Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Frozen Forest

Students:
Duarte Carvalho, up201503661, up201503661@fe.up.pt

October 17, 2018

Contents

1 Introduction **2**
- 1.1 Description . . . 2
- 1.2 How to play . . . 2
- 1.3 Winning conditions . . . 2
- 1.4 Game start . . . 2
- 1.5 Players movement . . . 3
- 1.6 Line of sight . . . 4

2 Implementation **5**
- 2.1 Internal game state representation . . . 5
  - 2.1.1 Board Initial state . . . 5
  - 2.1.2 Board after each player first move . . . 6
  - 2.1.3 Mid game board . . . 7
  - 2.1.4 End game board . . . 8
- 2.2 Game visualization . . . 9

1 Introduction

1.1 Description

A little girl named Mina and her best pal, a friendly yeti named Yuki, play hide-and-seek in a magical frozen forest. Not only do the trees grow perfectly aligned in a 10×10 grid, but the trees themselves grow straight up. This means that a skinny little girl could easily hide behind one. A yeti, however, is far too big not to be seen. So, that is why Mina always hides, Yuki always seeks, and they always have fun! But seeking takes a lot of energy for someone as big as Yuki, and the trees also happen to be very much to Yuki's tastes. So he cannot resist munching every last splinter of every tree he encounters. In fact, if he runs out of trees to eat, he gets too hungry and gives up.

1.2 How to play

One player plays Yuki and the other player plays Mina. In the end of the first round, the players must switch characters, making a total of two rounds.

1.3 Winning conditions

- If both players win playing Yuki the one that ate fewer trees wins.
- If both players win playing Mina, the one that ate more trees wins.
- If a player wins playing both Yuki and Mina this player is clearly the winner.

1.4 Game start

The Yuki player starts by putting Yuki wherever he wants, eating the tree that is in the selected position.

Then Mina player, puts Mina wherever on any other tree that is not in Yuki's line of sight.It must be at least one tree between Yuki and Mina along an imaginary straight line connecting both places.

After that, starting with Yuki, players alternate turns.

1.5 Players movement

As mentioned before, players move by turns. In each turn, players can move their character one time. That move must be as follows:

- Yuki: moves orthogonally or diagonally to an adjacent tree that has a clear line of sight to Mina. Before moving there, Yuki eats the tree. If Yuki can't make a valid move he loses the game. This happens if:
  - If there are no surrounding positions with trees.
  - None of the surrounding valid positions have a clear line of sight to Mina.

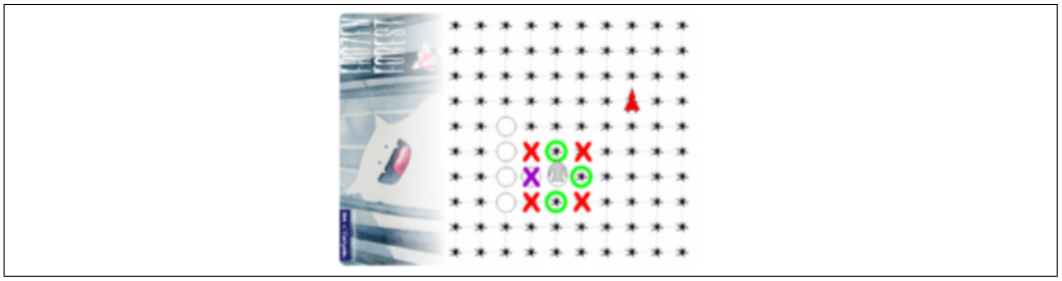

Figure 1: Green spot = Yuki valid spots. He cannot move to the red spots because he would have no clear line of sight to Mina.

- Mina: moves as many spaces as she wants in an orthogonal or diagonal1 straight line, always ending her move in a space (with or without a tree) that has a blocked line of sight to Yuki and at leats one tree in between. She can pass by other trees and empty spaces along the way. If Mina can't make a valid move he loses the game. This happens if there are no trees to witch she can move that have a blocked line of sight to Yuki.

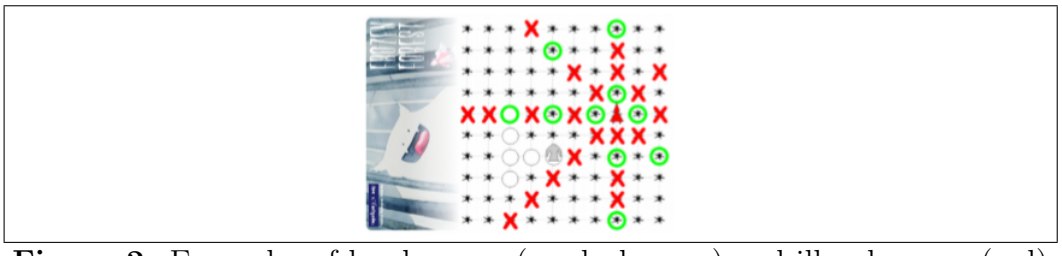

Figure 2: Examples of legal moves (marked green) and illegal moves (red) for Mina.

