Final Project Report

SpeciesChecker 1.0

by Cory McKay

I. INTRODUCTION

1.1 Overview

The goal of this project was to produce a piece of software that enables users to enter in simple compositions and have them automatically tested for technical errors. This is somewhat analogous to a word processor's spellchecker or a grammar checker. The software is able to analyze compositions for potential problems and generate output for the user indicating what types of errors were detected and where.

1.2 Scope

The types of compositions that can be analyzed are cantus firmi and first through fifth species two-part tonal counterpoint.

1.3 Intended Users

The software is meant for students in first year composition or music theory courses (such as MUSC*1250 at the University of Guelph), or anyone leaning the rudiments of counterpoint. The software was developed specifically for users writing in the framework of species counterpoint, so it is of limited utility to people wishing to writing more sophistical genres of music, although they could enter in fragments of compositions to have them checked.

The software is intended to be used primarily in an educational context. Music students are able to enter their work into the software and have it detect potential problems for them. This gives them immediate feedback on their work and helps them learn from their mistakes. It also helps them by acting as a final proofreading tool to avoid handing in work that contains careless errors. The software is also able to randomly generate cantus firmi which students can use as a base for writing counterpoint exercises.

II. FEATURES

2.1 Error Detection

The software detects technical errors in cantus firmi and tonal two-part species counterpoint compositions. A message is generated for each error detected, indicating the rule that was violated and, if appropriate, where the error occurred. The user can analyze the cantus or counterpoint individually, or can analyze both together, as desired.

2.2 Preferences

The software allows users to disable the application of certain rules. This is to accommodate users who wish to use different sets of rules. This is useful because there are several different theoretical perspectives on some issues in the literature. The preferences are saved to disk so that they will persist when the program is closed.

2.3 Random Music Generation

The software can randomly generate a cantus firmus that conforms to the set of rules that are selected in the preferences. This enables users to generate cantus firmi for use in writing counterpoint exercises.

2.4 Graphical User Interface

The software has a menu and table-based interface that allows the user to enter and edit compositions using a mouse and/or keyboard shortcuts. The interface uses combo boxes to prevent the user from being able to enter inappropriate information. The user is able to add or delete a counterpoint from a given cantus firmi so that the same cantus can be used in multiple compositions without needing to be reentered.

2.5 Read From File / Save To File Functionality

A specially designed file format is used to allow users to save their work so that they can review or edit it later.

2.6 On-line Help

There on-line help that the user can access to gain general information about the software, instructions on how to use the interface and explanations of the specific rules which are used in analyses.

III. OVERVIEW OF INTERFACE

3.1 Editing Compositions

The main window of the interface can be seen in Figure 1:

