CAP 4621

Final Report

Numbrix

Carlos Vasquez

11/23/2013

Table of Contents

I. Description of Game

The game implemented is called Numbrix. The goal of the game is to fill out an N by N grid with numbers between one and $N \times N$ and each number can only be used once. The player is given this grid with some of the cells already populated with a number. From there, the player must attempt to fill in the entire grid by finding every increasingly or decreasingly consecutive number in the vertical or horizontal directions (no diagonal directions) from a cell. Once the player completes the grid, the player should be able to start from the cell with value one and trace a non-terminating line of consecutive numbers in a non-diagonal direction up until the line reaches the number $N \times N$.

II. Description of Implementation Approach

Considering the competitive nature of the project, there was a focus on speed and memory. The goal was to solve the Numbrix grid quickly while consuming as little resources as possible. Due to the use of recursive objects in the program, I ended up utilizing multiple static variables to help reduce the memory that needs to be consumed when solving the grid.

III. Description of Programs, Procedures, Methods and Variables

Package Summary		*Page*
numbrixgame		*3*
numbrixgame.gui		*5*
numbrixgame.gui.leftdisplay		*10*
numbrixgame.gui.menubar		*14*
numbrixgame.system		*17*
numbrixgame.system.solver		*29*

Package numbrixgame

Class Summary		*Page*
numbrix	Numbrix will be the "main" of the project.	*3*

Class numbrix

numbrixgame

```
java.lang.Object
  └numbrixgame.numbrix
```

Direct Known Subclasses:

SearchTest

```
public class numbrix
extends Object
```

Numbrix will be the "main" of the project. It will construct the components needed to run and then run the program.

Field Summary	
protected static GUI	**gui**
protected static NumbrixSystem	**system** Class Attributes

Constructor Summary
numbrix()

Method Summary	
static GUI	**gui**() Returns gui
protected static void	**initializeSystem**() Initializes system
protected static void	**initializeUI**() Initializes gui
static void	**main**(String[] args) Class Methods
static NumbrixSystem	**system**() Returns system

Field Detail

system

`protected static NumbrixSystem system`

Class Attributes

gui

`protected static GUI gui`

Constructor Detail

numbrix

`public numbrix()`

Method Detail

main

`public static void main(String[] args)`

Class Methods

system

`public static final NumbrixSystem system()`

Returns system

Returns:

system

gui

```
public static final GUI gui()
```

Returns gui

Returns:

gui

initializeUI

```
protected static final void initializeUI()
```

Initializes gui

initializeSystem

```
protected static final void initializeSystem()
```

Initializes system

Package numbrixgame.gui

Class Summary		Page
BottomDisplay	BottomDisplay will take care of the bottom display in the GUI.	*5*
GUI	GUI will be the gui that acts as the "view" for Numbrix	*5*
HistoryDisplay	HistoryDisplay will display the history of the game	*7*
Table	Table will be the table created that will act as the UI for Numbrix.	*8*

Class BottomDisplay

numbrixgame.gui

```
java.lang.Object
  └java.awt.Component
      └java.awt.TextComponent
          └java.awt.TextArea
              └numbrixgame.gui.BottomDisplay
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, Serializable

```
public class BottomDisplay
extends TextArea
```

BottomDisplay will take care of the bottom display in the GUI. This will contain a text box that will be used to display messages to the user.

Field Summary	
private static long	**serialVersionUID** Class Cosntants
private static String	**START**

Constructor Summary
BottomDisplay() Class Methods

Field Detail

serialVersionUID

```
private static final long serialVersionUID
```

Class Cosntants

START

```
private static final String START
```

Constructor Detail

BottomDisplay

```
public BottomDisplay()
```

Class Methods

Class GUI

numbrixgame.gui

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └numbrixgame.gui.GUI
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, RootPaneContainer, Serializable, TransferHandler.HasGetTransferHandler, WindowConstants

```
public class GUI
extends JFrame
```

GUI will be the gui that acts as the "view" for Numbrix

Field Summary	
private BottomDisplay	**bottom**

private static int	DEFAULT_CLOSE_OP
static int	HEIGHT
private HistoryDisplay	history
private LeftDisplay	left
private static String	NAME
private static long	serialVersionUID Class Constants
private Table	table Class Attributes
static int	WIDTH

Constructor Summary
GUI() Class Methods

Method Summary	
void	addLeftDisplay(NumbrixSystem.Player playerType) Creates the LeftDisplay to be used given the playerType
void	addTable(int tableSize, boolean[][] staticData, Integer[][] startData) Overrides the current table with the startData and staticData
void	changeHistory(String newHistory) Change history to show the provided text
Table	getTable() Returns table
private void	initializeUI() Creates the basic UI
void	printMessage(String message) Sets the message as the text for bottom
void	removeLeftDisplay() Removes the contents in left and then removes left
void	removeTable() Removes the table from GUI
void	revalidateTable() Renders the table

Field Detail

serialVersionUID

private static final long **serialVersionUID**

Class Constants

NAME

private static final String **NAME**

WIDTH

public static final int **WIDTH**

HEIGHT

public static final int **HEIGHT**

DEFAULT_CLOSE_OP

private static final int **DEFAULT_CLOSE_OP**

table

private Table **table**

Class Attributes

bottom

private BottomDisplay **bottom**

left

private LeftDisplay **left**

history

private HistoryDisplay **history**

Constructor Detail

GUI

public **GUI**()

Class Methods

Method Detail

printMessage

```
public void printMessage(String message)
```

Sets the message as the text for bottom

Parameters:

`message` - the message being set

addTable

```
public void addTable(int tableSize,
                     boolean[][] staticData,
                     Integer[][] startData)
```

Overrides the current table with the startData and staticData

Parameters:

`tableSize` - the size of the table

`staticData` - the non-modifiable cells

`startData` - the data to populate

removeTable

```
public void removeTable()
```

Removes the table from GUI

revalidateTable

```
public void revalidateTable()
```

Renders the table

addLeftDisplay

```
public void addLeftDisplay(NumbrixSystem.Player playerType)
```

Creates the LeftDisplay to be used given the playerType

Parameters:

`playerType` - the type of player (COMPUTER or HUMAN)

removeLeftDisplay

```
public void removeLeftDisplay()
```

Removes the contents in left and then removes left

changeHistory

```
public void changeHistory(String newHistory)
```

Change history to show the provided text

Parameters:

`newHistory` - the text to be shown in history

initializeUI

```
private final void initializeUI()
```

Creates the basic UI

getTable

```
public Table getTable()
```

Returns table

Returns:

table

Class HistoryDisplay

numbrixgame.gui

```
java.lang.Object
  └java.awt.Component
      └java.awt.TextComponent
          └java.awt.TextArea
              └numbrixgame.gui.HistoryDisplay
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, Serializable

```
public class HistoryDisplay
extends TextArea
```

HistoryDisplay will display the history of the game

Field Summary

private static long	**serialVersionUID** Class Cosntants

Constructor Summary
HistoryDisplay()   Class Methods

Method Summary	
void	**setText**(String text)

Field Detail

serialVersionUID

`private static final long serialVersionUID`

Class Cosntants

Constructor Detail

HistoryDisplay

`public HistoryDisplay()`

Class Methods

Method Detail

setText

`public void setText(String text)`

Overrides:

`setText` in class `TextComponent`

Class Table

numbrixgame.gui

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JTable
                  └numbrixgame.gui.Table
```

All Implemented Interfaces:

Accessible, CellEditorListener, EventListener, ImageObserver, ListSelectionListener, MenuContainer, RowSorterListener, Scrollable, Serializable, TableColumnModelListener, TableModelListener, TransferHandler.HasGetTransferHandler

```
public class Table
extends JTable
```

Table will be the table created that will act as the UI for Numbrix.

Field Summary	
private static long	**serialVersionUID** Class Constants
private boolean[][]	**startData** Class Attributes

Constructor Summary
Table(int tableSize, boolean[][] staticData, Integer[][] grid)   Class Methods

Method Summary	
Integer[][]	**getGrid**() Returns a 2D array that represents the table
boolean	**isCellEditable**(int row, int column)
private void	**populate**(int tableSize, Integer[][] grid) Populates the table with the given grid
void	**setValueAt**(Object value, int row, int column)
void	**setValueAt**(Object value, int row, int column, boolean modifyGrid)

Field Detail

serialVersionUID

`private static final long serialVersionUID`

Class Constants

startData

`private boolean[][] startData`

Class Attributes

Constructor Detail

Table

```
public Table(int tableSize,
             boolean[][] staticData,
             Integer[][] grid)
```

Class Methods

Method Detail

isCellEditable

```
public boolean isCellEditable(int row,
                              int column)
```

Overrides:

`isCellEditable` in class `JTable`

getGrid

```
public Integer[][] getGrid()
```

Returns a 2D array that represents the table

Returns:

a 2D array that represents the table

setValueAt

```
public void setValueAt(Object value,
                       int row,
                       int column)
```

Overrides:

`setValueAt` in class `JTable`

setValueAt

```
public void setValueAt(Object value,
                       int row,
                       int column,
                       boolean modifyGrid)
```

populate

```
private void populate(int tableSize,
                      Integer[][] grid)
```

Populates the table with the given grid

Parameters:

`tableSize` - the size of the table

`grid` - the grid that will populate the table

Package numbrixgame.gui.leftdisplay

Class Summary		*Page*
CompleteActionListener	CompleteActionListener will populate the board with the correct solution as found by the Solver and update the history log with every move made by the solver to achieve the solution.	*10*
ComputerActionListener	Abstract class that will be used by the action listeners implemented when the computer is chosen as the player.	*10*
FinishActionListener	FinishActionListener will define the action listener to be used by the finish button.	*11*
LeftDisplay	LeftDisplay will create a toolbar on the left dislpay providing the player options.	*12*
NextActionListener	The NextActionListener will define the functionality of the Next button.	*12*

Class CompleteActionListener

numbrixgame.gui.leftdisplay

```
java.lang.Object
  └numbrixgame.gui.leftdisplay.ComputerActionListener
      └numbrixgame.gui.leftdisplay.CompleteActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class CompleteActionListener
extends ComputerActionListener
```

CompleteActionListener will populate the board with the correct solution as found by the Solver and update the history log with every move made by the solver to achieve the solution.

Fields inherited from class numbrixgame.gui.leftdisplay.ComputerActionListener
`grid, historyLog, logSize, next, solver, time`

Constructor Summary
`CompleteActionListener(Solver solver)` Class Methods

Method Summary		
`void`	`actionPerformed(ActionEvent e)`	

Constructor Detail

CompleteActionListener

`public CompleteActionListener(Solver solver)`

Class Methods

Method Detail

actionPerformed

`public void actionPerformed(ActionEvent e)`

Specified by:

`actionPerformed` in interface `ActionListener`

Class ComputerActionListener

numbrixgame.gui.leftdisplay

```
java.lang.Object
  └numbrixgame.gui.leftdisplay.ComputerActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

Direct Known Subclasses:

CompleteActionListener, NextActionListener

```
abstract public class ComputerActionListener
extends Object
implements ActionListener
```

Abstract class that will be used by the action listeners implemented when the computer is chosen as the player.

Field Summary		
`protected static Integer[][]`	`grid`	
`protected static ArrayList<Log>`	`historyLog`	
`protected static int`	`logSize`	

protected static int	**next** Class Attributes	
protected static Solver	**solver**	
protected static String	**time**	

Constructor Summary	
ComputerActionListener(Solver solver, boolean init) Class Methods	

Field Detail

next

`protected static int next`

Class Attributes

logSize

`protected static int logSize`

grid

`protected static Integer[][] grid`

historyLog

`protected static ArrayList<Log> historyLog`

time

`protected static String time`

solver

`protected static Solver solver`

Constructor Detail

ComputerActionListener

```
public ComputerActionListener(Solver solver,
                              boolean init)
```

Class Methods

Class FinishActionListener

numbrixgame.gui.leftdisplay

```
java.lang.Object
  └numbrixgame.gui.leftdisplay.FinishActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class FinishActionListener
extends Object
implements ActionListener
```

FinishActionListener will define the action listener to be used by the finish button. It should request from system a message that states the state of the grid and then return it to the player via the bottom text box.

Constructor Summary	
FinishActionListener()	

Method Summary		
void	**actionPerformed**(ActionEvent e) Class Methods	

Constructor Detail

FinishActionListener

`public FinishActionListener()`

Method Detail

actionPerformed

`public void actionPerformed(ActionEvent e)`

Class Methods

Specified by:

`actionPerformed` in interface `ActionListener`

Class LeftDisplay

numbrixgame.gui.leftdisplay

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JToolBar
                  └numbrixgame.gui.leftdisplay.LeftDisplay
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, Serializable, SwingConstants, TransferHandler.HasGetTransferHandler

`public class` **`LeftDisplay`**
`extends JToolBar`

LeftDisplay will create a toolbar on the left dislpay providing the player options. The options provided will depend on the type of user (player or computer).

