Description

Sim is a simple two-player game that uses two colored pens and a sheet of paper. Each game starts with six dots drawn at the vertices of a hexagon. Each player chooses one colored pencil at the start and then takes turns drawing a colored line between two dots. New lines may be drawn between dots only when an existing line does not already connect the dots. The first player who is forced to complete a triangle with their chosen color loses the game.

Sim will never end in a tie. This is a fact that may be proven using a branch of mathematics called Ramsey Theory. Ramsey theory shows that any two-coloring of the complete graph involving six vertices must contain a monochromatic triangle, which translates into a loss by one of the two players.

Write a software version of Sim using the DoodlePad package of graphic objects. Your program should start with six shapes in a hexagonal pattern that represent game dots. Clicking on two dot shapes in sequence will cause a Line object to be created that connect the selected dots. Line stroke color should indicate the player who created the Line. It is not necessary to check that a Line already exists between two dots.

Note: Source code for Java applets that implement Sim is available online. Don't be led astray by solutions you may find. They will not help you complete this project.

Hints

- Think of your game as alternating being two states: (1) waiting for a player to click the first dot which defines the starting coordinates of a line, and (2) waiting for a player to click the second dot for the ending coordinates that ultimately create a line with an appropriate color.
- My implementation of Sim involved three static methods, the first is the `main(…)` method which sets up the game including the creation of all Shapes. The second method handles mouse pressed events for all Shapes while the game is waiting for a line to be started (state 1). The third method handles mouse pressed events for all Shapes while the game is waiting for a Line to be created (state 2).
- When a player clicks on the first dot, your program should save the coordinates to be used to create a Line and set up your program for state 2. When a player clicks a second dot, the saved coordinates and new coordinates should be used to create a new Line object. New Line stroke color must be set to the color of the current player.
- Player number must be tracked during game play. Consider saving the player number as an integer, with value 0 indicating player 1 and the value 1 indicating player 2. At the appropriate time in your program, you must toggle player number. One way to do this is with the expression: `player = (player + 1) % 2`. Other solutions are possible.
- Player number must be used to determine color. To get appropriate player color RGB coordinates, consider multiplying player number (0 or 1) by 255 and use the result as R, G and/or B color coordinate values.
- Remember that the same method may be used as an event handler by multiple Shape objects. Because you cannot know which Shape a player will click on each turn, assign the appropriate event handler method to all dot shapes.
- Remember that Ovals, Lines, Rectangles, Texts, etc. are all kinds of Shape objects. The first parameter of a mouse event handler method is of type Shape, and may reference any kind of shape. Fortunately, the vast majority of methods are understood by all shape types. Alternatively, you may cast a Shape using standard techniques. For example, `oval1 = (Oval)shp;` where shp is a Shape that is being cast to an Oval.
- Shape event handler methods will probably need to be reassigned after each click in the body of the event handler method to implement your game's alternating state.
- New Shape objects are always created on top of all the other existing Shape objects. Also, when a Shape object is on top of an existing Shape object, the top Shape object will prevent the underlying Shape from receiving mouse events, even when the top Shape has no event handler methods assigned. To prevent this, you may tell

the top Shape not to receive events by invoking its setEventsEnabled(…) method. For example, `shp.setEventsEnabled(false);`

- You may send a Shape object behind all others by invoking its toBack() method. For example, `shp.toBack();`

Requirements

1. Your game must implement the rules of Sim.
    a. Game play alternates between players
    b. Lines are created between user-selected dots
    c. Distinct colors are assigned to player number and used when setting new line stroke color
2. Your game must indicate the current player number and what that player must do next (choose first dot or choose second dot). This may be accomplished with Shape fill color, a Text object, or some other indicator.
3. The first dot that a player selects must be indicated in some manner. This may be accomplished using fill or stroke color of the dot, or some other indicator. Your indicator must be cleared after the Line is created.
4. Line objects created by each player must have a stroke color that corresponds to the player's color.
5. Add header comments to your program file, including: (1) your name, (2) date, (3) file name, (4) A brief general description of what your program does. Also add comments throughout your program describing what it does.
6. As part of your program's header comments, you must include the terminal command you used to compile your program (`javac …`), and the terminal command you used to run your program (`java …`).
7. Submit only your Java source code file using Canvas.
    - Log in to Canvas and click on the link for this course
    - Click on "Assignments" in the left hand menu
    - Select "Projects" and then click on Project 1
    - Click the "Submit Assignment" button and the "Choose File" button
    - Find your `.java` file and submit

This is what my project looks like in the middle of game play.