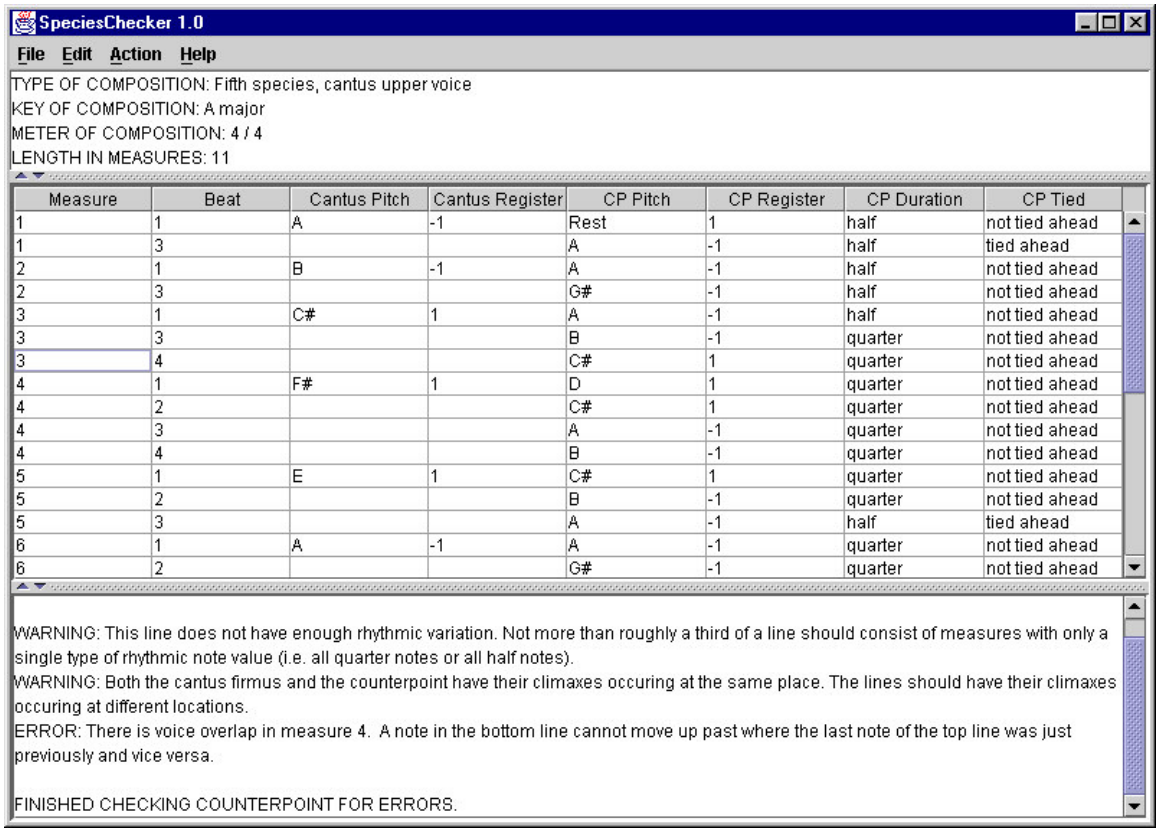

Figure 1: The main window of the interface

From top to bottom, the components of the interface are as follows:

- **Menu Bar:** used to cause the software to perform actions
- **Composition Settings:** displays the overall settings of the current composition
- **Notes Table:** allows the user to view and modify the notes in a composition
- **Analysis Results:** displays the results of the analysis of a composition as well as various error messages

The user can enter or modify compositions by clicking on the appropriate cell of the table. If the cell is modifiable (as are all non-empty cells not in the first two columns), a combo box will appear when the cell is clicked on, from which the user can select the new value.

Each row of the table corresponds to a rhythmic location in a composition. The cells in the first row of the table in Figure 1, for example, correspond to information about notes struck in the first beat of the first measure of the composition. The columns are as follows:

- **Measure:** the measure where notes in the given row are struck
- **Beat:** the beat where notes in the given row are struck
- **Cantus Pitch:** the pitch of the note belonging to the cantus firmus that is struck on the given beat of the given measure. It is also possible to place a rest at this location by choosing "Rest" from the combo box rather than a pitch.
- **Cantus Register:** the register of the pitch specified in the corresponding Cantus Pitch cell. For example, a C in the Cantus Pitch cell and a 1 in the Cantus Register cell refers to middle C. A value of 2 in the Cantus Register cell would correspond to the C an octave above middle C, and a value of –1 in the Cantus Register cell would correspond to the C an octave below middle C. Values of B and -1 in the Cantus Pitch and Cantus Register cells would correspond to the B a semitone below middle C, and a D and a 1 in these cells would correspond to the D a whole tone above middle C.
- **CP Pitch:** the pitch of the note belonging to the counterpoint that is struck on the given beat of the given measure. It is also possible to place a rest at this location by choosing "Rest" from the combo box rather than a pitch.
- **CP Register:** the register of the pitch specified in the corresponding CP Pitch cell. Functions similarly to the Cantus Register column.
- **CP Duration:** the rhythmic duration of the note belonging to the counterpoint that is struck on the given beat of the given measure
- **CP Tied:** specifies whether or not the note belonging to the counterpoint that is struck on the given beat of the given measure is tied ahead to the next note. It is not necessary to indicate whether or not a note is tied back, as this is determined automatically.

The table prevents the user from being able to make certain mistakes. No rhythmic duration is specified for notes belonging to the cantus firmus, for example, because these notes must always be whole notes. For similar reasons, it is not possible to tie notes belonging to the cantus firmus.

An example of a fifth species composition is shown in Figure 2:

