Chemistry Visualization Software

A program to assess and animate carbon compounds

Roland Heintze, Tim Elam, Chris Lansing, John Gibbons

4/27/2013

Table of Contents

1. Introduction 7
1.1 Team Introduction 7
1.1.1 Team Name 7
1.1.2 Team Members and Duties 7
1.2 Project Description 8
1.2.2 Screenshots 8
2. Feasibility Study 10
2.1 Revisions 10
2.2 Client 10
2.3 Task to be undertaken 11
2.4 Domain Analysis 11
2.5 Benefits 11
2.6 Preliminary Requirements Analysis 12
2.6.1 Interface 12
2.6.2 Backend 12
2.6.3 Drawing 12
2.7 Suggested Deliverables 12
2.7.1 Management 12
2.7.2 Technical 13
2.8 Process to be followed 13
2.9 Development Schedule 14
2.10 Outline Plan 15
2.10.1 Requirements Gathering 15
2.10.2 Design 15
2.10.3 Customer Feedback 15
2.10.4 Development 15
2.10.5 Testing 15
2.10.6 Release 16
2.10.7 Maintenance 16
2.11 Visibility Plan 16
2.11.1 External 16

2.11.2 Internal 17
2.12 Risk Analysis 17
2.12.1 Time Risk 17
2.12.2 Existing Software Risk 17
2.12.3 Resource Risks 17
2.12.4 Changing Requirements 18
2.13 Probable Technical Requirements 18
2.14 Conclusion 19
3. Requirements Documents 19
3.1 Revisions 19
3.2 Overall Description 20
3.2.1 Product Perspective 20
3.2.2 User Interfaces 20
3.2.3 Software Interfaces 21
3.2.4 Communication Interfaces 21
3.2.5 Memory Constraints 21
3.2.6 Operations 21
3.2.7 Site Adaption Requirements 22
3.3 Production Functions 22
3.4 User Characteristics 23
3.5 Constraints 24
3.5.1 Hardware Limitations 24
3.5.2 Interface to other applications 24
3.5.3 Parallel operation 24
3.5.4 Audit functions 24
3.5.5 Higher-order language requirements 24
3.5.6 Reliability Requirements 25
3.5.7 Criticality of the application 25
3.5.8 Safety and security considerations 25
3.6 Assumptions and Dependencies 25
3.7 Apportioning the Requirements 25
3.8 Specific Requirements 25

3.9 External Interfaces 26
3.10 Functions 27
3.11 Performance Requirements 27
3.12 Design Constraints 28
3.12.1 Standards Compliance 28
3.13 Software System Attributes 28
3.13.1 Reliability 28
3.13.2 Availability 29
3.13.3 Security 29
3.13.4 Maintainability 30
3.13.5 Portability 30
3.14 Use Cases 31
3.15 Prototypes 37
4. Project Schedule 39
4.1 Timeline 39
4.2 Work Breakdown Structure 40
4.3 PERT Diagram 41
4.4 Lines of Code Estimation 42
4.5 Revisions 42
5. Design Documents 43
5.1 UML Diagrams 43
Original UML 43
Updated UML 45
5.2 State Diagram 49
5.3 Prototype 50
5.4 Revisions 52
6. SQA Plan 53
6.1 Requirements 53
6.1.1 Revisions 53
6.2 Design 55
6.2.1 Revisions 55
6.3 Implementation 57

6.3.1 Revisions 57
6.4 Testing 60
6.4.1 Revisions 60
6.5 Installation 62
6.5.1 Revisions 62
6.6 Maintenance 63
6.6.1 Revisions 63
6.7 Checklists 64
6.7.1 Requirements 64
6.7.2 Design 67
6.7.3 Implementation 70
6.7.4 Testing 74
6.7.5 Installation 76
6.7.6 Maintenance 78
7 Test Plan 79
7.9 Revisions 79
7.1 Risk Assessment 80
7.2 Team Member's Responsibilities 80
7.3 Test Items 81
7.4 Features to be tested 81
7.5 System Functions 87
7.6 Testing Machine's System Requirements 87
7.7 Approach 88
7.7.1 Unit Testing 88
7.7.2 Integration Testing 88
7.7.3 Functional Testing 88
7.7.4 System Testing 88
7.7.5 Alpha Testing 89
7.7.6 Beta Testing 89
7.7.7 Acceptance Testing 89
7.8 Testing Deliverables 89
7.8.1 Unit Testing 89

7.8.2 Integration Testing 90
7.8.3 Alpha Testing 90
7.8.4 Beta Testing 90
7.8.5 Acceptance Testing 90
8 Testing Reports: 91
8.1 Unit Testing: 91
8.1.1 Alkane Test 91
8.1.2 Carbon Test 94
8.1.3 Substituent Test 102
8.1.3 Chemspipy Test 103
9 Post-mortem 104
9.9 Roland Heintze 104
9.10 Tim Elam 105
9.11 Chris Lansing 105
9.12 John Gibbons 105
10 Appendix 107
10.9 Source Code 107
chemistry/__init__.py 107
chemistry/alkane.py 108
chemistry/carbon.py 112
chemistry/chem_exceptions.py 118
chemistry/substituent.py 119
data/launch.ui 120
network/__init__.py 127
network/cactusAPI.py 128
network/chemspipy.apy 130
network/chemspipy_LICENSE.txt 136
ui/__init__.py 137
ui/carbon_view.py 138
ui/main.py 141
ui/ui.py 143
ui/AnimateView.py 147

ui/Frame.py 149
10.10 User Manual 150
Creating a Custom Molecule 150
Creating a Random Molecule 155
Checking your Nomenclature 158
Clearing the Nomenclature 159
Validating the Nomenclature 160

