Logic Puzzle Solver

University of Kentucky

Fall 2012 Masters Project Document

Darra Ricks

INTRODUCTION

The Logic Puzzle Solver (LPS) is a web application that serves as an aid for performing the deductive reasoning necessary to solve logic puzzles and problems. This program allows the user to create a custom logic grid and deduces information based on logic clues entered by the user. Logic Puzzle Solver can solve most puzzles of standard size and structure.

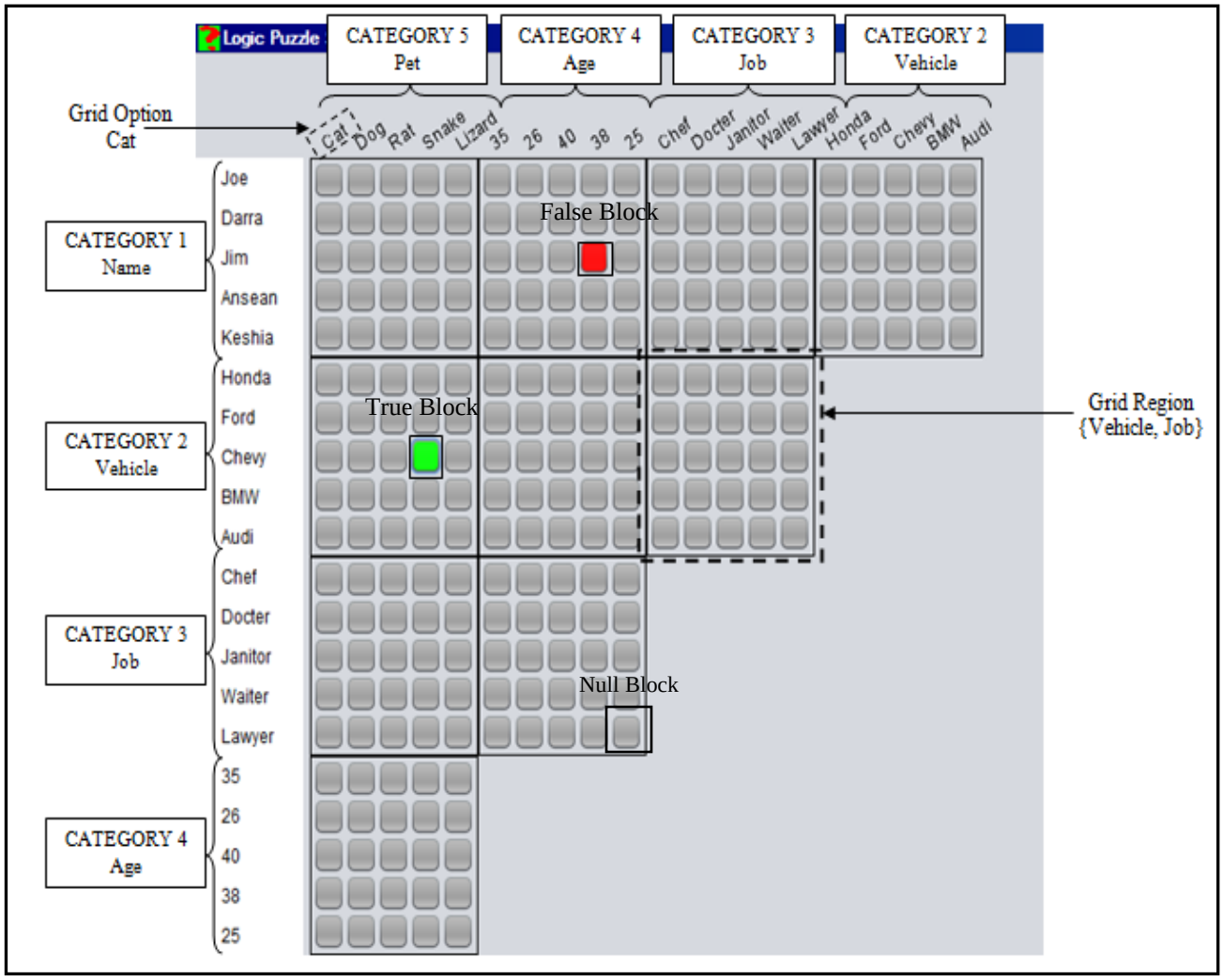
The size of a logic puzzle is C *P where C = Number of Categories and P = Number of Options. A common size for logic puzzles size is (5x5). For example, a puzzle with five categories such as (NAME, VEHICLE, JOB, AGE, and PET) and five options within each category would be considered a 5x5 logic puzzle. The NAME category, for example, could contain the five name options ("Joe", "Darra", "Jim", "Ansean" and "Keshia"). Puzzles of this size are usually ranked with a medium to high level of difficulty. Puzzles of smaller sizes are generally ranked with a difficulty level of medium to low. Standard logic puzzles contain an equal number of options for each category. Enthusiasts often solve logic puzzles by building a grid, as shown in Figure 1. This figure also defines the terms used in this document for the fundamental components of a logic puzzle.

FIGURE 1

Category: A category is a set of options. Every category contains the same number of options.

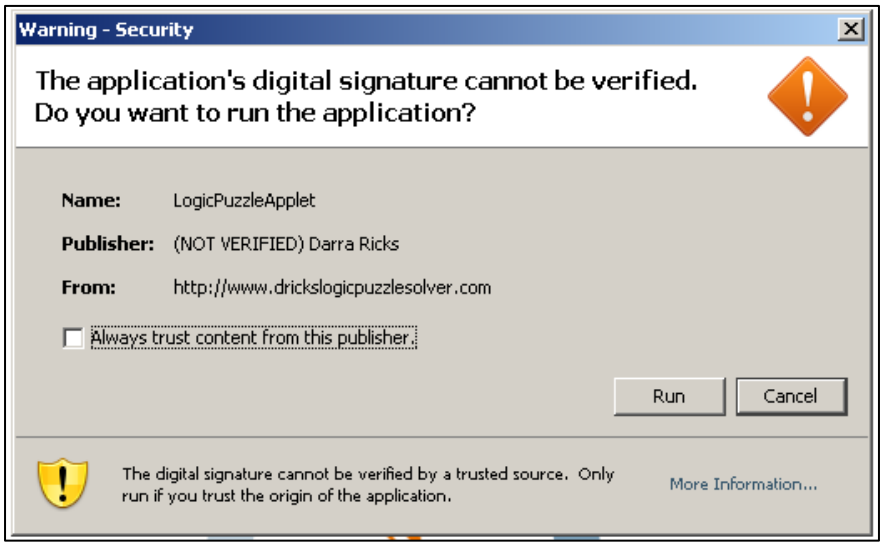
Grid Block: Each grid block corresponds to an option pair. A grid block can be marked true, false or blank. A red (false) grid block that corresponds to options {Jim, 38} means that NAME: Jim IS NOT AGE: 38. A green (true) grid block that corresponds to options {Chevy, Snake} means VEHICLE: Chevy IS PET: Snake.

