/127989/lot>
12. Mabula - <https://boardgamegeek.com/boardgame/346743/mabula>
13. Minefield - <https://boardgamegeek.com/boardgame/420797/minefield>
14. Replica - <https://boardgamegeek.com/boardgame/427267/replica>
15. Sight - <https://kanare-abstract.com/en/pages/sight>
16. STAQS - <https://boardgamegeek.com/boardgame/425529/staqs>
17. Storm Clouds - <https://boardgamegeek.com/boardgame/429340/storm-clouds>
18. Turtles - <https://turtlesgame.xyz/>
19. VLKNO - <https://boardgamegeek.com/boardgame/427638/vlkno>