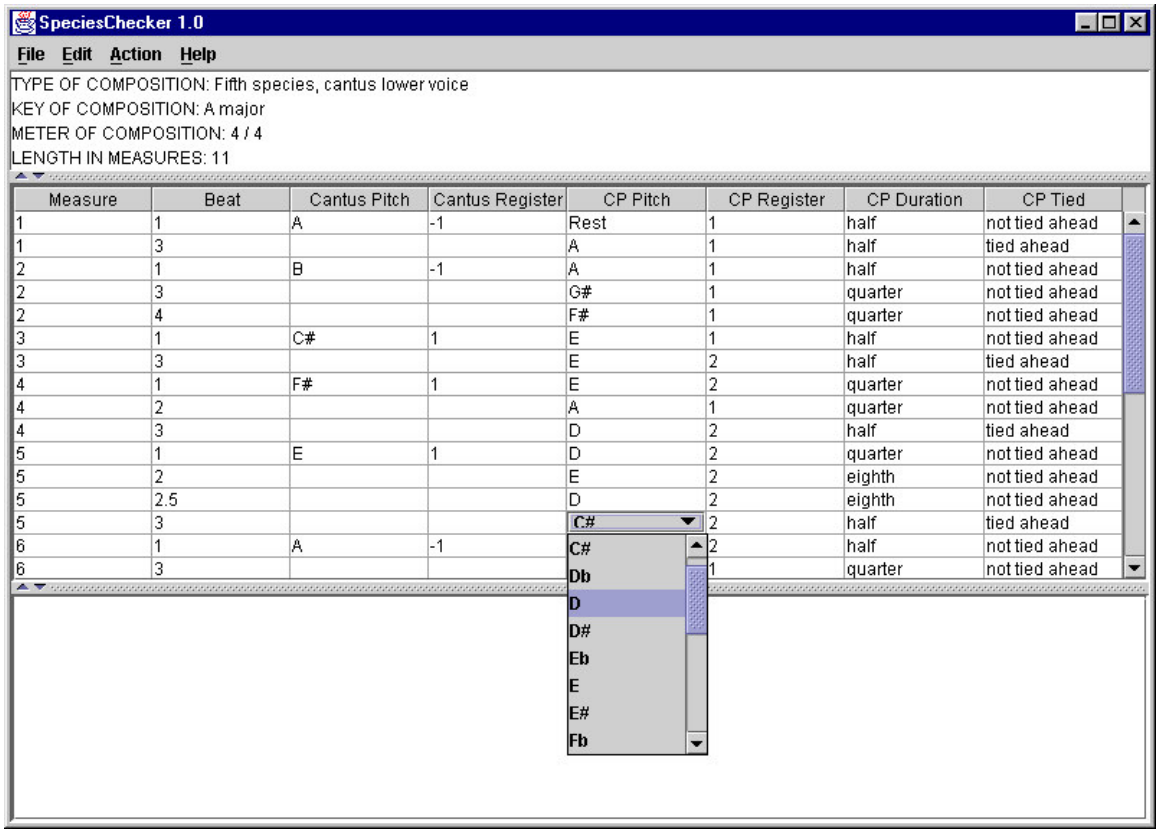

Figure 2: Sample composition

The first row of the table, for example, shows that the first beat of the piece has an A (just below middle C) struck in the cantus firmus, while there is a half rest in the counterpoint. The third beat of the first measure has an A (just above middle C) struck as a half note that is tied over to another A on the first beat of the second measure. It can be seen that the user is in the process of altering the pitch of the note in the counterpoint on the third beat of the fifth bar from a C# to a D.

As a final example, Figure 3 and Figure 4 show a sample second species composition, first in standard musical notation and then as displayed by the software:

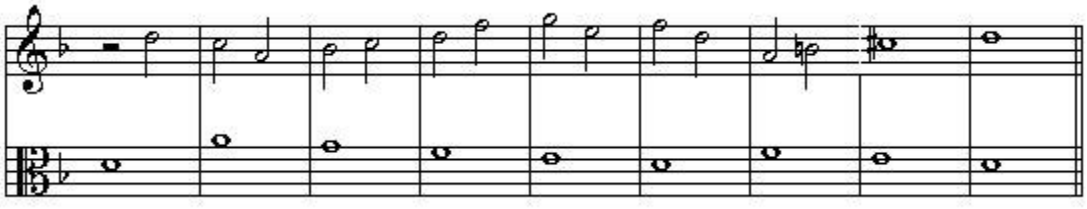

Figure 3: Sample second species composition

SpeciesChecker 1.0

File Edit Action Help

TYPE OF COMPOSITION: Second species, cantus lower voice
KEY OF COMPOSITION: D minor
METER OF COMPOSITION: 4 / 4
LENGTH IN MEASURES: 9

Measure	Beat	Cantus Pitch	Cantus Register	CP Pitch	CP Register	CP Duration	CP Tied
1	1	D	1	Rest	1	half	not tied ahead
1	3			D	2	half	not tied ahead
2	1	A	1	C	2	half	not tied ahead
2	3			A	1	half	not tied ahead
3	1	G	1	Bb	1	half	not tied ahead
3	3			C	2	half	not tied ahead
4	1	F	1	D	2	half	not tied ahead
4	3			F	2	half	not tied ahead
5	1	E	1	G	2	half	not tied ahead
5	3			E	2	half	not tied ahead
6	1	D	1	F	2	half	not tied ahead
6	3			D	2	half	not tied ahead
7	1	F	1	A	1	half	not tied ahead
7	3			B	1	half	not tied ahead
8	1	E	1	C#	2	whole	not tied ahead
9	1	D	1	D	2	whole	not tied ahead

Figure 4: Sample second species composition

3.2 Checking a Composition for Errors

The user can check a composition for errors by selecting the *Analyze Composition* command from the *Action* menu. This generates a list of problems detected in the Analysis Results section of the main window. This list is composed of two sections: The first section holds the results of an analysis of the cantus firmus itself, independently of the counterpoint. The second list consists of errors detected in the counterpoint line itself and in how the cantus firmus and the counterpoint interrelate.

