Game Zone

As you create some of the games in this section, you might find it convenient to add or remove components in a container after construction. Recall from Chapter 14 that in order for the user to see your changes, you might need to call the `validate()`, `invalidate()`, and `repaint()` methods. You will learn more about the `repaint()` method in the next chapter, *Graphics*.

1. a. Create a Mine Field game in which the user attempts to click 10 panels of a grid before hitting the "bomb." Set up a `JFrame` using `BorderLayout`, use the `NORTH` region for a congratulatory message, and use the `CENTER` region for the game. In the `CENTER` region, create a four-by-five grid using `GridLayout` and populate the grid with `JPanels`. Set the background color for all the `JPanels` to `Color.BLUE`. Randomly choose one of the panels to be the bomb; the other 19 panels are "safe." Allow the player to click on grids. If the player chooses a safe panel, turn the panel to `Color.WHITE`. If the player chooses the bomb panel, turn the panel to `Color.RED` and turn all the remaining panels white. If the user successfully chooses 10 safe panels before choosing the bomb, display a congratulatory message in the `NORTH JFrame` region. Save the game as **MineField.java**.

   b. Improve the Mine Field game by allowing the user to choose a difficulty level before beginning. Place three buttons labeled "Easy", "Intermediate", and "Difficult" in one region of the `JFrame`, and place the game grid and congratulatory message in other regions. Require the user to select a difficulty level before starting the game, and then disable the buttons. If the user chooses "Easy", the user must select only five safe panels to win the game. If the user selects "Intermediate", require 10 safe panels, as in the original game. If the user selects "Difficult", require 15 safe panels. Save the game as **MineField2.java**.

2. a. Create a game that helps new mouse users improve their hand-eye coordination. Within a `JFrame`, display an array of 48 `JPanels` in a `GridLayout` using eight rows and six columns. Randomly display an *X* on one of the panels. When the user clicks the correct panel (the one displaying the *X*), remove the *X* and display it on a different panel. After the user has successfully "hit" the correct panel 10 times, display a congratulatory message that includes the user's percentage (hits divided by clicks). Save the file as **JCatchTheMouse.java**.

   b. Review how to use the `GregorianCalendar` class from Chapter 4, and then revise the `JCatchTheMouse` game to conclude by displaying the number of seconds it took the user to click all 10 *X*s. When the application starts, create a `GregorianCalendar` object and use the `get(Calendar.SECOND)` and `get(Calendar.MINUTE)` methods with it to get the `SECOND` and `MINUTE` values at the start of the game. When the user has clicked all 10 *X*s, create a second `GregorianCalendar` object and get the `SECOND` and `MINUTE` values at the end of the game. If the user starts and ends a game during the same minute, then the playing time is simply the difference between the two `SECOND` values. Make sure your application times the game correctly even if the start and stop times do not occur during the same `MINUTE`. Save the file as **JCatchTheMouseTimed.java**.

c. In the `JCatchTheMouseTimed` game described in Game Zone exercise 2b, the timer does not work correctly if the user happens to play when the hour, day, or year changes. Visit the Java Web site to find out how to use the `GregorianCalendar` class method `getTimeInMillis()`, and then modify the game to measure playing time accurately, no matter when the user plays the game. Save the file as **JCatchTheMouseTimed2.java**.

If you were writing a professional timed game, you would test the timer's accuracy regardless of when the user decided to play. For example, if the user played over the midnight hour on New Year's Eve, you would either have to test the game then (which is impractical), or reset your system's clock to simulate New Year's Eve. If you are writing the programs in this book on a school's computer network, you might be blocked by the administrator from changing the date and time. Even if you are working on your own computer, do not attempt to change the date and time unless you understand the impact on other installed applications. For example, your operating system might assume that an installed virus-protection program is expired, or a financial program might indicate that automatically paid bills are overdue.

3. The game Corner the King is played on a checkerboard. To begin, a checker is randomly placed in the bottom row. The player can move one or two squares to the left or upward, and then the computer can move one or two squares left or up. The first to reach the upper-left corner wins. Design a game in which the computer's moves are chosen randomly. When the game ends, display a message that indicates the winner. Save the game as **CornerTheKing.java**.

4. Create a target practice game that allows the user to click moving targets and displays the number of hits in a 10-second period. Create a grid of at least 100 `JPanels`. Randomly display an *X* on five panels to indicate targets. As the user clicks each *X*, change the label to indicate a hit. When all five *X*s have been hit, randomly display a new set of five targets. Continue with as many sets as the user can hit in 10 seconds. (Use *www.oracle.com/technetwork/java/index.html* to find how to use the `GregorianCalendar` class method `getTimeInMillis()` to calculate the time change.) When the time is up, display a count of the number of targets hit. Save the file as **JTargetPractice.java**.