Grid Region: A grid region contains (P^2) grid blocks. Each grid region corresponds to an individual pair of categories. And each pair of categories is associated with exactly one grid region. The highlighted grid region below corresponds to Category 2 and Category 3 {Vehicle, Job}.



HOW TO USE LPS

LPS is written completely in Java as an applet. It is executed in a Web browser with Java compatibility or by a Java applet viewer. If using a web browser, the user must first grant the application rights to access their machine's local file directory in order for the application to save or open an existing logic puzzle file. The file extension for each logic puzzle file LPS generates is (.lps). The user can grant LPS access rights after seeing the digital certificate the web browser presents upon initialization of the applet. A sample Firefox warning security message is shown below:



By clicking the run button the user agrees to trust LPS.

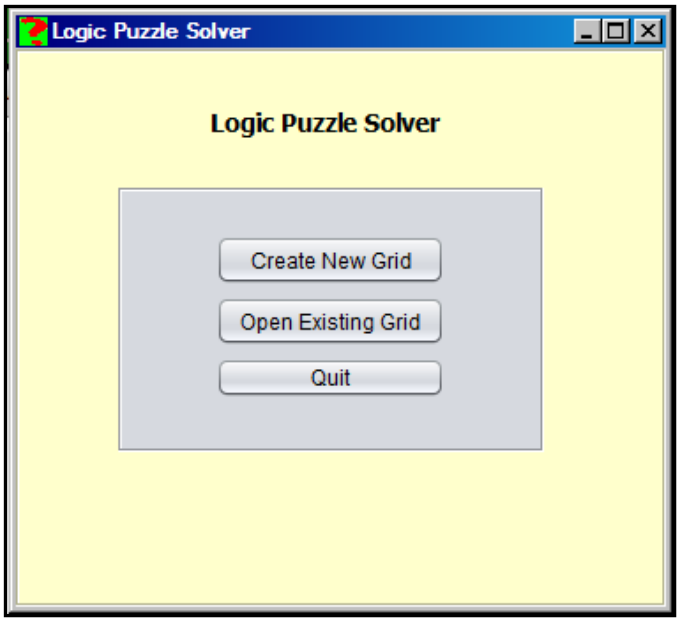
After access is granted, the Home Screen of the application appears. In the Home Screen the user has three choices:

1. Open an existing logic puzzle grid
2. Create a new logic puzzle grid
3. Exit

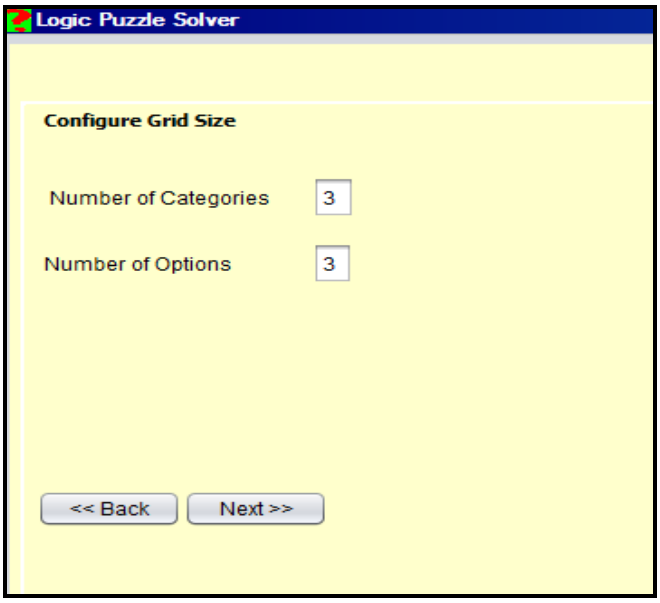
Create a New Logic Puzzle Grid

LPS prompts the user to enter the number of categories and number of options for the puzzle to be solved. After this information has been entered the user enters the names of the categories and the names of the options in each category. LPS then generates a blank logic grid based on this information. Below are screen shots of these steps.

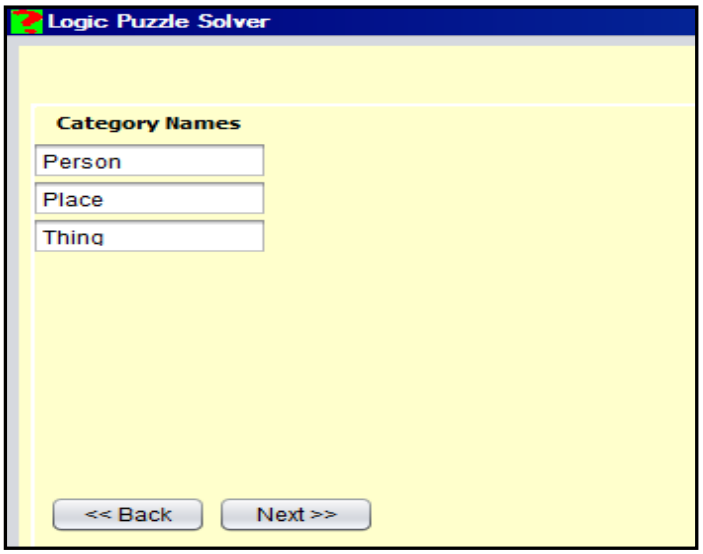
1. Click "Create New Grid"