Field Summary	
`private static long`	**`serialVersionUID`** Class Cosntants

Constructor Summary
`LeftDisplay``()`

Method Summary	
`void`	**`initialize`**`(NumbrixSystem.Player playerType)` Creates the appropriate JButton(s) based on the playerType
`private void`	**`initializeComputer`**`()` Creates a "NextMove" and a "Complete" button that will display the next move made by the solver or display the completed board and every move made by the solver (respectively)
`private void`	**`initializeHuman`**`()` Creates a Finish button that will check the board for completeness and correctness

Field Detail

serialVersionUID

`private static final long` **`serialVersionUID`**

Class Cosntants

Constructor Detail

LeftDisplay

`public` **`LeftDisplay`**`()`

Method Detail

initialize

`public void` **`initialize`**`(NumbrixSystem.Player playerType)`

Creates the appropriate JButton(s) based on the playerType

Parameters:

`playerType` - type of player (COMPUTER or HUMAN)

initializeHuman

`private final void` **`initializeHuman`**`()`

Creates a Finish button that will check the board for completeness and correctness

initializeComputer

`private final void` **`initializeComputer`**`()`

Creates a "NextMove" and a "Complete" button that will display the next move made by the solver or display the completed board and every move made by the solver (respectively)

Class NextActionListener

numbrixgame.gui.leftdisplay

```
java.lang.Object
  └numbrixgame.gui.leftdisplay.ComputerActionListener
      └numbrixgame.gui.leftdisplay.NextActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

`public class` **`NextActionListener`**
`extends ComputerActionListener`

The NextActionListener will define the functionality of the Next button. It will allow the user to step through the Solvers steps to see the approach the Solver took to obtain the solution. It will update the history with the next step and update the BottomDisplay with the step count and completion time.

Field Summary	
`private int`	**`totalMoves`** Class Attributes

Fields inherited from class numbrixgame.gui.leftdisplay.ComputerActionListener
`grid, historyLog, logSize, next, solver, time`

Constructor Summary
`NextActionListener``(Solver solver)` Class Methods

Method Summary	
`void`	**`actionPerformed`**`(ActionEvent e)`

Field Detail

totalMoves

`private int` **`totalMoves`**

Class Attributes

Constructor Detail

NextActionListener

`public` **`NextActionListener`**`(Solver solver)`

Class Methods

Method Detail

actionPerformed

`public void` **`actionPerformed`**`(ActionEvent e)`

Specified by:

`actionPerformed` in interface `ActionListener`

Package numbrixgame.gui.menubar

Class Summary		*Page*
FileMenu	FileMenu will create the file menu that will be added to the menu tool bar	*14*
Menubar	Toolbar will create the toolbar to be used by Numbrix	*15*
NewActionListener	The ActionListener that will be used by the New Menu Item.	*15*
ResetActionListener	ResetActionListener will define what is to be done when Reset is clicked.	*16*

Class FileMenu

numbrixgame.gui.menubar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.AbstractButton
                  └javax.swing.JMenuItem
                      └javax.swing.JMenu
                          └numbrixgame.gui.menubar.FileMenu
```

All Implemented Interfaces:

Accessible, ImageObserver, ItemSelectable, MenuContainer, MenuElement, Serializable, SwingConstants, TransferHandler.HasGetTransferHandler

```
public class FileMenu
extends JMenu
```

FileMenu will create the file menu that will be added to the menu tool bar

Field Summary		
private static long	**serialVersionUID** Class Constants	

Constructor Summary	
FileMenu() Class Methods	

Method Summary		
private static JMenuItem	**exitMenuItem**() Creates and returns the JMenuItem for "Exit"	
private static JMenuItem	**newMenuItem**() Creates and returns the JMenuItem for "New Game"	
private static JMenuItem	**resetMenuItem**() Creates and returns the JMenuItem for "Reset"	

Field Detail

serialVersionUID

```
private static final long serialVersionUID
```

Class Constants

Constructor Detail

FileMenu

```
public FileMenu()
```

Class Methods

Method Detail

newMenuItem

```
private static final JMenuItem newMenuItem()
```

Creates and returns the JMenuItem for "New Game"

Returns:

the JMenuItem for "New Game"

resetMenuItem

```
private static final JMenuItem resetMenuItem()
```

Creates and returns the JMenuItem for "Reset"

Returns:

the JMenuItem for "Reset"

exitMenuItem

```
private static final JMenuItem exitMenuItem()
```

Creates and returns the JMenuItem for "Exit"

Returns:

the JMenuItem for "Exit"

Class Menubar

numbrixgame.gui.menubar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └numbrixgame.gui.menubar.Menubar
```

All Implemented Interfaces:

Accessible, ImageObserver, MenuContainer, MenuElement, Serializable, TransferHandler.HasGetTransferHandler

```
public class Menubar
extends JMenuBar
```

Toolbar will create the toolbar to be used by Numbrix

Field Summary		
private static long	**serialVersionUID** Class Constants	

Constructor Summary	
Menubar() Class Methods	

Field Detail

serialVersionUID

```
private static final long serialVersionUID
```

Class Constants

Constructor Detail

Menubar

```
public Menubar()
```

Class Methods

Class NewActionListener

numbrixgame.gui.menubar

```
java.lang.Object
  └numbrixgame.gui.menubar.NewActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class NewActionListener
extends Object
implements ActionListener
```

The ActionListener that will be used by the New Menu Item. The action should open up a JFileChooser and interact with the system accordingly

Constructor Summary	
NewActionListener()	

Method Summary		
void	**actionPerformed**(ActionEvent e) Class Methods	

Constructor Detail

NewActionListener

```
public NewActionListener()
```

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Class Methods

Specified by:

`actionPerformed` in interface `ActionListener`

Class ResetActionListener

numbrixgame.gui.menubar

```
java.lang.Object
  └numbrixgame.gui.menubar.ResetActionListener
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class ResetActionListener
extends Object
implements ActionListener
```

ResetActionListener will define what is to be done when Reset is clicked. It will reset the grid back to its original values.

Constructor Summary	
`ResetActionListener()`	

Method Summary		
`void`	`actionPerformed(ActionEvent e)` Class Methods	

Constructor Detail

ResetActionListener

```
public ResetActionListener()
```

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Class Methods

Specified by:

`actionPerformed` in interface `ActionListener`

Package numbrixgame.system

Class Summary		*Page*
History	History will keep track of all the player made changes made to the grid.	*17*
Log	Log is a data structure that will be used by history to keep track of changes to the grid.	*19*
NumbrixSystem	NumbrixSystem will take care of the back end for Numbrix.	*20*
Parser	Parser will parse the file provided by the user to determine the grid size and static elements	*24*
Validator	Validator will check the grid for correctness.	*26*

Enum Summary		
History.Modification	Class Constants	
NumbrixSystem.Player	Class Constants	
Validator.State		

Class History

numbrixgame.system

```
java.lang.Object
  └numbrixgame.system.History
```

```
public class History
extends Object
```

History will keep track of all the player made changes made to the grid.

Nested Class Summary		
static enum	`History.Modification` Class Constants	

Field Summary		
private int	`gridSize`	
private boolean[][]	`hasVal`	
private ArrayList<Log>	`historyLog` Class Attributes	
private String	`incrementLog`	
private boolean[][]	`staticVals`	

Constructor Summary	
`History(int gridSize, boolean[][] staticVals)` Class Methods	
`History(int gridSize, boolean[][] staticVals, ArrayList<Log> historyLog, boolean[][] hasVal)`	

Method Summary		
ArrayList<Log>	`getHistoryLog()` Returns historyLog	
String	`getIncrementLog()` Returns incrementlog	
String	`getLog()` Returns a String format of the log	
int	`getSize()` Returns the size of the log	
void	`incrementLog()` Used by a process that knows the log has been updated and wishes to update the increment log string	
void	`logChange(int row, int column, Integer newVal)` Logs what kind of change occurred along with the change.	
private String	`logToString(Log log)`	

Field Detail

historyLog

```
private ArrayList<Log> historyLog
```

Class Attributes

staticVals

```
private boolean[][] staticVals
```

hasVal

```
private boolean[][] hasVal
```

gridSize

`private int gridSize`

incrementLog

`private String incrementLog`

Constructor Detail

History

```
public History(int gridSize,
               boolean[][] staticVals)
```

Class Methods

History

```
public History(int gridSize,
               boolean[][] staticVals,
               ArrayList<Log> historyLog,
               boolean[][] hasVal)
```

Method Detail

logChange

```
public void logChange(int row,
                      int column,
                      Integer newVal)
```

Logs what kind of change occurred along with the change.

Parameters:

`newVal` - value of change

getLog

`public String getLog()`

Returns a String format of the log

Returns:

a String format of the log

incrementLog

`public void incrementLog()`

Used by a process that knows the log has been updated and wishes to update the increment log string

logToString

`private String logToString(Log log)`

getIncrementLog

`public String getIncrementLog()`

Returns incrementlog

Returns:

incrementlog

getHistoryLog

`public ArrayList<Log> getHistoryLog()`

Returns historyLog

Returns:

historyLog

getSize

`public int getSize()`

Returns the size of the log

Returns:

the size of the log

Enum History.Modification

numbrixgame.system

```
java.lang.Object
  └java.lang.Enum<History.Modification>
      └numbrixgame.system.History.Modification
```

All Implemented Interfaces:

Comparable<History.Modification>, Serializable

Enclosing class:

History

```
public static enum History.Modification
extends Enum<History.Modification>
```

Class Constants

Enum Constant Summary	
ADD	
DELETE	
MODIFY	

Constructor Summary		
private	**History.Modification**()	

Method Summary		
static History.Modification	**valueOf**(String name)	
static History.Modification[]	**values**()	

Enum Constant Detail

ADD

public static final History.Modification **ADD**

DELETE

public static final History.Modification **DELETE**

MODIFY

public static final History.Modification **MODIFY**

Constructor Detail

History.Modification

private **History.Modification**()

Method Detail

values

public static History.Modification[] **values**()

valueOf

public static History.Modification **valueOf**(String name)

Class Log

numbrixgame.system

```
java.lang.Object
  └numbrixgame.system.Log
```

public class **Log**
extends Object

Log is a data structure that will be used by history to keep track of changes to the grid.

Field Summary		
private History.Modification	**change**	
private Integer	**val**	
private int	**x** Class Attributes	
private int	**y**	

Constructor Summary	
Log(int x, int y, Integer val, History.Modification change) Constructor	

Method Summary		
History.Modification	**getChange**() Returns change	
Integer	**getVal**() Returns val	
int	**getX**() Returns x	
int	**getY**() Returns y	
String	**toString**()	

Field Detail

x

private int **x**

Class Attributes

Y

```
private int Y
```

val

```
private Integer val
```

change

```
private History.Modification change
```

Constructor Detail

Log

```
public Log(int x,
           int y,
           Integer val,
           History.Modification change)
```

Constructor

Parameters:

`x` - the x value of the log

`y` - the y value of the log

`val` - the value of the log

`change` - the type of change

Method Detail

getX

```
public int getX()
```

Returns x

Returns:

x

getY

```
public int getY()
```

Returns y

Returns:

y

getVal

```
public Integer getVal()
```

Returns val

Returns:

val

getChange

```
public History.Modification getChange()
```

Returns change

Returns:

change

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class NumbrixSystem

numbrixgame.system

```
java.lang.Object
  └numbrixgame.system.NumbrixSystem
```

Direct Known Subclasses:

TestSystem

```
public class NumbrixSystem
extends Object
```

NumbrixSystem will take care of the back end for Numbrix.

Nested Class Summary		
static enum	**NumbrixSystem.Player** Class Constants	

Field Summary	
protected File	**file**
protected Integer[][]	**grid**
protected int	**gridSize** Class Attributes
protected History	**history**
protected int	**numOfObjects**
protected NumbrixSystem.Player	**player**
private Solver	**solver**
protected boolean[][]	**staticData**

Constructor Summary
NumbrixSystem() Class Methods

Method Summary	
void	**complete**(Integer[][] grid, ArrayList<Log> log) Applies the completed grid and history
Integer[][]	**getGrid**() Returns grid
int	**getGridSize**() Returns gridSize
String	**getHistory**() Returns the formatted string of hitsories log
ArrayList<Log>	**getHistoryLog**() Returns the log of history
String	**getIncrementLog**() Returns the formatted string of histories incrementLog
int	**getNumOfObjects**() Returns numOfObjects
NumbrixSystem.Player	**getPlayer**() Returns player
Solver	**getSolver**() Returns solver
boolean[][]	**getStaticData**() Returns staticData: a 2D array that tells which positions cannot be changed
Integer	**getVal**(int x, int y) Returns the value of the grid at the given x and y
void	**logChange**(int x, int y, Integer newVal) **Deprecated.**
Integer[][]	**makeGrid**() Creates an empty 2D array of size gridSize x gridSize
void	**modifyGrid**(int x, int y, Integer val) Modifies the grid and logs the change
void	**printGrid**() Print the system to standard out
void	**reset**() Undoes changes made and restores the grid to its original state
void	**resetData**() Resets the history and grid
void	**setup**(NumbrixSystem.Player player, File file) Sets up the grid and gui given the player and data
Validator.State	**verify**() Returns the validity of grid
Validator.State	**verify**(Integer[][] grid) Returns the validity of the grid

Field Detail

gridSize

`protected int gridSize`

Class Attributes

staticData

`protected boolean[][] staticData`

grid

`protected Integer[][] grid`

player

`protected NumbrixSystem.Player player`

file

`protected File file`

history

`protected History history`

numOfObjects

`protected int numOfObjects`

solver

`private Solver solver`

Constructor Detail

NumbrixSystem

`public NumbrixSystem()`

Class Methods

Method Detail

setup

```
public void setup(NumbrixSystem.Player player,
                  File file)
```

Sets up the grid and gui given the player and data

Parameters:

`player` - the type of player (HUMAN, or COMPUTER)

`file` - the grid data

reset

`public void reset()`

Undoes changes made and restores the grid to its original state

resetData

`public void resetData()`

Resets the history and grid

verify

`public Validator.State verify(Integer[][] grid)`

Returns the validity of the grid

Parameters:

`grid` - the grid being validated

Returns:

the validity of the grid

verify

`public Validator.State verify()`

Returns the validity of grid

Returns:

the validity of grid

makeGrid

`public Integer[][] makeGrid()`

Creates an empty 2D array of size gridSize x gridSize

Returns:

an empty 2D array of size gridSize x gridSize

modifyGrid

```
public void modifyGrid(int x,
                       int y,
                       Integer val)
```

Modifies the grid and logs the change

logChange

```
public void logChange(int x,
                      int y,
                      Integer newVal)
```

Deprecated.

Log the change to the position

Parameters:

`x` - the x position

`y` - the y position

`newVal` - the new value of the position

complete

```
public void complete(Integer[][] grid,
                     ArrayList<Log> log)
```

Applies the completed grid and history

printGrid

