Practical Assignment 2

2024/2025 PFL 2nd Assignment

Sight Board Game

Rodrigo Pinto Pesqueira Gaspar Pombo - up202105374
Pedro Jorge das Neves Pinto Vieira - up202206230

Identification of the topic and group

The game we implemented was Sight.

Our group is Sight_4, composed by the students up202206230 (Pedro Jorge das Neves Pinto Vieira) and up202105374 (Rodrigo Pinto Pesqueira Gaspar Pombo). Each student contributed 50% to the assignment.

Installation and Execution

Once the game is running (it runs on the terminal), we can just configure it with the in game menu, where we can select the board size, in which game state to start (new, intermediate, near final) and also if the white or black players are human or the computer level 1 (random) or the computer level 2 (greedy algorithm).

Windows:

From the root of the project folder (just outside the /src folder):

```
"C:\Program Files\SICStus Prolog VC16 4.9.0\bin\sicstus.exe" -l "src\game.pl" --goal "play, halt."
```

If necessary replace C:/Program Files/SICStus Prolog VC16 4.9.0/bin/sicstus.exe with a different path for the SICStus Prolog executable.

Linux:

From the root of the project folder (just outside the /src folder):

```
/usr/local/sicstus4.9.0/bin/sicstus -l ./src/game.pl --goal "play, halt."
```

If necessary replace /usr/local/sicstus4.9.0/bin/sicstus with a different path where SICStus Prolog is installed.

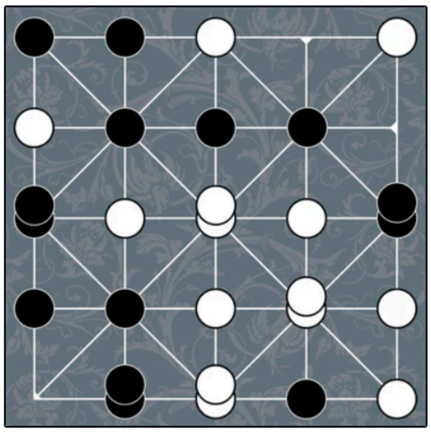
Description of the game

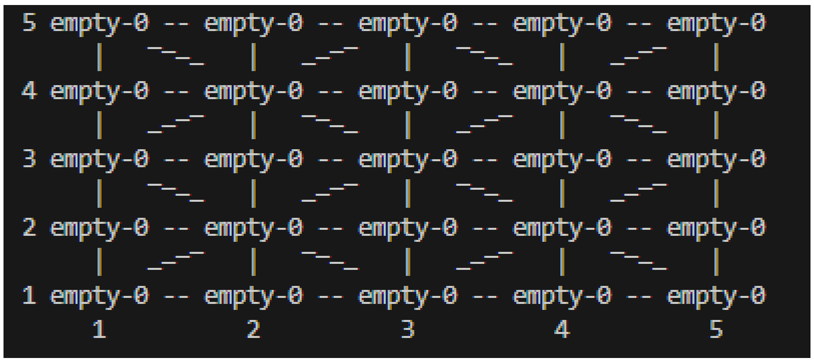
Sight is a 2-player game where one player has white pieces and the other one has black pieces, in a 5x5 squared board divided in 4, where each square has its horizontal, vertical and diagonal intersections represented. In their turns, the players must move their pieces to the next available intersection. If there is a “friendly” piece “in line of sight”, which means a piece that is on the same line (in all directions, indicated by the lines), a “stack” is formed, and other pieces are placed on the top of the pieces involved. In a turn where the player has stacks, the player must move a piece from the top of the stack to an adjacent empty cell, which also triggers the line of sight mechanic, except it doesn’t apply to the stack we just removed the cell from, and so on. The first player with no legal moves loses the game.

sight game storepage: https://kanare-abstract.com/en/pages/sight

sight game rules:
https://cdn.shopify.com/s/files/1/0578/3502/8664/files/Sight_EN.pdf?v=1713463846

Game is played in an alquerque board:


actual alquerque board



```
5 empty-0 -- empty-0 -- empty-0 -- empty-0 -- empty-0
     |   ‾-_    |   _-‾    |   ‾-_    |   _-‾    |
4 empty-0 -- empty-0 -- empty-0 -- empty-0 -- empty-0
     |   _-‾    |   ‾-_    |   _-‾    |   ‾-_    |
3 empty-0 -- empty-0 -- empty-0 -- empty-0 -- empty-0
     |   ‾-_    |   _-‾    |   ‾-_    |   _-‾    |
2 empty-0 -- empty-0 -- empty-0 -- empty-0 -- empty-0
     |   _-‾    |   ‾-_    |   _-‾    |   ‾-_    |
1 empty-0 -- empty-0 -- empty-0 -- empty-0 -- empty-0
     1          2          3          4          5
```

our implementation

Considerations for game extensions

We implemented a dynamic board size (the player can select the size of the board they wish to play the game with).

We also try to help players by providing them with the list of valid moves after they input an incorrect move to try to guide them. To help with that the menu also provides a lot of information about what is happening and what the error was so the user knows what's the problem and what is happening.

