PFL - Projeto Prolog

Replica

T13 G10

Group Members

Student	ID	Contribution
João Guedes	up202108711	50%
Eduardo Machado	up202105337	50%

João Guedes focused on the initial task of creating the game, and implementing the Player Vs Player mode. Eduardo Machado focused on the computer side of the game; creating the Player vs Computer and Computer vs Computer, in particular level 3 for Minimax. Some of the shared tasks involve creating level 2 for the greedy algorithm and other details of improvement to the program.

Installation and Execution

1. Clone the repository to your local machine.
2. Open SICStus Prolog.
3. Consult the `game.pl` file by running the following command in the Prolog interpreter: ```prolog ?- consult('path/../game.pl').
4. Start the game by running the following command: ?- play.

Description of the Game

Replica is a two player game played using a chessboard and 12 black and 12 white flippable checkers. Players setup the game by placing the checkers on the board in opposite corners (with a 2x2 square in the corner, flanked by a 2x2 square on each side). The pieces in the very corner start the game flipped over (indicating a king). Players make one move per turn, starting with White.

On each turn, players either step, jump, or transform. All moves (even captures) must go "forward" (1 of the 3 directions towards the opponent corner). For steps and for jumps, if there is already a piece on the destination square, it is captured by replacement. For a step, the piece moves forward one square. For a jump, the piece moves in a straight line forward over friendly pieces until it reaches a square not occupied by a friendly piece. For a transform, a friendly non-king piece in line-of-sight of a friendly king gets flipped (this creates another friendly king). Only enemy pieces block line of sight.

The game is over if a player wins by getting any friendly king into the opposite corner, or wins by capturing any enemy king.

Considerations for game extensions

For the game extensions, we add the option of choosing one of three board: 8x8, 12x12 and 16x16. To make sure the game functioned for each board, we add to implement a function called within_bounds which allows to verify the validity of each move independent of board size.

Another extension was the Graphical User Interface, we created a menu which allows the user to select five options: 1. Player VS Player; 2. Player VS Computer; 3. Computer VS Computer; 4. Rules; 5. Exit. Besides this the player has the ability to chose their own name that gets displayed in the beginning of each round.

For a final extension, we decided to implement an extra algorithm, the one proposed to groups of three: Minimax Algorithm. This way the player has the chance to select three difficulty levels to play against the machine.

Game Logic

Game Configuration Representation

The game configuration representation in the Replica game include the player names and the selected board size.

Internal Game State Representation

The internal game state is represented using a game_state structure, which include:

- the board that we've selected (A 2D list representing the current state of the board. Each element in the list represents a square on the board and can contain a piece (e.g., w, kw, b, kb) or be empty (e).);
- the current player that is going the select the current move;
- name of the white player;
- name of the black player.

This allows for the game to then switch between the white and black for the current player attribute, and the board is updated after each move until it's Game Over. This is done in our game cycle, which loops until the game ends by receiving the new updated game state and verifying if it matches the game over conditions, if it doesn't it continues.

To contrast among different game modes, we create three game cycles. Our main cycle, *game_cycle(GameState)*, the cycle for player vs computer which takes as argument the level chosen for the computer: *game_cycle_computer(+GameState, +Level)*. And for the computer vs computer, since we can choose two different levels, we create another game cycle: *game_cycle_computer_vs_computer(+GameState, +WhiteLevel, +BlackLevel)*.

Initial State:

```
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
8 |   |   |   |   | w | w | w | W |
  +---+---+---+---+---+---+---+---+
7 |   |   |   |   | w | w | w | w |
  +---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   | w | w |
  +---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   | w | w |
  +---+---+---+---+---+---+---+---+
4 | b | b |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
3 | b | b |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
2 | b | b | b | b |   |   |   |   |
  +---+---+---+---+---+---+---+---+
```

```
  1 | B | b | b | b |   |   |   |   |
    +---+---+---+---+---+---+---+---+
```

