Tantrix Game

Junyi Li

How to switch Models

Please go to RunTantrixGame.java file, find line 22 and line 23
// this.model = new Tantrix2DArrayModel();
this.model = new TantrixDoubleLinkedNodeModel();
Comment one of these two lines to switch models

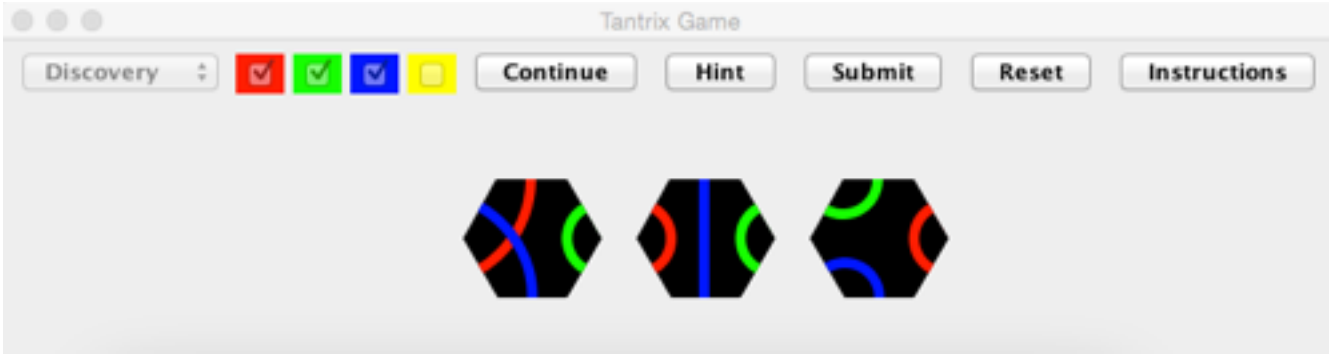
How To Play

The start of the game

The default game for the player is Discovery, the default colors are Red, Green and Blue.
When the user starts the program, the user will see an instructions screen,.Please read the instructions before you play the game. During the game the user can still press the **"Instructions"** button to see the game instructions without influencing the game.
Before the user presses **the "Play"** button, the user can choose a desired game from the drop down menu and a trio of colors from the check boxes. The "Hint", "Submit", and "Reset" buttons are disabled at this time.

After the user selects the **"Play"** button, the user can not change the game type and the game colors again unless the user resets the game. The **"Play"** button will change to
"Continue". The "Hint" ,"Submit" and"Reset" buttons will be enabled.

"Discovery" has 8 levels. In the first level the user will begin with three pieces. The user will receive an additional piece each time she/he progresses through the levels. The pieces are provided in order from index 0 to 13



If the player chooses the **"Solitaire"** game, the user will get a random piece from the 14 pieces in each level. No duplicate pieces will be provided during the game. Once the player selects the **"Play"** button, a **"Score"** label will appear.

Basic rules

How to move and rotate a piece

1. To drag a piece, the player first clicks the piece then drags the piece using the mouse or trackpad. To "Rotate" the piece, just click and release on a piece without dragging; the piece can rotate 60 degree each time it is clicked and released.
2. Since I used a snap to grid function, two pieces may snap to the same grid(I wish I have time to do some checking in my code). When you move a piece, the piece may overlap with the other pieces. If you are not sure ,you can count the number of pieces to see whether a piece is missing or not. If a piece overlaps another, just drag that piece to another grid.