If the user only wishes to analyze the cantus firmus, or if the composition does not contain a counterpoint, then the user can select the *Analyze Cantus* command from the *Action* menu. The user can also concentrate the analysis on the counterpoint by choosing

the *Analyze Counterpoint* command from the *Action* menu. This will cause the software to look for errors in the counterpoint line and how it relates to the cantus firmus, and will omit the independent analysis of the cantus.

The problems that are detected are given one of two labels: "errors" or "warnings." Warnings indicate problems that may be relatively minor, and errors indicate problems that are more likely to be significant.

3.3 Setting the Preferences

The user may wish to disable some of the rules that the software applies while looking for problems. This can be done by selecting the *Preferences* command from the *Edit* menu. The user is then presented with the dialog box shown in Figure 5.

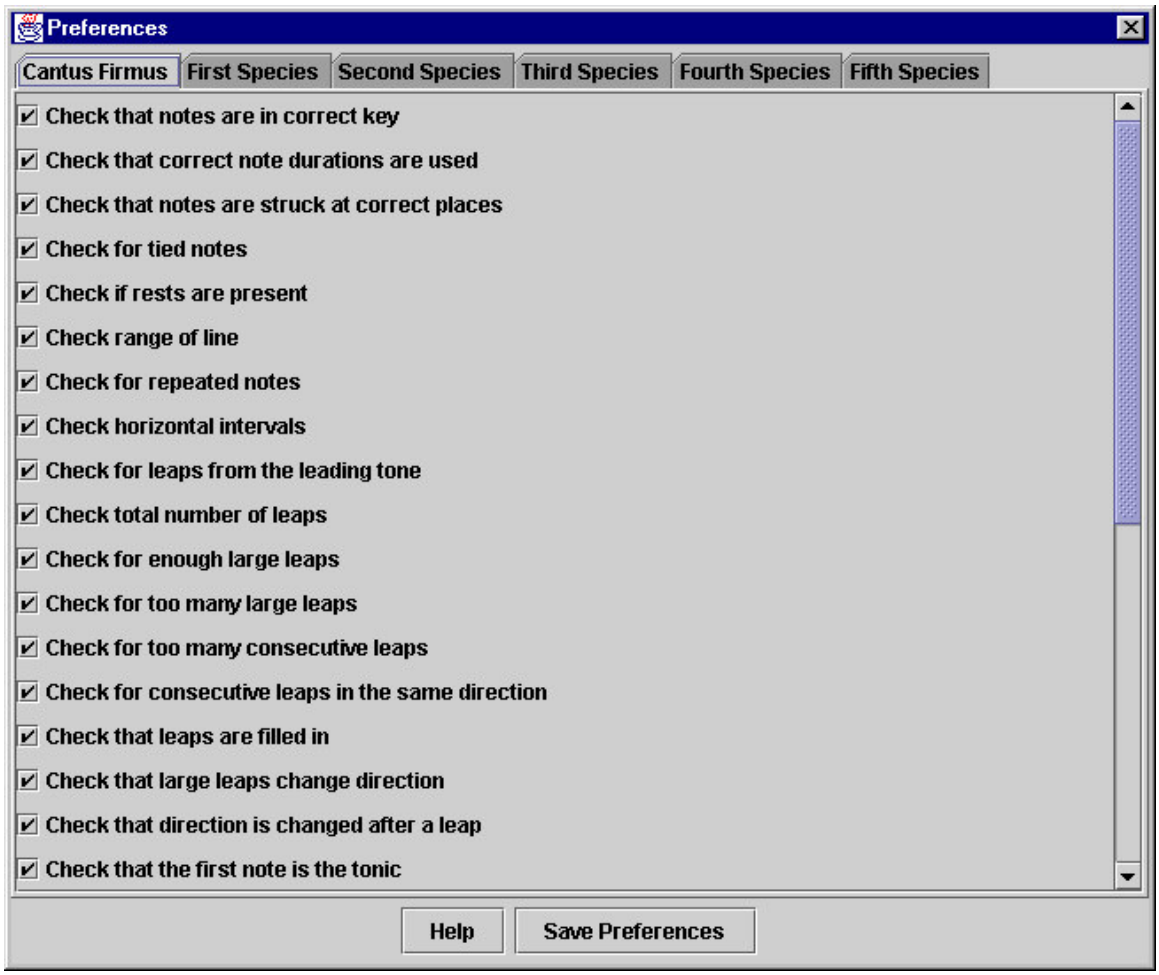

Figure 5: Preferences dialog box

The user may then choose to enable or disable any of the rules by clicking on the check box corresponding to each rule. The tabs at the top of the dialog box allow the user to

view the rules for each of the different types of compositions. Explanations of these rules are found in section 5 of the User Manual.

