Flume
Programação em Lógica
FEUP

Carolina Moreira - up201303494
Maria Teresa Ferreira - up201603811

November 18, 2018

1 Introduction

2 Game Description

In 2010, Mark Steere designed Flume. This is a two-player game, in which each player has different color pieces.
The board is a square with odd number positions, meaning there can not be ties.
The players take turns placing pieces of their own color on the empty board places.
If, when a player makes a move, the piece placed is adjacent to three or four other pieces (regardless of color), this player can play again until this does not happen.
The game ends when the board is full and the winner is the one who has more of their own pieces placed.

3 Game Logic

3.1 Game State Internal Representation

In this section we'll show how the board is stored on the program. We've chosen to use a list of lists, with the first list simbolizing the row, and the second one the column.

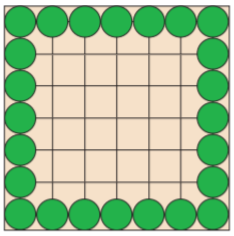
Figure 1: Image Representation of Initial Board

3.1.1 Initial Board

```
initial_board(Board) :-
        Board = [
                [green, green, green, green, green, green, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, green, green, green, green, green, green]].
```

Listing 1: Source Code for the Internal Representation of the Initial Board

3.1.2 Intermediate Board

```
example_board1(Board) :-
        Board = [
                [green, green, green, green, green, green, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blank, blank, green],
                [green, blank, blank, blank, blue, red, green],
                [green, blank, blank, red, blank, blank, green],
                [green, blank, blank, blue, blue, red, green],
                [green, green, green, green, green, green, green]].
```

Listing 2: Source Code for the Internal Representation of an Example Intermediate Board

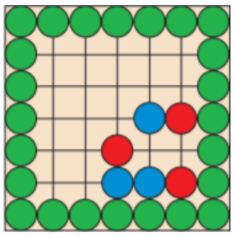

Figure 2: Image Representation of Example Intermediate Board

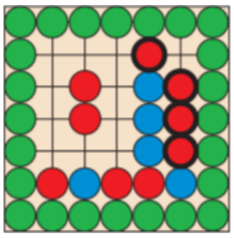

Figure 3: Image Representation of Example Winning Condition Board

3.1.3 Winning Condition Board

```
example_board2(Board) :-
        Board = [
                [green, green, green, green, green, green, green],
                [green, blank, blank, blank, red, blank, green],
                [green, blank, red, blank, blue, red, green],
                [green, blank, red, blank, blue, red, green],
                [green, blank, blank, blank, blue, red, green],
                [green, red, blue, red, red, blue, green],
                [green, green, green, green, green, green, green]].
```

Listing 3: Source Code for the Internal Representation of an Example Winning Condition Board

3.2 Text Visualization of Board

In this section we'll describe what the board looks like printed on the screen. We print it on the screen using capital letters for the pieces (G for green, R for red, B for blue, and an empty space for a blank space), and dashes and upright bars as separators.

```
print_board([]) :-
        print_separator.
```

```
print_board(Board) :-
          Board = [H|T],
          print_separator,
          print_line(H),
          print_board(T).

print_separator :-
                    print('*-*-*-*-*-*-*-*\n').

print_line([]) :-
          print('|\n').

print_line(Line) :-
          Line = [H|T],
          print('|'),
          print_piece(H),
          print_line(T).

print_piece(Piece) :-
          piece_text(Piece, Text),
          print(Text).

piece_text(green, 'G').
piece_text(red, 'R').
piece_text(blue, 'B').
piece_text(blank, ' ').
```

Listing 4: Source Code for the Text Representation

```
*-*-*-*-*-*-*-*
|G|G|G|G|G|G|G|
*-*-*-*-*-*-*-*
|G| | | | | |G|
*-*-*-*-*-*-*-*
|G| | | | | |G|
*-*-*-*-*-*-*-*
|G| | | | | |G|
*-*-*-*-*-*-*-*
|G| | | | | |G|
*-*-*-*-*-*-*-*
|G| | | | | |G|
*-*-*-*-*-*-*-*
```

```
|G|G|G|G|G|G|G|
*—*—*—*—*—*—*—*
```