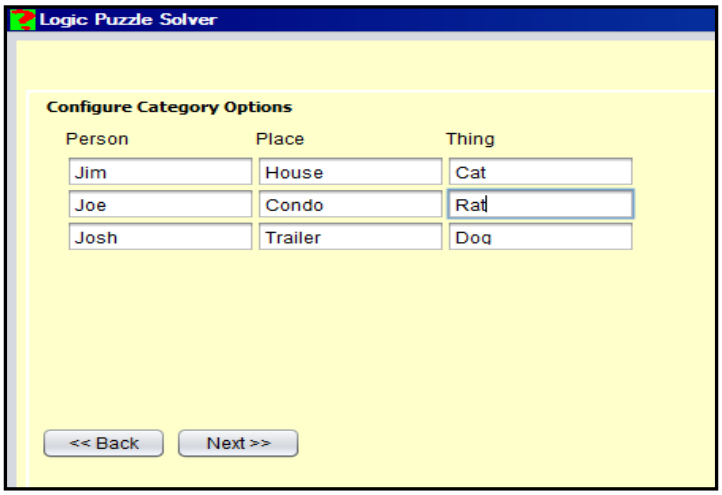
2. Enter the "number of categories" and the "number of options" and then press "Next" Button

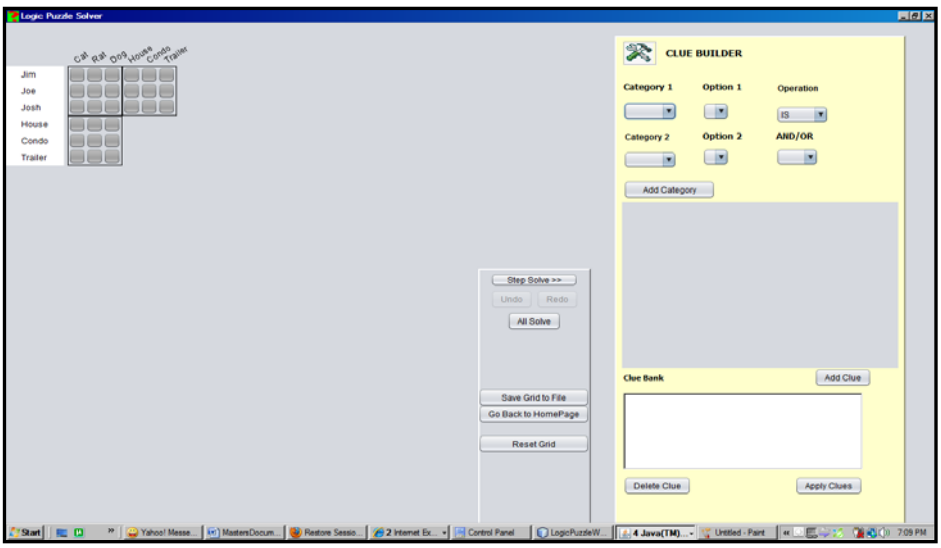


3. Enter the category names and then press “Next”



4. Enter the option names for each category and then click “Next”





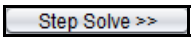
Grid Screen Functions

Undo 

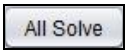
- Undo the last change to the logic grid. LPS keeps a stack of actions in order to undo multiple actions.

Redo 

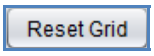
- Redo the last undo change to the logic grid. LPS keeps a stack of actions in order to redo multiple actions.

Step Solve 

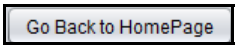
- Deduces new information from the selections currently in the logic grid and updates the logic grid based on this new information

All Solve 

- Attempts to solve the entire logic puzzle based on the selections currently in the logic grid

Reset 

- Clear all relationships chosen on grid by setting every grid block to blank

Exit to Home Screen 

- Prompts user to save current puzzle and navigates to the Home Screen

Save 

- Save current puzzle to a Logic Puzzle Solver .lps file

Grid Button 

- Each grid button has three different settings: blank (gray), false (red), true (green).These settings can be changed by clicking the grid button multiple times.

Clue Builder Functions

In order for LPS to aid in solving a given puzzle enter the known facts or clues of the puzzle. From these clues, information is deduced and represented in the logic grid interface. Enter clues using the clue builder interface pictured in Figure 2. The Clue Builder is on the main grid screen on the far left as shown on page 5.

