PFL – Blackstone Game

Group Identification

Group Designation: Class 13, Blackstone_9

Members:

- Tomás da Silva Custódio Teixeira (up202208041) – Contribution: 50%
- Rui Pedro da Silva Cruz (up202208011) – Contribution: 50%

Installation and Execution

To install the BlackStone game, first, download the game's zip file (*PFL_TP2_T13_Blackstone_09.zip*) and extract it. Inside the *src* directory, consult the *game.pl* file in Sicstus Prolog.

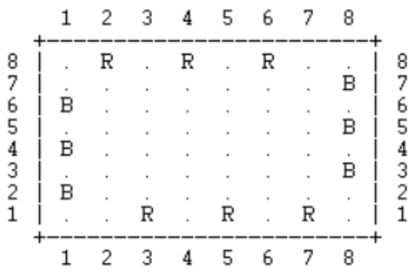
To start the game, use the *play/0* predicate, writing:

```
play.
```

The game is available for both Windows and Linux operating systems.

Description of the game

Blackstone is a two-player game played on a square board of any even size, 6x6 or larger. The perimeter cells are initially filled with red and blue stones, in the patterns shown in the figure below:



```
    1   2   3   4   5   6   7   8
  +-------------------------------+
8 | .   R   .   R   .   R   .   . | 8
7 | .   .   .   .   .   .   .   B | 7
6 | B   .   .   .   .   .   .   . | 6
5 | .   .   .   .   .   .   .   B | 5
4 | B   .   .   .   .   .   .   . | 4
3 | .   .   .   .   .   .   .   B | 3
2 | B   .   .   .   .   .   .   . | 2
1 | .   .   R   .   R   .   R   . | 1
  +-------------------------------+
    1   2   3   4   5   6   7   8
```

Players can only move their stones any number of squares along a straight, unobstructed path, vertically, horizontally, and diagonally. After the movement, in the original square, a **black** stone appears, which is neutral.

After you move your stone and place a black stone, if any **red** or **blue** stones are surrounded by adjacent stones of either three colors, and are completely blocked from moving, remove all of the red and blue stones are blocked, concluding your turn.

If the move removes all remaining stones of only one color (red or blue), the player of that color loses. However, if the move removes all remaining red and blue stones, the game ends in a **draw**, as neither player has any stones left on the board to continue the game.

The game also has the following variations:

- Medium Churn Variant: If a move completely blocks any red or blue stones, these stones and all black ones that contributed to the block are removed, concluding the turn.

- High Churn Variant: If a move completely blocks any red or blue stones, those stones and all black stones from the board are removed, concluding the turn.

To gather this information we used the following link: https://www.marksteeregames.com/Blackstone_rules.pdf

Consideration for game extensions

The user has the following options available in the game:

- Player vs Player mode
- Player vs Computer mode (player makes the first move)
- Computer vs Player mode (computer makes the first move)
- Computer vs Computer mode (each computer can have its own difficulty, depending on the user's input)
- Computer with random moves
- Computer with calculated moves, using a greedy algorithm
- Variable board size (6x6, 8x8 or 10x10)
- Standard Variant
- Medium Churn Variant
- High Churn Variant

These options allow the user to explore the game in various ways: playing with a friend, using boards of different sizes, watching a match between bots, or facing a greater challenge against the computer in greedy mode. The **greedy** algorithm calculates the best possible move at each turn by analyzing the computer's potential moves and considering the number of moves available to the opponent and the directions they can take. Additionally, in the late stages of the game, where fewer moves are available to significantly reduce the opponent's options, the algorithm also considers the computer's own mobility, prioritizing moves that maximize its own options to maintain strategic flexibility.

Game Logic

Game Configuration Representation

The *GameConfig* is composed solely of the *BoardSize*, which represents the board size selected by the user. This value is passed to the *initial_state/2* predicate, which calls the *board/2* predicate to construct the board with the desired dimensions. The generated board is then included in the *GameState*, thereby creating the initial state of the game.

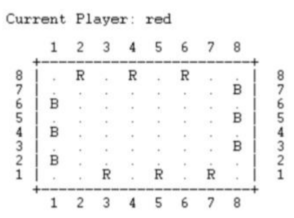
Internal Game State Representation

The ***GameState*** is a crucial argument in all implementation predicates. It is made up of a list of two elements:

- **Board:** The board is represented as a matrix containing all the stones in their respective positions, along with empty cells to display the game to the user and determine the possible moves. **Red** stones are represented with a **R**, **blue** stones are represented with a **B, black** stones are represented with a **X** and **empty** spaces are represented with a **period (.)**.

- **Player:** represents the player playing in the current round.