```
public void printGrid()
```

Print the system to standard out

getGridSize

```
public int getGridSize()
```

Returns gridSize

Returns:

gridSize

getPlayer

```
public NumbrixSystem.Player getPlayer()
```

Returns player

Returns:

player

getSolver

```
public Solver getSolver()
```

Returns solver

Returns:

solver

getGrid

```
public Integer[][] getGrid()
```

Returns grid

Returns:

grid

getVal

```
public Integer getVal(int x,
                      int y)
```

Returns the value of the grid at the given x and y

Parameters:

`x` - the x coordinate

`y` - the y coordinate

Returns:

the value of the grid at the given x and y

getStaticData

```
public boolean[][] getStaticData()
```

Returns staticData: a 2D array that tells which positions cannot be changed

Returns:

staticData

getHistory

```
public String getHistory()
```

Returns the formatted string of hitsories log

Returns:

the formatted string of hitsories log

getHistoryLog

```
public ArrayList<Log> getHistoryLog()
```

Returns the log of history

Returns:

the log of history

getNumOfObjects

```
public int getNumOfObjects()
```

Returns numOfObjects

Returns:

numOfObjects

getIncrementLog

`public String getIncrementLog()`

Returns the formatted string of histories incrementLog

Returns:

the formatted string of histories incrementLog

Enum NumbrixSystem.Player

numbrixgame.system

```
java.lang.Object
  └java.lang.Enum<NumbrixSystem.Player>
      └numbrixgame.system.NumbrixSystem.Player
```

All Implemented Interfaces:

Comparable<NumbrixSystem.Player>, Serializable

Enclosing class:

NumbrixSystem

```
public static enum NumbrixSystem.Player
extends Enum<NumbrixSystem.Player>
```

Class Constants

Enum Constant Summary
COMPUTER
HUMAN

Field Summary	
private String	**message**

Constructor Summary	
private	**NumbrixSystem.Player**(String message)

Method Summary	
String	**string**()
static NumbrixSystem.Player	**valueOf**(String name)
static NumbrixSystem.Player[]	**values**()

Enum Constant Detail

HUMAN

`public static final NumbrixSystem.Player HUMAN`

COMPUTER

`public static final NumbrixSystem.Player COMPUTER`

Field Detail

message

`private final String message`

Constructor Detail

NumbrixSystem.Player

`private NumbrixSystem.Player(String message)`

Method Detail

values

`public static NumbrixSystem.Player[] values()`

valueOf

`public static NumbrixSystem.Player valueOf(String name)`

string

`public String string()`

Class Parser

numbrixgame.system

```
java.lang.Object
  └numbrixgame.system.Parser
```

```
public class Parser
extends Object
```

Parser will parse the file provided by the user to determine the grid size and static elements

Field Summary		
`private Integer[][]`	`grid`	
`private int`	`gridSize` Class Attributes	
`private boolean[][]`	`staticElements`	

Constructor Summary	
`Parser(File file)` Class Methods	

Method Summary		
`Integer[][]`	`getGrid()` Returns grid	
`int`	`getGridSize()` Returns gridSize	
`boolean[][]`	`getStatic()` Returns staticElements	
`private void`	`parse(File file)` Takes in the formatted file and creates a Numbrix grid from the contents.	

Field Detail

gridSize

```
private int gridSize
```

Class Attributes

staticElements

```
private boolean[][] staticElements
```

grid

```
private Integer[][] grid
```

Constructor Detail

Parser

```
public Parser(File file)
```

Class Methods

Method Detail

getGridSize

```
public int getGridSize()
```

Returns gridSize

Returns:

gridSize

getStatic

```
public boolean[][] getStatic()
```

Returns staticElements

Returns:

staticElements

getGrid

```
public Integer[][] getGrid()
```

Returns grid

Returns:

grid

parse

```
private void parse(File file)
```

Takes in the formatted file and creates a Numbrix grid from the contents.

Parameters:

`file` - the file being parsed

Class Validator

numbrixgame.system

```
java.lang.Object
  └numbrixgame.system.Validator
```

```
public class Validator
extends Object
```

Validator will check the grid for correctness. It well then return a constant pertaining to the state of the board.

Nested Class Summary

static enum	**Validator.State**

Field Summary

private Validator.State	**state** Class Attributes
private static int	**X** Class Constants
private static int	**Y**

Constructor Summary

Validator(int gridSize, Integer[][] grid) Class Methods

Method Summary

private boolean	**checkVal**(int x, int y, int val, int gridSize, Integer[][] grid) Check to see if the coordinates are correct and if the provided val is at the coordinate
private int[]	**findNext**(int[] pos, int nextVal, int gridSize, Integer[][] grid) Used by trace to help find the next non-diagonal cell that contains the next number.
Validator.State	**getState**() Returns state
private Validator.State	**trace**(int gridSize, int[] pos, Integer[][] grid) Returns whether or not the grid is correctly completed.
private void	**validate**(int gridSize, Integer[][] grid) Validates the given grid
static Validator.State	**validateInput**(Integer value, int gridSize) Returns the validity of the given value.

Field Detail

X

```
private static int X
```

Class Constants

Y

```
private static int Y
```

state

```
private Validator.State state
```

Class Attributes

Constructor Detail

Validator

```
public Validator(int gridSize,
                 Integer[][] grid)
```

Class Methods

Method Detail

getState

```
public Validator.State getState()
```

Returns state

Returns:

state

validateInput

```
public static Validator.State validateInput(Integer value,
                                            int gridSize)
```

Returns the validity of the given value.

Parameters:

`value` - the value being validated

`gridSize` - the size of the grid

Returns:

the validity of the given value

validate

```
private final void validate(int gridSize,
                            Integer[][] grid)
```

Validates the given grid

Parameters:

`gridSize` - the size of hte grid

`grid` - the grid being validated

trace

```
private final Validator.State trace(int gridSize,
                                    int[] pos,
                                    Integer[][] grid)
```

Returns whether or not the grid is correctly completed. It does so by starting from the cell with the value 1 and attempting to create an unbroken path of consecutively increasing cells in a non-diagonal direction until the last value (gridSize x gridSize) is found.

Parameters:

`gridSize` - the size of the grid

`pos` - the position of the cell with value 1

`grid` - the grid being validated

Returns:

the validity of the grid

findNext

```
private final int[] findNext(int[] pos,
                             int nextVal,
                             int gridSize,
                             Integer[][] grid)
```

Used by trace to help find the next non-diagonal cell that contains the next number.

Parameters:

`pos` - the position of the cell that is being branched from

`nextVal` - the value that is being looked for

`gridSize` - the size of hte grid

`grid` - the grid

Returns:

the position of the next value (null if it does not exist)

checkVal

```
private final boolean checkVal(int x,
                               int y,
                               int val,
                               int gridSize,
                               Integer[][] grid)
```

Check to see if the coordinates are correct and if the provided val is at the coordinate

Parameters:

`x` - the x coordinate being checked

`y` - the y coordinate being checked

`val` - the value being looked for

`gridSize` - the size of the grid

`grid` - the grid

Returns:

whether or not the x and y coordinates are valid and contain val

Enum Validator.State

numbrixgame.system

```
java.lang.Object
  └java.lang.Enum<Validator.State>
      └numbrixgame.system.Validator.State
```

All Implemented Interfaces:

Comparable<Validator.State>, Serializable

Enclosing class:
Validator

```
public static enum Validator.State
extends Enum<Validator.State>
```

Enum Constant Summary	
CORRECT	
INCORRECT_ELEMENT	
INCORRECT_GRID	
INCORRECT_SIZE	

Field Summary		
private String	**message**	

Constructor Summary		
private	**Validator.State**(String message)	

Method Summary		
String	**string**()	
static Validator.State	**valueOf**(String name)	
static Validator.State[]	**values**()	

Enum Constant Detail

CORRECT

```
public static final Validator.State CORRECT
```

INCORRECT_GRID

```
public static final Validator.State INCORRECT_GRID
```

INCORRECT_ELEMENT

```
public static final Validator.State INCORRECT_ELEMENT
```

INCORRECT_SIZE

```
public static final Validator.State INCORRECT_SIZE
```

Field Detail

message

```
private final String message
```

Constructor Detail

Validator.State

```
private Validator.State(String message)
```

Method Detail

values

```
public static Validator.State[] values()
```

valueOf

```
public static Validator.State valueOf(String name)
```

string

```
public String string()
```

Package numbrixgame.system.solver

Class Summary		*Page*
ConstraintSearch	The constraint search to be used by the Solver.	29
HeuristicSearch	The heuristic search to be used by the solver.	32
SearchMethod		33
Snake	Data structure that will manage the data and segment it accordingly.	36
Solver	Solver will solve the Numbrix game	39
Triple	Data structure that holds unit, x, and y value	42

Enum Summary		
SearchMethod.Direction	Class Constant	
Snake.End	Class Constants	

Class ConstraintSearch

numbrixgame.system.solver

```
java.lang.Object
 └numbrixgame.system.solver.SearchMethod
    └numbrixgame.system.solver.ConstraintSearch
```

```
public class ConstraintSearch
extends SearchMethod
```

The constraint search to be used by the Solver. It will utilize constraints to help find possible cell values in the Numbrix grid.

Nested classes/interfaces inherited from class numbrixgame.system.solver.SearchMethod
`SearchMethod.Direction`

Field Summary		
`private Stack<Triple>`	**`additions`** Class Attributes	

Fields inherited from class numbrixgame.system.solver.SearchMethod
`snake, solver, system`

Constructor Summary	
`ConstraintSearch``(Solver solver)` Class Methods	

Method Summary		
`private void`	**`add`**`(Triple triple)`	
`private void`	**`constraintFound`**`(SearchMethod.Direction direction, Triple current, int increment)` Function called when constraint is found.	
`private Triple`	**`findNext`**`(int list, boolean forward)` Finds the next Triple to be searched for	
`private boolean`	**`firstDegreeSearch`**`(Triple previous, SearchMethod.Direction direction)` Just check to see if this node has the potential to be the next node by checking if it is empty and legal.	
`private boolean`	**`firstPrimeDegreeSearch`**`(Triple previous, SearchMethod.Direction direction)` Just check to see if this node is populated and legal	
`protected boolean`	**`search`**`(int increment, Triple current)` Recursive function that searches for the next constraint	
`private boolean`	**`secondDegreeSearch`**`(Triple previous, SearchMethod.Direction direction, int increment)` A check to see if the searched at node can contain the value sought for.	
`private boolean`	**`secondPrimeDegreeSearch`**`(Triple previous, SearchMethod.Direction direction, int increment)` **Deprecated.** *THIS IS NOT A FOR SURE SEARCH! DO NOT USE! A check to see if the searched at node cannot contain the sought after value.*	

boolean	**startSearch**(boolean forward) Start the constraint search	
private boolean	**thirdDegreeSearch**(Triple previous, SearchMethod.Direction direction, int increment) Searches for a node in which the only value that can fit in it is the sought for value.	
void	**undo**() Undoes additions made by this object	

Methods inherited from class numbrixgame.system.solver.SearchMethod
emptyAndLegal, fullAndLegal, legal, makeDirectionStack, makeDirectionStack, setSnake, setSystem

Field Detail

additions

```
private Stack<Triple> additions
```

Class Attributes

Constructor Detail

ConstraintSearch

```
public ConstraintSearch(Solver solver)
```

Class Methods

Method Detail

startSearch

```
public boolean startSearch(boolean forward)
```

Start the constraint search

Parameters:

`forward` - the direction that the constraint is searching in

Returns:

the success of the search

search

```
protected boolean search(int increment,
                         Triple current)
```

Recursive function that searches for the next constraint

Parameters:

`increment` - forward or backwards

`current` - the current node (node being looked at)

Returns:

the success of the search

firstDegreeSearch

```
private boolean firstDegreeSearch(Triple previous,
                                  SearchMethod.Direction direction)
```

Just check to see if this node has the potential to be the next node by checking if it is empty and legal.

Parameters:

`previous` - the callee

`direction` - the direction in which the callee called

Returns:

whether or not the node is empty and legal

firstPrimeDegreeSearch

```
private boolean firstPrimeDegreeSearch(Triple previous,
                                       SearchMethod.Direction direction)
```

Just check to see if this node is populated and legal

Parameters:

`previous` - the callee

`direction` - the direction in which the callee called

Returns:

whether or not the node is populated and legal

secondDegreeSearch

```
private boolean secondDegreeSearch(Triple previous,
                                   SearchMethod.Direction direction,
                                   int increment)
```

A check to see if the searched at node can contain the value sought for. It does so by looking to see if it can find the the increment node (two nodes after the previous node) in the surrounding nodes. If it can, then we say that this direction has potential.

Parameters:

`previous` - the callee

`direction` - the direction in which the callee called

`increment` - forwards or backwards

Returns:

whether or not the position can contain the searched for value

secondPrimeDegreeSearch

```
private boolean secondPrimeDegreeSearch(Triple previous,
                                        SearchMethod.Direction direction,
                                        int increment)
```

Deprecated. *THIS IS NOT A FOR SURE SEARCH! DO NOT USE! A check to see if the searched at node cannot contain the sought after value. If it cannot, then we can limit the nodes for which can contain the value.*

Parameters:

`previous` - the callee

`direction` - the direction in which the callee called

`increment` - forwards or backwards

Returns:

whether or not the position can contain the searched for value

thirdDegreeSearch

```
private boolean thirdDegreeSearch(Triple previous,
                                  SearchMethod.Direction direction,
                                  int increment)
```

Searches for a node in which the only value that can fit in it is the sought for value. It does so by looking at the surrounding nodes and seeing if they are all populated or legal. If they are, then this constraint can be applied. It then checks the surrounding populated and legal nodes to see which values they still need (ie, the end values). If only one pair of end values can be found, it must be the case that this node is the only node that can house the triple we are looking for. That is because this search is called after secondDegreeSearch, hence it must be the case that this node has the potential to at least house the sought after value. Hence, if only one pair of linking values can be found, it must be the case that they link the previous node with it's second increment node.

Parameters:

`previous` - the callee

`direction` - the direction in which the callee called

`increment` - forwards or backwards

Returns:

whether or not the position can contain the searched for value

findNext