1. Introduction

1.1 Team Introduction

1.1.1 Team Name

Chemistry Visualization Software

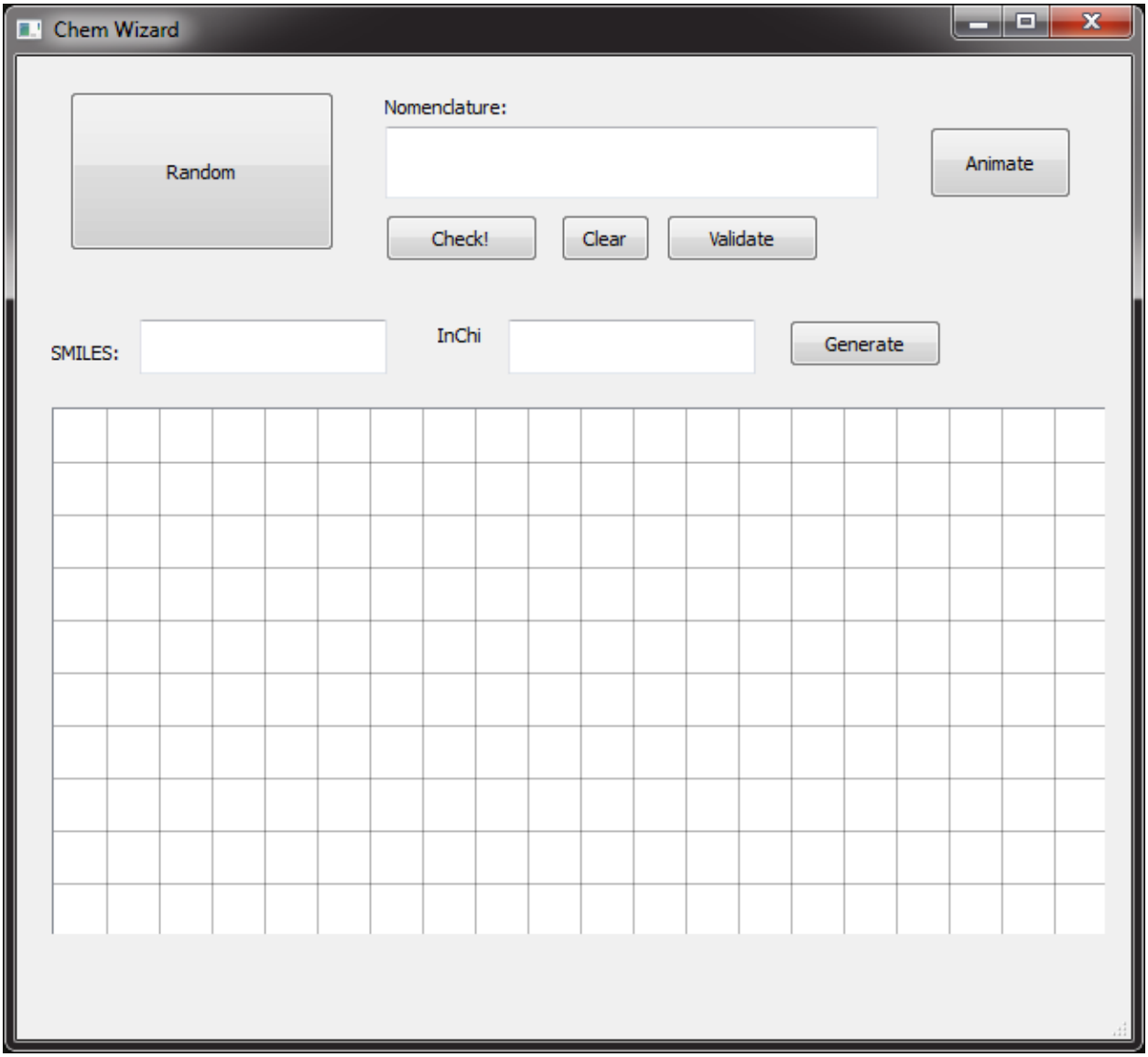
1.1.2 Team Members and Duties

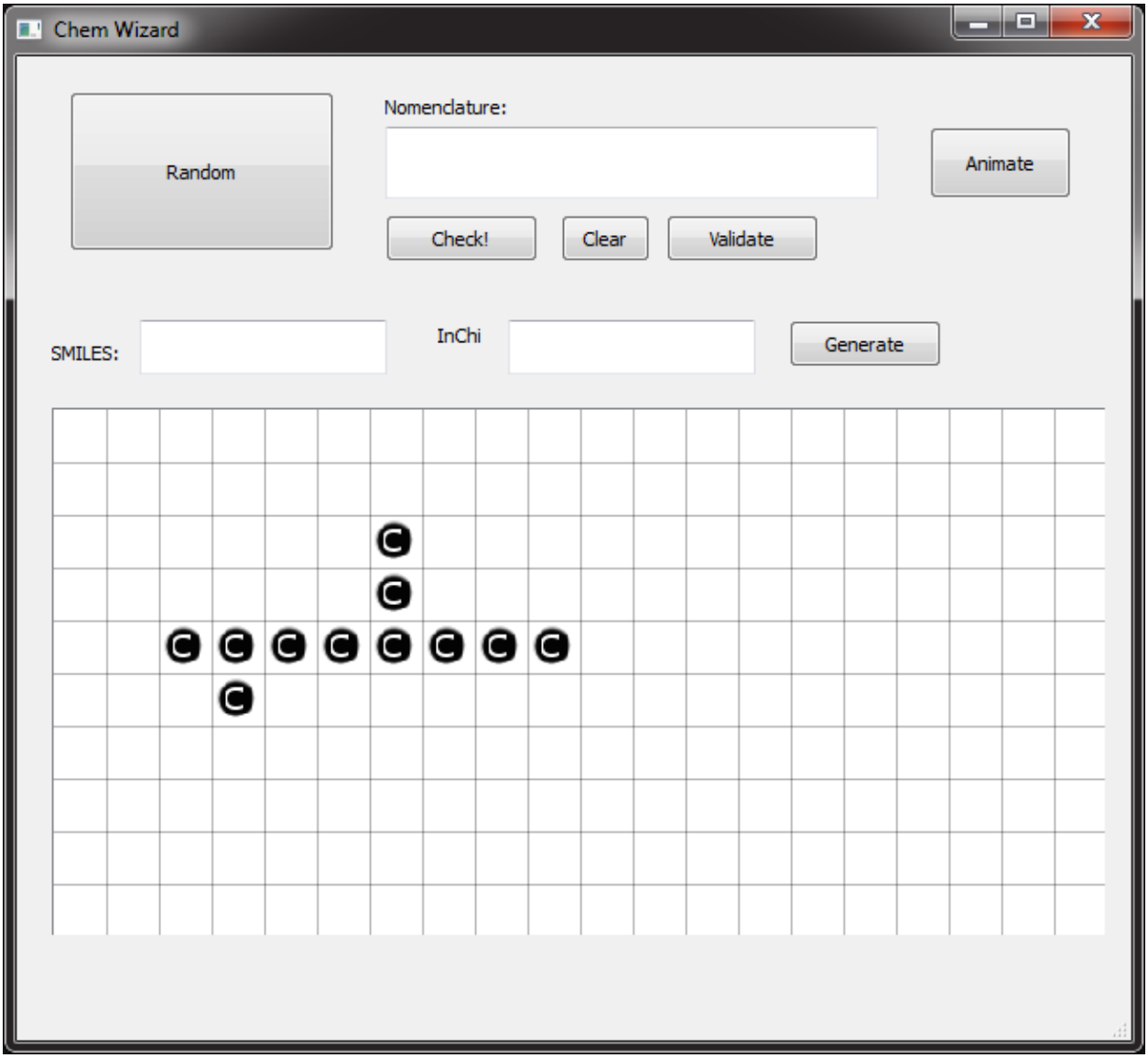
Name:	Email:	Role:
Dr. Shainaz M. Landge	slandge@georgiasouthern.edu	Client
Roland Heintze	dh02599@georgiasouthern.edu	Project Lead
Tim Elam	te00838@georgiasouthern.edu	Lead Developer
Chris Lansing	cl02064@georgiasouthern.edu	Developer/Clerk
John Gibbons	jg03039@georgiasouthern.edu	Quality Assurance Officer and paper work.

1.2 Project Description

Our project is to create a program for Dr. Landge that can read a chain of carbons, calculate the name of the compound that the chain forms, and then draw an animation based on that compound. The animation, as described by Dr. Landge, will be of a mailman walking along a road and delivering mail to mailboxes along that road. The road will made from the bonds of the compound and the mailboxes will be made from each carbon molecule. The objective of this program is to give a visual demonstration of the compounds and how they work in hopes that students will become familiar with the animation and have something to associate the compounds with in order to remember them more easily.

1.2.2 Screenshots




Chem Wizard
Random
Nomenclature:
Animate
Check!
Clear
Validate
SMILES:
InChi
Generate

2. Feasibility Study

2.1 Revisions

Revision Number:	Revision Date:	Author:	Summary of Changes:
1	2-28-13	John Gibbons	Added revision page and reworked/reworded first half of document.
2	2-29-13	John Gibbons	Reworked/reworded second half of document and added domain analysis.
3	3-26-13	John Gibbons	Third revision with requirements change.
4	4-9-13	John Gibbons	Fourth revision with requirements change and platform change.
5	4-25-13	John Gibbons	Changed requirements with GUI change for molecule custom creation.
6	4-26-13	Roland Heintze	Finish domain analysis and final revision.
7	4-27-13	Chris Lansing	Grammar corrections

2.2 Client

Dr. Shainaz Landge an Organic Chemistry and Principles of Chemistry professor of the Georgia Southern University Chemistry Department. Organic chemistry is a branch of chemistry that deals with the structure, properties, and reactions of compounds that contain carbon. Organic chemists can create new molecules never before proposed which, if carefully designed, may have important properties for the betterment of the human experience. Dr. Landge teaches students the current understand of

organic chemistry and sets them on a path to further develop within this field. As a principles of chemistry professor as well, Dr. Landge explains and shows the constituents of a substance and their attributes as well as possibly benefits and detriments they have or may have if combined with other elements.

2.3 Task to be undertaken

The project entails creating an animation that will demonstrate, through animations, the proper naming of organic compounds. The program will allow the user to arrange carbon elements and their bonds into a proper molecule. The user can also choose to randomly generate the organic compound with one button click. It will then display a short animation in accordance with Dr. Landge's present teaching analogy of a mailman traveling a route in a specific manner which follows the algorithm chemists use for the proper identification and naming of organic compound molecules.

2.4 Domain Analysis

In preparation for developing this software our group researched to see if any other software existed that would meet or partially meet the requirements given to us by Dr. Landge. There was a wealth of similar software to what she wanted but nothing specific enough to her needs. eMolecules was the most similar, providing a java applet interface for placing Carbons and then pulling information about the compound, to include the IUPAC name and various properties such as the molecular weight. Additionally, we found two other web APIs useful for gathering chemical compound properties and information. They were the ChemSpider database and CADD Group Chemoinformatics Tools and User Services, or Cactus. The problem with all of these pieces of software though, is that they relied upon an internal database so the user had no access to how the IUPAC name was resolved from the structure. Dr. Landge seeks to have a tool which will not only tell the student the name, but more importantly draw an animation to give the student a clear idea of how this name is generated. In order to accomplish this, we realized that it would be necessary for us to develop our own tool to take an organic compound, generate a data structure for it, and then pull from this graph the parts relevant to generating the proper IUPAC name. Using this information then, methods would need to be constructed to depict an animation of this process.

2.5 Benefits

The benefit of this program is that it will give students an elaborate tool for learning the proper means of finding an organic molecule's nomenclature. This will be used to reinforce the lecture they receive in class and make things more interesting and fun for the students. This will also allow the client to focus on other subjects in the classroom while students review the nomenclature conventions with this software at home. This has the potential to be a publicly released asset for other professors to use in the

classroom or to be downloaded by people interested in chemistry and the naming conventions used. In summary, our software will offer the following benefits:

- An automated teaching tool for students and professors.
- An illustration of a simple analogy for a complex topic.
- A means to quickly identify the name of a complex molecule for the public domain

2.6 Preliminary Requirements Analysis

2.6.1 Interface

An interface which allows the user to click in squares making up a grid to arrange carbon atoms and their bonds. Alternatively, they should be able to generate a random compound. It should then offer options to generate a proper nomenclature or to generate the animation.

2.6.2 Backend

A system must exist to discover the proper name for an organic carbon compound based on the user input or randomly generated compound. It must additionally provide a path for the purposes of animation.

2.6.3 Drawing

The software must be able to take the data from the backend and generate a proper animation.

2.7 Suggested Deliverables

2.7.1 Management

- Requirements Analysis - A formal requirement. Will contain all requirements delivered from the client, minimum specifications the machines need to run the software, design constraints and user characteristics.
- Status Reports - The customer will be kept updated with ongoing status reports that outline our progress with the program, mostly via e-mail. These reports will play a major role in keeping the customer involved and satisfied with the project.
- Source - The project will be hosted on a private git repository on GitHub. This site contains the source code as well as limits those who have access based on access levels granted by the creator of the repository. Any and all additions are time stamped as well as the user's name that made the changes. Users can choose to have the newly added

code highlighted. The creator then has the option to allow the code to remain permanent or revert the code back to the previous state.

