

CS1322 Homework 9

Due: Sunday, April 10 @ 6:00 pm

Mirror-Box Game

For this homework assignment, you'll write a simple computer game that will simulate a light beam traveling in a maze of mirrors. A maze that contains no mirrors looks like Figure 1.

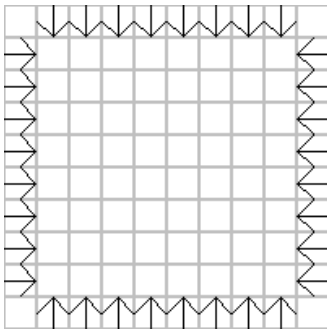


Figure 1

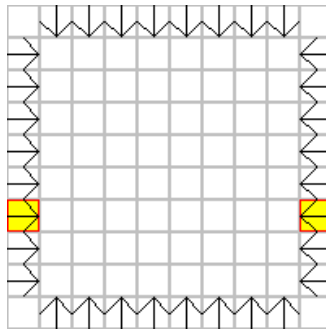


Figure 2.

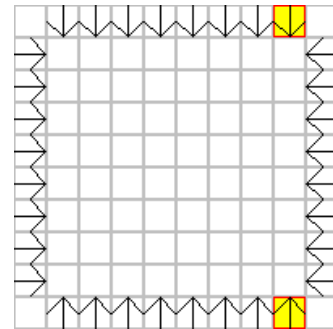


Figure 3

The arrows that appear along the edges of the board are simultaneously *lasers* and *light detectors*. If a laser is clicked, it fires a beam of light into the maze, and one of the detectors receives the beam of light.

Figures 2 and 3 show what happens when a beam of light is fired into a maze that has no mirrors. For example, in Figure 3, if the user clicks the rightmost laser on the bottom row, then the rightmost detector on the top row is highlighted. Alternatively, the user could have clicked the rightmost laser on the top row, causing the rightmost detector on the bottom row to be highlighted.

Now we'll add mirrors to the maze. The mirrors reflect the traveling light beam, making its path more interesting.

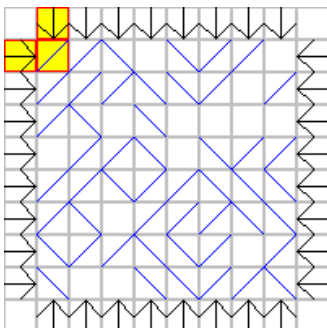


Figure 4

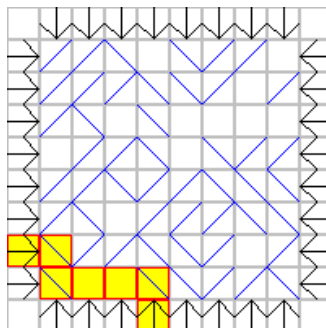


Figure 5

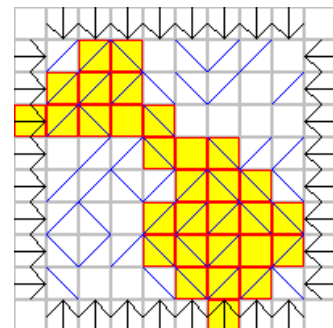


Figure 6

The blue diagonal lines represent the mirrors that reflect the light beam. To make this behavior easier to understand, we highlighted the squares that the light beam passes through as it traverses the maze.

Note: your final program *will not highlight the light beam's path* as we have in Figures 4-6.

Now that you understand how the lasers, detectors, and mirrors work, we'll turn the idea into a simple game.

- In the real game, the mirrors are *initially hidden* and are placed in random locations on the board. The user can click on a laser on the edge of the board, and only the selected laser and the receiving detector are highlighted.
- The user can also *guess which squares contain mirrors* by clicking on a square inside the maze. If that square actually contains a mirror (a correct guess), the mirror is *revealed*, and the mirror is shown for the rest of the game. The game should keep track of the number of correct and incorrect guesses for a game.

Figures 7-12 show a game in progress.