```
private Triple findNext(int list,
                        boolean forward)
```

Finds the next Triple to be searched for

Returns:

the triple being searched for

add

```
private void add(Triple triple)
```

constraintFound

```
private void constraintFound(SearchMethod.Direction direction,
                             Triple current,
                             int increment)
```

Function called when constraint is found. Modifies data accordingly

Parameters:

`direction` - the direction of the node found
`current` - the current triple
`increment` - the direction being traveled

undo

```
public void undo()
```

Undoes additions made by this object

Class HeuristicSearch

numbrixgame.system.solver

```
java.lang.Object
  └numbrixgame.system.solver.SearchMethod
      └numbrixgame.system.solver.HeuristicSearch
```

```
public class HeuristicSearch
extends SearchMethod
```

The heuristic search to be used by the solver. It will attempt a brute force search of the grid to solved the Numbrix grid.

Nested classes/interfaces inherited from class numbrixgame.system.solver.SearchMethod
`SearchMethod.Direction`

Fields inherited from class numbrixgame.system.solver.SearchMethod
`snake, solver, system`

Constructor Summary	
`HeuristicSearch(Solver solver)` Class Methods	

Method Summary		
`protected boolean`	`connects(Triple triple, SearchMethod.Direction direction, int increment)` Returns whether or not the triple can connect with a neighbor or is a terminal node	
`boolean`	`search(Triple triple, SearchMethod.Direction direction, int nodeCount, int increment)` A recursive search (of sorts) which checks to see if the triple can be placed and make sure that the nodeCount has not reached 1.	
`boolean`	`startSearch(Solver solver)` Stars the heuristic search by initializing variables and finding the shortest unsolved path in the snake.	

Methods inherited from class numbrixgame.system.solver.SearchMethod
`emptyAndLegal, fullAndLegal, legal, makeDirectionStack, makeDirectionStack, setSnake, setSystem`

Constructor Detail

HeuristicSearch

```
public HeuristicSearch(Solver solver)
```

Class Methods

Method Detail

startSearch

```
public boolean startSearch(Solver solver)
```

Stars the heuristic search by initializing variables and finding the shortest unsolved path in the snake.

Parameters:

`solver` - The solver doing the solving

Returns:

whether or not a path was found

search

```
public boolean search(Triple triple,
                      SearchMethod.Direction direction,
                      int nodeCount,
                      int increment)
```

A recursive search (of sorts) which checks to see if the triple can be placed and make sure that the nodeCount has not reached 1. If it is possible to place the triple in the cell in the direction, search will then search to the increment of the triple in the remaining direcetions.

Parameters:

`triple` - the cell calling the search

`direction` - the direction that triple is searching to populate

`nodeCount` - the number of nodes left to search for

`increment` - the "direction" the search is going in (forward or backward)

Returns:

the status of the solution

connects

```
protected boolean connects(Triple triple,
                           SearchMethod.Direction direction,
                           int increment)
```

Returns whether or not the triple can connect with a neighbor or is a terminal node

Parameters:

`triple` - the triple being checked

`direction` - the direction the triple is checking in

Returns:

whether or not the triple can connect with a neighbor

Class SearchMethod

numbrixgame.system.solver

```
java.lang.Object
  └numbrixgame.system.solver.SearchMethod
```

Direct Known Subclasses:

ConstraintSearch, HeuristicSearch

```
abstract public class SearchMethod
extends Object
```

Nested Class Summary	
static enum	**SearchMethod.Direction** Class Constant

Field Summary	
protected static Snake	**snake**
protected Solver	**solver**
protected static NumbrixSystem	**system** Class Attributes

Constructor Summary
SearchMethod(Solver solver) Class Methods

Method Summary	
boolean	**emptyAndLegal**(int x, int y)
boolean	**fullAndLegal**(int x, int y)
boolean	**legal**(int x, int y) Returns whether or not the provided coordinates are legal

protected LinkedList<SearchMethod.Direction>	**makeDirectionStack**() Returns a stack of unique directions
protected LinkedList<SearchMethod.Direction>	**makeDirectionStack**(SearchMethod.Direction remove) Returns a stack of unique directions with the provided direction omitted
static void	**setSnake**(Snake snake)
static void	**setSystem**(NumbrixSystem system)

Field Detail

system

```
protected static NumbrixSystem system
```

Class Attributes

snake

```
protected static Snake snake
```

solver

```
protected Solver solver
```

Constructor Detail

SearchMethod

```
public SearchMethod(Solver solver)
```

Class Methods

Method Detail

setSystem

```
public static void setSystem(NumbrixSystem system)
```

setSnake

```
public static void setSnake(Snake snake)
```

legal

```
public boolean legal(int x,
                     int y)
```

Returns whether or not the provided coordinates are legal

Parameters:

x - the x coordinate

y - the y coordinate

Returns:

the legality of the coordinates

emptyAndLegal

```
public boolean emptyAndLegal(int x,
                             int y)
```

fullAndLegal

```
public boolean fullAndLegal(int x,
                            int y)
```

makeDirectionStack

```
protected LinkedList<SearchMethod.Direction> makeDirectionStack()
```

Returns a stack of unique directions

Returns:

a stack of unique directions

makeDirectionStack

```
protected LinkedList<SearchMethod.Direction> makeDirectionStack(SearchMethod.Direction
 remove)
```

Returns a stack of unique directions with the provided direction omitted

Parameters:

remove - the direction to omit

Returns:

the stack of unique directions with the provided direction omitted

Enum SearchMethod.Direction

numbrixgame.system.solver

```
java.lang.Object
  └java.lang.Enum<SearchMethod.Direction>
      └numbrixgame.system.solver.SearchMethod.Direction
```

All Implemented Interfaces:

Comparable<SearchMethod.Direction>, Serializable

Enclosing class:

SearchMethod

```
public static enum SearchMethod.Direction
extends Enum<SearchMethod.Direction>
```

Class Constant

Enum Constant Summary	
BOTTOM	
LEFT	
RIGHT	
START	
TOP	

Field Summary		
protected int	**position**	
int	**x**	
int	**y**	

Constructor Summary		
private	**SearchMethod.Direction**(int x, int y, int position)	

Method Summary		
static SearchMethod.Direction	**valueOf**(String name)	
static SearchMethod.Direction[]	**values**()	

Enum Constant Detail

BOTTOM

`public static final SearchMethod.Direction BOTTOM`

TOP

`public static final SearchMethod.Direction TOP`

LEFT

`public static final SearchMethod.Direction LEFT`

RIGHT

`public static final SearchMethod.Direction RIGHT`

START

`public static final SearchMethod.Direction START`

Field Detail

x

`public final int x`

y

`public final int y`

position

`protected final int position`

Constructor Detail

SearchMethod.Direction

```
private SearchMethod.Direction(int x,
                               int y,
                               int position)
```

Method Detail

values

```
public static SearchMethod.Direction[] values()
```

valueOf

```
public static SearchMethod.Direction valueOf(String name)
```

Class Snake

numbrixgame.system.solver

```
java.lang.Object
  └numbrixgame.system.solver.Snake
```

```
public class Snake
extends Object
```

Data structure that will manage the data and segment it accordingly. The snake will keep a list of lists where each sub list contains elements in consecutively increasing order. Each parent list will contain lists in consecutively increasing order.

Nested Class Summary	
static enum	**Snake.End** Class Constants

Field Summary	
private LinkedList<LinkedList<Triple>>	**snake** Class Attributes

Constructor Summary
Snake(int gridSize, Integer[][] grid) Class Methods

Method Summary	
void	**add**(Triple triple) Add new value and positions
int	**count**() Returns the number of Triples in the snake
Triple	**find**(int value) Returns the triple with the given value
Integer[]	**findEnds**(int value) Returns the missing ends (if any) of the provided value
Triple	**findTip**(int value, boolean last) Returns the head or tail of the list within which the value is a part of
Triple	**getFirst**(int list) Returns the first Triple in the list
Triple	**getLast**(int list) Returns the last Triple in the list
boolean	**hasEmpty**() Returns whether or Snake has any empty LinkedLists
boolean	**isEnd**(int value) Returns whether or not the provided value is a tip
Triple	**remove**(int value) Remove the given value.
int	**size**() Returns the number of lists in the snake
int	**sizeOf**(int position) Returns the size of the list at the position

String	toString() Returns a String representation of Snake	

Field Detail

snake

```
private LinkedList<LinkedList<Triple>> snake
```

Class Attributes

Constructor Detail

Snake

```
public Snake(int gridSize,
             Integer[][] grid)
```

Class Methods

Method Detail

add

```
public void add(Triple triple)
```

Add new value and positions

remove

```
public Triple remove(int value)
```

Remove the given value.

Parameters:

`value` - the value of the triple being removed

Returns:

the triple that was removed

find

```
public Triple find(int value)
```

Returns the triple with the given value

Parameters:

`value` - the value of the triple being searched for

Returns:

the triple with the given value

findTip

```
public Triple findTip(int value,
                      boolean last)
```

Returns the head or tail of the list within which the value is a part of

Parameters:

`value` - the value being searched for

`last` - tail or tip

Returns:

the tail or tip of the list within which the value is a part of

size

```
public int size()
```

Returns the number of lists in the snake

Returns:

the number of lists in the snake

sizeOf

```
public int sizeOf(int position)
```

Returns the size of the list at the position

Parameters:

`position` - the position of the list queried

Returns:

the size of the list at the position

count

```
public int count()
```

Returns the number of Triples in the snake

Returns:

the number of Triples in the snake

isEnd

```
public boolean isEnd(int value)
```

Returns whether or not the provided value is a tip

Parameters:

`value` - the value being searched for

Returns:

whether or not the provided value is a tip

findEnds

```
public Integer[] findEnds(int value)
```

Returns the missing ends (if any) of the provided value

Parameters:

`value` - the value for which the ends are being searched for

Returns:

the ends of the provided value

getFirst

```
public Triple getFirst(int list)
```

Returns the first Triple in the list

Parameters:

`list` - the list being searched

Returns:

the first Triple in the list

getLast

```
public Triple getLast(int list)
```

Returns the last Triple in the list

Parameters:

`list` - the list being searched

Returns:

the last Triple in the list

hasEmpty

```
public boolean hasEmpty()
```

Returns whether or Snake has any empty LinkedLists

Returns:

whether or Snake has any empty LinkedLists

toString

```
public String toString()
```

Returns a String representation of Snake

Overrides:

`toString` in class `Object`

Enum Snake.End

numbrixgame.system.solver

```
java.lang.Object
  └java.lang.Enum<Snake.End>
      └numbrixgame.system.solver.Snake.End
```

All Implemented Interfaces:

Comparable<Snake.End>, Serializable

Enclosing class:

Snake

```
public static enum Snake.End
extends Enum<Snake.End>
```

Class Constants

Enum Constant Summary	
FIRST	
LAST	

Field Summary		
protected int	**increment**	
protected int	**position**	

Constructor Summary		
private	**Snake.End**(int position, int increment)	

Method Summary		
static Snake.End	**valueOf**(String name)	
static Snake.End[]	**values**()	

Enum Constant Detail

LAST

```
public static final Snake.End LAST
```

FIRST

```
public static final Snake.End FIRST
```

Field Detail

position

```
protected final int position
```

increment

```
protected final int increment
```

Constructor Detail

Snake.End

```
private Snake.End(int position,
                  int increment)
```

Method Detail

values

```
public static Snake.End[] values()
```

valueOf

```
public static Snake.End valueOf(String name)
```

Class Solver

numbrixgame.system.solver

```
java.lang.Object
  └numbrixgame.system.solver.Solver
```

```
public class Solver
extends Object
```

Solver will solve the Numbrix game

Field Summary		
private ConstraintSearch	**constraint**	
private long	**endTime**	
private static HeuristicSearch	**heuristic**	
private static Snake	**snake**	
private boolean	**solutionFound**	
private long	**startTime**	
private static NumbrixSystem	**system** Class Attributes	

Constructor Summary
Solver()
Solver(NumbrixSystem system) Class Methods

Method Summary	
void	**add**(int x, int y, int val) Add the given values to the grid and structures
void	**add**(Triple triple) Add the given triple to the grid and structures
boolean	**check**() Check to see if the gird has been solved
protected boolean	**constraintSatisfactionSearch**() A Constraint Satisfaction Search on Numbrix that will be used recursively by HeuristicSearch and return whether or not a solution was found.
protected boolean	**constraintSearch**() Performs the constraint search and returns if the grid is solved.
protected ConstraintSearch	**getConstraint**() Returns constraint
protected HeuristicSearch	**getHeuristic**() Returns heuristic
boolean	**getSolutionFound**() Returns solutionFound
String	**getTimeElsapsed**() Returns a formatted string of the time spent in MM:SS:mm
long	**getTimeSpent**() Returns the amount of time spent solving the Numbrix grid
protected void	**initialize**() Initialize datastructures for search
void	**remove**(int x, int y, int val) Remove the given values from the grid and structures
void	**remove**(Triple triple) Remove the triple from the grid and structures
String	**snakeString**() Returns a String representation of snake
void	**solve**() Solves the Numbrix problem
void	**undo**() Removes modifications made form the constraint search

Field Detail

system

`private static NumbrixSystem` **`system`**

Class Attributes

snake

`private static Snake` **`snake`**

heuristic

`private static HeuristicSearch` **`heuristic`**

constraint

`private ConstraintSearch` **`constraint`**

startTime

`private long` **`startTime`**

endTime

`private long` **`endTime`**

solutionFound

`private boolean` **`solutionFound`**

Constructor Detail

Solver