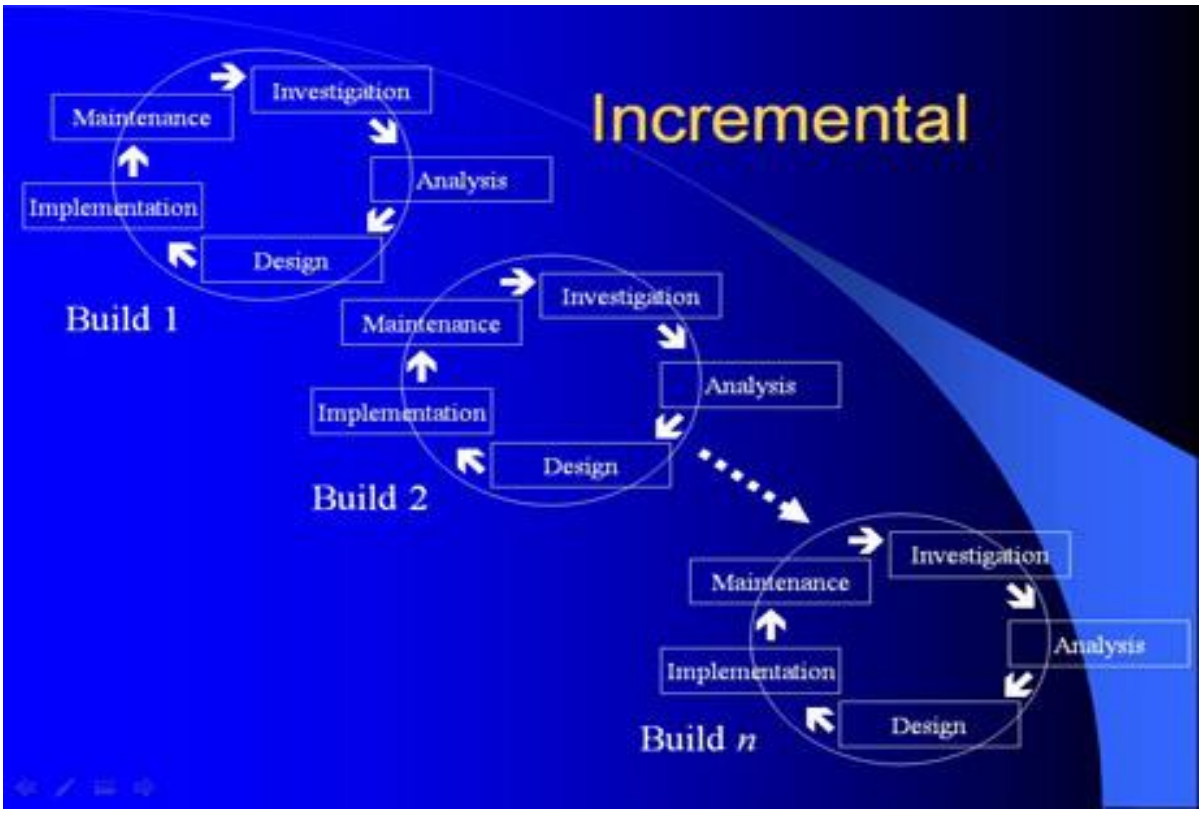
- User Manual - A guide or tutorial to explain the software and its use.

2.7.2 Technical

A portable version of the software would need to be available.

2.8 Process to be followed

The team will utilize the Incremental Model for the development life-cycle. In this way we can continually develop the software while having the client review parts of the implementation. In this way we can continually receive feedback to ensure that the client remains part of the development process. This will address any confusion or misunderstandings pertaining to the requirements or deliverables at any stage of development.



2.9 Development Schedule

Milestone 1: Feasibility Study (1/22/13): Begin Feasibility study draft. Begin Domain analysis research.
Milestone 2: Feasibility Study (1/29/13): Final version of our groups Feasibility study.
Milestone 3: Client Meeting (2/1/13): Begin extensive requirements gathering and design models.
Milestone 4: Requirements Documentation (1/30/13): Begin constructing the requirements documentation and ensuring all requirements are fully documented.
Milestone 5: Requirements Documentation (2/7/13): Final version of requirements documents with use cases each containing at least one use case scenario and GUI prototypes.
Milestone 6: Project Schedule (2/8/13): Begin creating a comprehensive project schedule for development.
Milestone 7: Project Schedule (2/12/13): Final version of project schedule with milestones.
Milestone 8: Project Design (2/13/13): Begin creation of UML diagram and state diagrams. Start creating classes and variables with types.
Milestone 9: Project Design (2/16/13): Final version of UML and State Diagrams are to be completed. All foreseen variable names and method names are to be included.
Milestone 10: Software Quality Assurance (2/22/13): Begin constructing the checklists to ensure each stage of development is professional and are performed according to IEEE and standard Object Oriented Design principles.
Milestone 11: Software Quality Assurance (2/28/13): Final versions of checklists with their own documents are to be completed.
Milestone 12: Implementation (2/22/13): begin implementing the code in accordance to all documentation and designs.
Milestone 13: Implementation (4/20/13): Desired date to have completed implementing all code to ensure plenty of time for testing.
Milestone 14: Test Plan (4/1/13): Begin designing test plan and discerning possible areas of concern.
Milestone 15: Test Plan (4/21/13): Final version of test plan.
Milestone 16: Alpha Testing (4/22/13): Conduct alpha testing with the client to flesh out any functional or design issues the client may have with the current software.
Milestone 17: Beta Testing (4/24/13): Conduct beta testing to ensure all previous issues have been resolved and the program functions according to the client's wishes on platforms the client previously stated for the requirements documents.
Milestone 18: Test Reports (4/27/13): Completion of all test report documentation.
Milestone 19: Notebook (4/28/13): Completion of the notebook and all documents within to submit on schedule.
Milestone 20: Presentation (5/7/13): Completion of PowerPoint and rehearsal for presentation.

2.10 Outline Plan

2.10.1 Requirements Gathering

We will need to communicate with the customer to draft the requirements document. This document will include everything that the customer believes we need in the program and we will follow it precisely. Our team may provide additional suggestions to the customer as to requirements and functionality of the program in order to deliver a more robust program. However, the requirements will ultimately be up to the customer. To avoid any future disagreements with the customer, we will have the client sign off on the requirements document confirming that the stated requirements will be what our team will be developing from.

2.10.2 Design

We will create the necessary diagrams for our program, such as UML, class, and use case diagrams. This way we will be able to flesh out the entire program and have a solid understanding of how we will development it. We will be able to quickly and efficiently complete development by following the models, and all members of the team will have an outline to follow should they ever misinterpret the code or its design.

2.10.3 Customer Feedback

Though we will be getting feedback from the customer throughout the entire development cycle, it is crucial that we get detailed feedback at this point from the customer. We want to ensure the customer and our development team is in full agreement and understanding on all aspects of the project before we start development in regards to how the program is going to function and the desired final deliverables.

2.10.4 Development

The overall structure and function of the program will be implemented from the design documents. The bulk of the development for this project will be creating the necessary animations to illustrate the story that the customer requires. Some or all of the code that Dr. Cook has created for the project may need to be rewritten. This process should not be time consuming due to the fact that all of the algorithms that are used, which we will be using to name the compounds, are all well-defined, scientifically precise and straightforward.

2.10.5 Testing

Once our team is satisfied with the usability and functionality of our program, we will begin unit and integration testing. Unit testing will be done using PyUnit to verify that all of the classes, methods,

objects, and variables are functioning correctly. Integration testing will be done by running many (possibly automated) simulations of our program to ensure that it does not fail on any edge cases. Changes to the program may be made as necessary during this phase in order to prepare it for release.

2.10.6 Release

Once our product is finished, we will deliver a complete, portable, copy and paste-able version of our software to the customer via USB drive. This program will be small enough to transfer to students' computers via USB drive, a shared folder on the school network, or allows for the professors to send the program directly to students. After the file is on a student's computer, the program will run simply by double clicking on the icon of the file from the desktop.

2.10.7 Maintenance

While currently there is no planned ongoing maintenance for the software after the class is over, the customer will have our team members' emails in case future questions or concerns, attempts to get one or more members to continue development, or to fix any bugs in the software. The customer will also be given the entire source code for the project. This source code will be school property and the customer may enlist the help of any student(s) to take over the project once our team is finished working with it.

2.11 Visibility Plan

2.11.1 External

It is vital that at least one member of our team make weekly/bi-weekly check-ins with the client to ensure the program is constructed to all specifications and requirements and that any new questions or concerns from the client are addressed. The base code that generates the molecules has already been written by Dr. Cook and has been approved by the client. Our development team will be constructing an animation to explain how each combination is named. With the procedures being well-defined, we have to ensure that the integrity of the naming process stays intact. Also, we need to ensure that our client's models come to life in a way that will best convey the messages to the students properly who will be using this program as a study tool. One or more members of our group will report our progress to the client and explain current status of designs and coding up to that point. If at all possible, our team wishes to show working examples and demonstrations of functionality of the software within one and a half months into development. We will make sure the client agrees to and signs off on each step of the development process such that there will be records of approval and satisfaction.

2.11.2 Internal

In addition to regular contact with the client, regular contact between the group members is vital in completing our task on time, professionally and to ensure the clients' wishes are fully met. The way our team has set up communication involves using IRC for chat purposes. Our group is also using Google documents to keep track of all documentation and participation. Finally, the team is using GitHub to keep track of the source code, as well as times and sections of code that were recently edited and by whom. This gives each member of our group instant access to all sections of the project from any machine that is connected to the internet. We will also schedule meeting times and places for face to face group work/discussions when necessary.

2.12 Risk Analysis

2.12.1 Time Risk

As our team has limited time to complete this task, time management will be crucial. Our group will be setting up multiple deadlines to ensure work is completed early or on time. A checks and balance system will be implemented where group members can check on other group members' progress to ensure deadlines will be met. Roland has the ability to reassign work or assign additional group members onto a section of work anytime he feels necessary to help ensure the deadlines are met.

2.12.2 Existing Software Risk

The existing software our group will be working from was created by Dr. Cook. If it takes longer than expected for our software development team to fully grasp the code and its functionality, Dr. Cook has offered to help explain those areas to one or more members of our group at any point he is available. However, Roland and Tim have spoken with Dr. Cook already and stated that they have a good grasp on Dr. Cook's code and its functionality and thus our group does not foresee this being an issue.

2.12.3 Resource Risks

The main resource risk in this project would be if one or more members of our group decide to not manage their time properly or decide not to contribute to the project at all. If either or both of those events transpire then the whole project could fall behind or run the risk of failure to be completed within the given time schedule. With the above mentioned checks and balance system as well as the group leader, Roland, checking on progress at certain intervals, our group hopes to prevent this or become aware of this situation far enough in advance to compensate for it. This project consists of extensive animation, where our team will be relying on Roland and Tim, who have taken courses and have experience in this field, to do the majority of the coding. If both those members get sick and/or become unable to code, that situation would cause our team and the project serious issues. However, if one becomes unable to code, the other would be able to compensate for the work load with support

from Chris and John. As far as the documentation is concerned, all four members of the group have experience with creating, editing and completing documentation, so our group should not encounter any issues pertaining to this area. Additional risks include the customers' hardware and additional costs revealed after the initiation of the project. The hardware will not be an issue because the machines in the school labs as well as the majority of machines owned by the students and professors will be able to run the .exe the program will be made into. However, if the .exe file does not function properly, the Python interpreter Py2exe will be used to allow the program to run on those machines. On top of the interpreter, the machines will need to possess the processing power to generate and run the animations. As for additional costs, there are none. In fact, this program will be completely free to develop. All programs and/or applications needed to run this program are free to download and use. Also, the machines in the labs do not need to be upgraded for this program and have dedicated hours for students to use so there is no additional costs that may occur after development.