The software comes pre-loaded with the preferences configured to apply the rules most commonly found in the literature. Pressing on the *Save Preferences* button will save the currently selected preferences to disk, so that the user will not have to reset the preferences the next time that the software is run.

3.4 More Information

Section 4 of the User's Manual contains the information just presented, along with more details.

IV. TASKS ACCOMPLISHED

4.1 Research

The first stage of the project was to research the various perspectives and rules of counterpoint, voice leading and simple harmony. Computer music publications were also consulted to investigate approaches that may have been taken in related work and the types of data structures that could be used to represent music. Some reading was also done on context-free grammars, to investigate their applicability to analyzing shifts between species in fifth species compositions and the harmonic analysis of four-part compositions, the latter of which was initially being considered as one of the features of the software.

Various commercially available notation programs were also investigated for ideas in designing an effective user interface. Research was also done on Swing, as I had never written a GUI in Java before.

A bibliography of the sources consulted is presented in Section VII.

4.2 Formalization of Rules

Once the rules of counterpoint had been researched, it was necessary to compile them, remove redundancies, distinguish between absolute rules and guidelines, resolve differing perspectives in the literature and formalize the rules so that they could be implemented in the software as conditions that are either met or not met.

4.3 Software Design

It was necessary to design data structures that could be used to store and analyze compositions. A file format was also needed to represent these data structures as saved files. Techniques were developed to apply the rules to these data structures. It was also necessary to develop a way of using these rules to randomly generate cantus firmi. Finally, an infrastructure was needed to allow the GUI, data structures and rules to communicate and interact with each other.

All of this design was done keeping in mind basic software design principles, so that the software would be easy to test and potentially upgrade in the future. The data structures were thus made much more general and powerful than was necessary for analyzing species compositions. The design of the analysis portions of the software is also easily adaptable to more sophisticated types of compositions, such as four-part harmonies. A modular design was used throughout the project so that integration and addition of features would be simple. More information on the specifics of the design is available in the Mid-Project Report.

4.4 Graphical User Interface Design

A GUI was needed that would be easy to use, easy to implement (due to time constraints) and would prevent the user from making types of errors that would compromise the analysis portion of the software. A table-based interface was arrived at as the best compromise between these competing interests. This interface is well-suited to species counterpoint, but it is not easily adaptable to four-voice compositions, as the increased number of voices would require a large and unwieldy table.

4.5 Implementation

The implementation process was fairly straight-forward. The design worked well, with only a few minor modifications being needed. This part of the project was very time consuming, however, partly due to my inexperience with implementing GUIs in Java and partly due to the large number of rules that needed to be implemented. The time required for implementation and testing was the main factor in the decision to limit the project to species counterpoint rather than dealing with four-part harmonies as well.

4.6 Testing

The data structures, file saving and GUI portions of the software were thoroughly tested before the actual analysis portion of the software was integrated with them. The rules were tested one by one with various compositions as they were implemented, for both fail and pass conditions. Once all of the rules for a given species had been completed, numerous flawed examples and model compositions from the literature were used to test the rules as a whole and ensure that the software produced results that were in agreement with the literature. Sample corrected compositions were also collected from music students and analyzed to make sure that the results of the analysis were similar to the corrections. It was hoped that students would be able to use the software themselves and provide feedback, but the late date of the implementation made this last testing strategy impossible.

4.7 Documentation

A project proposal, mid-project report and on-line user manual were all written, in addition to this document.

V. MAJOR PROBLEMS ENCOUNTERED

5.1 Theoretical Disagreement

Although the theory involved in the types of exercises dealt with here is for the most part standardized, there are still several differing perspectives on certain issues. Opinions on what is correct and what is not can vary from one ear to another, and can depend on period and theoretical perspective. This problem was dealt with by including all of the rules that were found in at least two sources, but allowing the user to disable whatever rules are desired with the preferences option. I also implemented several different versions of some of the rules with varying degrees of strictness. Although this required a fair amount of extra work in the implementation phase, I felt that theoretical flexibility was essential to the success of the software, and this also facilitated testing.