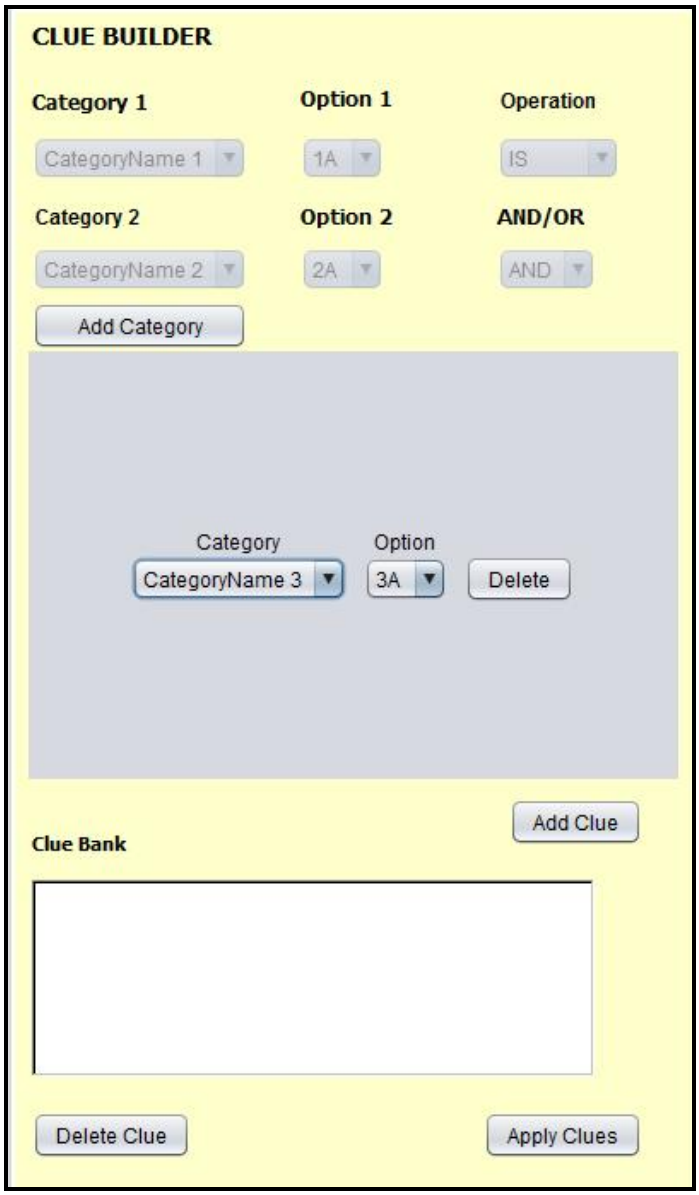


FIGURE 2: Clue Builder Interface

To enter a clue, select the first category from the Category 1 dropdown box, then select the option from the Option 1 dropdown box. Next select an operation from the Operation drop down box. The available operations to select from are IS (=) and IS NOT (!=). Next, select the Category 2 option, then click the Add Clue button. Once the Add Clue button is pressed, the string version of the clue is added to the clue bank.

The clue builder interface can also be used to build composite clues like "Jim IS in the Condo AND he does NOT own a rat". To create a composite clue, choose AND or OR from the AND/OR drop down box and then click the Add Category button. When the Add Category button is pressed, an additional row of category and option drop-down boxes appears. Choose the correct category and option and press the Add Clue button. The clue builder allows the user to create composite clues up to six categories long. Add all necessary clues to the clue bank and press the Apply Clue button to apply these clues to the logic grid.

HOW IT ALL WORKS

Grid Structure

The main logic grid implements four primary Java classes:

1. Block
2. Grid Region
3. Main Grid
4. Clue

Block *(block.java)*

The block object is a single grid block, which can be true, false or blank. This object can return and set its own (true/false/null) value. Also, it can return its grid location. The grid location is the number of the row and column the block belongs to in the grid region.

Grid Region *(gridblock.java)*

The grid region object contains a two-dimensional array of grid block objects. The object's constructor takes the number of options in each category, as input, which allows it to allocate a sufficient number of grid block objects upon initialization. The grid region object contains two primary methods: Auto Complete and Reset. The Auto Complete method sets all blocks in the same row and column as a true block to false. The reset method sets all blocks in the grid region object to null.

Main Grid *(maingrid.java)*

The main grid object contains a 2-dimensional jagged array of grid-region objects. This object contains (C -1) rows of grid regions. The number of grid regions in each row r is C - r. For example, a four-category main grid object has three rows of grid regions: 3 grid regions in the first row, 2 grid regions in the second row and 1 grid region in the third row. The main grid object contains methods to save the logic grid to a file, reset the entire grid, undo and redo recent changes to the grid, deduce new information from existing information selected in the logic grid and exit to the homepage of the application.

Clue *(clue.java)*

This object stores a clue entered by the user through the clue builder. A clue consists of the following: a base category, base option, an IS/IS NOT Boolean choice and at least one additional category/option pair. The base category/option pair is the component of the clue which all other category/option pairs in the clue build upon.

For example, in the clue:

"John IS NOT tall" = (PERSON, John) IS NOT (CHARACTERISTIC, Tall)

The base category is PERSON and the base option is John. Additional category 1 is CHARACTERISTIC and additional option 1 is Tall. Therefore, the pairs (PERSON, John) and (CHARACTERISTIC, Tall) are created. The boolean choice in this clue is "IS NOT".

Additional categories are used for composite clues, such as:
”John is NOT tall or obese or from KY”

The additional categories vector component stores these additional categories. Every composite clue must use an AND or an OR Boolean option. In LPS OR is inclusive when using IS NOT and exclusive using IS.

For example:
“John is NOT tall or obese or from KY” = (PERSON, John) IS NOT (CHARACTERISTIC, Tall) OR (CHARACTERISTIC, Short) OR (STATE, KY)

A clue can be either pending or active. An active clue’s information can be entered immediately into the logic grid. Pending clues cannot be applied to the grid immediately because they require more information, because they are composite.

An example of a pending clue is as follows:
“John IS tall OR from KY OR drives a Mazda”

More information is needed before this clue can be applied because IS/OR means John is exactly one of the options listed. As more information is entered into the logic grid, pending clues are updated. If the user enters into the grid”John IS NOT tall” then (CHARACTERISTIC, Tall) removes from the pending clue’s additional category and additional options vectors. Once the pending clue has only one category/option pair remaining the clue becomes active and can be applied to the grid.

Another component of the clue object is the associate clue vector. The associate clue vector applies when all the additional categories in an IS/OR composite clue are the same.
For example:
“John is tall or obese or short”

All these option choices come from the same category CHARACTERISTIC. Therefore, it is derived that John IS NOT the remaining unlisted options within this category. If the CHARACTERISTIC category contains options (obese, tall, short, stocky and slim) it is derived

that “John IS NOT stocky OR slim”. This derived clue is an associate clue to the main clue “John is tall or obese or short”. Associate clues are active and are applied immediately to the logic grid. Finally, the clue object is serializable, so it is saved to a file in its current instance state. When the user presses the Save Grid button, a file creates, containing a list of all the clue instances in the current grid. The file extension of the clue data file is .clue_data and the file name is the same file name as the logic puzzle solver file (.lps).

Grid Solving Logic

The logic grid is solved by propagating equality relations as in the following:

A = B, B = C => A = C

For example, if we have a simple four-category puzzle to solve with four options in each:

Person	**Place**	**Thing**	**Pet**
Jim	Condo	Thing_1	Cat
Joe	House	Thing_2	Dog
Mike	Trailer	Thing_3	Rat
Jimmy	Tent	Thing_4	Mouse

And the given facts are:

Jim = Thing_3, Jim != House and Jim != Cat

Propagation deduces that since Jim = Thing_3 then Thing_3 must also not be equal to House and Cat so:

Jim = Thing_3 => Thing_3 !=House and Thing_3 != Cat

Say we also know another fact:

Thing_3 = Trailer

this means,

Thing_3 = Trailer => Trailer = Jim and Trailer !=Cat

The logic grid in this application is solved by propagating existing information to corresponding category regions horizontally and vertically.

