Minesweeper Port to J2ME
Brian Cavanaugh
CSCI 188/297
14 December 2009

Introduction

A minesweeper game application was written using the Java 2 Micro Edition (J2ME) Software Development Kit 3.0 programming platform from SUN microsystems. The application was developed within the J2ME wireless toolkit (WTK) for immediate use on the emulators supplied therein. The application encompasses three java files: MineCanvasSquare.java, MinesweeperMidlet.java, and GridSquare.java. The remainder of this report will cover background about the game, the purpose of the aforementioned java files, how they interact, issues and resolutions as well as next steps for completeness.

Background

The game of minesweeper is a single player game in which the player must find landmines using clues from neighboring grid squares and demarcate these mines, using flags without setting the mines off. When the player 'sweeps' a spot on the grid it is revealed how many mines are within the adjacent grid squares. When the player 'flags' a spot on the grid, that spot will be marked as safe, and the player cannot 'sweep' that spot unless the flag is removed. To win the game, the player must flag all the mines without setting any of them off.

The Classes

MinesweeperMidlet.java

The MinesweeperMidlet class extends MIDlet and implements CommandListener. This class provides the basic skeleton of a J2ME MIDlet, describing the most rudimentary methods startApp, pauseApp, and destroyApp. MinesweeperMidlet also appends the basic menu to the application and through the use of the CommandListener, allows these menu items to call various Forms. These options – Exit, Help, About, Configure, New Game, and for some screens Back—are all detailed in MinesweeperMidlet. All of the choices with the exception of New Game, and Exit implement javax.microedition.lcdui.Forms and MinesweeperMidlet details the creation and manipulations of these forms. New Game calls startApp which calls mineStart in one of two ways using a try-catch statement; either by passing the number of mines selected in the Configure Form, or if the Configure Form has not been run yet, by passing the default number of ten (10) mines. mineStart then creates a new MineCanvasSquare object and this object handles the main game logic. Exit calls the Alert class.

MineCanvasSquare.java

MineCanvasSquare is the class that handles all of the game logic from creating and placing mines, to handling user keypad inputs, to drawing the game to the screen. The class extends GameCanvas and implements Runnable, and was modified from DogGameCanvas4.java written by Dan Eisenreich. Due to the fact that the number of mines can be decided by the user, it became prudent to wait to make key declarations in the constructor method.

The following analysis follows this class' methods from top to bottom. The render method adds the Graphics g to the LayerManager. The start method initializes the main (and only) thread. The stop method provides the switch to end the thread loop.

The next method is initGame which sets up the cursor sprite and the initial background tilelayer. Since each square of the initial grid is the same, the fillCells method of the TiledLayer class was a nice time saver. However, a loop over each grid square was necessary to construct a GridSquare at each x and y grid location. The next section of this paper covers GridSquare in more depth. The last bit of the initGame method establishes the location of the mines, adds the background layer to the LayerManager and sets it to visible.

The keyPressed method retrieves and decodes the key that is pressed by the player. If it is a directional key, this method simply calls the move_cursor method with the correct direction. If the FIRE or '5' key is pressed, the sweepspot method is called and if the GAME_A or '1' button is pressed, the flag method is called. It is interesting to note that before calling either the sweepspot or the flag method the mine_sprite is set to invisible so that the player can see the result of their button press. This will be discussed further in the paragraphs regarding those methods.

The move_cursor method moves the cursor one grid square in whichever direction is appropriate using a case statement. However, it does not allow the cursor to go beyond the bounds of the game grid.

The sweepspot method is called when the player presses the FIRE or '5' key. This is where most of the game logic happens. Stepping through the method sequentially, the grid square on which the cursor is when the button is pressed is checked for a flag. If that square is flagged, the button press is ignored. If the square has no flag, and contains a mine, the background on that square is changed to an exploded mine using the setCell function and gameplay is ended by calling the stop method. Since the win switch was initialized to false, and never switched to true, the lose method will be called displaying the lose screen and leaving the player to select New Game, or exit. If the square is neither flagged nor a mine, the surrounding squares are checked for mines. Depending on the number of mines in the eight (8) adjacent squares, the appropriate graphic (1-8) replaces the background tile again using the setCell function. If no mines are in the surrounding eight (8) squares, they are all opened using a quasi-recursive call to sweepspot. Initially this was done using a fully recursive call with the intent of opening a vast canyon of free squares, however this approach led to a StackOverflowError.

The flag method is called when the player presses the GAME_A or '1' key. The first thing this method does is convert display coordinates to grid coordinates. This is done in order to index the squares array of GridSquares and for ease of reference. Next, the flag method toggles the state of the current GridSquare. If the square is now flagged and was previously un-swept (this is checked so as to not allow the flagging of previously cleared squares) then the numFlags counter is incremented and the background_layer at that cell is set to the flag graphic. And if the square that is flagged also contains a mine, the count is decremented. If the flag is toggled off, the counters are incremented or decremented in the reverse manner and the graphic goes back to being an un-swept square. If the count is decremented to zero and numFlags is equal to the number of mines, the player has marked all of the mines as flagged, and therefore won! This causes the win to be made true and exits the loop with stop, so that the win screen is displayed.

The win method details the 'You Win!' screen. And the lose method details the 'You Lose!' screen. hideNotify, showNotify, and sizeChanged are all largely ignored in this program, however future versions will handle game pause events and save the games state when the game is exited or paused.

The run method contains the loop and flushes the graphics to the screen.

GridSquare.java

The GridSquare class is a data structure for storing certain attributes about one grid square. Each object created using GridSquare is kept in an array in MineCanvasSquare. The attributes stored in each object are the X and Y coordinate in grid space, and the two Boolean arguments denoting if the square contains a mine and if it is flagged. The X and Y coordinates are computed in grid space in order to make it easier to enumerate. Methods such as setMine and getMine, flag and getFlag make it immutable.

Conclusions

In conclusion, the Minesweeper application is a simple one player game with basic, limited functionality. The application has a help screen, an about screen, an exit alarm with confirmation and a configuration screen. The game reveals information about adjacent grid squares that allows the player to mark squares that are suspected mines.

Future versions of the game will allow the user to create bigger mine fields with more mines, a scrollable game board, and save the players' state of completion when paused or exited.