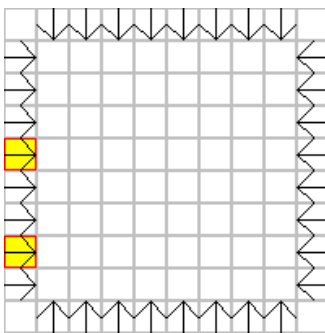


Figure 7

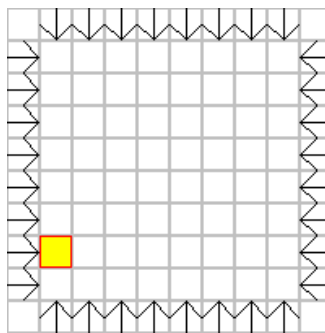


Figure 8

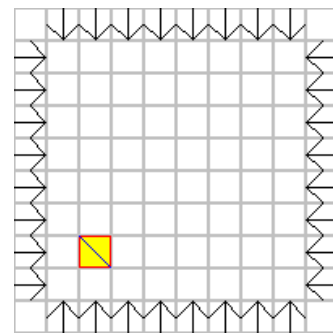


Figure 9

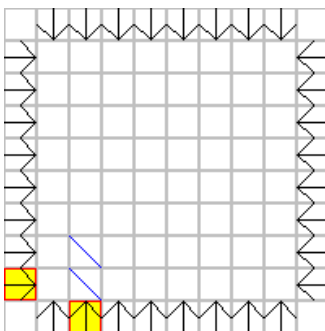


Figure 10

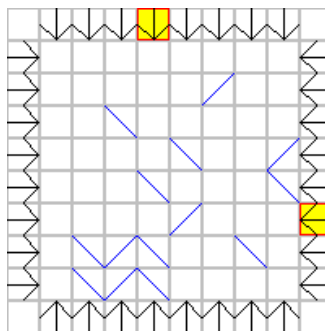


Figure 11

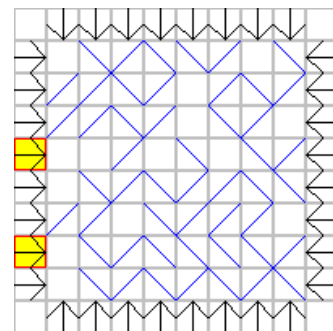


Figure 12

- **Figure 7** - The game has just begun. The player has initially clicked on one of the two highlighted lasers, causing the other light detector to be highlighted.
- **Figure 8** - After observing the two lasers/detectors in Figure 7, the player has incorrectly guessed that there is a mirror adjacent to the lower laser. Notice that the highlighting of the lasers has disappeared, and only the empty square is highlighted.

- **Figure 9** – The player has correctly guessed that there is a mirror in location shown. Again, all previous highlighting is removed. From now on, this mirror will always be shown.
- **Figures 10, 11, and 12** – After more correct guesses, the game reveals more and more of the board to the player. Try tracing the path in Figure 12. Where in the maze are more mirrors possible?

After the player is satisfied that all of the mirrors have been discovered, the game automatically brings up a **Victory** message box and resets the game board, as shown in Figures 13 and 14.

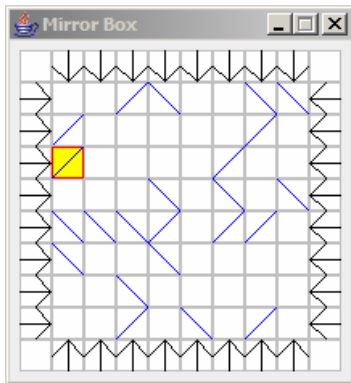


Figure 13

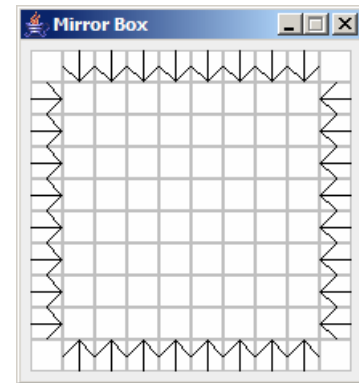


Figure 14

- **Figure 13** – The player has just found the last mirror in the maze. The **Victory** message box appears, telling the user how many mirrors were in the maze, and how many incorrect guesses (clicks on empty squares) were made.
- **Figure 14** – After the OK button in the message box is clicked, the game board is cleared. The board is randomized again, and all mirrors are hidden.

Requirements

Gameplay:

- The area of the game board that can contain mirrors should have a size of 8x8 squares. Lasers/detectors should appear along the edge of the board; the entire board should have a size of 10x10 squares.
- When the game board is created, each square should have a 50% chance of containing a mirror. If a square has a mirror, it should have an even (50%-50%) probability of being a left- or right-oriented mirror (i.e., whether it would appear as / or \). All mirrors are initially hidden.
- Clicking on any square on the board will highlight it.
- Clicking on a laser/detector square will also highlight the square of the detector that receives the light beam.
- If there is a mirror in the square that the user clicked, then that mirror should be visible for the rest of the game.
- Once all the mirrors have been discovered, the total number of mirrors in the maze and the number of incorrect guesses should be displayed in a message box. The game board should then be created again with a new random set of mirrors.

User interface:

- Use a single `JPanel`-derived class to implement the game board.
- Use a `MouseListener`-derived class to handle clicks on the game board. See section 7.9 (page 412) in the textbook for an example of how to intercept mouse events.
- Use a message dialog to display the number of mirrors and incorrect guesses after the game is over. See section 5.11 (page 260) in the textbook for an example of how dialog boxes work.

Turn-in:

- The class used to run your program must be named `MirrorBox`. In other words, the `MirrorBox` class must have a `main()` method that runs the game.
- Include all of the Java source code files that are required to compile your project.
- **Remember to retrieve your submission after uploading it.**

Hints and Suggestions

You are free to design your program however you wish. If you need help getting started, here is an overview of how the example solution was designed:

class MirrorBox

Used to run the game. It instantiates a `GamePanel` to display the user interface.
`public void main (String[] args)`

class GamePanel extends JPanel

The graphical representation of the game board. This class contains no game logic; it only handles mouse events and the drawing of the board. It instantiates a `GameBoard` to handle game logic.
`public void paintComponent (Graphics g)`

class GameBoard

Handles the game logic. Uses a two-dimensional array of `Squares` to represent the board.
`public ArrowSquare fireLightBeam (int x, int y, ArrowSquare in)`
Returns the `ArrowSquare` corresponding to the light detector that would receive the beam of light fired from the laser in at (x, y).
`public void clickSquare (int x, int y)`
Called whenever the user clicks on a square.

class Square

Represents a blank square on the game board. Provides default functionality for an empty square.
`public void draw (int x, int y, Graphics g)`
Draws this square at the specified location.
`public LightBeam collide (LightBeam in)`
Returns the new position and direction of a traveling particle of light (in) that moves into (or collides with) this `Square`.

class MirrorSquare extends Square

Represents a mirror square on the game board.
`public void draw (...)`
`public LightBeam collide (...)`

class ArrowSquare extends Square

Represents a laser/light detector square on the game board.
`public void draw (...)`
`public LightBeam collide (...)`

class LightBeam

Represents the current location and direction of a traveling particle of light.

The table above contains most of the useful methods in each class. Feel free to experiment with your own design.