```
public Solver(NumbrixSystem system)
```

Class Methods

Solver

```
public Solver()
```

Method Detail

solve

```
public void solve()
```

Solves the Numbrix problem

initialize

```
protected void initialize()
```

Initialize datastructures for search

constraintSatisfactionSearch

```
protected boolean constraintSatisfactionSearch()
```

A Constraint Satisfaction Search on Numbrix that will be used recursively by HeuristicSearch and return whether or not a solution was found.

Returns:

whether or not a solution was found

constraintSearch

```
protected boolean constraintSearch()
```

Performs the constraint search and returns if the grid is solved.

Returns:

state of solution

getTimeSpent

```
public long getTimeSpent()
```

Returns the amount of time spent solving the Numbrix grid

Returns:

the time spent solving the Numbrix grid

check

```
public boolean check()
```

Check to see if the gird has been solved

Returns:

whether or not the grid has been solved

add

```
public void add(int x,
                int y,
                int val)
```

Add the given values to the grid and structures

Parameters:

`x` - the x position of the object being added

`y` - the y position of the object being added

`val` - the value of the object being added

add

```
public void add(Triple triple)
```

Add the given triple to the grid and structures

Parameters:

`triple` - the triple being added

remove

```
public void remove(int x,
                   int y,
                   int val)
```

Remove the given values from the grid and structures

Parameters:

`x` - the x position of the object being removed

`y` - the y position of the object being removed

`val` - the value of the object being removed

remove

`public void **remove**(Triple triple)`

Remove the triple from the grid and structures

Parameters:

`triple` - the triple to be removed

undo

`public void undo()`

Removes modifications made form the constraint search

snakeString

`public String snakeString()`

Returns a String representation of snake

Returns:

a String representation of snake

getConstraint

`protected ConstraintSearch getConstraint()`

Returns constraint

Returns:

constraint

getHeuristic

`protected HeuristicSearch getHeuristic()`

Returns heuristic

Returns:

heuristic

getTimeElsapsed

`public String getTimeElsapsed()`

Returns a formatted string of the time spent in MM:SS:mm

Returns:

a formatted string of the time spent in MM:SS:mm

getSolutionFound

`public boolean getSolutionFound()`

Returns solutionFound

Returns:

solutionFound

Class Triple

numbrixgame.system.solver

```
java.lang.Object
  └numbrixgame.system.solver.Triple
```

All Implemented Interfaces:

Comparable<Triple>

```
public class Triple
extends Object
implements Comparable<Triple>
```

Data structure that holds unit, x, and y value

Field Summary		
private int	**value** Class Attributes	
private int	**x**	

private int	**y**

Constructor Summary	
Triple(int value, int x, int y) A data structure that contains a value, x, and y coordinate	

Method Summary		
int	**compareTo**(Triple triple) Compare in ascending order based on value	
int	**getValue**()	
int	**getX**()	
int	**getY**()	
String	**toString**()	

Field Detail

value

`private int value`

Class Attributes

x

`private int x`

y

`private int y`

Constructor Detail

Triple

