CSIS10B Ideas for Final Projects

I recommend checking a few of these projects out before coming up with your own project idea. For one thing, these projects all have a code base already established, meaning you can get to an interesting and educational experience much sooner than starting your own thing from scratch. Once you decide on a project let me know and I'll gather up the source files for you to begin.

The projects from Bailey's book are more text based than graphical. If you want to develop animated projects, Hoot's manual provides several interesting possibilities.

Nifty Projects:

1) Evil Hangman using 120,000 word scrabble dictionary
   Students write a program in which the computer cheats at Hangman by actively dodging players' guesses. Associative arrays, string processing, file I/O.
2) Traveling Salesman Problem using Genetic Algorithm start-code
3) Simpler Traveling Salesman Algorithm (using linked lists)
4) Avogadro's Number Calculation based on Image Analysis
   Recursion, depth-first search, image processing, nearest neighbor, data analysis, science.
5) Guitar Heroine synthesize guitar sounds using Karplus-Strong algorithm (demo). Checklist/hints
   Relies on the standard libraries from Introduction to Programming in Java for real-time audio and GUI interaction

Projects Involving Dictionaries:

1) Discovering the authorship of certain famous pieces of literature is an interesting problem. Comparisons are made between pieces whose authorship is disputed and those of known authorship. One approach is to compare the frequency of pairs of letters. There are 26 x 26 different pairs of letters. Not all of them will appear in a piece of writing. For example, "qz" is unlikely to appear, while "th" is likely to appear often. Design a program, similar to the frequency counter of Segments 19.12 through 19.17, that counts all the pairs of letters that appear in a given piece of text, and creates a 26x26 "thumbprint" of the letter frequencies. This could be a simple graphical applet like we made in CSIS10A. Pairs that appear more often could be colored a dark shade of gray, while those that don't appear at all could be colored white. Pick two authors from the same era (such as Herman Melville and Nathaniel Hawthorne) and create "thumbprints" for their work (much of which is available freely online). Then choose a piece of text that wasn't used in the making of the thumbprints and classify it by determining which author has a closer match.

2) Write a program that plays the game tic-tac-toe. Represent the game board by an array of nine values. Each location in the array contains either an X, an O, or a blank. The total number of possible board configurations is 3^9, or approximately 20,000. Associated with every possible configuration is a best move.
   Generate all possible board configurations, and let them be search keys in a dictionary. For each search key, let the next best move be its associated value. Once you have created the dictionary, use it to decide the moves for a computer-based player in a game of tic-tac-toe.

3) Create a program that would form the core of a typing assistant. This program would create a word frequency dictionary based on samples from a user's emails or other writing. Then open a console and let the user type a few letters. Whenever the "enter" key is pressed, the letters already typed are used to search the dictionary for possible matches (words that begin with the same few letters). The top ten possible matches are presented (in decreasing word frequency order) in a menu allowing the user to select the desired word by typing 0 thru 9. Once this works, for an extra (optional) challenge develop an application that does this using a GUI, which continually updates the top ten list as letters are typed and inserts the remaining letters of the word after the user selects the desired number.

From Bailey's Java Structures

10.5 Laboratory: A Stack-Based Language Page 247 (challenging) Download

This project builds an extended language processor that includes symbols, functions and if statements

10.6 Laboratory: The Web Crawler Page 251 (medium) Download

This project creates a working web crawler that determines the distance between two pages on a website. Uses queue. NOTE—the HTML class in this project does not support modern web page parsing, so you can only apply it to simple sites like http://tomrebold.com. A fun application is to search for dead links on http://tomrebold.com.

12.11 Laboratory: Playing Gardner's Hex-a-Pawn, Page 313 (hard) Download

This project uses up a game tree to allow choosing the best response at any point in this simplified computer chess game.

13.7 Laboratory: Simulating Business, Page 341 (medium)

This is a more elaborate queue based simulation than the StoreSim exercise we did. The simulation sets up multiple tellers, and determines whether it's best to have a single line served by multiple tellers, or to have multiple lines. Customers have random arrival times and service times, making the coding much more complex.

15.8 Laboratory: The Soundex Name Lookup System, Page 401 (medium) Data Source File

This project involves creating an algorithm for classifying names by their phonetics and creates a database of surnames using either a SortedArrayDictionary (from Lab10) or a sjava.util.TreeMap that allows geneologists to retrieve alternate spellings for names that sound very similar to each other. The key would be the soundex code of a name, and the value would be a linked list of surnames that match the soundex value.

A nice app would be to read in the top 15000 surnames in the U.S., and store them in the manner just described. Then ask the user to enter a surname, compute the soundex (key), and show the neighboring soundex entries with all the names within 5 keys before and after the entered surname's soundex key.

16.6 Laboratory: Converting Between Units, Page 439 (medium)

Uses a graph to handle unit conversions that may require multiple steps to complete. Conversions trace pathways on a graph of unit conversions.

From Hoot, Lab Manual to accompany "Data Structures and Algorithms in Java"

Lab 11 Dictionary Client, Page 165

Uses a dictionary as a spell checker, with the animation engine described in appendix. (see below)

Lab13 Maze Recursion, Page 203 (easy if no animation, harder with animation, Page209)

Uses recursion to "backtrack" during maze searching. Solves a maze.

Lab14 Bank Line Simulation, Page 211

Uses a queue to base a more realistic line simulation. Uses the animation framework to show the queue(s) as it evolves in time

Lab15 Tree Client, Page 223 Initial Files

Uses a tree to create a Huffman Code, which is used to compress data files based on mapping binary sequences to letters according to their frequency. Uses the animation framework to illustrate how the tree is accessed during compressing or uncompressing.

Appendix A: Animation Framework, Page 265 Source Files

This package is used to animate certain lab solutions. You can use if for the maze solver and other projects to watch the program solutions evolve in real time.

The final project is due on the last day of class. You are invited to present your work to the class so it pays to finish early.

The final project is worth 10% of your grade.