2.12.4 Changing Requirements

It is possible that during the semester, the client may want more animations, higher end graphics or a different sequence of animations. To help ensure that our group can deliver the best possible software to the client, the group leader will have the client sign off at the beginning of the development on the requirements our group will be given. If the client wishes to add additional requirements, our development team will assess the changes and provide a detailed explanation as to which new requirement(s) are possible to implement in the timeframe allotted and which are not. As for the client being sick, this may delay the progress reports done in person which entail showing current functionality but one member of our group can still email the current progress and possibly include screen shots of the animations depending on their completion. Another factor would be if the client becomes disinterested in the project at any period during development. If that occurs our team would work with closely with the client to hopefully re-inspire them back into the project by doing demonstrations (if possible) and by showing current models of what the program is currently doing and what it will be doing as well as going into how students can greatly benefit from the lessons this program can provide them.

2.13 Probable Technical Requirements

The software will be required to operate on multiple platforms to facilitate its' availability to a wide range of students from many different classrooms. In order to meet this requirement, our group has chosen to write the application in Python 3 with PyQt4. This way the program will be capable of running on all platforms when converted to an .exe file at compile or with a Python Interpreter called Py2exe. This includes virtually all available platforms to the user population. Our application requires no additional applications or libraries outside of its own byte code so portability will be transparent on the development end.

2.14 Conclusion

This is a great project with outstanding rewards. The cost for this project is non-existent. The project, within the specified guidelines and requirements, is able to be completed within the given timeframe of this semester. When this project is finished, it will deliver a product to the customer that will satisfy the clients' requirements and expectations for the software and more. Also, our group believes that it will be a program in which other professors and students will enjoy and benefit greatly from.

3. Requirements Documents

3.1 Revisions

Revision Number:	Revision Date:	Author:	Summary of Changes:
1	2-28-13	John Gibbons	First revision of draft. Reworded sentences and corrected grammatical errors.
2	3-4-13	Roland Heintze	Second revision of draft. Moved document into Word. Added use cases, cover page, formatting and a table of contents.
3	4-9-13	John Gibbons	Third revision with requirements change and platform change.
4	4-25-13	John Gibbons	Fourth revision while updating requirements and GUI change for custom molecule creation.
5	4-26-13	Roland Heintze	Final revision.

6	4-27-13	Chris Lansing	Grammar corrections

3.2 Overall Description

This document specifies the Software Requirements for the application for teaching organic compound nomenclature methodology. It describes the scope, requirements, constraints, and interfaces.

3.2.1 Product Perspective

The product will operate as an application for assisting students in learning the proper naming conventions of organic compounds. It will allow users to generate random organic molecules or create their own. The user can then choose to be given the proper nomenclature of the molecule or provide a guess. The user will submit guesses of what the proper name of the compound is. Whether deciding to guess the name or not, the user may elect to watch a brief animation of an extended metaphor that explains the naming process from the perspective of a hypothetical mailman.

The dual purpose of this document is to gather requirements for our groups planned solution to deliver to Dr. Landge and to track deviations from our initial development plans. The audience of this document is Dr. Landge and her peers.

3.2.2 User Interfaces

a) The logical characteristics of each interface between the software product and its users.

The GUI must be tailored in such a manner as to be easy to use on the target platform. It must also be easy to use for people with limited technical experience. The GUI must have complete functionality without being overly difficult or involved.

b) All the aspects of optimizing the interface with the person who must use the system.

The User Interface (U.I.) should be easy to learn and use.
The U.I. should be simple and organized in a neat manner.

The U.I. should provide obvious and direct links to other forms/menus.

The U.I. should have error feedback such that the user can know what they did wrong and what they need to do correctly the next time in case of an error.
The U.I. should contain vocabulary in which the user will be familiar with.

The U.I. design and colors should be pleasant to the user.

3.2.3 Software Interfaces

Operating System:

Windows XP, Vista, Windows 7, Windows 8, Linux, Solaris, and Mac OS.

Windows: RAM - 128mb, 64mb for Windows XP (32 bit), Disc Space - 124 MB, Browsers - Internet Explorer 7.0 and above, Firefox 3.6 and above, Chrome

Mac OS X: Intel-based Mac running Mac OS X 10.7.3 (Lion) or later. Administrator privileges for installation, 32/64-bit browser.

Linux - Oracle Linux 5.5+, Oracle Linux 6.x (32-bit), 6.x (64-bit), Red Hat Enterprise Linux 5.5+, 6.x (32-bit), 6.x (64-bit), Ubuntu Linux 10.04 and above, Suse Linux Enterprise Server 10 SP2, 11x, Ram - 64 MB, Disk space - 58 MB, Browsers - All OS that support versions of Firefox 3.6 and above.

3.2.4 Communication Interfaces

The user needs to have a working internet connection to be able to download the software or a working USB drive. The user may also need to download Py2exe, a Python Interpreter, in the event the of the software's exported .exe file does not work on their machine. The connection will be provided by Georgia Southern University via the campus network or the software will be provided by the professor via USB or emailed to the student's Georgia Southern emails.

3.2.5 Memory Constraints

At least 128mb of RAM and at least 50mb of persistent storage are needed.

3.2.6 Operations

a) The various modes of operations in the user organization (user-initiated operations)

The user clicks a button to generate a random bonded organic molecule. After viewing the animation that shows the molecule's naming procedure, the user can click a button to re-watch the animation. The user can also click within boxes arranged in a grid to generate a set of molecules in a window to form an organic compound and then watch the animation of its naming process. Before choosing to watch the animation, the user has the option to enter in what they believe to be the name of the organic compound and check to see if their guess is correct. This is completely optional and is not required to begin the animation. The user can enter in and check the validity of their guess as many times as they wish before clicking on the button to begin the animation. At the end of the animation,

the correct answer for the name of the organic compound will be displayed. If the user wishes to see the name of the molecule before the animation starts, they have the ability to do so.

b) Periods of interactive operations and periods of unattended operations.

The user can choose to participate in an interactive series of operations by making their own organic compound. This will involve clicking in boxes laid out in a grid format to custom form their own compound. This step is optional and is not required to be completed should the user choose to not utilize this feature. In the initial release of the software, the users will be limited to single bonds of organic molecules. There are no unattended operations.

c) Data processing support functions.

Generating random organic molecules and determining the proper name for the given bonded organic molecule.

d) Backup and recovery operations.

There is no backup option. The data is randomly generated or created by the user and is not critical to the program or later instances of the program. Each new randomly generated organic compound or user created organic compound is sufficient data for the program to use and for the user to learn from.

3.2.7 Site Adaption Requirements

The program will run on any machine that has Py2exe installed. However, Linux and Unix machines already have Python Interpreters installed and do not require this software.

3.3 Production Functions

This program will allow any user to generate a random organic compound and view an animation for the scientific nomenclatures for carbons up to and including pentadecanes. The user can also create their own organic compound through a series of clicking events into a series of boxes laid out in a grid format. Either process delivers an animation for the compounds' naming process. Before the user clicks the button to view the animation, they have the option of entering in the name of the bonded molecule and

checking to see if their guess is correct. The user can repeat the process of entering in a name and checking its validity as many times as they wish before viewing the animation. To begin the animation, the text field where the name may be entered does not have to contain anything or contain the correct answer to begin the animation. The design of the animation will be generated to exactly mimic the shape of the randomly generated or custom created organic molecule. A mailman will appear at the far left and upper position of the generated animation which resembles a road. The mailman will then walk the correct path down the roads while delivering mail to show how the naming process functions. After the animation has completed, the correct name of the compound will be displayed.

3.4 User Characteristics

The intended users for this software will be Organic Chemistry professors and students. Each instance of the program will support one user at a time.

Professors:

i. Education Level
   1. They possess a PHD in Organic Chemistry and have research experience in this field.
   2. They have intimate knowledge of how the naming process works and how it was created.

ii. Experience
   1. Dr. Landge has experience with the teaching method that generates the organic compound. She has no experience with software which contains animations or the ability to craft organic compounds.
   2. The professors have experience with an operating system capable of running the software.

iii. Technical Expertise.
   1. Ability to navigate common desktop environments.
   2. Understanding of virtual form controls.
   3. Understanding of the mouse clicking interaction metaphor.

Students:

i. Education Level.
   1. Students who are currently studying Organic Chemistry and can range from freshmen to postgraduate levels.

2. Students possess little to no knowledge of how organic compound naming process works or how it was created.

ii. Experience.

1. They possess no experience with this software.
2. Assumed to possess basic to high level experience with common desktop applications.

iii. Technical Expertise.

1. Ability to navigate common desktop environments.
2. Understanding of virtual form controls.
3. Understanding of the mouse clicking interaction metaphor.

3.5 Constraints

3.5.1 Hardware Limitations

Monitors should support 4:3 and 16:10 aspect ratios. A minimum resolution of 800x600 is recommended. The program should run on all common graphical displays.

3.5.2 Interface to other applications

The program will not interface with any external applications.

3.5.3 Parallel operation

The application will run as a single process running two separate threads. One thread will be used to receive input events from the user and update the display, while a second thread will be responsible for maintaining the internal state of the program and handling any calculations that might slow down the user interface.

3.5.4 Audit functions

Does not apply

3.5.5 Higher-order language requirements

Our development team is using Python 3 with the PyQt4 library to generate the randomly bonded or custom designed organic compounds and also to generate the animations for the naming process. The software requires a cross-platform graphical user interface and support for multithreading and synchronization.

3.5.6 Reliability Requirements

This software will be used for teaching and studying purposes. It follows that the application should perform consistently and correctly uses naming conventions of organic molecules.

3.5.7 Criticality of the application

This program will teach how organic compounds are named. It is not considered critical.

3.5.8 Safety and security considerations

The program does not contain any information that would need to be guarded or encrypted.

3.6 Assumptions and Dependencies

Our team assumes that the users will primarily using this program on the school computers, which have Windows 7 installed. The school computers are known to support 4:3 and 16:10 displays and support a resolution that exceeds 800px:400px.

3.7 Apportioning the Requirements

The client requires the development of animation software to illustrate the naming process for organic compounds. Our software will allow for the generation of random compounds as well as user-defined ones. The software will then generate an animation. The largest foreseeable improvements on the software for future versions would be to increase the quality of the animations and other graphical artifacts. The initial release will have simplistic animations and user interface flourishes.

