README for GemMatch.jar

Kathryn Tocci

CS251-003

May 2, 2016

1 How to Play

Game play for this game is very simple. The board will appear blank at the start. Once the Start button is clicked, the board will fill and the timer will start. The object of the game is to make as many swaps as possible and gain as many points as you can before the time runs out.

1.1 Swapping Gems

Gem swaps are conducted using a series of mouse clicks. The first click highlights a Gem you intend to swap with one of its horizontal or vertical neighbors. If the same gem is clicked a second time, the highlight will be cleared and the game will await a new first selection. If a gem is selected that is not adjacent, that new gem will then become the highlighted selection. Once two adjacent gems are selected they will swap, and, if a match of 3 or more is created by the swap, be removed from the board. The board will then shift, removing any newly created 3 or more matches and repeating the process until there are no more matches to remove.

NOTE: Gems do not have to create a match of 3 or more in order to be swapped.

1.2 Pausing Gameplay

This Gem Match game comes equipped with a pause button to stop the running timer. When clicked it will stop game time and remove visibilty of the gems, thus preventing cheating by planning moves. Simply click the continue button to resume gameplay.

1.3 Timer

The timer will provide 100 seconds of gameplay to create as many matches as possible to gain a high score.

NOTE: Cascades will continue until finished even if the time has reached 0.

1.4 Scoring

Each gem removed is worth 100 points with a bonus provided for strings longer than three. The score for each string removed is given by the equation:

$$stringScore = 100 \times numberOfGems + 100 \times (numberOfGems - 3)$$

There is also a multiplier for cascades. For every cascade that occurs after the initial swap you receive an incremental multiplier ($turnScore = stringScore \times multiplier$). For example, when you swap the gems, the multiplier is 1. After the board refills and looks for new matches, the score from those matches receives a multiplier of 2, the next is a multiplier of 3, and so on and so forth.

2 Program Internals

2.1 GemMatch

This class simply contains a main method that initializes the GUI and runs the game.

2.2 Gem

This class defines the `Gem` objects that make up the board. They hold three values, the x and y coordinate of its location in the board that does not change throughout the game, and an integer type that may or may not change throughout the course of the game. The number zero is reserved as marking the Gem for removal from the board.

2.3 ImageArrays

This class imports a buffered image that contains all of the images for the game. The main image is cut into arrays of sub-images, organized into categories. These are given as an alternative display to generic ovals in the GUI version of the game. Its only job is the importing and splicing of the image, which is read into the GUI through a get method.

2.4 GemManager

This class holds bookkeeping for the game as well as the algorithms for board checking, gem removal, and gem swaps. Each instance of a `GemManager` object holds:

- Integers for the number of rows and columns in the game board
- An integer for the number of different symbols that will be used in the game
- A 2-D array of `Gem` objects that represents the board
- A `Random` object to instantiate and refill the board with random gem types
- An integer array that stores how many gems will be removed per column, per turn

- A `Collection` of the `Gem` objects that will be removed each turn
- A `Collection` of `Gem` objects to check for matching strings on the following turn
- The integer score for each turn

This class performs a variety of duties. It will initialize the board with random Gem types at the start of each new game. On that initial fill it will check the entire board for matches of 3 or more, remove them and shift the gems until the game starts with a board that has no strings of 3 or more.

After two gems have been selected to be swapped, the `GemManager` goes through the checks, removals, and shifting of gems, also looking for any new matches that may have been created from those shifts, until once again the board contains no matches.

This class also has a method to ensure that any coordinate (x,y) exists within the board. This helps prevent null pointer exceptions.

Algorithm for Generating Random Gems The constructor for GemManager creates a Random object and instantiates it with a seed. For testing purposes, the seed is static, ensuring the same board every time a new game is started. For live gameplay, System.nanoTime is used as the seed, ensuring a different seed with every new game. The random number for the type is selected based on the number of symbols in play + 1, because 0 is reserved for flagging the Gem for removal. When the board is initialized, every Gem is filled with a random type. When shifting Gems, new random types are generated based on how many are necessary to fill each column.

Algorithm for Detecting Horizontal and Vertical Matches of 3+ An enum filled with the four cardinal directions was created, to represent the four directions necessary to check around the swapped gem, or any gems that have shifted location during game play. The horizontal check starts with a single point (x,y) on the board that is to be checked. It looks for matching types in one direction, then the opposite direction, storing both numbers in separate integers. If those numbers, plus the value of 1 for the initial point itself are greater than or equal to 3, indicating they need to be removed from the board. Similarly, the vertical check works in exactly the same way, only checking along the y-axis instead of the x.

Algorithm for Removing and Shifting Gems First the program finds the number of Gems to remove per column. It does this by looping over the Gems flagged for removal, setting their type to zero, and incrementing the number to be removed in the numDeleteCol array for the proper column.

Once that is finished, we start shifting. In the shiftGems method there is a local Collection of types that will hold the Integer types of the Gems that will be added to the board. It is reset before each column. Then it is filled with random gem types based on the number of Gems to be removed from that column. If there are 3 Gems flagged for removal, the Collection is filled with 3 random gem types. If there are no gems to remove, it skips to the next column. Otherwise, we create a new reference to the gem at whatever set of (x,y) coordinates we are looking at, and store its type. If the type is not 0, we add that type onto the end of our Collection, set the type of the Gem that we were looking at to whatever was first on the list, and ensure we add that gem to the list of Gems to check for the next turn.

For example. There are 3 Gems to be remomved from the bottom of a column:

The Collection of Types starts as {N,N,N} (0 = Empty Space)

Y-Value	Old Type	Collection of Types	New Type
0	X	{N,N,X}	N
1	X	{N,X,X}	N
2	X	{X,X,X}	N
3	0	{X,X}	X
4	0	{X}	X
5	0	{ }	X

Once the size of the Collection is zero, the loop breaks. This prevents having to check every gem in the column every time.