During vertical propagation the application examines each row of every grid region for true blocks. A true block means that an option from a category 1 is equal to an option from a category 2. Since option1 = option2, every (true/false) choice in option 1 propagates to the blocks in the option 2 row and every (true/false) choice in the option 2 row propagates to the option 1 row. Also, every blank block in the same row or column as the true block at hand is set to false.

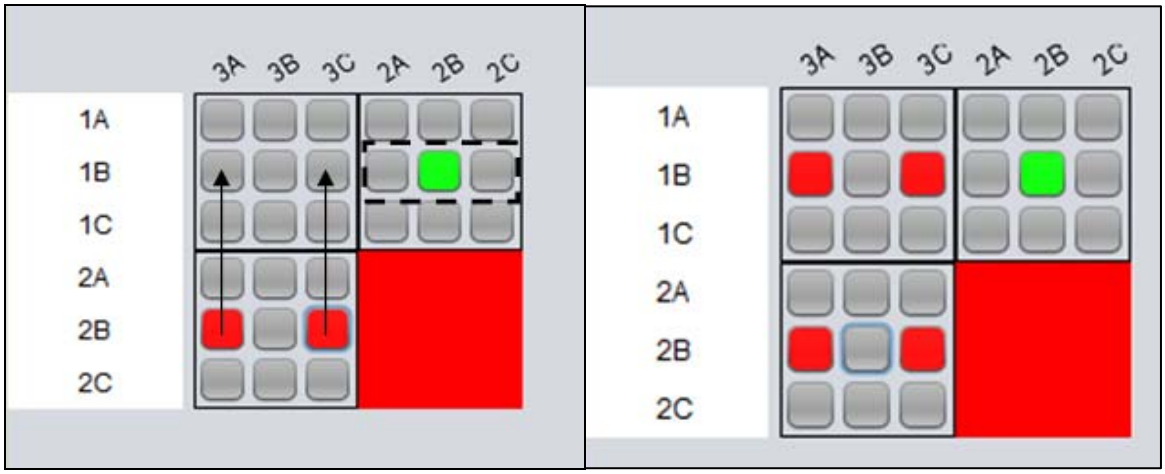


FIGURE 3: Example 1 of Vertical Propagation in Logic Grid

2B = 1B , 2B !=3A, 2B != 3C => 1B != 3A 1B != 3C, 1B != 2A, 1B != 2C

Next, each row in every grid region is examined for false blocks. If a false block is found, an option from a category 1 is NOT equal to an option in a category 2. Therefore, if there are any true choices in the corresponding rows for option 1 (from category 1), these need to be set to false in the corresponding rows for option 2.

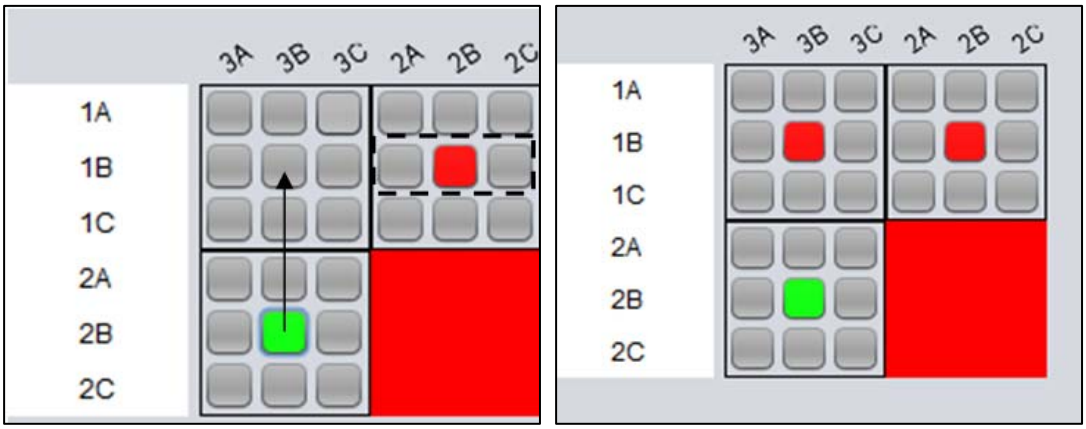


FIGURE 4: Example 2 of Vertical Propagation

2B != 1B, 2B=3B => 3B !=1B

Horizontal propagation is similar to vertical propagation, but instead of examining the rows within the grid regions, examine each column. The deduced information propagates across to the corresponding columns instead of rows.