3.8 Specific Requirements

This project entails creating software to generate and animate different organic compounds. The user will also have the option of generating their own compounds. They will then be capable of entering in what they believe to be the name of the displayed or made organic molecule and check if their answer is correct.

1. Creating the backend.

    a. Having it create a data structure to represent the organic compound.
    b. Adding in the ability for the user to create their own organic compound.

c. Adding in the ability for the user to enter in what they believe to be the name of the organic compound and checking to see if the answer is correct.

2. Creating the animation.

a. Design and implement a new U.I. to handle the animation and all controls that go with it. This includes a button to re-watch the animation, a textbox to enter in the name of the compound as well as a button to check the answer, and buttons to generate another random compound or exit the program.
b. Creating the algorithm which will define the data structure for the compound and its various components.
c. Creating the design of the animation to mimic the exact organic compound that is stored.

3.9 External Interfaces

1. Creation of Randomly Generated Organic Bonded Molecule:

a. Name of item:

Randomly generated or user created organic compound.

b. Description of purpose:

An organic compound which the user will learn how to name.

c. Source of input or destination of output:

The application will randomly generate this compound or allow the user to construct their own.

d. Valid range, accuracy, and/or tolerance:

The input will be a correct version of an organic compound limited by the current functionality of the program.

e. Units of Measure:

The only units of measurement will be the atoms themselves.

f. Timing:

Does not apply.

g. Relationships to other input/outputs:

The animation will mimic the generated organic compound to show the naming process. The nomenclature displayed at the end of the animation will be calculated by a separate naming algorithm.

h. Screen formats/organization:

The program will have focus and be in the forefront of the window until the user clicks the animation button. A canvas section will then show the animation and upon completion will display the forming of the compound's nomenclature. After the animation is finished, the animation screen will disappear and the programs' main form will regain forefront attention.

i. Window formats/organization:

One window for the application with appropriate frames.

j. Data Formats:

The organic compound will have its own class and have the appropriate attributes needed for the different algorithms.

k. Command formats:

Does not apply.

l. End Messages:

None are needed. The user is brought back to a menu screen with options.

3.10 Functions

The system shall accept and check the input of the generated organic compound and ensure it is a valid structure. It will not accept further action until a valid compound is provided by the user.

The system shall accept and check the name for the currently displayed organic compound. The system will then check to see if the name is correct or incorrect and inform the user of the result. The system will allow the user to repeat this process as many times as they desire.

The system shall generate an animation, in which it will show an analogy of a mailman delivering mail. The mailman will perform the same steps as are taken in the process of naming an organic compound. Roads will be representative of the longest chain while houses will be representative of all substituents.

3.11 Performance Requirements

The program will support one user at a time per client.

3.12 Design Constraints

The system should be able to run the exported .exe file. In the event there is an issue, the system shall have Py2exe, a Python Interpreter, installed to run the file. The system shall also have Internet Explorer 7.0 or newer, Firefox 3.6 or newer or Google Chrome installed to download the Py2exe software if it is not currently installed on the machine. The rest of the constraints and limitations will be met due to the specifications required for windows 7/Mac OS/Linux/Unix on the machines.

3.12.1 Standards Compliance

The only standard applicable to this project is the IEEE standard. This project is not subject to FERPA or HIPPA.

a) Report format:

All reports and paperwork pertaining to this project will follow guidelines set by IEEE and other Software Engineering practices and standards.

b) Data naming

The names of all classes, methods, functions and variables will be modeled after standard Object Oriented Design principles. They will be concise and related to the actions performed or information stored. This will assist later development teams in adding functionality to the software to include other variations and types of organic compounds.

c) Audit tracing

Our Software Development group will be using Google Docs for documentation collaboration and git for code progress. Additionally, we will be utilizing GitHub, a project management and bug-tracking application. This will provide group members with an online interface for the groups' repository as well as collaboration tools such as tickets, a wiki, and a forum. These tools will assist the team in tracking member participation and progress.

3.13 Software System Attributes

3.13.1 Reliability

This program should be reliable and able to catch and throw errors with easily understood directions for resolving them. If the user enters in invalid information into the nomenclature guessing box, the

program shall tell the user if their input is correct. If the user decides to manually generate an organic compound, the program shall inform the user if their molecule is invalid.

3.13.2 Availability

This program will be accessible on all machines. If some machines have issues with the exported python file to an .exe file, then any machine that contains Py2exe software will be able to successfully run the program. The packaged executable will include all necessary libraries and dependencies. There is no persistent data so no database or network connection is necessary. For the purposes of demonstration, the application will be served as an applet on the internet. As such, a network connection will be necessary.

3.13.3 Security

a) Utilize certain cryptographic techniques

No cryptographic techniques need to be used for this program or subsequent later versions unless the requirements drastically change.

b) Keep specific log or historical data sets

No logs or historical data needs to be collected as the information generated and used by the program is disregarded after use or when new information is generated. No logs need to be created to keep track of users and time used on the program as per the current requirements.

c) Assign certain functions to different modules

Does not apply.

d) Restricted communications between some areas of the program.

No such restrictions exist or are needed in this program.

e) Check data integrity for critical variables.

If the user decides to create their own organic compound, the program will have to validate it. Once the integrity and validity of the compound are ensured, all other functionality can be expected to operate without error.

3.13.4 Maintainability

The code will be broken up into well-developed methods and functions as well as be professionally commented. Any and all methods and functions will have comments describing their functionality and usability. This will allow future development teams to make additional features and/or changes to the program with relative ease and understanding of how the current version and previous versions function.

3.13.5 Portability

a) Percentage of components with host-dependent code.

The host of the program may require Py2exe software to be installed in order to run the program if the exported .exe file does not execute properly. Thus, there is no component with host-dependent code.

b) Percentage of code that is host dependent.

Zero percent of the code is host dependent.

c) Use of a proven portable language.

Since the application is written in Python 3, all major operating systems that contain a Python Interpreter, if the exported .exe file is nonfunctional, will be able to run the software.

d) Use of a particular compiler or language subset.

Because our team wanted this program to be easily portable to other machines, we chose Python 3 with PyQt4 library.

e) Use of a particular operating System.

The most used operating system will be Windows 7 as the computers in the school labs have it installed. Most students who will use this program generally have Windows 7 installed. The program will be able to run on any mainstream operating system as long as the exported .exe file is functional or Py2exe Python Interpreter software is installed.

3.14 Use Cases

Generating a Random Organic Molecule and Watching the Animation.

I. Input
   a. First button click.
   b. Second button click.

II. True Black Box Testing
   a. Description
      i. Open program.
      ii. Click 'Generate Random Compound' button.
      iii. Click 'View Animation' button.
      iv. Exit program.
   b. Expected Outcome
      i. Program will open.
      ii. A random organic compound up to and including pentadecane is generated.
      iii. The compound is passed to the animation generator and the animation plays according to the naming process.
      iv. Program exits.

III. False Black Box Testing
   a. Description
      i. Open program.
      ii. Click "Generate Random Compound' button.
   b. Expected Outcome
      i. Program will open.
      ii. Error message displays stating an error has occurred while trying to generate the organic compound.

IV. True White Box Testing (Same as black as all inputs are automated button clicks)

V. False White Box Testing
   a. Description
      i. Open program.
      ii. Click "View Animation" button
   b. Expected Outcome
      i. Program opens.
      ii. Error message is displayed stating that there is no current organic compound to base the animation from.

Generating a Random Organic Molecule, Entering in What is Believed to be the Name and Watching the Animation.

I. Input

a. First button click.
b. Name of organic molecule.
c. Second button click.
d. Third button click.

II. True Black Box Testing
    a. Description
        i. Open Program.
        ii. Click “Generate Random Compound’ button.
        iii. Enter in Supposed name of organic compound.
        iv. Click ‘Check Name’ button.
        v. Click ‘View Animation’ button.
        vi. Exit program.
    b. Expected Outcome
        i. Program opens.
        ii. Random organic molecule up to and including pentadecane is generated.
        iii. Program informs user their guess is correct.
        iv. Program displays correct animation for naming process.
        v. Program Exits.

III. False Black Box Testing
    a. Description
        i. User clicks ‘Generate Random Compound’ button.
        ii. User enters in guess for name of molecule.
        iii. User clicks ‘Check Name’ button.
    b. Expected Outcome
        i. Dialog box will appear informing the user their guess for the name is incorrect.

IV. True White Box Testing
    a. Description
        i. User clicks ‘Generate Random Compound’ button.
            1. Makes sure compound generated is up to and including pentadecane and correct format compared to pre gathered data.
        ii. User enters in guess for name of molecule.
        iii. User clicks ‘Check Name’ button.
            1. Makes sure the guess is correct compared to pre gathered data.
        iv. User clicks “View Animation” button.
            1. Makes sure animation follows correct paths according to scientific nomenclature.
        v. Exit program.
    b. Expected Outcome
        i. A correct random organic compound is generated.
        ii. User is informed their guess is correct.
        iii. Correct animation is displayed.
        iv. Program exits.

V. False White Box Testing
    a. Description
        i. User clicks “Generate Random Compound’ button.
            1. Makes sure compound generated is up to and including a pentadecane and correct format compared to pre gathered data.
    b. Expected Outcome
        i. The generated organic compound is not in the allowed compound list.

Custom Creating an Organic Compound and Viewing the Animation.

I. Input
    a. User goes through a series of mouse clicks in a grid.
    b. First button click.
    c. Second button click.

II. True Black Box Testing
    a. Description
        i. Open program.
        ii. User clicks in a series of boxes laid out in a grid.
        iii. User clicks ‘Validate Compound’ button.
        iv. User clicks ‘View Animation’ button.
        v. Exit program.
    b. Expected Outcome
        i. Program opens.
        ii. User forms an acceptable organic compound.
        iii. User is informed their custom compound is indeed correct.
        iv. User views animation for organic compound.
        v. Exits program.

III. False Black Box Testing
    a. Description
        i. Open program
        ii. User clicks in a series of boxes laid out in a grid.
        iii. User clicks ‘Validate Compound’ button.
    b. Expected Outcome
        i. Program opens.
        ii. A message box is displayed stating that the user created a compound not allowed by the current functionality of the program.