2.5 SelectPanel

This class displays the customization component of the game. It contains drop down boxes to allow the user to choose how many rows, columns, number of symbols, and type of symbols the board will display. It has get methods to return each of the highlighted values to the listener for the Start Button. If nothing is changed, it defaults to an 8x8 board with 3 types of circles. The board can be as big as a 12x12 with up to 8 types of gems.

2.6 InfoPanel

This class is nested within GemGUI and is a custom JPanel that displays all of the bookkeeping the user should be aware of, including the countdown timer and score board. This is the panel where the Start and Pause/Continue buttons are located. It is in this panel that the player customizes the look of their game. This is also where the Display Message pops up after the timer reaches zero, letting the user know that the game is over.

2.7 GemGUI

This class creates the GUI for a Gem Matching game, interacts with user clicks, "animates" the removal and dropping of gems, keeps track of the score and a countdown timer. It lets the player know when the time is up by stopping game play and displaying a "Game Over" dialog.

The basic interface includes a Start Button that doesn't create and paint a board until clicked. A Pause/Continue button that stops the timer and clears visibility of the board while retaining all of the information therein before the game was paused. It also has a countdown timer and a running score board.

The majority of the GUI consists of the board where the actual game play takes place. It is fully clickable and interactive. After a swap that creates a match of 3 or more, it has a simple animation that first makes the matching gems "disappear" and then reprints the board with all of the proper gems shifted down and new gems in any space that may have been made empty by those shifts.

2.8 GameBoard

This class is nested within GemGUI and is a custom JPanel that paints and displays the game board, while responding to mouse clicks from the player. This is where the bulk of the animation and board updates take place. The only exceptions are the animations initiated by the buttons found in the InfoPanel class.

Algorithm for Selecting Gems Via Clicks This is the heart of the user interacting with the GameBoard since clicking and swapping Gems is the only way users can create matches of 3 or more. Its implementation is fairly simple. A player clicks in the board, and an x and y coordinate are generated from that mouse click. From those coordinates, the x and y of the gem affected are deduced by merely dividing both numbers by the size of rectangle that represents each gem area. Because we are only working with integers, it conveniently concatenates to the appropriate number. If the new x and y exist within the parameters of the board, a black rectangle is drawn around that gem.

The second click is read much the same way as the first, but in this case we check first to know if it was placed within the area where gems are populated. If not, it is as if we never made the second click.

Then the program checks if the x and y coordinates are the same on the second click as the first, the values reset to what they were before the first click, repainting the board without the highlight.

If the second click isn't on the same Gem, the program then checks to see if the square clicked is adjacent to the first. This is done by ensuring the absolute value of the difference between the x or y values is equal to 1. If this is the case, the stage flag for the timer animations is initiated and the timer starts.

If the second click isn't on the same Gem, and isn't on an adjacent Gem, the program assumes it is on a non-adjacent gems and moves treats the Gem selected as the new first click, deselecting the first and highlighting the new and waiting for a new second click.

Algorithm for Animating Cascades of Matched Gems When the timer is activated and the stage flag is set to 1, the program initiates the Gem swapping and cascading sequences. For each tick of the timer, the program goes into the proper if statement based on what integer the stage flag is set to.

Stage 1 is initiated after a valid swap has been selected by the player. The gem types are swapped between the two and the board is repainted, showing the gems in their new positions, and the stage flag is incremented to 2.

Stage 2 is where the board is checked and types for those gems flagged for removal are set to zero. If the number of gems to delete is ¿ 0, every gem flagged for removal has its type set to zero, which is set to match the background color, thus when the board is repainted it looks as if it has been erased from the board. Then the stage flag is set to 3. This repainting only happens if there are gems to remove, if there are none, the timer stops and the stage flag is set to zero.

Stage 3 is where the Gems are shifted according to the Gem Shift Algorithm. It is only after all the gems are shifted and the board is repopulated that the board is repainted. During this shift, every gem that moved is flagged to be checked again. This is why the stage is then set to 2, repeating the flashing removal of gems with the accompanying repainting of a new fully filled board until there are no more matching chains remaining.

3 Extras

3.1 Pause/Continue Function

I thought it was important to have a working pause button for my game since my end of game play is purely time based. But I also wanted the board to not be visible if the game is in pause mode, to curtail any cheating by the player. This was achieved with a simple boolean flag around the insides of my paintComponent method. I also had to ensure that the game board area was not clickable while in paused mode.

3.2 Imported Images

This game now has actual images to use instead of drawn circles. I put all of the images onto a single JPEG file, and spliced the sub-images accordingly. For ease, each grouping that goes together is in a single row, with the first index that is fed into an array reserved as a "blank" space for my gem removal animation. This sheet located inside the .jar file.

3.3 Board Customization

After getting the imported images to work, I thought it would be fun if the player could customize the game. I used JComboBox objects to allow the player to choose the number of rows, columns, how many different symbols will be used, and what the symbols themselves would look like. This showcases a little extra versatility and also demonstrates the flexibility of my GemManager class by having it handle any of these variations with ease.

4 Bugs & Future Features

4.1 Bugs

As of this writing, there are no known bugs in the game.

4.2 Future Features

- Image sets for countdown timer and score board
- Background Music

- Audio file of Strong Bad (of Homestar Runner fame) stating ”It’s Over!” when the time reaches zero and the end of game dialog pops up.
- Scalability, I would have loved to have the window resize property so there wasn’t the extra white space when the board is smaller than the max size.