Discovery game

1. After the user gets the first three pieces , the user has to form a loop using these three pieces. The user can select the **"Hint"** button to get the possible loop color for the current level. The user can also choose to ignore the hint and form a completely different loop.
2. If the player is ready to enter the next level, the player has to press the **"Submit"** button. The program will tell the player whether the current level was completed or not. If completed, the user will select **"Continue"** to enter the next level; otherwise the player has to stay in the current level and try it again.
3. When the user enter a new level, the player breaks up the previous loop, adds the next piece and forms a new loop.
4. The hint color is the easiest loop to form

Solitaire Game:

1. The user gets a random piece in every level, if the player wants to see the current score, the player has to press **"Submit".** Once the player submits the current solution, the pieces that are currently placed on the screen are not allow to change. The drag and rotate functions are disabled.
2. Press **"Continue"** to get another piece.
3. The score is accumulated by the computer. The player will receive one point for each piece in the longest line or 2 points for each piece in the longest loop. The maximum score is 28.

The user can see the game instructions when ever they press “Instructions” and can also press the “Reset” button to reset the game. During the game, pop-up messages will be shown at an appropriate time to remind the player to progress to the next level.

Data Structure Selection

1. For each game, the number of total game pieces are fixed(40 for the discovery game or 56 for the solitaire game). The 56 pieces only include 14 different patterns. Once the piece colors are set, only 14 pieces need to be considered. Since the data structure does not need to resize, it can have a fixed size.
2. During the game, the model needs to update the game status(check for loops, calculate the score, etc.) based on the user’s behaviors, which means the structure should be able to update. Random access to the information stored in the data structure is also necessary. An array is good for this.
3. In order to use the 14 patterns to represent the 56 pieces, a color mapping is needed for the 4 group of pieces, which can be represented as a hashMap.
4. Every piece includes six edges; pairs of edges have the same color. A another mapping is needed for the color and edge indexes.
5. For the algorithm, I need to check whether the player formed a loop or not and also find the longest line or longest loop. The data structure should provide a way to do this. A linked list is a good structure to use.

Considering the above situations, the following are the data structures that I utilized:

Array, Double-linked list, HashMap

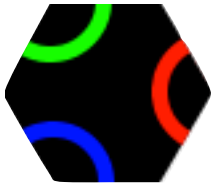
Another issue is representing the information that is stored on the tantrix piece, and,based on the representation, choosing the appropriate data structure.

I had the following ideas to represent the pieces:

Idea NO.1

-Using an array to store the color information
-Using another 2D array to store the corresponding edge indexes for each piece
-Mapping these two arrays correctly
For example:
The above pieces are the third piece from the 4 groups, assign edge indexes for each piece. upper right conner is index 0, upper left conner is index 5.
Pairs store the same color edge indexes






1 **2** **3** **4**

```
Color [3]  colorArray;
Pair[14][3] edgeIndexes;
```

For the first group:

```
colorArray = {yellow, red, blue}
edgeIndexes = {…
               …
              {(5,4), (1,0), (3,2)}
              …}
```

yellow —>(5,4)
red —>(1,0)
blue —>(3,2)

Do the same thing for the other 13 pieces in the first group.

Then map the colors in the first group to the other groups

group 2: yellow->yellow; red->green; blue->blue
group 3: yellow->green; red->red; blue->blue
group 4: yellow->yellow; red->green; blue->red

for other groups, I just need to map the color correctly. The edge indexes will stay the same, since the pattern does not change.
By using the two arrays, and keep a mapping between them, I can represent all the 56 pieces, without using extra spaces. It also provides an efficient random access (O(1))to each piece's edge index mapping information. If I want to compare whether two edges in different pieces have the same color, I need to get the two pieces first(O(1)), and then find the closest edge.

Idea NO.2

Using a hashMap:
key: color
value: all corresponding edge indexes(from piece 1 to piece 14)

this method gives a clear relationship between the color and the edge. But a hashMap is not efficient for random access and information update.
For edge matching, we need to check the edge colors, which means we need to find the key based on the value. The hashMap is not very efficient So this ideas was not used.

