Search Algorithms in Java language (N-puzzle)

Luís Miguel Pedrosa de Moura Oliveira Henriques (up201604343)  
Mestrado Integrado de Engenharia e Computação, MIEIC  
Faculdade de Engenharia da Universidade do PortoSanta Maria da Feira, Portugal  
up201604343@fe.up.pt

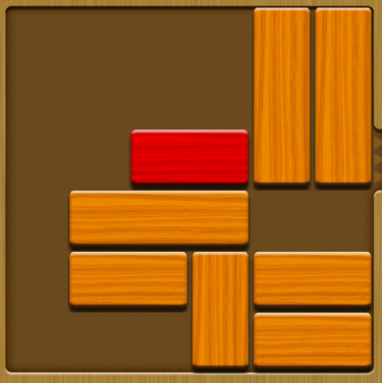
Abstract—In this document we describe a simple game, formulate it as a search problem, solve it with different algorithms, then we analyze the results and how we can benefit from them.

Keywords—Artificial Intelligence, Search, A\* Algorithm, Uniform Cost Algorithm, Greedy Algorithm and Breadth First Algorithm

# Introduction

In this project, an application capable of resolving instances of the game “N-Puzzle” was implemented, without any external interaction, using search algorithms of Artificial Intelligence, namely A\* Algorithm, Uniform Cost Algorithm, Greedy Algorithm and Breadth First Algorithm.

# Problem Description

* The game is played on a 6x6 board, with one special piece (typically 1x2), and several other that can vary in size and orientation, but always 1xY (pieces that move vertically) or Xx1 (pieces that move horizontally). The objective of the game is to move the special piece to the right side of the board, moving the other pieces out of the way to make room for it. The challenge lays in the fact that the small size of the board does not allow the pieces to move much without being obstructed by others, sometimes several pieces need to be moved before making room for a certain one to move just one position.
* Released in 2009, the game is the most popular release of the Thai company Kiragames [2]. Although hard to find evidence, the game resembles an early 20th-century game called Klotski, which was included in Windows3.1’s Windows Entertainment Pack [3].

# Formulation of the problem

State Representation: A board can be represented with a matrix of *x* width and *y* height, whose positions range from (0, 0) on the topmost left to (*x*, *y*) on the bottommost right. Every piece can be represented with letters ranging from “A” to “Z” where every piece that occupies multiple positions is represented with the same letter. Two or more pieces cannot occupy the same position and blank space is represented with a dot (“.”).

Initial State: A board will always start with a key piece facing a target side and there might be pieces obstructing its movement but, by moving the surrounding pieces, it’s always possible to achieve a target state. In the earlier levels played, the key piece is always located on the third row.

Target State: The game ends when a key piece reaches a target side. In the earlier levels played, this target side is always the rightmost of the board and can be achieved if the key piece reaches the matrix position (2, 5) (third row, sixth column). As with most puzzles of this nature, the goal should be to finish a level within the least amount of moves possible.

Operators:

P – Letter of the current piece

xPos – Position of the current piece within the horizontal axis.

yPos – Position of the current piece within the vertical axis.

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Pre-conditions: | Effects: | Costs: |
| MoveUp | yPos > 0;  (xPos, yPos-1)  = “.” | (xPos, yPos-1)  = P;  (xPos, yPos) = “.” | 1 |
| MoveDown | yPos < 5;  (xPos, yPos+1)  = “.” | (xPos, yPos+1)  = P;  (xPos, yPos) = “.” | 1 |
| MoveLeft | xPos > 0;  (xPos-1, yPos)  = “.” | (xPos-1, yPos)  = P;  (xPos, yPos) = “.” | 1 |
| MoveRight | yPos < 5;  (xPos+1, yPos)  = “.” | (xPos+1, yPos)  = P;  (xPos, yPos) = “.” | 1 |

# Related Work

During our research, there were several projects that we found which will likely assist us throughout the course of the development of the project. The first example details the development of a solver of the Unblock Me game [4]. It uses a Breadth First algorithm and it is implemented in C++ [5]. The second project is also an implementation of this game, but with a couple of differences. It is developed using the Python language and the algorithm implemented is the A\* [6].

# game Implementation

This project was developed using the Java programming language. We chose to implement this game in this specific language, not only because all the members already felt comfortable using it, but also because it is one of the most used programming languages in the world. Java is also widely used in the Artificial Intelligence context. Therefore, we believed that utilizing this language would benefit us over all other languages.

Moving on to the overall organization of our game implementation, we can begin by describing our approach to the illustration of the game board. As specified in the third section, a board is represented by a matrix of variable size. Below, we demonstrate how a 6x6 matrix is represented using our implementation.

#AA.GGG#

#...B.F#

#CXXBDF#

#C..BDF#

#C.EEE.#

#......#

Figure 2 – Board Representation

In our implementation, each board is represented in the *Board* class. The code below represents the constructor of this class.