```
public Triple(int value,
              int x,
              int y)
```

A data structure that contains a value, x, and y coordinate

Parameters:

`value` - value of triple

`x` - x coordinate of triple

`y` - y coordinate of triple

Method Detail

getValue

`public int getValue()`

getX

`public int getX()`

getY

`public int getY()`

compareTo

`public int compareTo(Triple triple)`

Compare in ascending order based on value

Specified by:

`compareTo` in interface `Comparable<T>`

toString

`public String toString()`

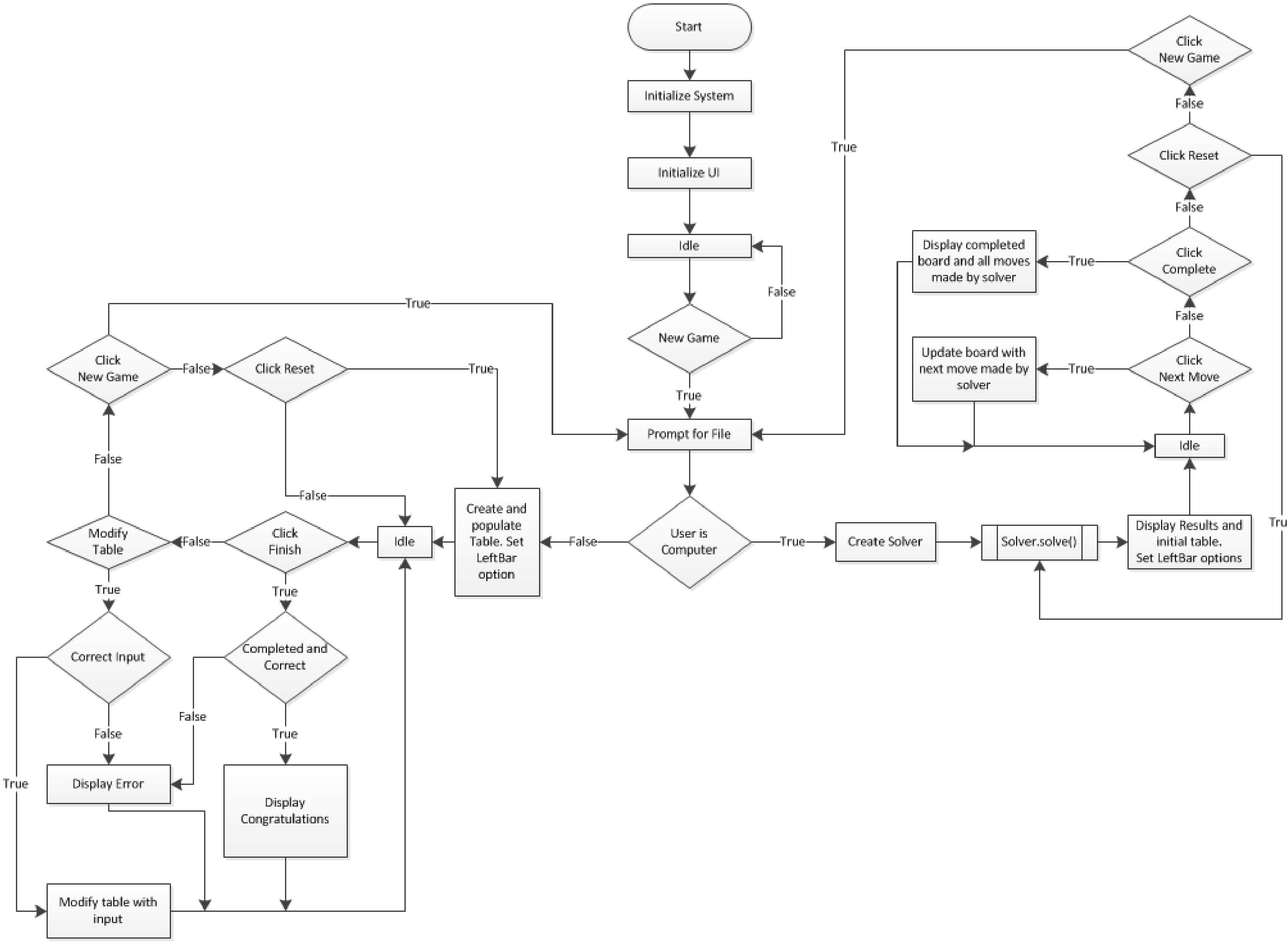
Overrides:

`toString` in class

I. FLOWCHART

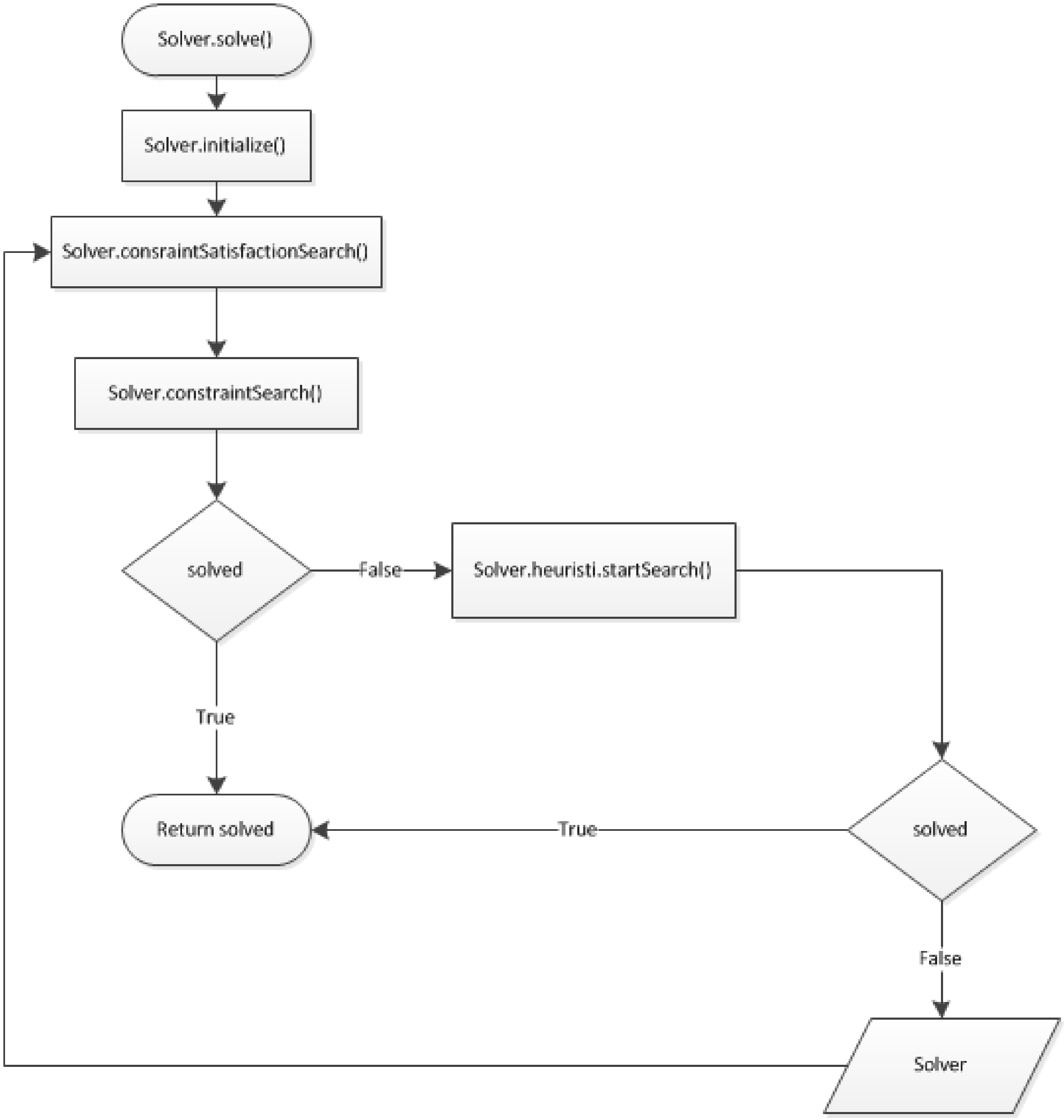
i. GENERAL OVERVIEW

The below flowchart is a basic representation of how the program starts and handles basic user input and output interactions.



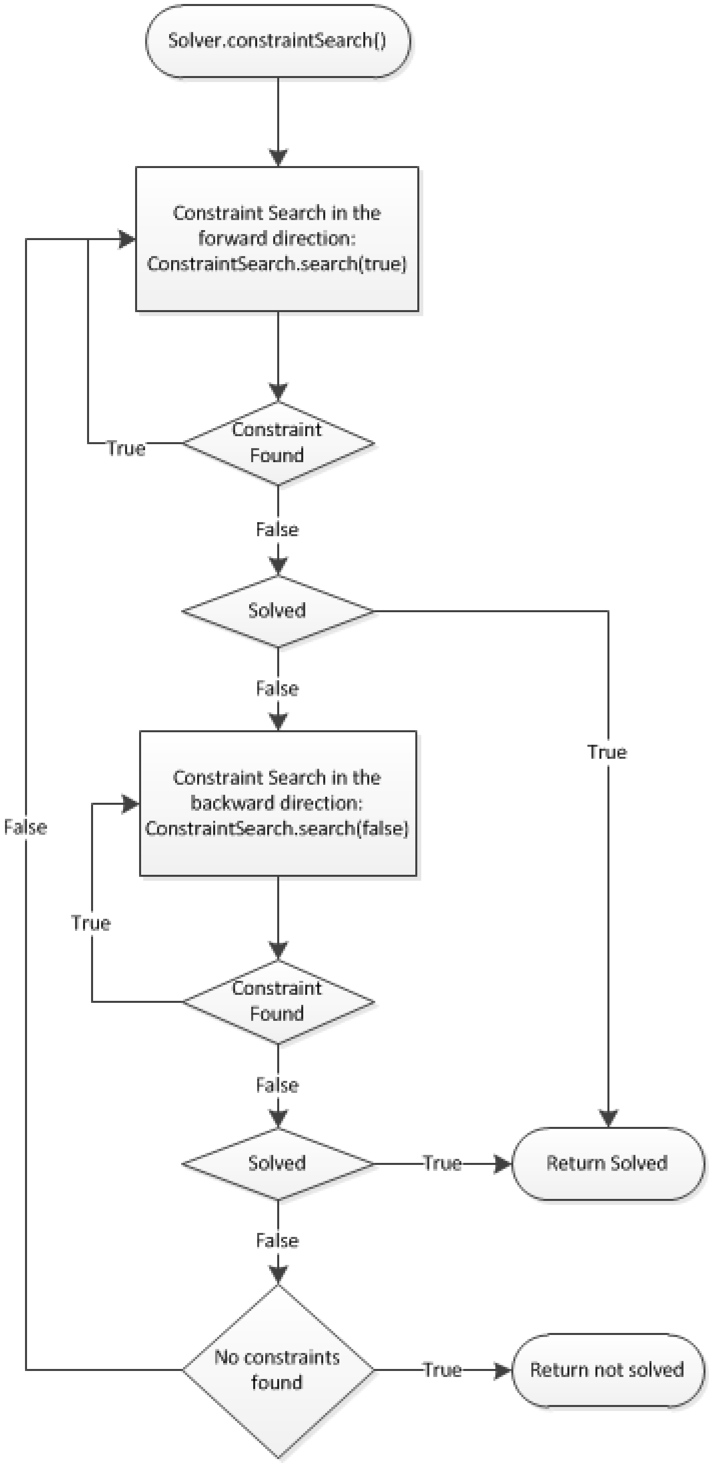
ii. SOLVER

The below flowchart is a high level representation of the steps taken by the Solver class to attempt to find a solution to the Numbrix grid.


Solver.solve()
Solver.initialize()
Solver.consraintSatisfactionSearch()
Solver.constraintSearch()
solved
False
Solver.heuristi.startSearch()
True
Return solved
True
solved
False
Solver

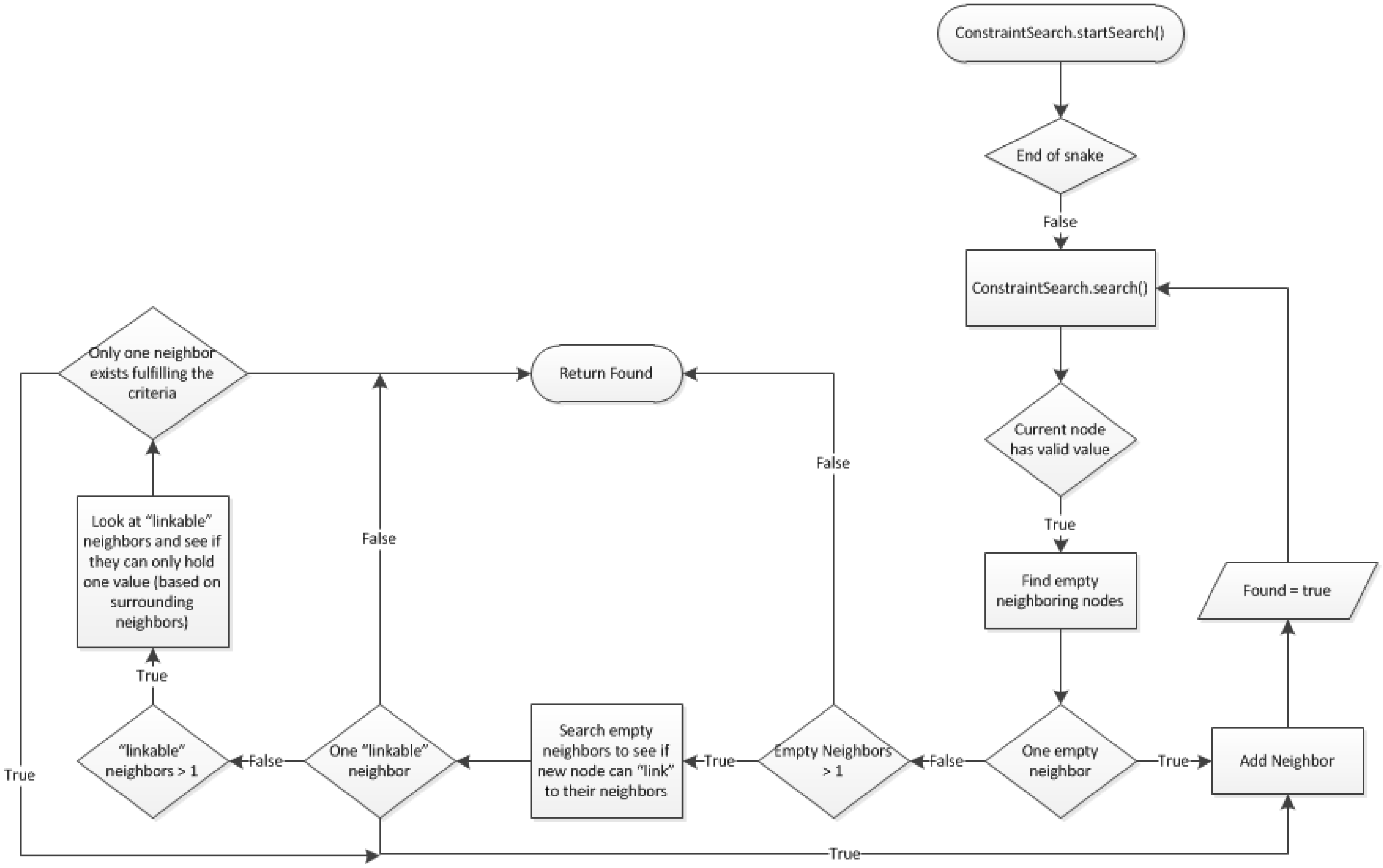
iii. SOLVERS CONSTRAINT SATISFACTION SEARCH

The below flowchart is a more low level representation of the process by which the solver utilizes the ConstraintSearch class via the Solver.constraintSearch() method.


Solver.constraintSearch()
Constraint Search in the forward direction: ConstraintSearch.search(true)
Constraint Found
True
False
Solved
False
True
Constraint Search in the backward direction: ConstraintSearch.search(false)
False
True
Constraint Found
False
Solved
True
Return Solved
False
No constraints found
True
Return not solved

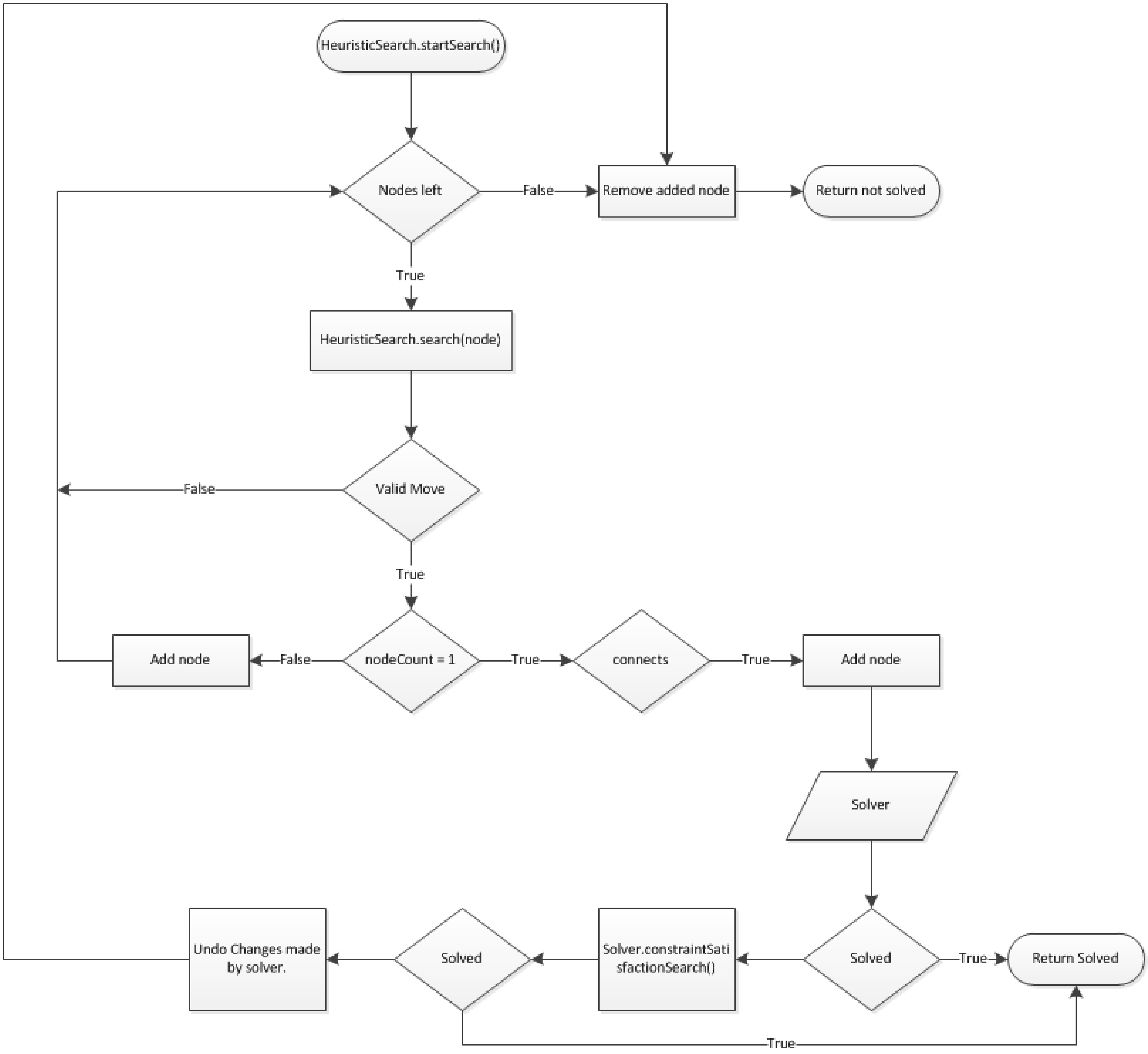
iv. CONSTRAINT SEARCH'S SEARCH METHOD

The below flowchart is a low level representation of how the ConstraintSearch class attempts to find constraints in a Numbrix grid and by extension find new nodes based on those constraints.


ConstraintSearch.startSearch()
End of snake
False
ConstraintSearch.search()
Current node has valid value
True
Find empty neighboring nodes
One empty neighbor
True
Add Neighbor
Found = true
False
Empty Neighbors > 1
False
Return Found
True
Search empty neighbors to see if new node can "link" to their neighbors
One "linkable" neighbor
True
False
"linkable" neighbors > 1
False
True
Look at "linkable" neighbors and see if they can only hold one value (based on surrounding neighbors)
Only one neighbor exists fulfilling the criteria
True

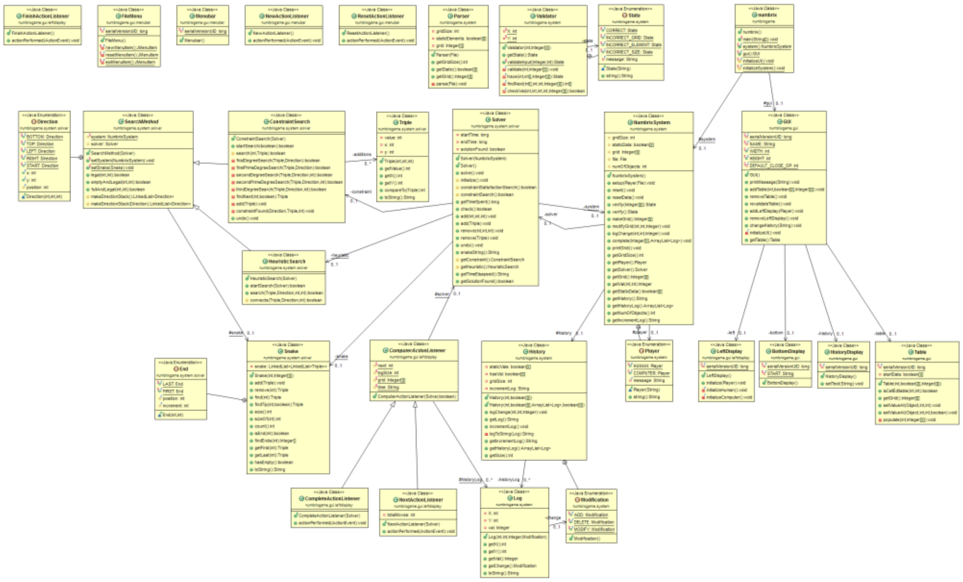
v. HEURISTIC SEARCH'S SEARCH MEHTOD

The below flowchart is a low level representation of how the HeuristicSearch class attempts to perform a mixture of a brute force search and constraint search (via the Solver class) to find a solution to the Numbrix grid.

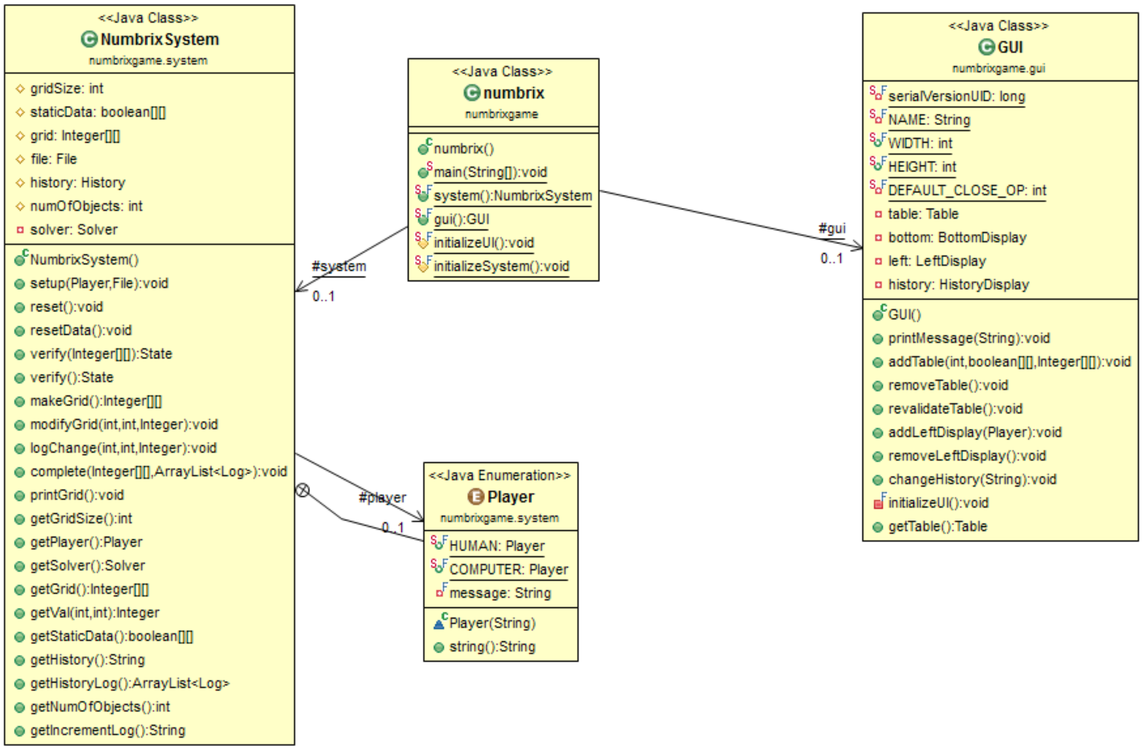


II. UML

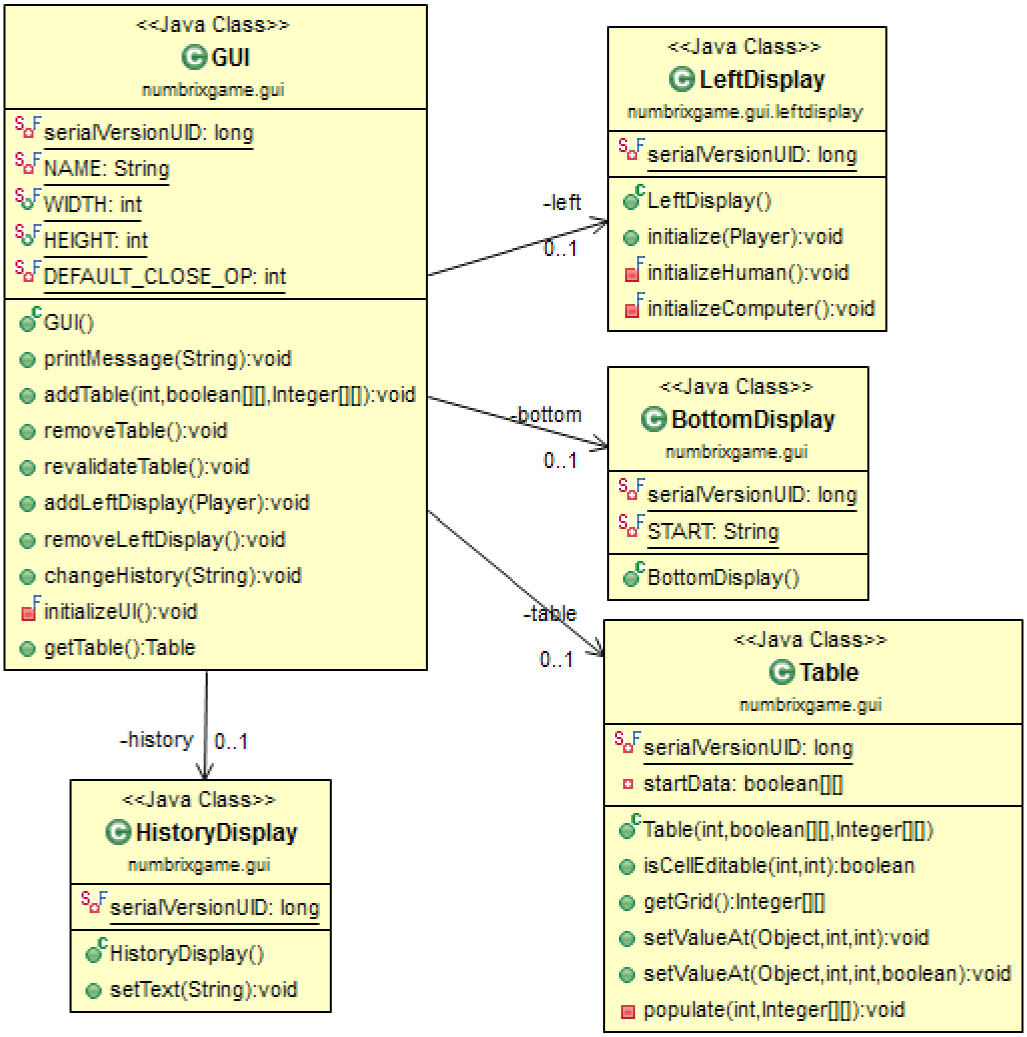
The UML diagrams in this section will be sorted by package so as to maintain a level of coherence and simplify the structure of the program. However, a UML diagram of the entire program is presented below so that a basic understanding of the interconnections between structures can be seen.



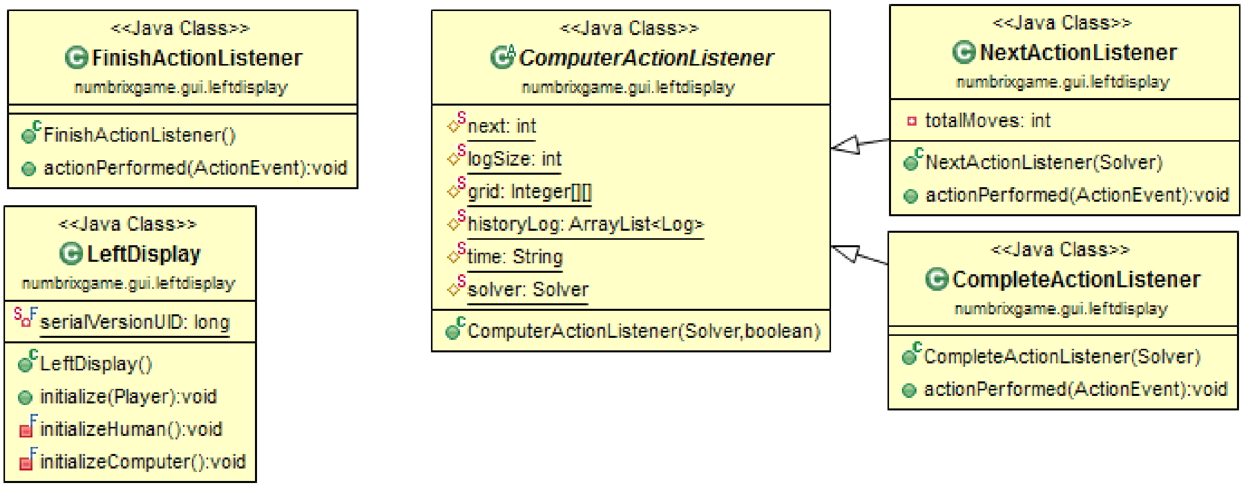
i. NUMBRIX



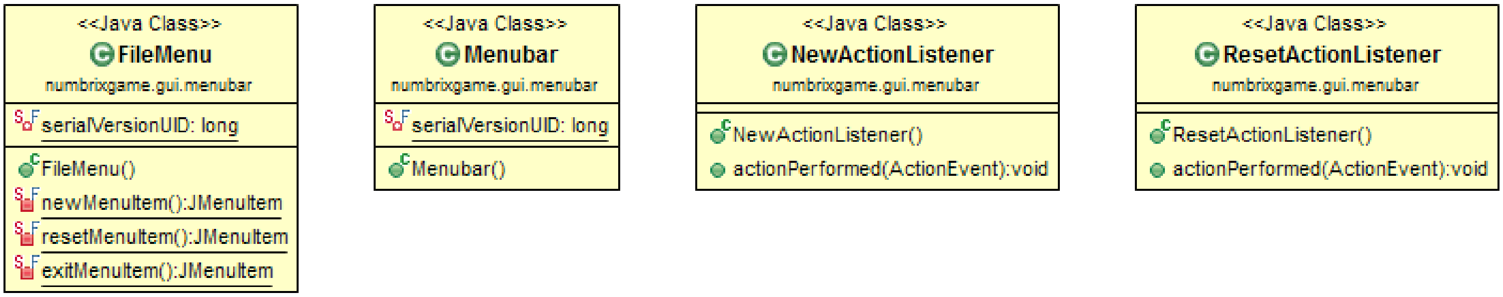
i. GUI



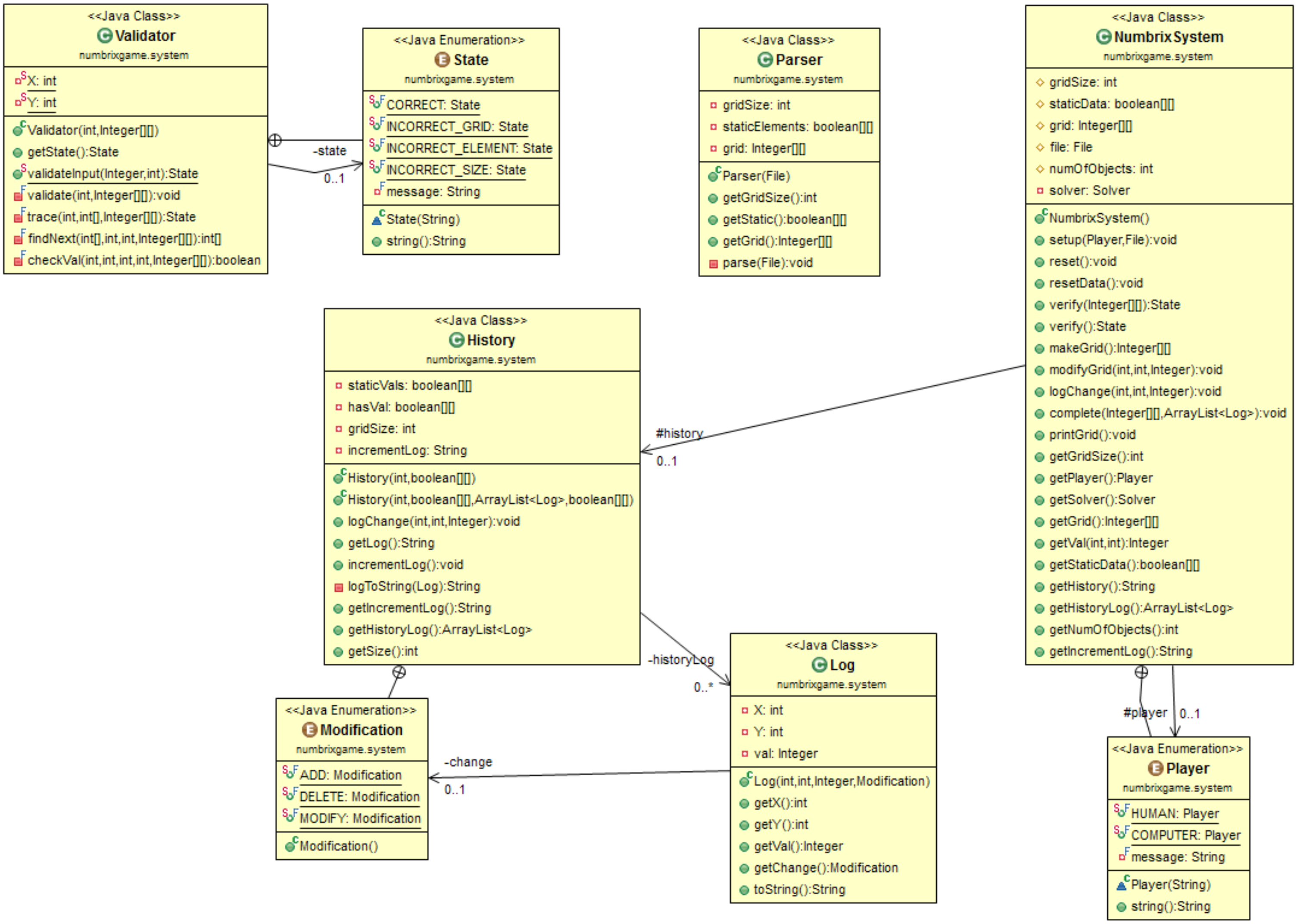
a. LEFTDISPLAY


<<Java Class>>
FinishActionListener
numbrixgame.gui.leftdisplay
FinishActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
ComputerActionListener
numbrixgame.gui.leftdisplay
next: int
logSize: int
grid: Integer[][]
historyLog: ArrayList<Log>
time: String
solver: Solver
ComputerActionListener(Solver,boolean)
<<Java Class>>
NextActionListener
numbrixgame.gui.leftdisplay
totalMoves: int
NextActionListener(Solver)
actionPerformed(ActionEvent):void
<<Java Class>>
LeftDisplay
numbrixgame.gui.leftdisplay
serialVersionUID: long
LeftDisplay()
initialize(Player):void
initializeHuman():void
initializeComputer():void
<<Java Class>>
CompleteActionListener
numbrixgame.gui.leftdisplay
CompleteActionListener(Solver)
actionPerformed(ActionEvent):void

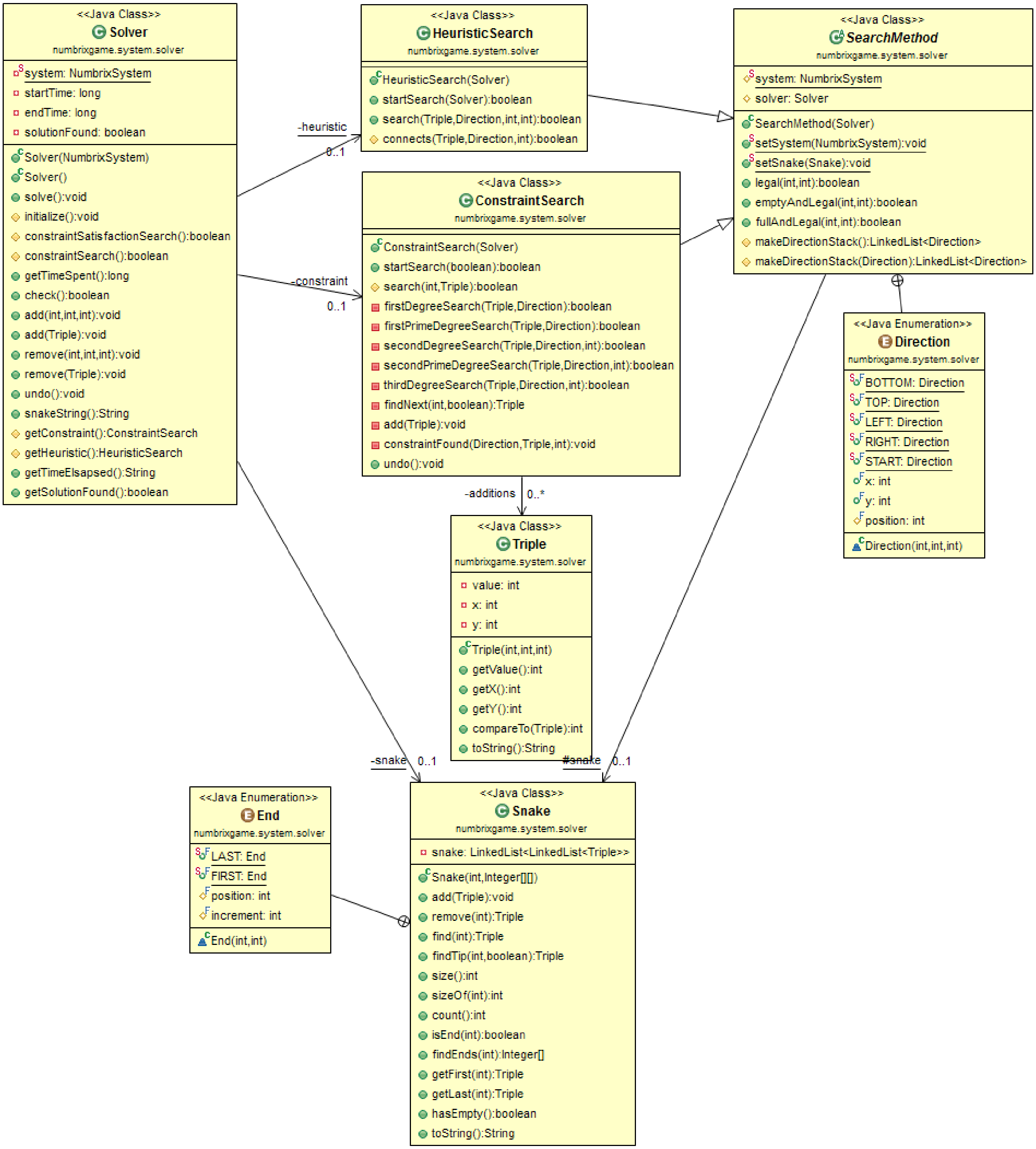
b. MENUBAR



ii. NUMBRIX.SYSTEM



i. SOLVER


<<Java Class>>
Solver
numbrixgame.system.solver
system: NumbrixSystem
startTime: long
endTime: long
solutionFound: boolean
Solver(NumbrixSystem)
Solver()
solve():void
initialize():void
constraintSatisfactionSearch():boolean
constraintSearch():boolean
getTimeSpent():long
check():boolean
add(int,int,int):void
add(Triple):void
remove(int,int,int):void
remove(Triple):void
undo():void
snakeString():String
getConstraint():ConstraintSearch
getHeuristic():HeuristicSearch
getTimeElsapsed():String
getSolutionFound():boolean
-heuristic
0..1
-constraint
0..1
<<Java Class>>
HeuristicSearch
numbrixgame.system.solver
HeuristicSearch(Solver)
startSearch(Solver):boolean
search(Triple,Direction,int,int):boolean
connects(Triple,Direction,int):boolean
<<Java Class>>
SearchMethod
numbrixgame.system.solver
system: NumbrixSystem
solver: Solver
SearchMethod(Solver)
setSystem(NumbrixSystem):void
setSnake(Snake):void
legal(int,int):boolean
emptyAndLegal(int,int):boolean
fullAndLegal(int,int):boolean
makeDirectionStack():LinkedList<Direction>
makeDirectionStack(Direction):LinkedList<Direction>
<<Java Class>>
ConstraintSearch
numbrixgame.system.solver
ConstraintSearch(Solver)
startSearch(boolean):boolean
search(int,Triple):boolean
firstDegreeSearch(Triple,Direction):boolean
firstPrimeDegreeSearch(Triple,Direction):boolean
secondDegreeSearch(Triple,Direction,int):boolean
secondPrimeDegreeSearch(Triple,Direction,int):boolean
thirdDegreeSearch(Triple,Direction,int):boolean
findNext(int,boolean):Triple
add(Triple):void
constraintFound(Direction,Triple,int):void
undo():void
-additions
0..*
<<Java Enumeration>>
Direction
numbrixgame.system.solver
BOTTOM: Direction
TOP: Direction
LEFT: Direction
RIGHT: Direction
START: Direction
x: int
y: int
position: int
Direction(int,int,int)
<<Java Class>>
Triple
numbrixgame.system.solver
value: int
x: int
y: int
Triple(int,int,int)
getValue():int
getX():int