IV. True White Box Testing
    a. Description.
        i. Open program.
        ii. User clicks in a series of boxes laid out in a grid.

iii. User clicks 'Validate Compound' button.
    1. Checks pre gathered compound structures to ensure compound structure is indeed correct.
iv. User clicks 'View Animation' button.
    1. Checks pre gathered compound information to ensure animation follows the exact nomenclature procedure.
v. Exit program.

b. Expected Outcome
    i. Program opens.
    ii. User constructed a valid organic compound within the current functionality of the program.
    iii. User is informed their custom compound is correct.
    iv. User views animation of naming process.
    v. Program closes.

V. False White Box Testing
    a. Description
        i. Open Program.
        ii. User clicks in a series of boxes laid out in a grid.
        iii. User clicks 'Validate Compound' button.
    b. Expected Outcome
        i. Program opens.
        ii. Error message displays the currently constructed compound is not within current constraints.

Custom Creates Organic Compound, Guesses Name and Views Animation.

I. Input
    a. User goes through a series of mouse clicks in boxes laid out in a grid.
    b. First button click.
    c. User enters in guess of name for the compound.
    d. Second button click.
    e. Third button click.

II. True Black Box Testing
    a. Description
        i. Open program.
        ii. User clicks in a series of boxes laid out in a grid.
        iii. User clicks 'Validate Compound' button.
        iv. User enters in a guess for the name of the compound.
        v. User clicks 'Check Name' button.
        vi. User clicks 'View Animation' button.

vii. Closes program.

b. Expected Outcome.
   i. Program opens.
   ii. User constructs valid organic compound.
   iii. User is informed their compound is indeed valid.
   iv. User enters in the correct name for compound.
   v. User is informed their guess was correct.
   vi. User views animation for the compound.
   vii. Exits program.

III. False Black Box Testing
   a. Description
      i. Open program.
      ii. User clicks in a series of boxes laid out in a grid.
      iii. User clicks 'Validate Compound' button.
      iv. User enters in a guess for the name of the compound.
      v. User clicks 'Check Name' button.
   b. Expected Outcome.
      i. Program opens.
      ii. User creates a valid organic compound.
      iii. User enters in weird characters and numbers.
      iv. Error message is displayed informing the user their guess name contained invalid characters.

IV. True White Box Testing
   a. Description
      i. Open program.
      ii. User clicks in a series of boxes laid out in a grid.
      iii. User clicks 'Validate Compound' button.
         1. Checks pre gathered data to ensure the custom organic compound is indeed allowed in current functionality and is indeed a compound.
      iv. User enters in a guess for the name of the compound.
      v. User clicks 'Check Name' button.
         1. Checks pre gathered data to ensure the name of the organic compound and format is indeed correct.
      vi. User clicks 'View Animation' button.
      vii. Exits program.
   b. Expected Outcome
      i. Program opens.
      ii. User indeed creates an acceptable organic compound.
      iii. User is informed their custom compound is acceptable.
      iv. User enters in a guess for the name of the compound.
      v. User is informed their guess is indeed correct.
      vi. User views animation on naming procedure of the compound.

vii. Exits program.

V. False White Box Testing

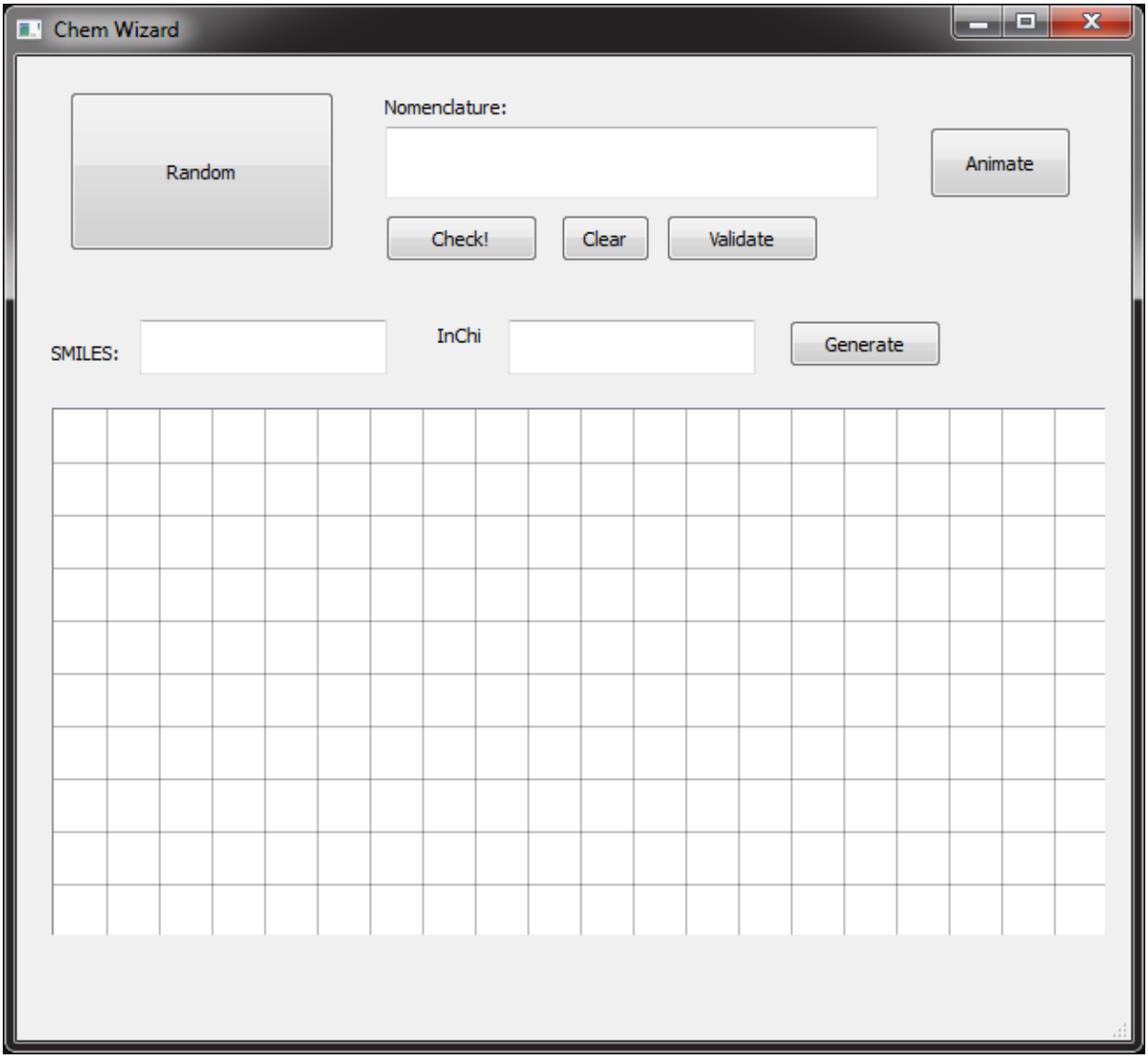
a. Description

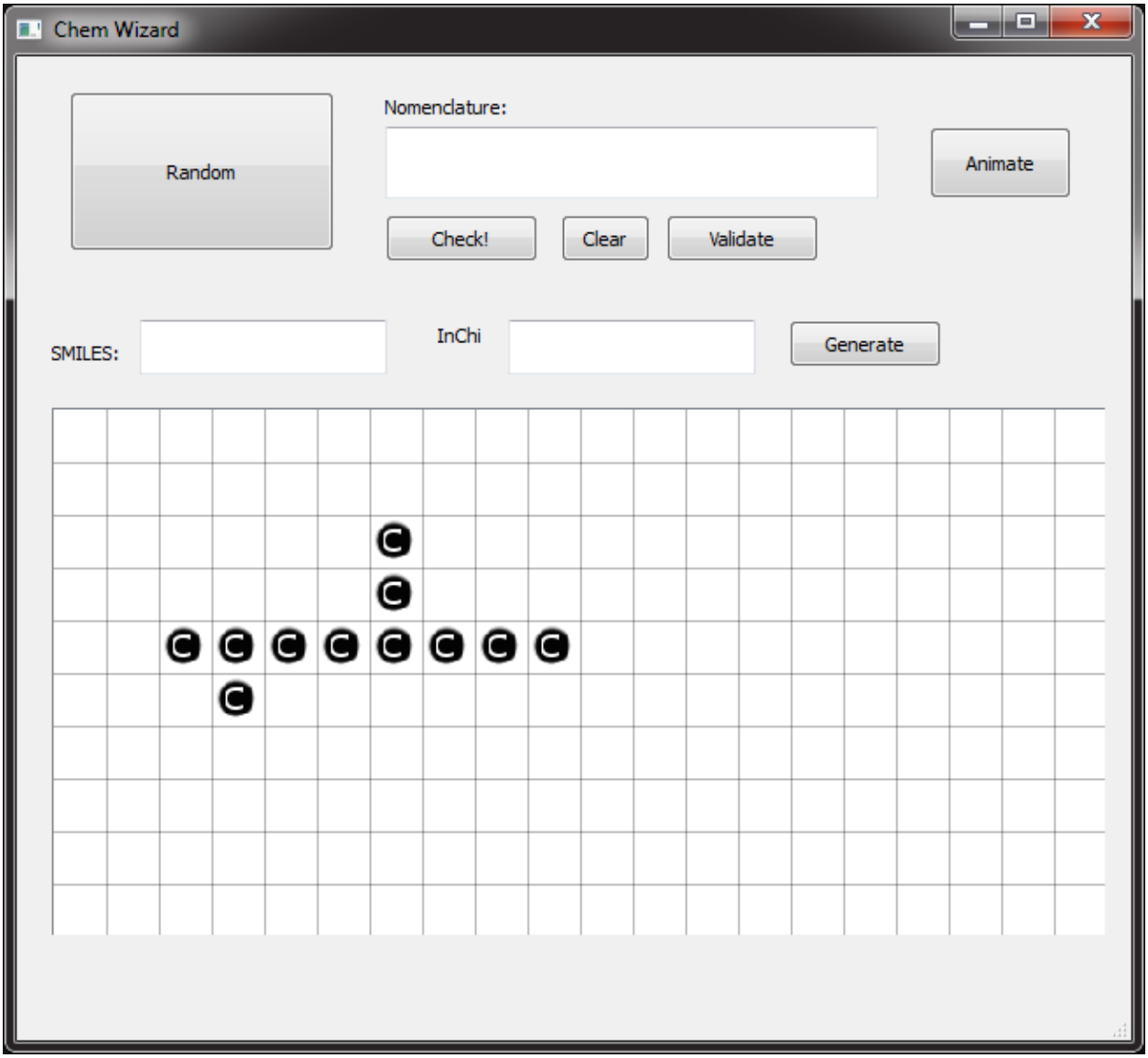
i. Open program.
ii. User clicks in a series of boxes laid out in a grid.
iii. User clicks ‘Validate Compound’ button.
iv. User enters in guess for the name of the compound.
v. User clicks ‘Check Name’ button.

b. Expected Outcome

i. Program opens.
ii. User creates a valid organic compound within the functionality of the program (pentadecane).
iii. User is informed their custom compound is indeed acceptable.
iv. User enters in guess for the name of the compound.