1 public Board(int targetX, int targetY, int height,

2 int width, Piece mainPiece) {

3 *this*.height = height;

4 *this*.width = width;

5 board = new char[*this*.height][*this*.width];

6 for (char[] heightIterator : board) {

7 Arrays.fill(heightIterator, '.');

8 }

9 boardPieces = new ArrayList<>();

10 boardPieces.add(mainPiece);

11 *this*.updateBoard();

12 *this*.targetX = targetX;

13 *this*.targetY = targetY;

14 }

Figure 3 – Board class constructor

Analyzing the code above, the variables *targetX* and *targetY* represent the *x* and *y* coordinates of the target position, *width* and *height* represent the size of the board and *mainPiece* represents the piece that needs to be moved to the target position. Additionally, each *Board* is also associated with a multidimensional array of *chars* (used to print the board in the console and an *ArrayList* which stores every piece that is in the board.

The pieces mentioned in the paragraph above are represented by the Piece class. The following code represents the constructor of this class.

1 public Piece(int x, int y, int size, Boolean

2 IsHorizontal, char identificationLetter) {

3 *this*.size = size;

4 *this*.isHorizontal = isHorizontal;

5 *this*.identificationLetter =

6 identificationLetter;

7 *this*.x = x;

8 *this*.y = y;

9 }

Figure 4 – Piece class constructor

Analyzing the code above, the variables *x* and *y* represent the coordinates of the piece, *size* represents its length and *identificationLetter* represents the char that identifies the piece in the board. If we look at the second figure, we can observe that the X represents the main piece.

In order to move a piece, we’ve decided to create a class Move that, as soon as a new object is instantiated, moves a piece and updates the board. The code below represents the constructor of this class.

1 public Move(Board oldBoard, Piece piece, int

2 distance, char direction) {

3 *this*.oldBoard = oldBoard;

4 *this*.oldPiece = piece;

5 *this*.newBoard = new Board(*this*.oldBoard);

6 for(Piece p : newBoard.getPieces()) {

7 if(p.equals(piece))

8 *this*.newPiece = p;

9 }

10 *this*.distance = distance;

11 *this*.direction = direction;

12

13 movePiece();

14 }

Figure 5 – Move class constructor

Analyzing the code above, the variable *oldBoard* stores the current board, i.e. the board that existed before a piece is moved, *piece* represents the piece that is going to be moved, the variable *distance* represents the number of spaces the piece is going to be moved and *direction* represents the direction the piece is going to be moved in. These directions can be ‘u’ (up), ‘d’ (down), ‘l’ (left), ‘r’ (right). The first two directions can only be used if the piece is in a vertical orientation and the last two can only be used if the piece in a horizontal orientation.

In order to play and access different levels of the game more easily, a parser for analyzing text files containing distinct puzzles was implemented. This parser can read files that follow the format below.

TamanhoX, TamanhoY, ObjectiveX, ObjectiveY; //BOARD

KeyX, KeyY, KeySize, KeyOrientation, KeyChar; //KEY

//PIECES

PieceX, PieceY, PieceSize, PieceOrientation, PieceChar;

PieceX, PieceY, PieceSize, PieceOrientation, PieceChar;

PieceX, PieceY, PieceSize, PieceOrientation, PieceChar;

. . .

PieceX, PieceY, PieceSize, PieceOrientation, PieceChar;

Figure 6 – Puzzle format for parser

The first line contains the information that needs to be sent to the *Board* class constructor. The remaining lines have information regarding all the pieces, although the line immediately after the board declaration is always contains the information of the main piece.

Finally, a Human Mode was also implemented in the *UI* class. This class is only responsible for the interaction with the user, allowing it to solve one of the puzzles by itself.

# Search algorithms

Descrevendo os vários algoritmos de pesquisa utilizados e a sua implementação de modo a calcular a próxima jogada do PC ou retornar a solução final (conjunto de operações para transformar o estado inicial no estado objetivo). Devem ser implementados algoritmos para cálculo da solução utilizando pesquisa em largura, pesquisa em profundidade (se aplicável), aprofundamento progressivo, custo uniforme (se aplicável), pesquisa gulosa e Algoritmo A\* (estes último método utilizando várias heurísticas).

# Experiences and results

In order to easily visualize the results obtained, each one of the tables below will illustrate the outcome of the different algorithms for 5 experiments of different complexity.

Experience 1:

Descrevendo as experiências realizadas com os vários algoritmos para resolver diversos puzzles e os resultados obtidos a nível de tempo e custo da solução obtida em cada nível, por cada um dos métodos experimentados. Devem ser incluídas tabelas comparativas dos resultados obtidos na aplicação dos vários métodos aos vários puzzles (níveis do jogo) e discutidos os resultados.

# Conclusions and Development Prespectives

Sumário do trabalho realizado e conclusões que retira deste projeto. Análise crítica dos resultados obtidos em comparação com os resultados teóricos que seriam esperados. Trabalho futuro, ou seja, formas de melhorar o trabalho desenvolvido.

##### References

1. Stuart Russel and Peter Norvig, “Artificial Intelligence: A Modern Approach”, Third Edtition, Pearson Education Inc., 2010, ISBN: 978-0-13-604259-4.
2. Stuart Russel and Peter Norvig, “AimaCode - Code for the Book Artificial Intelligence: A Modern Approach", 2019, [online], available at <https://github.com/aimacode>, consulted in March 2019.

Livros, artigos e páginas Web utilizados para desenvolver o trabalho. Todos os elementos bibliográficos devem ser citados no texto do trabalho, incluindo qualquer código fonte adaptado de uma dada fonte para a realização do trabalho.