1.6 Line of sight

Yuki and Mina are in a clear line of sight if there are no trees in the straight line connecting both locations. The tree that Mina occupies does not count as 'blocking' the line of sight (it's not thick enough to fully cover Mina). There will be a clean line of sight between the two if the horizontal and vertical distances are pairwise coprime; in other words, they don't share a common factor other than 1.

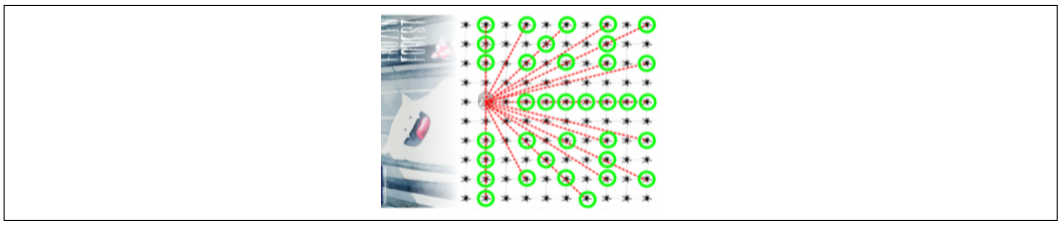

Figure 3: All safe spots for Mina considering only Yuki's current location.

2 Implementation

2.1 Internal game state representation

To internally represent the game I decided to use a 2D Matrix with atoms representing the various game elemets:

- X - cell containing a tree.
- O - cell containing an eaten tree.
- Y - cell containing Yuki.
- M - cell containing Mina.

2.1.1 Board Initial state

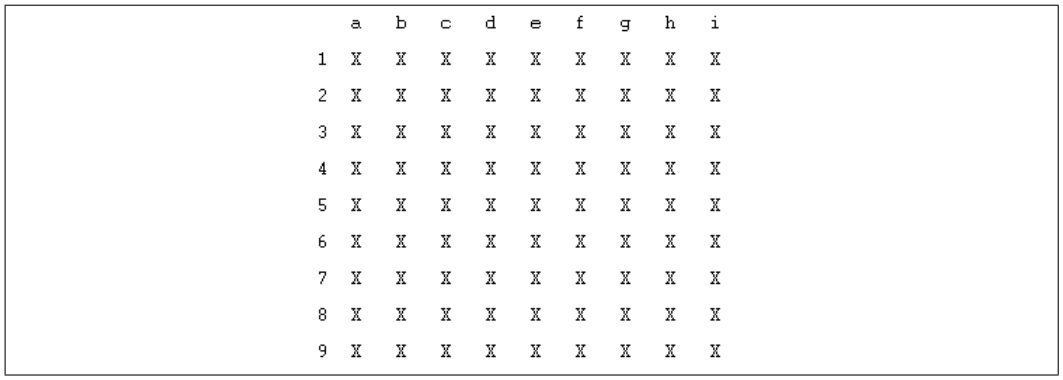

Figure 4: Console view of the initial state of the game board.

```
[
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
]
```

2.1.2 Board after each player first move

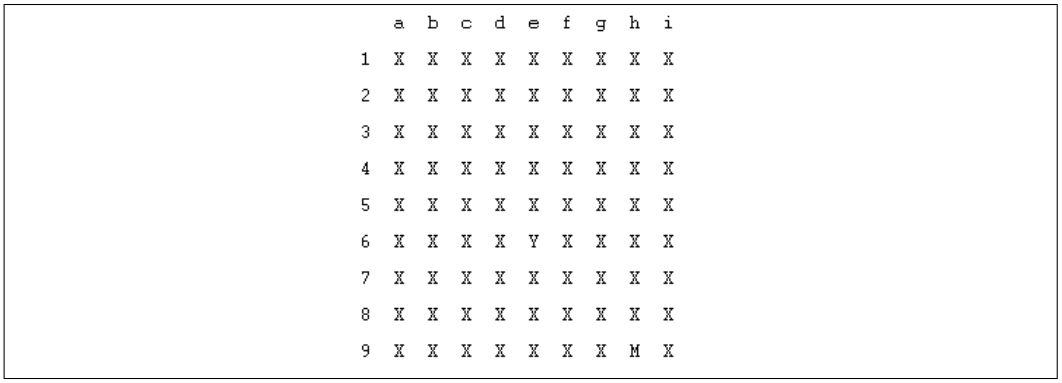

Figure 5: Console view of the board after each player decided their starting points.

```
[
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'Y', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'M', 'X']
]
```