Game Logic

We used something similar to a MVC (Model View Controller) design with the game_controller.pl where most of the logic of the program is implemented, game_model.pl where we get the initial representation of the GameState and game_view.pl where printing the GameState to the terminal is handled, while also maintaining the game.pl file for the managing of the game cycle. We also used an additional file menu.pl where the initial menu to configure the game is implemented.

Essentially the flow of execution starts at play/0 at the game.pl file that accesses the menu(-GameConfig) inside menu.pl, gets the initial GameState based on the GameConfig from the menu by accessing the initial_state(+GameConfig, -GameState) inside the game_model.pl, then it displays the GameState by calling display_game(+GameState) implemented inside game_view.pl, after that it starts the recursive game_cycle where it handles user input and changes made to the GameState by accessing multiple predicates inside game_controller.pl, until the predicate game_over(+GameState, -Winner) succeeds, after which the winner is printed and the execution is stopped.

Game Configuration Representation

The Game Configuration is chosen by the user. When he opens the game in the terminal, he can choose from 3 different game states: a new game, an intermediate state and a near-final state. Then, he can choose the board size (if he selects the initial state option, starting a new game). Finally, he'll decide who will play white and black, between a human player, a computer level 1 (chooses move randomly among the possible moves) or a computer level 2 (chooses the best move based on a greedy algorithm):

After this, the Game Configuration is saved into the array GameConfig, made with the user's inputs.

Internal Game State Representation

The Game State is handled in the "game_model.pl" file. Regardless of the user's choice, a game that starts is our initial state (this allows us to keep track of the moves without causing conflicts). The correct predicate is selected by Pattern Matching. If the user decides to start a brand new game, an empty board is built from scratch using a "maplist". In the other two states, a list of lists representing the board with some pieces already placed is used. All boards have the 1,1 index at the bottom left and the column index and row index both start at 1. Each cell of the board is represented with a compound term (for example empty-0 or white-1 or black-2 or black-1) indicating the player the pieces in the block belong to and the number of pieces that block has stacked on top of each other.

In the intermediate game state representation there are multiple pieces placed on the board for both players but no stacks (no more than 1 piece per cell) and in the near final game representation we have multiple single pieces placed on the board but also multiple stacks of 2 for both players.

Move Representation

There are 2 types of moves:

- place(ColumnIndex, RowIndex) for placing a piece on an empty cell
- move_stack(SourceColumnIndex, SourceRowIndex, DestinationColumnIndex, DestinationRowIndex) for moving the top piece from a stack to an adjacent empty cell

When using the place move we check if the space is empty and then add a piece to each cell that already has at least 1 piece belonging to the player that is executing the move and that is in line of sight (we check to see if there are any pieces that aren't blocked by other pieces horizontally, vertically and diagonally if the cell has diagonal lines) with the piece that was just placed.

When using the move_stack move we check if the move is valid by making sure that there is a stack at the source coordinates, that the stack belongs to the player that is trying to make the move, that the destination coordinates are adjacent to the stack and empty, after that we decrement the stack, place a piece in the destination coordinates, do the line of sight mechanic of adding pieces to other pieces of the same player by checking if they are pieces that aren't blocked by other pieces horizontally, vertically and diagonally with the piece that was placed at the destination coordinates. Then at the end we need to remove another piece from the source stack because it was affected by the line of sight mechanic and it's not supposed to be, which always happens because they are both adjacent to one another, so we revert it by doing that.

User Interaction

The program will prompt the user again if the user input was invalid and will try to help them by displaying why it was invalid and in the case of moves it also displays the valid moves after the error message so that the user can simply choose from one of the moves.

On the initial menu wrong user inputs are also handled by displaying an error message and repeating the prompt to the user.

Conclusions

We managed to make the game function as it should, including the ability to choose whether a player is human, computer level 1 (random) or computer level 2 (greedy), all working correctly, we also implemented a dynamic board size and didn't notice any remaining bugs.

Possibly there is a better way to implement the greedy algorithm in the move selection process, in particular the part of giving a specific GameState a value, since in this game we are essentially trying not to lose we consider that the more moves we have and the less stacks we have the better, and opposingly the less moves the opponent has and the more stacks he has the better for us too. We also take into account that states that lead to victory because the opponent wouldn't have any possible moves are prioritised over all else by giving it a value of 10 000 and that states where we have no more possible moves are avoided at all costs by putting a value of -10 000, except if it also makes it so that the opponent has no more moves, in which case he will lose our turn so we would win. But it still remains really inconsistent, so we tried also adding and subtracting to the value in relation to the amount of moves a player can make by moving his highest stack piece, to try to make it so we get don't into a situation where we don't have a lot of options and the opposite for the other player, trying to get him into a position where he hasn't got a lot of options. But even after the random computer is still able to win quite frequently against it. We could maybe further improve the algorithm by fine tuning the weights of each metric (multiplying by 2 the most relevant one for example).

Bibliography

https://github.com/Fabio-A-Sa/Y3S1-ProgFuncionalLogica

https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_19.html

https://www.swi-prolog.org/

https://www.youtube.com/watch?v=SykxWpFwMGs

https://www.youtube.com/watch?v=GH4Tg5CYa1k

We also used ChatGPT especially for debugging, but didn't save the queries.