CS 251 Intermediate Programming
Gem Match: Part 2 – Complete Gem Match Game

Brooke Chenoweth

Spring 2025

Goals

To carry forward with the gem matching game we have been working on, we are going to make a final push toward the end goal. You will notice that the due date is nearly the end of the term, but this does not mean that you can sit around and relax until the last day. This project will take quite a bit of programming to get done. I strongly recommend talking to course assistants, tutors, and myself, to hash out any problems that you might have, and to do so early on.

The overall goals of this project are:

- Create a complete working gem match game
- Write a larger program that actually does something
- Put the knowledge of many small individual pieces together into a larger system
- Find out how inheritance can really help you, and why it's so neat
- Get some experience writing a GUI program that utilizes mouse and keyboard input
- Master the process of breaking a large problem down into smaller pieces. You do *not* want to write this entire program in a single main method.

Program Description

I've described the basic gem matching game, but I encourage you to be creative with the game design. Go ahead and use candy or charms instead of gems, just make sure the basic game mechanic stays the same.

The game area should have around 8-12 rows/columns and 4-6 gem types. (Your choice for the exact values. Pick something that makes the game fun.)

Selecting and Swapping Gems

When no gem is selected and the user clicks on a gem, there should be some visual indication that the gem is now selected. In my example program, I just drew a yellow rectangle outlining the selected gem.

When a gem is currently selected and a clicks on...

- the currently selected gem, deselect the gem.
- a gem that is adjacent to the currently selected gem, swap the two and check for any three in a row matches formed as a result. No gem is selected as the matches are removed.
- a gem that is not adjacent to the currently selected gem, deselect the current gem and select the new gem.

You must have at least this mouse click method of selected gems, but if you want you can add additional controls. (Click and drag, perhaps? Arrow keys and spacebar?) If you do, be sure to document it in your readme file.

Gem Removal Animation Timer

I want to be able to see the steps of the cascade of removals and drops, so you'll need to make use of a Timer to "animate" this. At the minimum, I want to see the removed gems vanish for a tick before the gems above them appear in their new dropped positions. Then the dropped gems should be in their new positions for a tick before any new matches disappear, and so on until the cascade ends.

If you want to get fancier and have smoothly falling gems, explosion animations, etc., feel free to go above and beyond. What I don't want is to swap two gems and suddenly have the final result of the cascade appear without showing any of the intermediate steps.

If you chose to make a timed version of the game where you try to get as high a score as possible in a minute or so, you will also want to use a Timer for that. (It doesn't necessarily have to be the same Timer object as the one for the gem animation, though it could be depending on how you structure your program.)

GUI Layout

You have a lot of freedom in designing this GUI, but I require at least the following elements.

- A panel where the "gameboard" is drawn.
- Labels for current score information.
- A start button that begins a new game.

Game Play and Scoring

- Game begins when user presses the start button.
- The game starts with a board full of gems without any matches and no gem selected.
- Score increases as gems are removed. (More points for larger matches and longer cascades, perhaps?) Feel free to display additional statistics. (Length of current/longest cascade, largest group removed, level up after some number of matches, etc.)
- When the game is over, somehow let the user know. In a timed game, this would be when the time runs out. In free-form game, it could be when after the user removes a certain number of gems or scores a certain number of points.

Extras

Adding extra features can make your game more fun. Just make sure you have the basic game functionality first. Here are a few ideas to get you started.

- Custom background image instead of plain color.
- Sound effects and/or background music.
- Fancy game over notification.
- Increase difficulty as game progresses. Add more colors? Require ever more points to get to the next level?
- Add special types of gems with additional powers. For example, a colored bomb gem could remove all gems of the given color on the entire board.
- Gracefully handling resizing the window, so you can rescale the game while maintaining the aspect ratio.
- Save high scores to a file.
- Hints! Show a potential swap to the user if they need a hint.
- Easter eggs. (Really important to document those if you want us to find them!)

Suggestions and Hints

Rather than telling you exactly what to do, I will provide a number of hints that you will hopefully benefit from.

- Split things up into smaller pieces! My sample solution that you have seen in class, consists of roughly 600 lines of code. Among these lines of code I have at least 10 classes defined. Some are nested, some are anonymous, and some are higher level classes. Again, what I'm trying to say, this is not a problem that you can just write in a single method.

- Build off of previous code. Obviously, I expect you to use the GemManager you wrote for part one, possibly adding additional functionality as you need it. You should be able to use a lot of the work you did for the GUI layout practice to at least get you going on the GUI for the game.