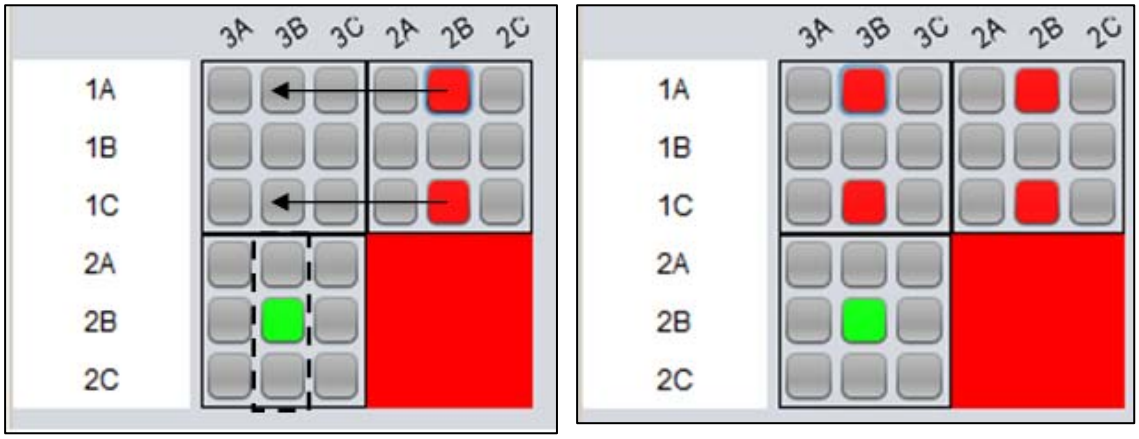


FIGURE 5: Example 1 of Horizontal Propagation

2B = 3B, 2B != 1A, 2B != 1C => 3B != 1A, 3B != 1C

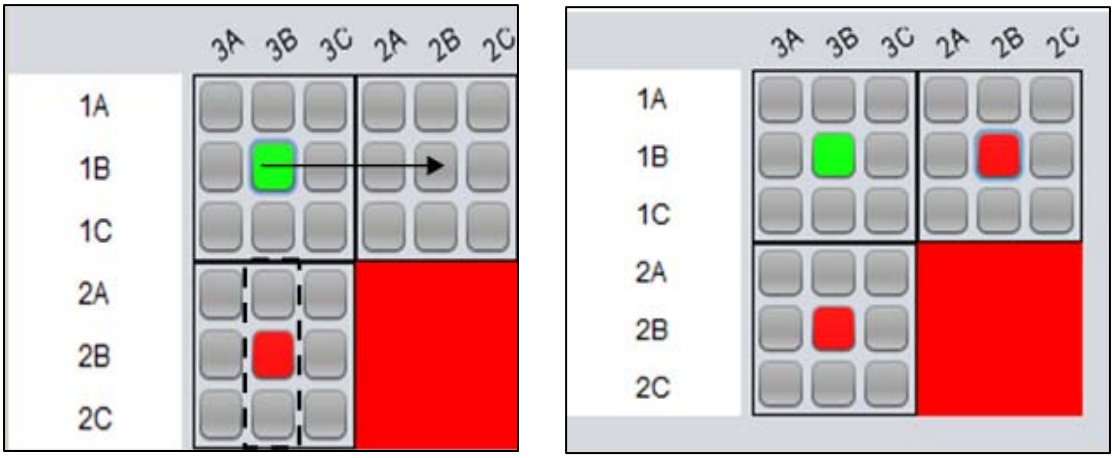


FIGURE 6: Example 2 of Horizontal Propagation

2B != 3B, 3B = 1B => 2B !=1B

The application also recognizes conflicts in the logic grid. Figure 7 demonstrates an example of such a conflict.

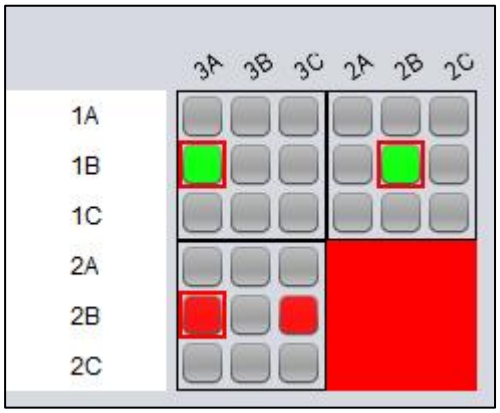


FIGURE 7: Example of Conflict in Logic Grid

2B = 1B, 2B != 3A, 2B!=3C => 1B !=3A

In the illustration 2B = 1B, 2B != 3A and 2B!=3C => 1B !=3A but the grid says that 1B = 3A. This is a conflict and the application always notifies the user of such conflicts.

By recursively performing the above horizontal and vertical propagation and resolving logical conflicts this application solves most logic puzzle grids completely.

Demonstration Example

To demonstrate the functionality of this application a logic puzzle entitled “Lazy Final Saturday” from http://www.braingle.com/brainteasers/45491/lazy-final-saturday.html is used. The puzzle is as follows:

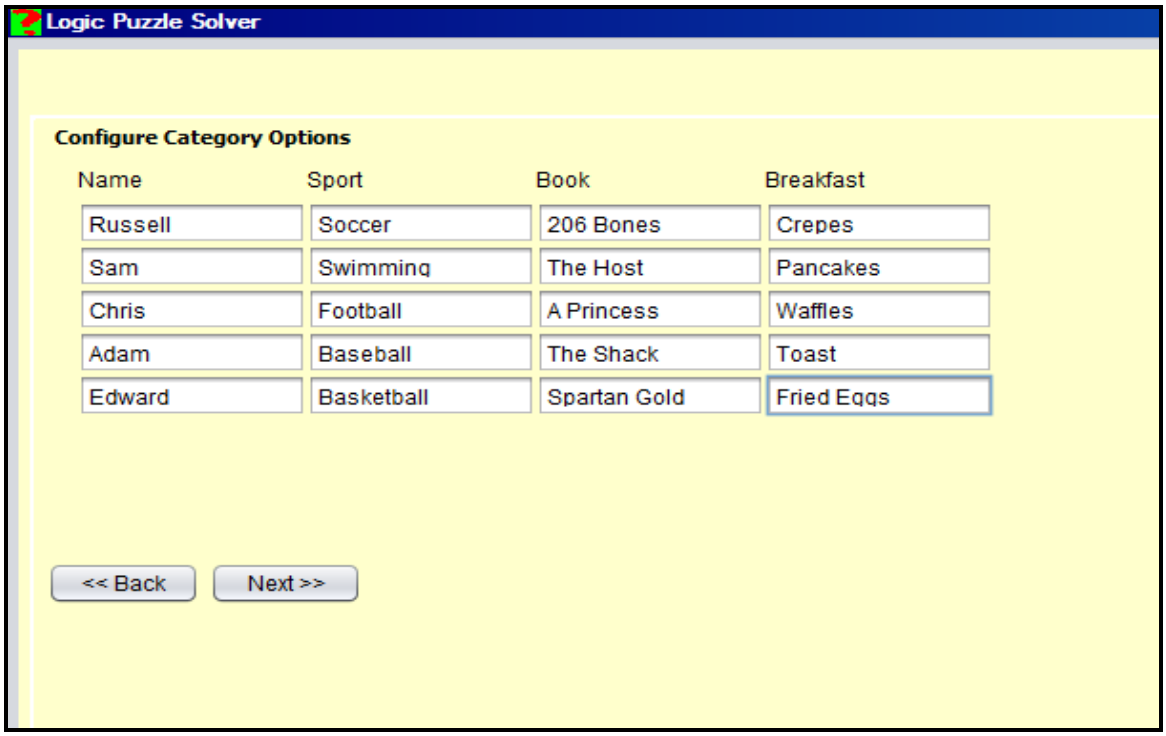
The Saturday after final's week, five roommates were feeling lazy. So they decided to wake up late, eat breakfast, read a book, and then watch a sport on the TV. From the clues given can you determine who ate what breakfast, the book they read, and the sport they watched.