Intermediary State:

```
      1   2   3   4   5   6   7   8
    +---+---+---+---+---+---+---+---+
  8 |   |   |   |   | w | w | w | W |
    +---+---+---+---+---+---+---+---+
  7 |   |   |   | w | w |   | w | w |
    +---+---+---+---+---+---+---+---+
  6 |   | b |   |   |   |   |   | w |
    +---+---+---+---+---+---+---+---+
  5 |   |   | b |   |   | w | w |   |
    +---+---+---+---+---+---+---+---+
  4 | b |   |   |   |   |   |   | w |
    +---+---+---+---+---+---+---+---+
  3 | b | b | b |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
  2 | B |   | b | b |   |   |   |   |
    +---+---+---+---+---+---+---+---+
  1 | B | b |   | b |   |   |   |   |
    +---+---+---+---+---+---+---+---+
```

Final State:

```
      1   2   3   4   5   6   7   8
    +---+---+---+---+---+---+---+---+
  8 |   |   |   | b |   | w |   |   |
    +---+---+---+---+---+---+---+---+
  7 |   |   |   |   | b |   | W |   |
    +---+---+---+---+---+---+---+---+
  6 | b |   |   |   |   | W |   |   |
    +---+---+---+---+---+---+---+---+
  5 |   |   |   |   |   |   | w |   |
    +---+---+---+---+---+---+---+---+
  4 |   |   |   |   |   |   |   | w |
    +---+---+---+---+---+---+---+---+
  3 | b |   | b |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
  2 | b |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
  1 | b |   | w |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
```

Winner: white.

Move Representation

Moves are represented as move(MoveType, CX, CY, NX, NY) for steps and jumps, and move(transform, X, Y, KX, KY) for transforms. During each round the player gets to choose which type of move they want to make, this way to system of movement select becomes less ambiguous and similar to older text based games, and allows us to work on them and test them separately.

The player selects the move type, for step:

- the player inputs their initial position X-Y, and then the final position NX-NY (the step only allows to move one square, up, down, sideways, diagonally);

For the jump:

- the player inputs their initial position and then the final position, a piece can only jump over friendly pieces, so by selecting the initial and final, the program then verifies if between coordinates there are only friendly pieces, no empty spaces or enemy pieces;

For the transform:

- the player inputs the position of the piece they want to transform and then the position of the King, if there are no pieces between them, then the piece transform into a King (we set a rule that only allows to have a maximum of two kings).

In *move(+GameState, +Move, -NewGameState)*, after receveing the player's move (step, jump, transform), it starts by looking at all valid moves (*valid_moves/2*), then checks if our chosen move is in that list, it performs the move (*perform_move(+Board, +CX, +CY, +NX, +NY, -NewBoard)*), and, finally, it switches player.

For the moves made by the computer we have the predicate *choose_move(+GameState, +Level, -Move)*, for each level it calls for different algorithms. For level 1, it's random, for level 2 it calls our greedy algorithm and level 3 it calls minimax.

User Interaction

The game provides a menu for the user to choose between different game modes. The player only has to input the number of the option to select. If they choose aa option that involver a human player, then they get to choose the display name. If they choose an option with a computer then they get to choose the difficulty: 1, 2, 3. For the board selection it's a similar process, pick: 8, 12, or 16 for the size.

When playing, the user chooses what type of move they want: *step.*, *jump.*, or *transform.*. After selecting one of them, they input the current position of the pieces they want to interact with, ie: *4-5.* and then the next position *4-6.*.

Conclusions

The implementation of Replica in Prolog demonstrates the power of logic programming for developing board games. The game logic, move validation, and AI strategies are all implemented using Prolog's declarative syntax, making the code easy to understand and extend. With the full implementation of the modes Human VS Human, Human VS Comp and Comp VS Comp, three difficulties, and three boards, to us the hardest part was the implementation of the greedy algorithm.

Bibliography

- **Replica**. *BoardGameGeek*. Retrieved from https://boardgamegeek.com/boardgame/427267/replica.
- **SICStus Prolog Documentation**. *SICStus Prolog*. Retrieved from https://sicstus.sics.se/sicstus/docs/latest/html/sicstus.html/index.html#SEC_Contents.