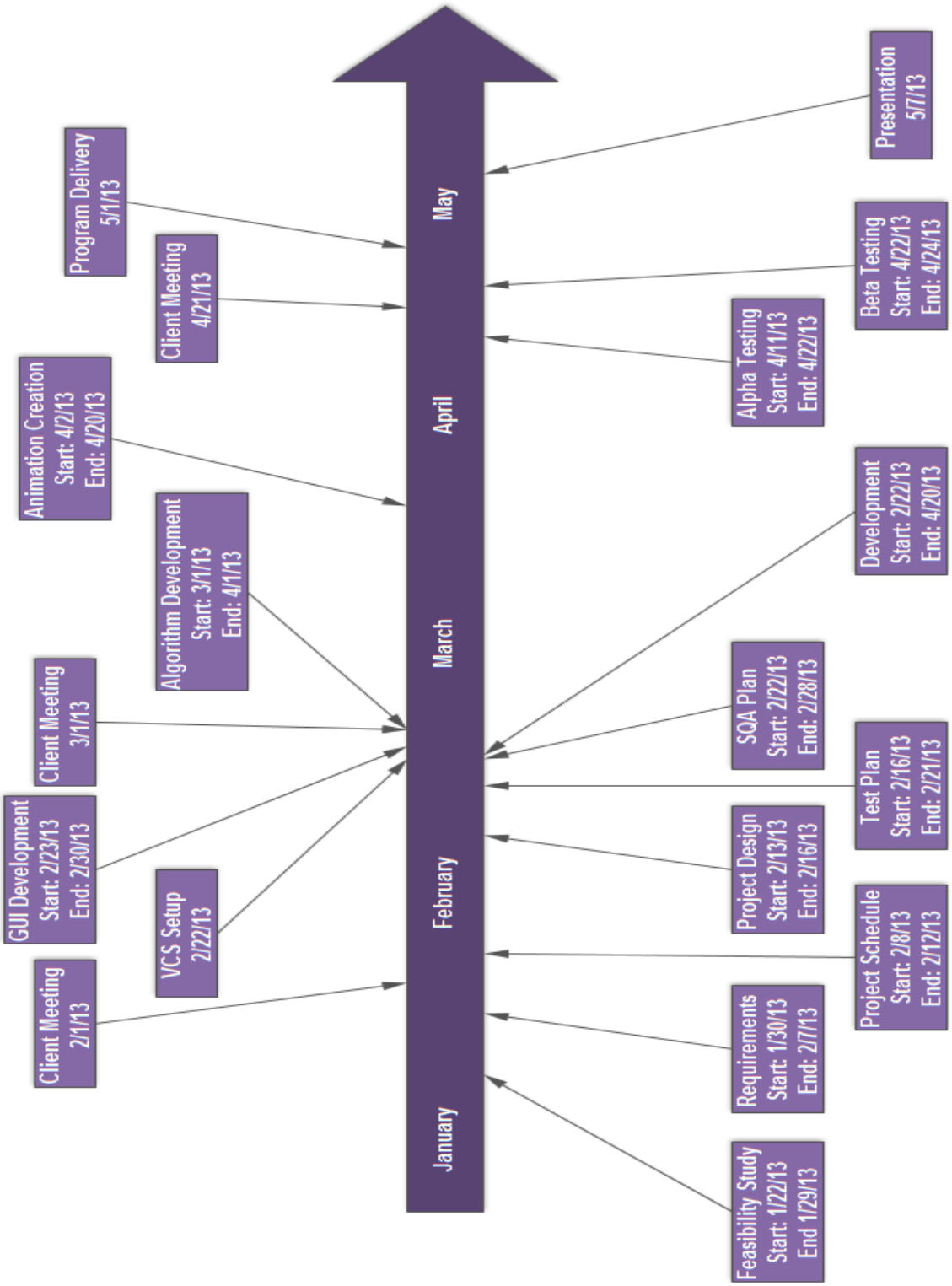
3.15 Prototypes



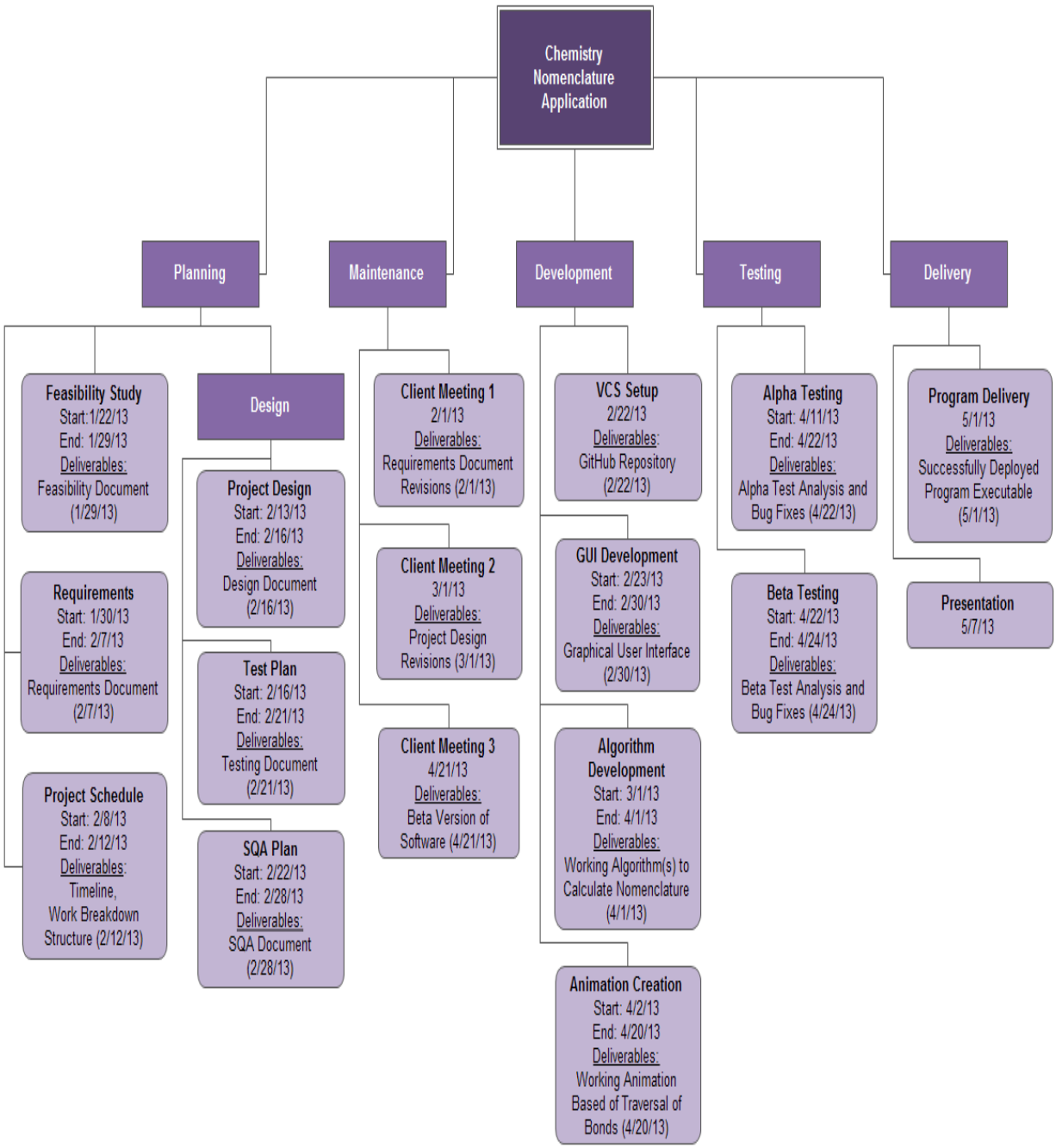

Chem Wizard
Random
Nomenclature:
Animate
Check!
Clear
Validate
SMILES:
InChi
Generate

