LonPro Puzzle Solver

Abstract

The LonPro solver is a program designed to solve puzzles presented in the LonPro board game. LonPro is the Taiwanese version of the game Quadrillion. Quadrillion is a puzzle game where players are given several different puzzle pieces of different shapes and colors. These puzzle pieces have to all fit onto a game board in order for the player to win.

Introduction

The original idea of this program was for it to be a Quadrillion game solver. The Quadrillion game presents users with different puzzles and then provides them with the answers, similar to a Sudoku booklet or a word search puzzle where the answers can be found in the book. I am more familiar with the Taiwanese version of the game. I would play it when I was younger. LonPro has a different board which is triangle shaped. As far as the puzzles go, LonPro has a somewhat different approach and starts off with simple puzzles that have answers that the player can look up, but then as the player progresses through the booklet the puzzles get more and more difficult. LonPro has a last section of high level puzzles and does not give the player solutions. The LonPro Research Working House even held contests in the past to see if people could figure out these puzzles and submit the answers to the company for chances to win various prizes and badges. While this contest is no longer still going on, I thought it would be interesting to write a program that can solve these higher level puzzles. This document describes the layout of the program and how it works. Below are pictures of LonPro.

Detailed System Description

The system works by printing the board and then filling it with the different puzzle pieces until they are all put in successfully. Matrices are used for the board, game pieces, and puzzle inputs. The system then uses a series of loops to try to put all the pieces in using brute force. The pieces are tried through different orientations and positions until a fit is found. This program

follows the same method that a person would use when trying to solve the puzzle, only the program can do it a lot faster.

User Interface and Graphics

When the program is run, the user is prompted on the screen to enter the input file name. Once the input file name is entered, graphics window is displayed separately. A text output is displayed on the console and also written to an output file. See the "User Manual" section for an example of the output from a sample run.

Abstract Windows Toolkit (AWT) and Swing package are used to display graphics and simple user interface. MouseClick is used to control/toggle graphics display of puzzle solving process. For example, sometimes the graphics display of the game pieces placement movements can prolong the puzzle solving process for certain difficult puzzle problems, the graphics display can be disabled by by clicking the mouse inside the game board window, and the solving process will speed up. The referenced graphics and mouse events tutorials are listed in the References section.

Classes and logic flow

The **gameBoard** class defines the LonPro game board where the game pieces can be placed. It draws windows using JFrame, and exits the program when window is closed.

The **Handlerclass** is defined to implement mouse events for the game board. The mouse click is used to toggle enabling and disabling the display of game pieces placement attempt in the puzzle solving process. Disabling the game piece movement graphics can be useful when the graphics appears to slow down the puzzle solving process too much for certain difficult puzzles. LonPro default is set to have graphics off.

The setBoard method sets initial value and title for the frame. It records the given pieces for puzzle. The setColorSymbol method sets symbol and color of each piece based on the LonPro game. It can be enhanced to support Quadrillion and other similar games. The drawBoard method displays graphics output in addition to text output.

The **GamePiece** class determines the game piece's unique orientations and the footPrint in each orientation.

The buildTrack function builds build a track with stops where the game piece in a particular orientation can actually fit in. The sequenceUnique function determines if an orientation is unique by calling the footPrintEqual function to compare two footPrints to see if they are identical. For example, symmetrical game pieces (v W V U in p in picture on page 7) don't have full set (8) of unique orientations.

The **main** function logic does the following:

- Read input file LPxxx.txt and store input data into an array
- Initialize data. Game pieces will be tried in the same sequence in each run. (Change the code to try different sequences.)
- Draw gameBoard

- Establish a track for each game piece, each orientation. It contains all pit stops that piece, at an orientation, can fit in.
- Measure running time of the solver by storing start/end time
    - Record start time
- Invoke tryPutIn(startingPiece, myGamePiece, myGameBoard)
- When all pieces are tried, record end time and calculate total time. Output to a file.

The tryPutIn function logic does the following:

- Attempt to place a game piece on the game board.
- Loop through each unique orientations for that game piece.
- keepDrawing the game board based on click mouse event.
- *Call tryPutIn function recursively*
- If finished all game pieces, return true
- If we have reached a dead end, remove the last piece tried, then try something else.

UML

gameBoard
+colorSeries: int +symbol: char +onBoard: boolean +howManyOnBoard: int +widthOfRow: int +results: int +gameBoardHeight: int
+gameBoard() constructor +keepDrawing: boolean +setBoard(String[][] inputData, String inputGameTitle): void +setColorSymbol(int[][] specifiedColorSeries, char[] specifiedSymbol): void +drawBoard(boolean solid, boolean outputTextToo): void

Handlerclass
+paintComponent(g: Graphics): void

The Handlerclass extents implements MouseListener, MouseMotionListener.

gamePiece
+unique: boolean +howManyUnique: int +trackH: int

+trackV: int +howManySteps: int +footPrintH: int +footPrintV: int +howManyToes: int +symbol: char
+gamePiece(theSymbol: char, initialShape: int[][]) constructor

Circle
-x: int -x: int -height: int -width: int
+draw(g: Graphics): void

The Circle class extends java.awt.Color, Graphics, Graphics2D, and Ellipse2D classes.

drawingComponent
+paintComponent(g: Graphics): void

The drawingComponet extents JComponent and many other classes.