Idea NO.3

still use two arrays, but represent the pieces differently

```
Color [3] colorArray;
int [14][6] edgeColor;
```

Using 3 integers to represent 3 different colors

In the first group: 0 —>yellow; 1—>red; 2—>blue
In the second group: 0—>yellow; 1—>green; 2—> blue
In the third group: 0—>green; 1—>red; 2—>blue

In the fourth group: 0—>yellow; 1—>green; 2—>red

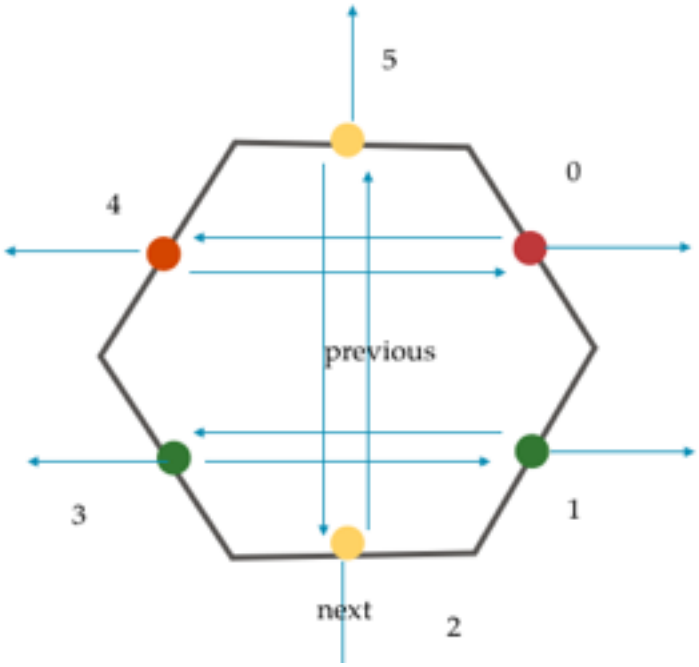
for example:
The third piece in the first group

```
colorArray = {yellow, red, blue}
edgeColor = {…
             …
               {1,1,2,2,0,0}
             …
            }
```

This way if two edges in different pieces have the same color, the color information from the edge color array can be access directly.

Idea NO.4

Since we need to check the loop, the easiest way to check a loop is to determine whether the start and end points meet after iterating through the edges. A double linked list can do this. For the tantrix piece, we need to check every possible loop from six directions. Thus it is possible to use double linked nodes.



Every node has the following information:
An index.
Two pointers: previous pointer, next pointer.
Color information

The previous pointer points to the node which has the same color inside its own piece. the next pointer points to the node which belongs to another piece. These two nodes have the same color. Starting from a node, the pointers can be iterated until the final pointer is null(the nodes will have the same color). If the final pointer points to the initial node, it means the graph has a loop.

This data structure is good for the loop checking algorithm.

In order to represent all the pieces, 14 pieces were created and store into an array.

Another data structure to store the mapping information was needed.
The representation of this structure is identical with idea NO.3.

The data structures looks like the following:

```
pieceArray = {piece0, piece2,……piece13}
edgeColor = {…
                          …
                         {1,1,2,2,0,0}
                         …
            }
```

Once the user selects the desired color, the edge-color mapping initializes the colors for each piece's nodes. Based on the color of each node, set the adjacent pointer's color to be the same. After the game begins, if two like-colored edges touch, set the corresponding nodes's next pointer to each other.

The following has been implemented in the program:
Idea NO.1: a color array + a 2D color- edge indexes mapping array(edge pairs)
Idea NO.4: six double linked nodes + 2D color-edge mapping array

Model View Controller Design

Abstract model: `TantrixAbstractModel.java`
First model : `Pair.java, Tantrix2DArray.java`
Second model : `Node.java, TantrixDoubleLinkedNode.java`