2.1.3 Mid game board

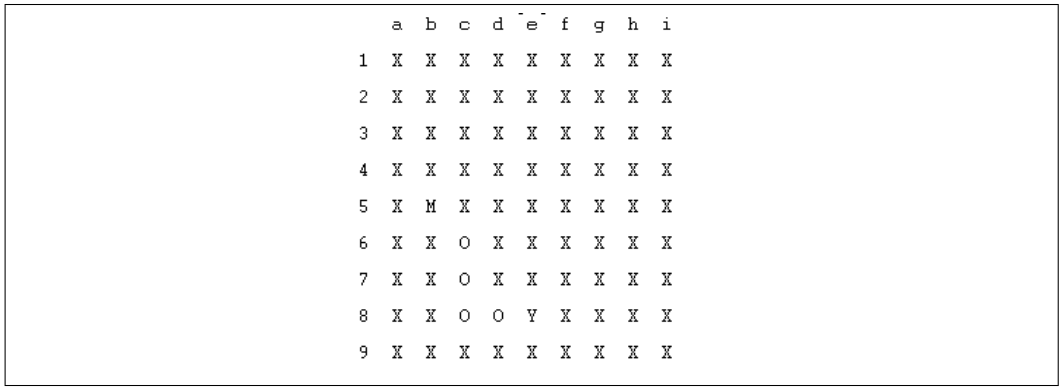

Figure 6: Console view of the board in a mid game state.

```
[
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'M', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'O', 'X', 'Y', 'X', 'X', 'X', 'X'],
['X', 'X', 'O', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'O', 'O', 'Y', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
]
```

2.1.4 End game board

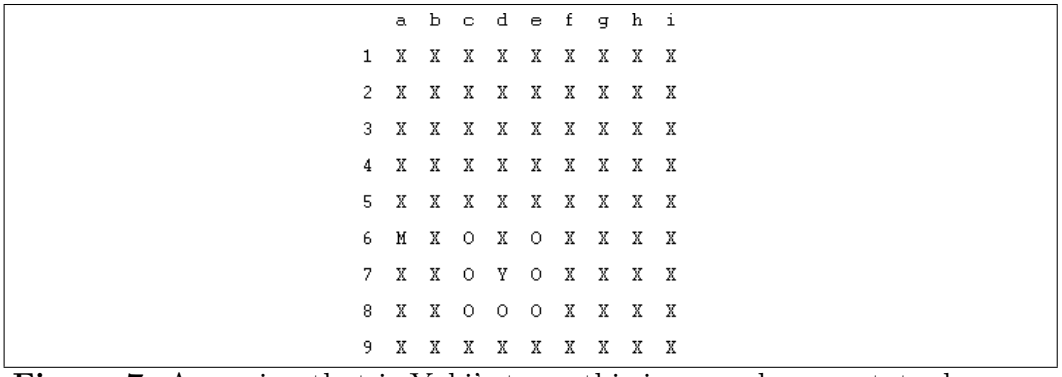

Figure 7: Assuming that is Yuki's turn, this is an end game state, because he can't have a clear line of sight to Mina.

```
[
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
['M', 'X', 'O', 'X', 'O', 'X', 'X', 'X', 'X'],
['X', 'X', 'O', 'Y', 'O', 'X', 'X', 'X', 'X'],
['X', 'X', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
]
```

2.2 Game visualization

In order to see the game and get the images I presented previously, I developed the following code, that probably will be used in this project:

```
rowGuides(['1', '2', '3', '4', '5', '6', '7', '8', '9']).

printColumnGuide :- write(' a b c d e f g h i').

printRowValues([]).

printRowValues([Value | OtherValues]) :-
                                    write(Value),
                                    write(' '),
                                    printRowValues(OtherValues).

printBoardRow(Row, Identifier):-
                            format('~w ', Identifier),
                            printRowValues(Row).

printRestOfTheBoard([], []).

printRestOfTheBoard([Row|OtherRows],[Identifier|OtherIdentifiers]):-
              printBoardRow(Row, Identifier),
              nl,
              nl,
              printRestOfTheBoard(OtherRows, OtherIdentifiers).

display_game(Board) :-
                  printColumnGuide,
                  nl,
                  nl,
                  rowGuides(RI),
                  printRestOfTheBoard(Board, RI),
                  nl.
```