getY():int
compareTo(Triple):int
toString():String
-snake
0..1
#snake
0..1
<<Java Enumeration>>
End
numbrixgame.system.solver
LAST: End
FIRST: End
position: int
increment: int
End(int,int)
<<Java Class>>
Snake
numbrixgame.system.solver
snake: LinkedList<LinkedList<Triple>>
Snake(int,Integer[][])
add(Triple):void
remove(int):Triple
find(int):Triple
findTip(int,boolean):Triple
size():int
sizeOf(int):int
count():int
isEnd(int):boolean
findEnds(int):Integer[]
getFirst(int):Triple
getLast(int):Triple
hasEmpty():boolean
toString():String

II. DESCRIPTION OF INTELLIGENCE IMPLEMENTED

The Solver can be broken down into three parts: the Solver class, the ConstraintSearch class, and the HeuristicSearch class. However, one of the most important parts to this program is the Snake class. Each of these four classes will be discussed below with an explanation of how they contribute to obtaining a solution to a Numbrix grid and any basic intelligence behind each class.

i. SNAKE

The Snake class is a data structure that keeps track of the nodes that have been "found" or "guessed". It does so by keeping a LinkedList of LinkedLists that each hold multiple Triple data structures. A Triple is a simple data structure that simply holds the value of a node, the x coordinate of a node, and the y coordinate of a node. The important part here is the LinkedList of LinkedLists which is stored in a variable aptly named snake. Specifically, the Snake class will only store consecutive nodes (based on value) in a single LinkedList. Should a new node that does not consecutively follow or head another node in the snake variable, a new LinkedList will be created to hold the new node. Similarly, should a node be removed from a LinkedList such that it breaks the consecutive pattern of the list, snake will break the broken LinkedList into two LinkedLists that only contain consecutively increasing nodes.

It is also important to note that the LinkedLists are stored in ascending order. This makes insertions, deletions, and searches for particular nodes easier and faster to do. Specifically, a search takes O(N) time to search. This is especially important because it makes it easy and quick to find the heads or tails of particular lists as well as find lists and nodes with the shortest gap between them.

ii. SOLVER

The Solver class is rather simple in nature. Its main purpose is to drive the ConstraintSearch and HeuristicSearch classes. However while it may not do much, it is this class that encapsulates the search method of the program. Solver utilizes the constraint satisfaction approach to find a solution to the Numbrix grid. This is done by first applying ConstraintSearch in a forward direction until no more constraints can be found. Once no more constraints can be found, the solver attempts to apply ConstraintSearch in a backwards direction until no more constraints can be found. It repeats this forward and backward search until no more constraints can be found. This repetition is done based off the hope that a new node found by a constraint will reveal new constraints that were not apparent beforehand. Once no more constraints can be found, the solver relies on the HeuristicSearch class to look for more nodes in the grid. More detail on the constraint search itself can be found in part c of this section.

Once the HeuristicSearch starts its search, the solver object is done. However, as soon as the HeuristicSearch finishes its search and no solution has been found, the HeuristicSearch will create a new instance of Search and call the new instances constraintSatisfactionSearch()

method. This call brings the Solver class and HeuristicSearch together into a recursive call. Until a solution can be found, HeuristicSearch will continue to create a new solver and call its constraintSatisfactionSearch() method which will in turn call HeuristicSearches startSearch() method. One subtle difference of note is that HeuristicSearches search is called from a static context whereas a new instance of solver is created every time HeuristicSearch finishes searching. This is partly an attempt to conserve memory. However, there is one more key reason for the utilization of a new solver. This will be discussed in part d of this section.

iii. CONSTRAINT SEARCH

The ConstraintSearches search method utilizes three important constraints when looking for the placement of a new node. However, before moving on to these techniques, one important concept must be covered first. It is here where the Snake class shines. When searching for a new node, one must first ask which node to look for. It is unnecessary to look for nodes that have a consecutively larger and a consecutively smaller node next to it. This is because all the nodes that this example node needs have already been found. Hence, because the Snake class keeps lists of only consecutively connected nodes, one can simply use the snake to find the first and last elements of each list in order to create a pool of nodes to search through. This is where the idea of forward and backward searches comes into play. A constraint search in the forward direction is simply a constraint search where the search starts with the last element in a list and progresses to the next lists last element until the final list is reached. Similarly, a constraint search in the backwards direction starts with the first node in the last list of the snake and works its way to the first node in the first node of the list.

The constraint search thus takes a node (A) from the tip of a list in the snake and looks for its neighbors. This is where the three constraints come into play. First, the search will look for all empty and legal neighboring nodes surrounding A. A legal node is a node that is not placed out of the bounds of the grid. A neighbor is any node that is directly on top of, under, to the right of, or to the left of a node. If there is only one empty and legal node next to A, then it must be the case the empty node holds the value that is the increment of A or decrement of A if going in the forward or backwards direction respectively.