1) Sam White watched soccer, while the basketball watcher had crepes.
2) Adam Brown read The Shack.
3) Edward Black had toast for breakfast; he didn't read The Host or watch football.
4) Fried eggs and 206 Bones took up most of the morning of one of the roommates, but not Sam
5) Football was watched after reading Spartan Gold.
6) The reader of The Host didn't eat pancakes, and the reader of the A Princess of Landover didn't watch baseball.

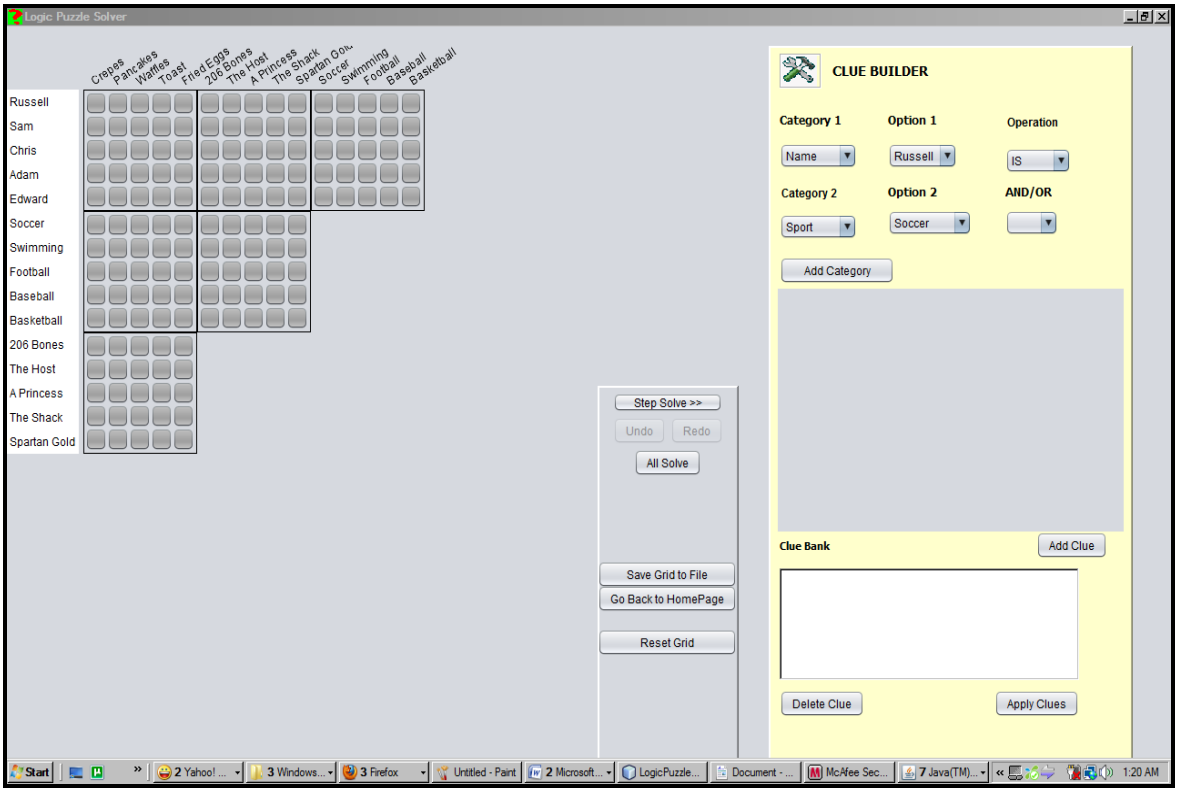
STEPS

1. Navigate to application *http://www.drickslogicpuzzlesolver.com/LogicPuzzle.html*
2. Click the Launch Logic Puzzle Solver link
3. Click the Create New Grid Button
4. Enter the number of categories and options, this puzzle has 4 categories and 5 options
5. Press the Next button
6. Enter the Category names (Name, Sport, Book, Breakfast)
7. Press the Next button
8. Enter the following option names in the corresponding text fields.

Name	**Sport**	**Book**	**Breakfast**
Russell	Soccer	206 Bones	Crepes
Sam	Swimming	The Host	Pancakes
Chris	Football	A Princess	Waffles
Adam	Baseball	The Shack	Toast
Edward	Basketball	Spartan Gold	Fried Eggs



9. Press the Next button to navigate to the main grid screen



10. Enter the following clues into the logic grid using the Clue Builder User Control

Each of the following clues can be broken down into multiple clues for entry into the Clue Builder. After entering a clue, press the Add Clue button.

Clue 1

Sam White watched soccer, while the basketball watcher had crepes.

- Sam IS Soccer
- Basketball IS crepes

Clue 2

Adam Brown read The Shack.

Adam IS The Shack

Clue 3

Edward Black had toast for breakfast; he didn't read The Host or watch football.

- Edward IS toast
- Edward IS NOT The Host OR football

Clue 4

Fried eggs and 206 Bones took up most of the morning of one of the roommates, but not Sam White or Russell Green.

- Fried eggs IS 206 Bones
- Sam IS NOT fried eggs OR 206 Bones
- Russell IS NOT fried eggs OR 206 Bones

Clue 5

Football was watched after reading Spartan Gold.

Football IS Spartan Gold

Clue 6

The reader of The Host didn't eat pancakes, and the reader of the A Princess of Landover didn't watch baseball.