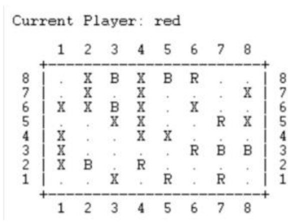
Initial Game State



```
Current Player: red

    1  2  3  4  5  6  7  8
  +------------------------+
8 | .  R  .  R  .  R  .  . |  8
7 | .  .  .  .  .  .  .  B |  7
6 | B  .  .  .  .  .  .  . |  6
5 | .  .  .  .  .  .  .  B |  5
4 | B  .  .  .  .  .  .  . |  4
3 | .  .  .  .  .  .  .  B |  3
2 | B  .  .  .  .  .  .  . |  2
1 | .  .  R  .  R  .  R  . |  1
  +------------------------+
    1  2  3  4  5  6  7  8
```

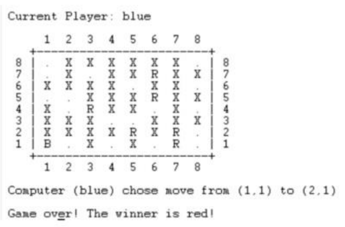
Intermediate Game State



```
Current Player: red

    1  2  3  4  5  6  7  8
  +------------------------+
8 | .  X  B  X  B  R  .  . |  8
7 | .  X  .  X  .  .  .  X |  7
6 | X  X  B  X  .  X  .  . |  6
5 | .  .  X  X  .  .  R  X |  5
4 | X  .  .  X  X  .  .  . |  4
3 | X  .  .  .  .  R  B  B |  3
2 | X  B  .  R  .  .  .  . |  2
1 | .  .  X  .  R  .  R  . |  1
  +------------------------+
    1  2  3  4  5  6  7  8
```

Final Game State



```
Current Player: blue

    1  2  3  4  5  6  7  8
  +------------------------+
8 | .  X  X  X  X  X  X  . | 8
7 | .  X  .  X  X  R  X  X | 7
6 | X  X  X  X  .  X  X  . | 6
5 | .  .  X  X  X  R  X  X | 5
4 | X  .  R  X  X  .  X  . | 4
3 | X  X  X  .  .  X  X  X | 3
2 | X  X  X  X  R  X  R  . | 2
1 | B  .  X  .  X  .  R  . | 1
  +------------------------+
    1  2  3  4  5  6  7  8

Computer (blue) chose move from (1,1) to (2,1)

Game over! The winner is red!
```

Move Representation

For a move to occur, a piece must first be selected. For the users, the predicate ***choose_piece/2*** is used, which captures the coordinates of the piece the user wants to move and confirms if a piece belonging to that user is being selected.
To assist the user and prevent invalid moves, after selecting the piece they want to move, a list of valid moves with the selected piece is displayed. This is achieved through the predicate ***valid_moves/3***, which considers all directions and the board's edges to calculate all possible moves for a specific piece.

For the computer, the predicate ***choose_move/3*** is used, which selects, depending on the chosen difficulty level (random or greedy), the piece to be moved as well as the destination coordinates, also using the predicate *valid_moves/3*.

The predicate **move** contains the following 3 arguments:

- GameState: already explained above, contains the current Border and the current Player.

- Move: a list [PieceCoords, NewCoords] where **PieceCoords** represents the current coordinates of the piece chosen by the current player to move and **NewCoords** represents the coordinates where the chosen piece will move.

- NewGameState: contains the edge updated with the chosen movement.

In the ***move/3*** predicate, the atom of the origin coordinate (*PieceCoords*) is changed from *blue* or *red* (depending on the player making the move) to *black*, the neutral piece. The destination coordinate changes from empty to the color of the current player. Thus, the board is updated, and finally, the player is alternated to complete the state transition.

User Interaction

When the user starts the game, they encounter the **main menu**, which allows them to **play**, read the **game instructions**, or **exit**. Interaction with the menu is done through **numbers**. If the user attempts to input a letter or an unavailable number, an error message is displayed.

When players are involved in the game, they are prompted to enter the **X** coordinate first, followed by the **Y** coordinate. If the coordinates are invalid, an error message appears, and the player can correct them. In **Computer vs. Computer** mode, after each move, the user has to input any key to proceed, allowing them to follow the game's progress.

To read user input, the predicate ***read_input_number/1*** is used, which relies on ***get_code/1*** to read the input in ASCII. The ASCII code is then converted into the number entered by the user. This approach eliminates the need to add a period at the end of the input and is suitable since the input is always a single number.

Conclusions

The BlackStone game was successfully implemented in Prolog. The Player vs. Player, Player vs. Computer, and Computer vs. Computer modes were developed, including the *Random* and *Greedy* difficulty levels, as well as two additional game variants beyond the standard one.

The most challenging part of the project was implementing the *Greedy* difficulty, as it required designing an efficient algorithm to evaluate and prioritize moves dynamically.

As a result, this project allowed us to consolidate the knowledge acquired during both theoretical and practical classes.

Bibliography

- https://www.marksteeregames.com/Blackstone_rules.pdf

- Prolog Powerpoints available on Moodle