If, however, there is more than one empty legal neighboring node, this claim cannot be made. Hence, the next constraint comes into play. This second constraint looks at the empty neighboring nodes and checks to see if these nodes are neighbored by a legal and populated node that has a value equal to double the increment (or decrement) of node A. This means that the populated neighboring node can act as a hint as to whether or not the empty node can “connect” node A and the populated node. If there exists only one empty node that can “connect” node A and the populated node, then it must be the case that the empty node contains the incremented (or decremented) value of A. Note that the only time two empty nodes will be capable of this “connection” are when the node A and the populated node are diagonally adjacent to each other.

If it is the case that there exists two empty nodes that can “connect” A and the populated node, then ConstraintSearch applies one final constraint. In this final constraint, ConstraintSearch once again looks at the neighbors of the remaining empty nodes and checks to see what values the empty node can hold. If it is the case that there exists only one empty node than can “connect” only one pair of nodes together, then it must be the case that this single empty node must hold the increment (or decrement) of A.

iv. HEURISTIC SEARCH

Once the HeuristicSearch starts its search, the solver has given up on finding “sure” answers and takes a brute force approach to the solution. However, that is not to say the solver completely relies on guesses from here on out. This search starts, once again, with the Snake object. HeuristicSearch will find, from the Snake, the two nodes at the ends of each list that have the shortest gap from each other. It will then attempt to bridge this gap. The reasons for this are two fold. First, because the solver will be making guesses, it would be better to make the smallest number of guesses possible. Hence, the smallest gap is chosen to be filled in. Second, this search utilizes a depth first search and it is quicker to do a depth first search with a shorter known distance than a longer one.

So, given a starting node with a known distance, HeuristicSearch attempts to find a path to the next node in the snakes list. It does so by taking a node and searching its neighbors (in an arbitrary order). If the neighbor is empty and legal, it will then perform the search again. Hence, HeuristicSearch utilizes a recursive search method to look for a path that connects to the head of the next list in the snake. If no more paths can be taken, the method will simply terminate and move on to the next neighbor. Once the node count has reached one and the correct neighbor is found, HeuristicSearch creates a new solver and calls its constraintSatisfactionSearch(). The reason for creating a new instance of solver is so that it can create a new instance of ConstraintSearch. This is because there is a possibility that the search will prove impossible with the given guessed path and so the HeuristicSearch will need to back track and remove any changes made.

While it is easy to remove and add changes from the HeuristicSearch due to its recursive nature, additions made by ConstraintSearch are not so easy to keep track of. Hence, by creating a new instance of ConstraintSearch and having each instance keep track of its changes while also only having each instance perform a search and make changes between heuristic searches, it is possible to keep track and undo changes made by the ConstraintSearch class. Thus, once a HeuristicSearch finds that it can no longer progress any further down a given path, it tells its solver to undo changes made by the ConstraintSearch and then the HeuristicSearch undoes the node it added at its given search method. By taking this approach, the Solver should be able to cover every plausible path (brute force) while cutting out unnecessary paths in the process by doing a combination of guessing and constraining on the grid.

III. WHAT I WOULD HAVE DONE DIFFERENTLY

One of my major gripes with the project were in fact minor mistakes made on my part. Small logical errors and bad implementations of code were rampant. Among these mistakes was my forgetting the coordinate convention. When creating the solver, I used one convention for coordinates and when creating the first part of the project I used a different convention. This led to some initially confusing output when I attempted to combine the solver and the Numbrix system. My one other issue with my project is the structure of my system (everything other than Solver). Towards the end, it felt like I was hacking together my code in order to get certain output to display correctly. In hind sight, I would like to have had the chance to remake the system and increase the separation between the GUI and the System classes while creating well defined access to different elements of the program.

Thankfully, there is not much I am unhappy with in regards to the solver. If I had time, I would like to have looked for and implemented more constraints in the ConstraintSearch class. I would also like to have added a bit more intelligence in the HeuristicSearch so as to weed out more bad paths before taking them. Most notably, I noticed that when creating a new connection between nodes, it was possible to create “islands”. That is, there were times when combining nodes that two or more separate empty areas would be made. After forming this connection, instead of continuing to apply constraints and heuristic searches until it is impossible to do so, I could instead weed out the path entirely (and save much time) by checking to see if the islands were fillable. That is, if the borders of the islands only contained non-tip nodes from the snake, then it would be impossible to fill them. Hence, I would know right then that the path I created was a bad path. There are other constraints I could use to check the islands as well, but the important thing of note is that one could use this concept of islands to help prune bad paths from the search as opposed to taking the bad path and every branch along the bad path.