- The Host IS NOT pancakes
- A Princess IS NOT baseball OR pancakes

11. After all clues have been added to the Clue Builder clue bank. Press the Apply Clues button
12. Press the All Solve Button and LPS should solve the entire puzzle completely.

The solution to the example puzzle is as follows

Name	Sport	Book	Breakfast
Russell	Football	Spartan Gold	Pancakes
Sam	Soccer	The Host	Waffles
Chris	Baseball	206 Bones	Fried Eggs
Adam	Basketball	The Shack	Crepes
Edward	Swimming	A Princess	Swimming

Pending and Associate Clue Demonstration

Person	Place	Thing
Jim	KY	Bottle
Keshia	OH	Mouse
Sally	GA	Phone
Roger	FL	Pencil

As mentioned earlier, LPS handles certain clues that require more information before being applied to the logic grid. To demonstrate this function, create a new grid based on the three-category table above. Use the Clue Builder to enter the following IS/OR composite clue:

"Keshia IS KY OR Bottle OR GA OR Pencil"

Click the Apply Clues button. Notice that no grid blocks select which correspond to this clue. The clue cannot apply immediately since it is an IS/OR clue composed of two different categories Place and Thing.

Now set the false grid blocks that correspond to:
"Keshia IS NOT KY"
"Keshia IS NOT Pencil"
"Keshia IS NOT GA"

And press the Step Solve button

The selection “Keshia IS Bottle” should now fill the logic grid, which means LPS has applied the pending clue to the logic grid, since there is no more information to be obtained for the clue.

Press the Reset Grid button and delete all clues in Clue Builder clue bank by clicking the clue entry and pressing the Delete Clue button. Now to demonstrate the associate clue functionality enter the following clue into the logic grid using the Clue Builder control.

“Keshia IS GA OR KY”

Apply this clue to the logic grid by pressing the Apply Clue button. Notice how the grid blocks which correspond to “Keshia IS NOT FL” and “Keshia IS NOT OH” are selected in the logic grid. This is an example of an associate clue. Since Keshia can only be either GA or KY the associate clue formed is:
“Keshia IS NOT FL OR OH”

LESSONS LEARNED

Since this was my first time working with a project that required an extensive user interface, I learned about general user interface principles such as structure and visibility. Structure requires an organized and meaningful user interface design laid out in such a way that it is apparent to the users where functionality is located. Visibility means a user interface in which all necessary user components for a task are evident, visible and distinguishable from any user components that are unnecessary for a given task.

I wrote and organized the project source code using the Netbeans Integrated Development Environment. Therefore, I learned how to manage a project of significant size using this development environment. Netbeans provides a user-friendly interface for a forms designer, which is quite useful for basic form creation. This project also enhanced my knowledge of Java object oriented programming and the Java Swing Library.

Another new process I learned was “signing” a Java applet. For its saving functionality, the applet accesses the file directory of the users machine through a web browser. The signing process begins by first packaging the applet’s classes in a .jar file. A JAR file (**J**ava **Ar**chive) is a compressed file which can contain many files, such as all the classes and data needed for the applet. Compressing a java application’s class files into a single JAR makes it easier to distribute the application. Once the applet’s jar file is created, a public/private key pair must be generated

using the keytool utility provided by the Java Sun Development kit. An applet can be self signed by the developer or it can be signed by a Certificate Authority. For self signing, the keytool utility generates a private and public key based on an alias name and password provided by the developer or Certificate Authority. This key is stored in a file called a keystore file. After the keystore file is generated, a digital public certificate must be associated with the keystore file. A digital certificate is a statement signed digitally using the alias and password for the keystore file. Finally, the JAR should be signed using the signed certificate and the public/private key pair contained within the keystore file. This task is done using the jarsigner tool also provided by the Java Sun Development Kit. Jarsigner uses the private key from the keystore file and a unique string of bits from data within the JAR to create a digital signature. Jarsigner signs each file within the JAR file with this signature and the "signing" process is completed.

CONCLUSION

One of the major limitations of LPS is that the user interface is configured to handle logic puzzles with no more than 6 categories and 6 options. Although logic puzzles of sizes larger than 6 x 6 are uncommon, it would be interesting to be able to create and solve larger logic puzzles with LPS. Also, since the level of difficulty increases with the size of the logic puzzle there is a greater need for the aid of this application to solve puzzles of larger size. The size of the puzzle is limited mainly due to time constraints of this project. A puzzle of a larger size than 6x6 would create a larger grid than the current configuration of the user interface can handle.

Many logic puzzles have clues that specify quantities and how much a certain option 1 is greater than or less than another option 2. Take for example a puzzle with categories PERSON (Ansean, Gloria, Darra, Troy), WEIGHT (150 lbs, 180 lbs, 233 lbs, 152 lbs), POSITION (1,2,3,4) and AGE (23,33,40,17). An example clue for this type of logic puzzle would be "Ansean is heavier than the person who is standing two positions to the left of a person who is heavier than Gloria". LPS cannot process clues based on quantities. To implement this into the application, categories which are numeric would be flagged as such prior to generating the logic grid. Either the user or the application itself could set this numeric flag. Once the categories are marked as numeric then some type of quantity comparison mechanism must be implemented in the deductive reasoning algorithm of the application. The numeric operators <, >, = and +/- (a numeric amount) should be included in the clue builder interface.

The (+/-) operation would be used to process clues such as:

"Bill is 7 years older than Tom"

(PERSON, Bill) +(AGE,7) (PERSON, Tom)

It also would be beneficial to add a MATCH clue type to LPS. Add the category SPORT(Swimming, Basketball, Baseball, Tennis) to the example puzzle This MATCH clue type would handle clues such as "Troy and Gloria are, in some order, the swimmer and the person who is 23 years old". The MATCH clue type would signal LPS to exclude all other PERSON options except Troy and Gloria to match with option SPORT(swimming) and AGE(23). Another feature that could be added to the application is a logic puzzle database so that the user can simply load an existing puzzle instead of using a logic puzzle from an outside source such as a book or the internet. Enhancing this application to include such functionality would definitely be advantageous because it would decrease the level of work for the user.