How to draw Tantrix Piece: `TantrixPiece.java`
View: `TantrixView.java`

Controller: `TantrixController.java`

Model

An abstract model was used for the program to make it easy to switch between different models. The actual models inherits from this abstract model
It includes these abstract methods:

```
//set the color for the model(the tantrix piece) based on the user's choice
    abstract void setColor(Color[] color, Color[] loopColor);

//  set the current game level
    abstract void setCurrentLevel(int level);

 //   show player the messages, including game instructions, game status
    abstract String getMessage(String inforType);

  //  update model after the user rotate the piece
    abstract public void colorUpdateAfterRotate(int pieceIndex);

   // get the edge index pairs which are mapped to three color for the required
piece
    abstract public Pair[] getEdgeIndexesForCurrentPiece(int pieceIndex);

   // reset model
    abstract public void resetModel();

    // shuffle the piece index, so the computer can provide a random piece
    abstract void shufflePieceIndex();

    //get the random piece index
    abstract int getRandomPieceIndex(int i);



    //find possible path for discovery game
```

```
    abstract void createPossiblePathForDiscovery(ArrayList<Point2D.Double>
locations, int length);

//check whether these possible paths include a longest loop
    abstract public Boolean checkLoopForDiscovery();


 //find possible path for solitaire game
    abstract void createPossiblePathForSolitaire(int length,
ArrayList<Point2D.Double> pieceLocations);

//calculate score for solitaire game
    abstract int caculateScore(int currentPieceIndex);
```

The 2DArray model also includes these main methods:

```
//check whether two edges have the same color
public Color checkForSameColor(int pieceIndex1, int edgeIndex1, int
pieceIndex2, int edgeIndex2);

//a generic function to check whether the given pieces which are connected
by the same color edge can form a loop
public boolean findLoophelper(ArrayList<Pair> l);

// After the player place a new piece, if a new piece connect two same
color edges, call this function to link the two path together
public void connectLinks(Color c, int currentPieceIndex, int
currentPieceNeighbor, int index) ;
```

The linked nodes model includes these extra methods:

```
//this method is for discovery game, since every level is independent,
before the next loop checking, break the previous level's pointer
connections.
public void breakConnection(int index);

//algorithm to check whether the user formed a longest loop
public void findLoopHelperForDiscovery();

//algorithm to find longest loop or longest line for solitaire game
public void findLoopOrLineHelperForSolitaire();
```