- How to approach the problem... One of the harder things to do when implementing a game like this is to figure out where to start, and what to do first. The first thing you need to understand, is that the game isn't going to write itself, and it's not going to be completely done the first time you sit down to write code for it. So the trick is to work on pieces that you can finish and test, individually first, then putting them together into a usable system. For example – when writing the gem matching game, start by just drawing the gems to make sure they will show up, and then once that is working, figure out how to click on them and make swaps.

- Then what... Well, you hopefully know what you want your game to do, how it will work, and what is going to happen as results of something that you do. How many points should be added for a removed group of gems, etc... I encourage you to sit down and think out your own set of rules and policies for the game. I do not want to impose any specific standard in terms of this for your implementation. But, what I'm saying is that if you have a good idea of what you want your program to do, it's easier coming up with the design for that program on your own. I encourage you to come talk to your CA or to me, about your design before you start writing a lot of code. I also encourage you to talk about design decisions on the discussion board for the program. Sometimes, it helps venting ideas. Feel free to brainstorm high level ideas (not sharing code!) with your classmates, both in person and on the Learn discussion board.

  If you come up with a design that you think is reasonable, you will likely do well on this assignment as well – that being said, *pleeease* don't hesitate to ask for help, and to pose questions in class. It will benefit everyone.

- Soooo... What kind of stuff do we need in order for this game to work? Partially it's up to you, but I can list a few things that I used and that you may feel are useful to you as well.

  - Instance variables - I have quite a few in order to keep track of the state of the game. Examples are the gem manager, scores, currently selected gem, and such. These typically need to be initialized at the beginning of a game.
  - Private helper methods - I have lots of them, these are methods that do small tasks that you may be performing often, but you don't want to write the code for them over and over again. If you find yourself copying and pasting a lot of code, you should probably be thinking – "Hmmm, I should probably make a method for that!", and then figure out what the method is going to look like, and what parameters it needs, etc.
  - Timer. I use a timer to keep track of how fast the game progresses, but... not every object responds to every tick of the timer. You'll have to decide what class(es) should pay attention to the Timer event.

– Mouse Listener – Probably one of the more important things for this game, since that is how the user selects the gems. Remember that methods called from listeners should usually not be computation heavy as it may slow down the response time to the next event.

Turning in your assignment

For this project, I want all the code and resources to be packaged into a self-contained jar file. I also expect you to turn in a readme document describing your project. These are the only two files you will submit.

Jar File

Create a jar file with all the necessary files that you used for your assignment. The jar file must of course include your source files, as well as code from all packages that you used. I.e., the jar should be self-contained and you should be able to run the game completely from the jar.

To make a jar file with a entry point of the `GemMatch` class:

1. Compile all your classes. (`javac *.java` will compile them if all your source files are in the current directory.)

2. Use the `jar` command to create a jar with all your files. (This should include both your `.java` source files and your compiled `.class` files.) Use the `e` option to specify the entry point.
   `jar cvfe JarFileName.jar EntryPointName <List of files and directorys to include>`

   So, if all your source files and class files are in the current directory, you can use:
   `jar cvfe GemMatch.jar GemMatch *`

   If you are using any images, sounds, or other files like that, make sure you include them in your jar. You want the jar to contain all the files your program needs to run in the single jar.

3. Make sure your program runs from the jar file. Use the `-jar` option with java.
   `java -jar GemMatch.jar`

   To properly test this, you should move your jar file to a new location and try running it there to make sure you are not accidentally running from the files you used to make it instead of the jar itself.

README file

You have enough freedom with this project that we'll need some documentation. Submit a readme file that explains how to use your program and any special features we should be aware of.

At the very least, your readme should include:

- Game play
  - How to play the game.
  - How is the game scored.
- Description of program internals
  - Description of classes. (Where are game logic, data structures, etc.?)
  - Algorithm details, such as:
    * Detecting the horizontal and vertical lines of same color gems.
    * Removing gems and dropping gems to fill in.
    * Generating random gems
    * Managing and animating the cascades of gem matches.
    * Detecting end of game
- Any extras. It is especially important to point out the clever things you do so the grader will know to look for them while testing your program.
- Known bugs and feature requests – I know that no matter how long you have work on this assignment, there will be some bug that you can't quite fix or some feature that you won't quite have time to implement. Tell us about them. What would be your next step?

Submit to Canvas

Submit both your jar file and readme document to Canvas. Make sure that your jar file includes your source code!