Requirements

The LonPro puzzle game package did not provide solution to players. This LonPro solver program can solve the puzzle quickly and reliably. A specific puzzle can be fed into the program as an input text file. When following pre-defined syntax rules, additional input text files can be created by the user. Below is an example of an input file (LP324.txt):

```
LonPro324
j,j,_,_,_,_,_,_,_,_
j,_,_,_,_,_,_,_,_,_
j,_,_,_,_,_,_,_,_,_
L,L,L,L,_,_,_,_
_,_,_,L,_,_,_
_,_,_,_,_
_,_,_,_
_,_,_
_,_
_
```

User Manual

When the program is run, the user is prompted on the screen to enter the input file name. Once the input file name is entered, graphics window is displayed separately. A text output is displayed on the console and also written to an output file. Below is some sample output from a run:

----- Example ----------

```
Name of your input file (examples: LP282.txt  LP324.txt):
LP282.txt
You entered: LP282.txt
first line: LonPro282


The Quest:

I_________
I________
I_______
I______
LLLL__
L____
y___
y__
yy
y
Averaging 14.25 steps for each (game piece * orientation)


Finished! Time span: 935 milliseconds

IVVVDDjhhh
IVvvDDjhh
IVvzzxjj
Izzzxxx
LLLLWx
LUUWW
yUWW
yUU
yy
y
```

-----End Example ----------

A few input text files have been supplied with the program. They are listed below:

- LP217.txt
- LP241.txt
- LP282.txt
- LP284.txt
- LP318.txt
- LP319.txt
- LP321.txt
- LP324.txt

The files must locate in the directory where the *src* folder resides. Each of the input files represents a unique puzzle corresponding to a specific game given in the LonPro puzzle booklet. The user

can create additional input text files based on the LonPro Pyramid Creator puzzle booklet, as long as file naming convention and the puzzle text “specification” (syntax) rules are followed.

Input text file name must be in the format of LPnnn.txt. (examples: LP282.txt LP324.txt)
First line identifies the specific puzzle name as laid out in the game booklets. It is used in this program as the title of the game board.

Below is an example of puzzle 324

represented by a input file (LP324.txt):

LonPro324
j,j,_,_,_,_,_,_,_,_
j,_,_,_,_,_,_,_,_
j,_,_,_,_,_,_,_,_
L,L,L,L,_,_,_
_,_,_,L,_,_
_,_,_,_,_
_,_,_,_
_,_,_
_,_
_

A underscore denotes an unoccupied hole or pit where a game piece can be placed in. Specific upper and lower case characters are used to designate positions on the board where the game pieces have been placed at the start of the puzzle. See figure below for the specific character associated with each LonPro game piece. A comma is used as a delimiter.

j J 2x3 Little L

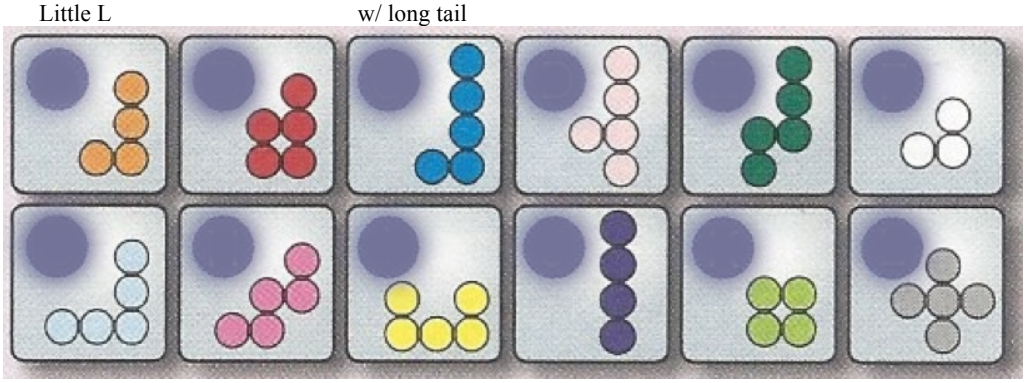
h A wedge

L Big L 4x2, w/ long tail

y Little y,

z Little z

v Little angled



V Big angled,

W Stair,

U U shaped,

I Straight 1x4,

D Square 2x2,

X Little cross

Conclusion

With the LonPro solver, puzzles presented in the LonPro booklet can be solved quickly and accurately.

Some of the additional ideas for enhancements to the program are as follows:

- Allow user to input specific puzzle by using a mouse-to select game pieces and click positions on the game board
- Solve the Quadrillon puzzles
- Solve 3D pyramid puzzles in LonPro, Quadrillion, and other similar puzzle games

References/Bibliography

- LonPro Pyramid Creator user manual by LonPro Research Working House 2002
- Java Graphics basic:
  - Java Programming - Beginning Graphics in Java, https://www.youtube.com/watch?v=llMIT6No7N8 by Isay Katsman, published Mar 21, 2012, Standard YouTube License
  - Dynamic Graphics Object Painting, https://stackoverflow.com/questions/10628492/dynamic-graphics-object-painting May, 2012
- Java Programming Tutorial - Mouse Event, https://www.youtube.com/watch?v=hsHqhX0s7Rs by thenewboston thenewboston, published Oct 10, 2009, Standard YouTube License

How to measure the running time of my program? https://stackoverflow.com/questions/5204051/how-to-calculate-the-running-time-of-my-program Mach, 2011