Listing 5: Text Representation of the Initial Board

3.3 Valid Moves Listing

In this section we show the code used for listing all the valid moves. First, we compile all the positions on the board into a list called MoveList. Then we call on another predicate, which will from that list choose which moves are valid, depending on whether the head of the list at the time is a valid move or invalid move. We use the predicate validmove as well, which verifies whether a certain cell is empty.

```
valid_move(GameState, Pos):-
        get_board(GameState, Board),
        get_piece(Board, Pos, blank).

valid_moves(GameState, ValidMoveList) :-
        all_positions(MoveList),
        choose_valid_moves(GameState, MoveList, ValidMoveList).

choose_valid_moves(_, [], []).
choose_valid_moves(GameState, [ValidMove|MoveList], [ValidMove|ValidMo
        valid_move(GameState, ValidMove),
        choose_valid_moves(GameState, MoveList, ValidMoveList).

choose_valid_moves(GameState, [InvalidMove|MoveList], ValidMoveList):-
        \+ valid_move(GameState, InvalidMove),
        choose_valid_moves(GameState, MoveList, ValidMoveList).
```

Listing 6: Source Code for Valid Moves

3.4 Move Usage

In this section we proceed to moving one piece, first by getting that move from the user in the predicate getmove, then by verifying whether it's a valid move, and then by actually making the move, by setting the piece and changing the board accordingly.

```
move(GameState, NextGameState):-
        get_move(Pos),
        valid_move(GameState, Pos),
        make_move(GameState, Pos, NextGameState).

make_move(GameState, Pos, NextGameState) :-
        get_board(GameState, Board),
        get_player(GameState, Player),
        set_piece(Board, Pos, Player, NewBoard),
        set_board(GameState, NewBoard, NewGameState),
        increment(NewGameState, Player, NewGameState_temp),
        value(NewGameState_temp, Blank),

        decide_next_player(NewGameState, Pos, NewPlayer),
        set_player(NewGameState_temp, NewPlayer, NextGameState),

        format('Next play by ~w', NewPlayer), nl.
```

Listing 7: Source Code for Moving

3.5 End of Game

In this section, we verify that the game has ended when the sum of blue pieces and red pieces totals 25, the maximum number of pieces that can be placed. Then, depending on who has the larger number of pieces, that player wins.

```
game_over([_Board, _Player, Red, Blue], Winner) :-
          Red + Blue =:= 25,
          get_winner(Red, Blue, Winner),
          format('~w wins', Winner), nl.

get_winner(Red, Blue, red) :-
          Red > Blue, !.

get_winner(_Red, _Blue, blue).
```

Listing 8: Source Code for Ending the Game

3.6 Board Evaluation

In this section, we evaluate the board, showing how many blue pieces, red pieces, and blank spaces there are.

```
value(GameState, Blank):-
        get_red_num(GameState, Red_Num),
        get_blue_num(GameState, Blue_Num),
        Total is Red_Num + Blue_Num,
        Blank is 25 - Total,

        nl,
        format('Red: ~w ', Red_Num),
        format('Blue: ~w ', Blue_Num),
        format('Blanks: ~w', Blank), nl.
```

Listing 9: Source Code for Evaluating the Game

3.7 Computer Play

In this section, the computer plays a round as well, by choosing a random position. We'd like to improve on this by having the computer choose a position that would utilize the adjancency functions of the game, but we didn't manage to finish it in time.

```
choose_move(GameState, NextGameState, blue, '2'):-
        valid_moves(GameState, List),
        listLength(List, Length),
        random(1, Length, Index),
        nth1(Index, List, Pos),
        make_move(GameState, Pos, NextGameState).
```

Listing 10: Source Code for the Computer Play

4 Conclusion

During this project, we learnt a lot about prolog, as well as how to think in a different way than what we'd done so far. There were many concepts we explored while working in this game that we hadn't encountered before, so it was good to learn new things. If we were to develop this project further, we'd like to improve our AI, as well as better develop the graphical interface of the program.

5 Bibliography

https://sicstus.sics.se/documentation.html