5.2 Ill-Definedness of Theoretical Rules

Many of the rules found in the literature are loose guidelines and factors that should be considered rather than absolute rules. Considerations of taste and context are often essential. Although this is certainly understandable and desirable for artistic reasons, it is necessary for all rules to be strictly defined in order for them to be implemented on a computer. Although I did make my rules sensitive to the entirety of the compositions where possible, this was not always sufficient or possible. I was forced to make judgment calls on some rules. For example, if the literature said that there should not be "too many" consecutive notes moving in the same direction in a line, I formalized this as "more than five," based on sample exercises and models in the literature. Another step that I took towards solving this problem was to define two classes of errors: warnings and errors. The latter refers to absolute rules that the literature defines reasonably strictly and attaches a great deal of importance to, and the former refers to general guidelines that are either only suggestions or are not strictly defined in the literature. Despite all of this, however, the analyses produced by the software are not be able to take into account all of the information that an informed and experienced human ear would, so the analysis results produced by the software should be regarded as useful guidelines rather than absolute truth.

5.3 Data Structures

Musical data is fairly complex, and is difficult to easily store and process. I wanted to use data structures that would be easy to conceptualize and that contained information about the structure of the music itself, while also being general enough to have applicability to types of music beyond just species counterpoint. I did a fair amount of research on this topic, but was unable to find any data structures in the literature that were appropriate for what I wanted to do. Most commercial software use MIDI which, although useful in a performance context, is event driven and does not encapsulate the structural information that I needed. The other approach that I

encountered was the use of simple two-dimensional arrays based on timing and pitch which, although useful for tasks such as harmonic analysis, did not encapsulate enough information to deal with the voice leading information needed in the analysis of counterpoint such as the differentiation between voices or between repeated notes and suspensions.

My solution to this problem was to take an object-oriented approach. Each composition is stored in a Composition object, which holds general information about the composition and has an array of Voice objects. Voice objects contain information about general aspects of a voice as well as an array of Note objects. Each Note object represents a note, and holds information about pitch, duration and whether or not the note is tied and how. These data structures ended up being more general than required, since I did not implement the four-part harmonic analysis that I was hoping to, but nonetheless functioned very well and are easily expandable for use in projects of a greater scope.

The only real problem that I had with my data structures was that the type of analysis I did required me to treat enharmonic intervals as being exactly equivalent pitches. Although this has no negative impact on the actual analysis of compositions, it can cause notes to be incorrectly displayed in the GUI (e.g. G# might be displayed as Fb). I solved this problem by causing the GUI to consider the key signature of a piece when displaying notes, but notes that fall outside of key signatures may still be incorrectly displayed. This manifests itself only in the case of raised 6^{th} and 7^{th} scale degrees at the end of some minor compositions, and could be solved with additional GUI processing, but is nonetheless an inconvenient aspect of the data structures.

5.4 Design

I wanted to design the software in such a way that it would be easily expandable to more sophisticated genres of music in the future. I also wanted to make it easy to test and modify and to make it user customizable. I was able to accomplish these goals by implementing each rule independently of each other rule, and then have controller classes call the particular rules that are selected in the preferences. This made it easy to fit a new interface over the analysis portion of the software, and made it easy to control which rules were activated or to add more rules.

I was also hoping to use inheritance to avoid having to re-implement rules that various species had in common. This last goal was difficult to accomplish, however, since the rhythmic considerations of each species meant that the same rule sometimes had to be implemented in different ways for different species. I made the implementation of my rules as general as possible, but making the implementation species dependant was sometimes unavoidable. In addition, the rules were not consistently carried through the species. For example, a rule might be present in second and forth species, but not third. The result of this was that using inheritance among the analyzer classes for the different species would have

ended up making the design overly complex and difficult to understand. I therefore chose to make the controller and analyzer classes for each species entirely independent. This resulted in a certain amount of repetition, which made bug fixing a little bit inconvenient, but overall this strictly modular approach made the design easier to understand.

5.5 Fifth Species

Fifth species was difficult to implement because it often switches between other species, and temporarily uses the rules of these species.

5.6 Graphical User Interface

This was one of the most difficult areas of the project. It is very difficult to implement an interface that is easy for music students with potentially little computer background to understand and use. Most commercial software that deals with music tries to simulate, often with only limited success, the writing of music by allowing users to enter data with the mouse onto screens that look something like manuscript paper. Although this approach is by far the easiest type of interface available for people who are musically literate to use, it is very difficult to implement, with many difficult details to deal with, such as barring and spacing.