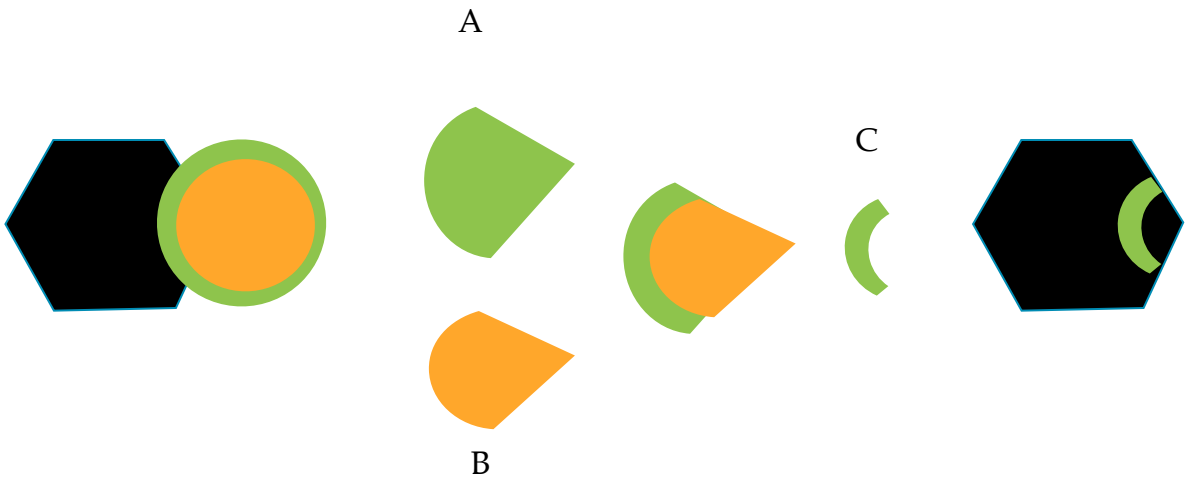
View

The view is responsible for all the drawing and layout.
The tantrix piece class extends JComponent. all necessary
methods and properties that this tantrix piece may need have been encapsulated.
Pieces can be dragged and rotated. When a piece is placed by a user, the snap to grid method will place the piece to the closest grid.

```
//every piece has an index
private int pieceIndex;

// use polygon to draw hexagon
private Polygon p = new Polygon();

//the edge length of the hexagon
private final int edgeLength;

//colorEdgeIndexMap : key:color - value: corresponding edge indexes information
private final HashMap<Color, Pair<Integer, Integer>> colorEdgeIndexMap;

//coordinatesList : store six vertices's coordinates of the hexagon
private final ArrayList<Pair> coordinatesList;

//draw tantrix Piece inside an Image
BufferedImage sprite;

//whether the piece needs to be rotated
Boolean rotate;

//the rotate count
int rotateCount;
```

Basic idea to draw the Tantrix piece: using shape intersection and subtraction

First draw a hexagon. Based on the relationship between the hexagon and the lines inside the hexagon, create two circles which have different radii but the same center. Do an intersection between the two circles and hexagon. You will get shape A and B. Then do intersections between A and B, you will get shape C, Then fill shape C within a hexagon.



Controller:

The controller is responsible for handling user interaction. Based on the user's actions, the controller talks to the view and the model. It calls the procedures in the view or model to finish some tasks. For example, when the user selects the game type and game color, the controller tells the model to initialize itself. When the user presses **"play"** or **"continue"**, the controller gets the data from the model and passes the data to the view. The view will draw the game pieces on the screen. When the user selects **"submit"**, the controller will get the piece location information from the view and pass the location to the model. The model will check whether two pieces touch and whether two edges have the same colors. Finally, it will check loops and calculate the game score.

Challenges

One of the biggest challenge was drawing Tantrix pieces. I tried two different ways. One is to calculate the coordinates of the lines inside the hexagon and draw the lines on the hexagon. This way, I have to do complicated math very carefully, and it is hard to control the position of the line.
So I tried another method; using the intersection, subtraction between shapes. If I do this way, I only care about how to find the center of the circles. And the line I draw inside the tantrix piece will not go outside the shape.

The next challenge was determining whether the player formed a loop. In the 2D array model, I needed an algorithm to check whether the given graph has a loop or not. I used an extra array to mark whether the nodes have been visited. I used an

arrayList as a stack. If the edge has been visited, I pushed the two nodes onto the stack. Then I find another unvisited edge which shares the same node with the previous one, repeating this procedure until all nodes are marked as visited. Finally, if the first node and last visited node are the same, the graph has a loop. In the double linked node model, every node has two pointers. I initialized the previous pointer to point to the node that has the same color. When I do the loop checking for the discovery game, I just needed to pick a random piece, and check the six nodes(edges), keeping track of the next pointer of each node. If one of the nodes returns to the beginning and the track count equals to the number of pieces, the user formed a longest loop.
For the solitaire game,I needed to keep track all possible loops and lines, and compare these values.

Reference

I had some issues with laying out the pieces correctly after I moves the pieces and repainted the panel. The default Java layout manager would move the pieces into incorrect locations. So I searched the internet and applied Rob Camick's component mover interface to my tantrix pieces, though I implemented the interfaces by myself. (ComponentMover.java)
http://tips4java.wordpress.com/2009/06/14/moving-windows/

I used his dragLayout to keep the components in the right location.
http://tips4java.wordpress.com/2011/10/23/drag-layout/

I also used a customized label for my score label from a java tutorial.
http://www.java2s.com/Code/Java/Swing-JFC/Labelwithvariouseffects.htm