5. You set up the card game Concentration by placing pairs of cards face down in a grid. The player turns up two cards at a time, exposing their values. If the cards match, they are removed from the grid. If the cards do not match, they are turned back over so their values are hidden again, and the player selects two more cards to expose. Using the knowledge gained by the previously exposed cards, the player attempts to remove all the pairs of cards from play. Create a Java version of this game using a `GridLayout` that is four rows high and five columns wide. Randomly assign two of the numbers 0 through 9 to each of 20 `JPanels`, and place each of the

20 `JPanels` in a cell of the grid. Initially, show only "backs" of cards by setting each panel's background to a solid color. When the user clicks a first card, change its color and expose its value. After the user clicks a second card, change its color to the same color as the first exposed card, expose the second card's value, and keep both cards exposed until the user's mouse pointer exits the second card. If the two exposed cards are different, hide the cards again. If the two turned cards match, then "remove" the pair from play by setting their background colors to white. When the user has matched all 20 cards into 10 pairs, display a congratulatory message. Save the game as **JConcentration.java**.

6. Create a Mine Sweeper game by setting up a grid of rows and columns in which "bombs" are randomly hidden. You choose the size and difficulty of the game; for example, you might choose to create a fairly simple game by displaying a four-by-five grid that contains four bombs. If a player clicks a panel in the grid that contains a bomb, then the player loses the game. If the clicked panel is not a bomb, display a number that indicates how many adjacent panels contain a bomb. For example, if a user clicks a panel containing a 0, the user knows it is safe to click any panel above, below, beside, or diagonally adjacent to the cell, because those cells cannot possibly contain a bomb. If the player loses by clicking a bomb, display all the numeric values as well as the bomb positions. If the player succeeds in clicking all the panels except those containing bombs, the player wins and you should display a congratulatory message. Figure 15-44 shows the progression of a typical game. In the first screen, the user has clicked a panel, and the display indicates that two adjacent cells contain a bomb. In the second screen, the user has clicked a second panel, and the display indicates that three adjacent cells contain bombs. In the last screen, the user has clicked a bomb panel, and all the bomb positions are displayed. Save the game as **MineSweeper.java**.