An additional problem is that this approach allows the user a great deal of freedom to enter types of musical data that are inappropriate for the type of analysis done by my software. This means that the error checking of the analysis portion of the software would need to be much more complex and involved to compensate for inappropriate user input.

Unable to find any reasonably easy alternative to the musical notation approach, I decided to design my own table-based interface. This interface allows the user to enter notes into a table where the rows are indexed based on rhythmic location and the columns contain information about pitch, duration and ties. The user selections options from combo boxes that limit the kind of data that can be entered. This strikes a good balance between restricting the user from entering types of input that might cause the analysis to malfunction, while at the same time allowing the user enough freedom to enter incorrect data so that he or she can learn from his or her mistakes.

Although a notation based interface would certainly be easier to use than this table-based interface, this approach is still fairly logical and easy to learn. The amount of work that would have been needed to implement a notation based interface made it unfeasible, and the table based approach is a good compromise between usability and feasibility.

5.7 Implementation

Although there were no significant problems encountered during implementation, the quantity of the rules which needed to be implemented made the process quite time consuming, which limited the scope of the project to species counterpoint.

5.8 Testing

The essentially infinite possible compositions that could be input made the analysis portion of the software difficult to test. To help deal with this problem, I designed the GUI to prevent the user from being able to enter certain kinds of erroneous input. I also made each rule modular and independent of all others so that it could be tested individually, both logically and with a number of compositions custom designed to test it thoroughly. Once all of the rules for a given species were tested this way, I tested all of the rules together on flawed examples and model examples from the literature. Unfortunately, although the testing that was carried out makes me confident that the rules will all function properly under most circumstances, the number of possible compositions that could be input and the time needed to thoroughly test all rules made it impossible to be absolutely certain that they will operate correctly in unusual circumstances. The best way to detect any remaining problems would be through user testing, but time is not available for this. So, while I am sure that the analyses will be correct under most circumstances, users should be aware that the analyses may not be absolutely correct 100% of the time.

VI. WHAT I LEARNED

6.1 Differing Theoretical Perspectives

Although I already had a fair amount of knowledge about counterpoint and harmony when I started this project, I was mostly familiar with the perspectives that I had learned in school. The musical research done for this project familiarized me with some of the historical theoretical debates that have happened among theorists of different perspectives, and exposed me to some of the subtleties of how approaches to dealing with and teaching counterpoint have changed with time.

6.2 Computer Music

My studies up until this point have focused on either computer science or music, but have not integrated the two. I read a number of books dealing with various techniques and approaches to computer music analysis and generation during the research phase of the project, which I found very interesting. Although in the end I made fairly limited use of what I read, since most of it was not directly applicable to my project, it taught me a lot about an area that I plan to study at the graduate level and prevented me from designing in some ways that might have been caused difficulties.

6.3 Musical Representation on Computers

Representing musical data in a computer in a way that represents musical structure and meaning is an area where a lot of research is currently being done, and I feel that I learned a lot about different approaches to using computers to represent and analyze music in this project. The research that I did into existing approaches and the different ideas that I considered when designing my data structures taught me about ways that this could be approached, and the problems that I encountered while working on the analysis portion of the project taught me a lot about the strengths and weaknesses of the approach that I chose.

6.4 Software Design

Although I have taken a number of courses on software design, this was the first time that I ever worked on a project this complex and involved. It was also the first time that I worked on a software project anywhere near this size on my own, or came up with my own project and design independently. This project gave me a chance to work on all aspects of the project by myself and gave me a chance to implement some of the design principles that I had learned about. I feel that the my experiences on this project increased my ability to design and implement software systems of this size, gave me experience within the entire development process from start to finish and gave me better judgement of the time and work required to accomplish various types of tasks. The relatively problem-free progress of the

implementation and testing phases also reinforced my understanding of the benefits of spending time on good design.

6.5 Musical Interfaces

My work on designing the GUI for this project made me think a lot about the kinds of ways in which humans can interact easily with musical data. My previous experience with dealing with music on computers had involved notation based interfaces, and designing the GUI for this project made me consider alternatives and their strengths and weaknesses.

6.4 Graphical User Interface Implementation

Although I did have some experience designing GUIs, I had never used Swing before, and I had only ever implemented one GUI previously, in TCL / TK. This GUI was somewhat sophisticated, and working on it taught me a lot about writing GUIs in general, and about using them to present and take in specifically musical information.

VII. AREAS FOR FUTURE EXPANSION

7.1 Overview

The data structures and general structure of the software are all designed to be fully usable for music involving four-part harmonies. They could also easily be adapted for the error checking or analysis of other genres of music. This section outlines features that could be added or modifications that could be made in the future to improve the software or increase its scope.

7.2 Analysis of Four-Part Harmonies

As just mentioned, the infrastructure is already present to analyze four-part harmonies, although the interface would require some modifications. The software could use many of the existing rules to check for voice leading errors. It could also analyze the harmony of pieces and output the names, figured bass and harmonic functions of chords. Context-free grammars could be very useful here, particularly if modulation is present.

7.3 Music Generation

The rules that are programmed for the five species could be used to generate full species counterpoint instead of just cantus firmi, as is the case in the current implementation. The random generation of cantus firmi is done by randomly generating candidate solutions in a moderately intelligent fashion (the problem is otherwise intractable) and then testing them using the rules until a piece is generated where no errors are detected. This process could be sped up by using more intelligent transition tables for the generation of candidate compositions. In addition, penalty point values could be associated with the rules to reflect their relative importance, and candidate solutions could be accepted as long as the penalty points for all of the rules that they violate are below some threshold. These values would be arbitrary to a certain degree, but this approach would be an improvement over the current system of rejecting a candidate solution even if it only violates one minor rule in a minor way, as many excellent human compositions do.

7.4 Notation-Based GUI

The GUI could be redone so that it follows the paradigm of notation based interfaces. Although the current GUI is adequate, and this notation-based approach would require a great deal of time to implement properly, it would increase the usability of the software.

7.5 Web-Based Interface

It would be relatively easy to implement a web-based interface for the software that uses tables. This would enable the software to be housed on a music department's web site, for example, where it could be easily accessed by students. A more sophisticated interface could also be designed, using Flash or JavaScript, for example, or the system could be made into an applet.

VIII. BIBLIOGRAPHY

8.1 Music Theory

Aldwell, Edward and Carl Schachter. *Harmony and Voice Leading*. San Diego, California: Harcourt Brace Jovanovich, 1979.

Davis, Ferdinand and Donald Lybbert. *Essentials of Counterpoint*. Norman, Oklahoma: University of Oklahoma Press, 1977.

Forte, Allen. *Tonal Harmony in Concept and Practice*. Montreal: Holt, Rinehart and Winston, 1979.

Kitson, C. H. *The Art of Counterpoint*. New York: Da Capo Press, 1975.

Kitson, C. H. *Contrapuntal Harmony for Beginners*. London: Oxford University Press, 1931.

Kitson, C. H. *Elementary Harmony*. Oxford: Clarendon Press, 1961.

Lester, Joel. *Harmony in Tonal Music*. New York: Random House, 1982.

Morris, R. O. *Introduction to Counterpoint*. New York: Oxford University Press, 1963.

Orem, Preston Ware. *The Art of Interweaving Melodies*. Philadelphia: T. Presser, 1937.

Salzer, Felix. *Counterpoint in Composition*. Toronto: McGraw-Hill Book Company, 1969.

Thakar, Markand. *Counterpoint: Fundamentals of Music Making*. New Haven, Connecticut: Yale University Press, 1990.

8.2 Computer Music

Dodge, Charles and Thomas A. Jerse. . *Computer Music, Synthesis, Composition and Performance.* New York: Schirmer Books, 1985.

Mathews, Max V. and John R. Pierce eds. *Current Directions in Computer Music Research.* Cambridge: MIT Press, 1989.

Moore, F. R. *Elements of Computer Music.* Englewood Cliffs, New Jersey: Prentice-Hall, 1990.

Roads, C. *The Computer Music Tutorial*. Cambridge: MIT Press, 1996.

Rowe, Robert. *Interactive Music Systems.* Cambridge: MIT Press, 1993.

Rumsey, Francis. *MIDI Systems and Control.* Toronto: Focal Press, 1994.

8.3 Computer Science

Appel, Andrew W. and Maia Ginsburg. *Modern Compiler Implementation in C*. New York: Cambridge University Press, 1998.

Arnold, Ken and James Gosling. *The Java Programming Language.* Mountain View, California: Addison-Wesley, 1997.

Flanagan, David. *Java Examples in a Nutshell.* Sebastopol, California: O`Reilly & Associates, 2001.

Hopcroft, John E. and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Don Mills, ON: Adison Wesley Publishing, 1979.

Hopkin, David and Barbara Moss. *Automata.* Surrey, UK: MacMillan Press, 1976.
Horstmann, Cay S. and Gary Cornell. *Core Java 2 – Fundamentals.* Palo Alto, California: Sun Microsystems Press, 2001.
Horstmann, Cay S. and Gary Cornell. *Core Java 2 – Advanced Features.* Palo Alto, California: Sun Microsystems Press, 2000.
Howie, John M. *Automata and Languages.* Oxford: Clarendon Press, 1991.