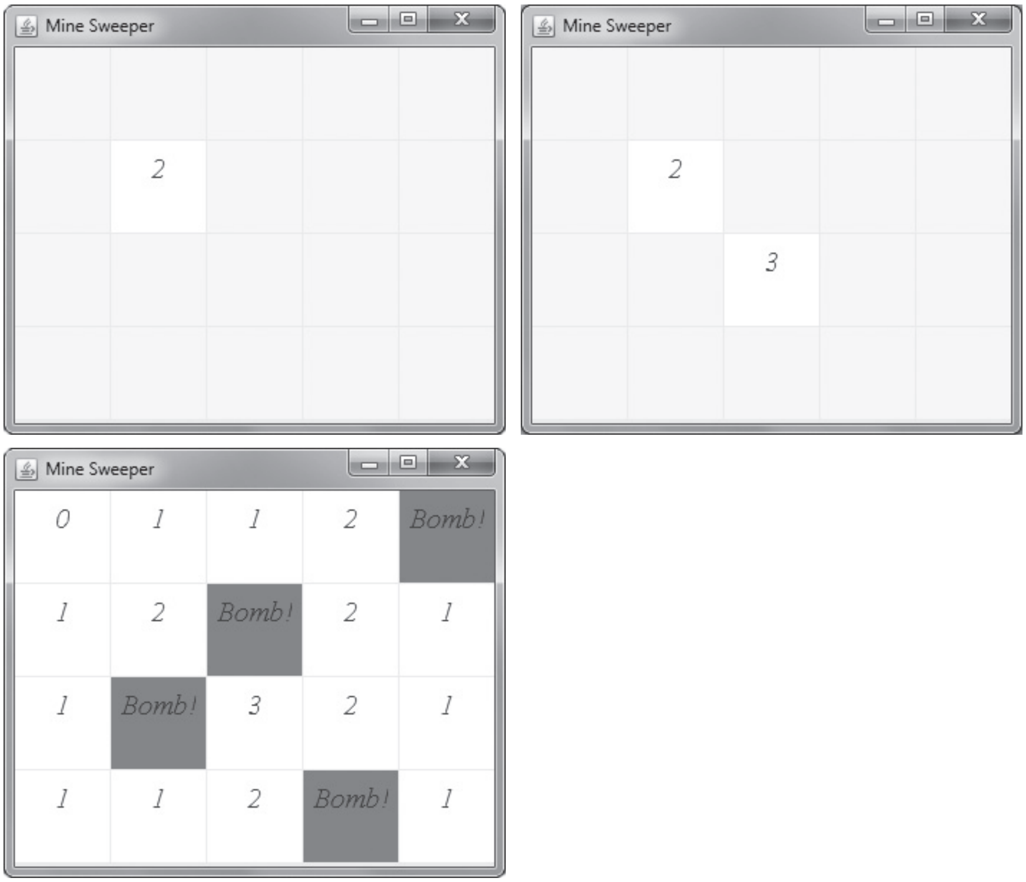

Figure 15-44 Typical progression of MineSweeper game

7. Create the game Lights Out using a `BorderLayout`. Place a five-by-five grid of panels in one region, and reserve another region for a congratulatory message. Randomly set each panel in the grid to a dark color or light color. The object of the game is to force all the panels to be dark, thus turning the "lights out." When the player clicks a panel, turn all the panels in the same row and column, including the clicked panel, to the opposite color. For example, if the user clicks the panel in the second row, third column, then darken all the light-colored panels in the second row and third column, and lighten all the dark-colored panels in that row and column. When all the panels in the grid are dark, all the lights are out, so display a congratulatory message. Save the game as **LightsOut.java**.

8. The game StopGate is played on a checkerboard with a set of dominoes; each domino is large enough to cover two checkerboard squares. One player places a domino horizontally on the checkerboard, covering any two squares. The other player then places a domino vertically to cover any other two squares. When a player has no more moves available, that player loses. Create a computerized version of the game in which the player places the horizontal pieces and the computer randomly selects a position for the vertical pieces. (Game construction will be simpler if you allow the player to select only the left square of a two-square area and assume that the domino covers that position plus the position immediately to the right.) Use a different color for the player's dominoes and the computer's dominoes. Display a message naming the winner when no more moves are possible. Figure 15-45 shows a typical game after the player (blue) and computer (black) have each made one move, and near the end of the game when the player is about to win—the player has two moves remaining, but the computer has none. Save the file as **StopGate.java**.

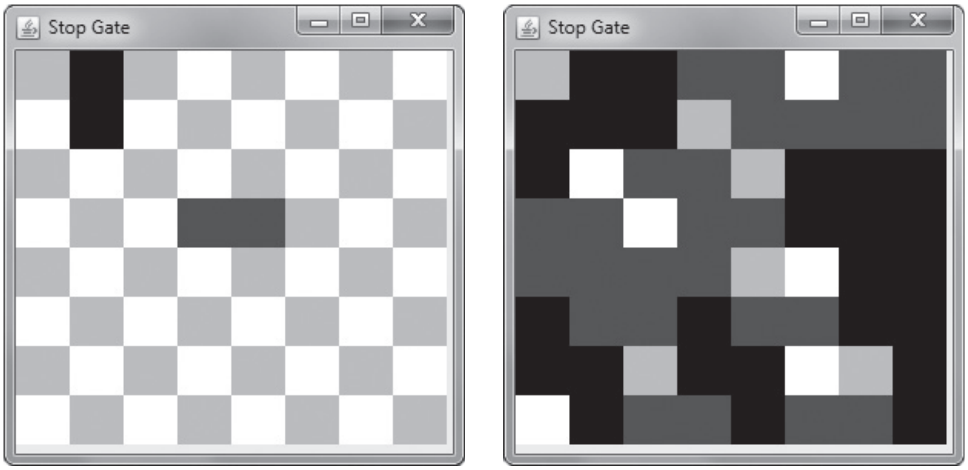

Figure 15-45 A typical game of StopGate just after play begins and near the end of the game

Case Problems

1. In Chapter 14, you created an interactive GUI application for Carly's Catering that allows the user to enter a number of guests for an event and to choose an entrée, two side dishes, and a dessert from groups of choices. Then, the application displays the cost of the event and a list of the chosen items. Now, modify the interface to include separate panels for the guest number entry, each group of menu choices, and the output. Use at least two different layout managers and at least two different colors in your application. Save the program as **JCarlysCatering.java**.

2. In Chapter 14, you created an interactive GUI application for Sammy's Seashore Rentals that allows the user to enter a rental time in hours, an equipment type, and a lesson option. Then, the application displays the cost of the rental and rental details. Now, modify the interface to include separate panels for the hour entry, each group of menu choices, and the output. Use at least two different layout managers and at least two different colors in your application. Save the program as **JSammysSeashore.java**.