4. Project Schedule

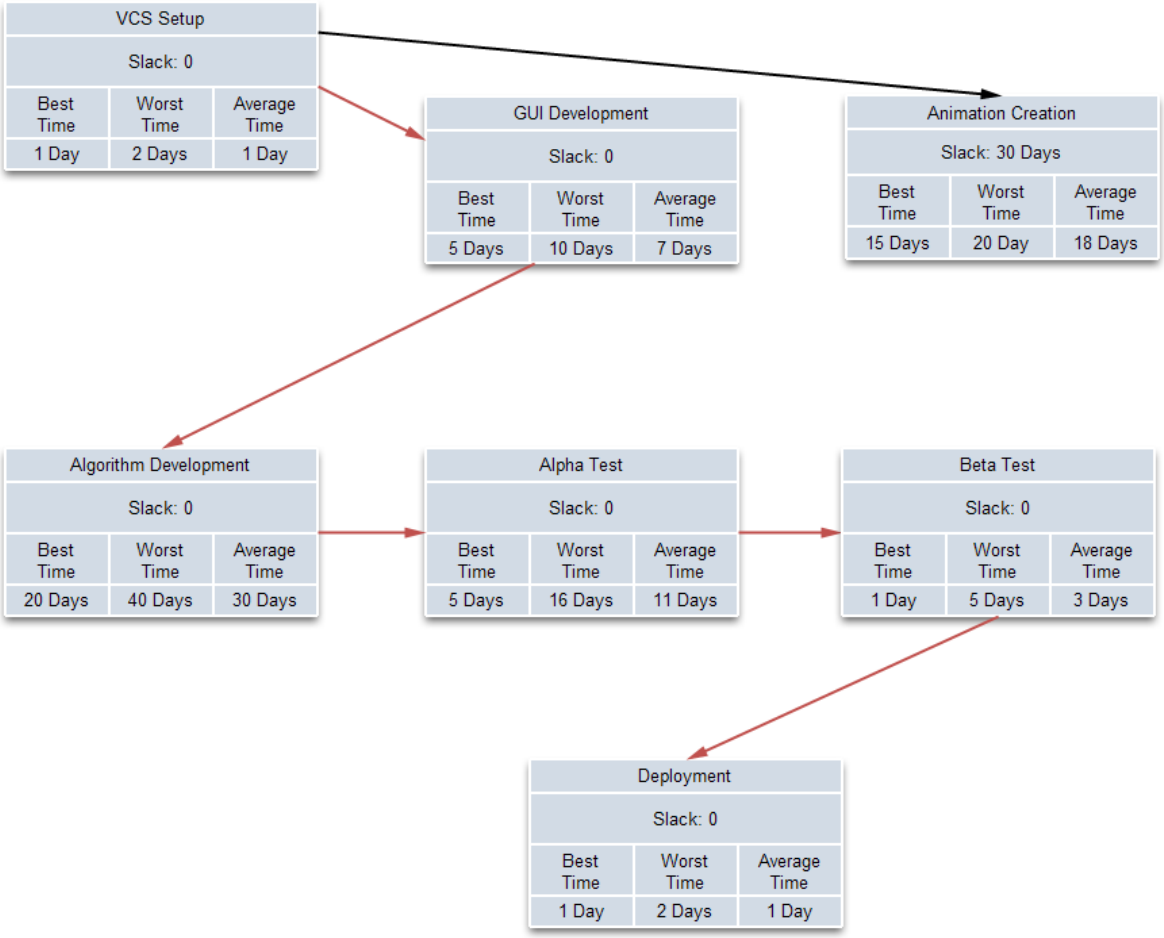
4.1 Timeline



4.2 Work Breakdown Structure



4.3 PERT Diagram



4.4 Lines of Code Estimation

Code Section	Min	Best	Max
GUI	200	400	800
Algorithms	500	800	1500
Animations	400	500	600
Total	1100	1700	2900

Code Estimate = (Min + 4*Best + Max) / 6 = (1100 + 4*1700 + 2900) / 6 = 1800 Lines of Code

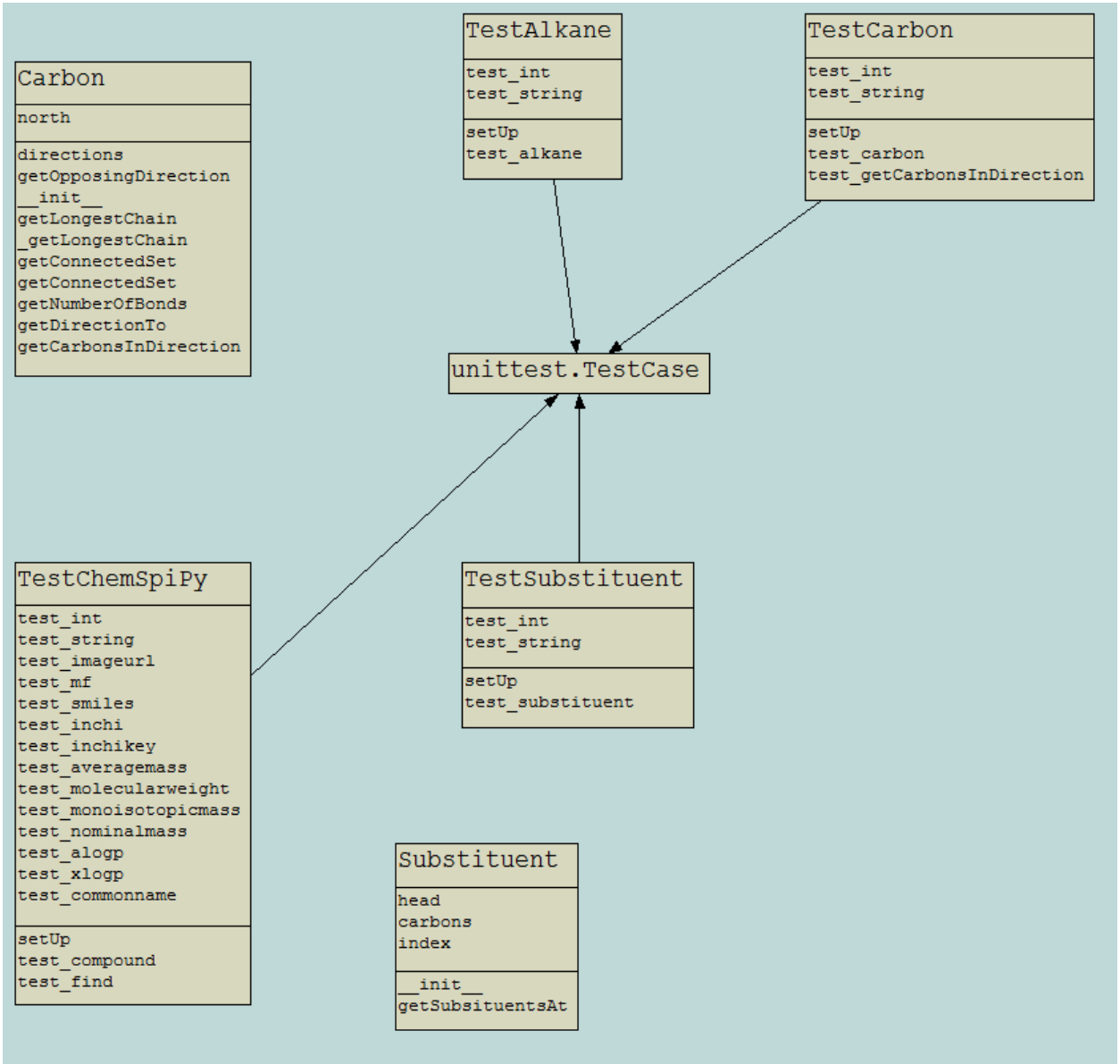
4.5 Revisions

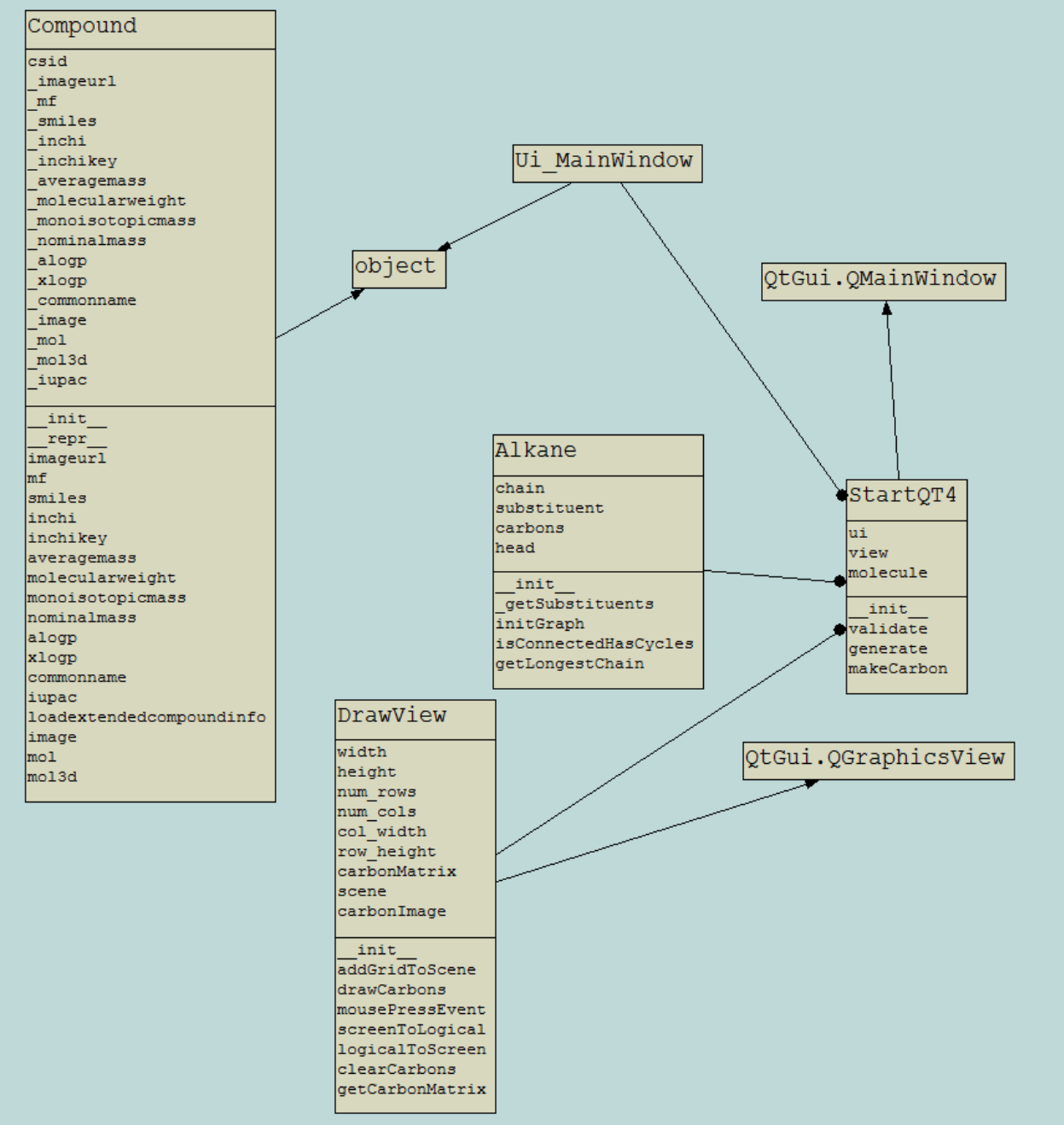
Revision Number:	Revision Date:	Author:	Summary of Changes:
1	3-11-13	Chris Lansing	Created Timeline
2	3-13-13	Chris Lansing	Created WBS

5. Design Documents

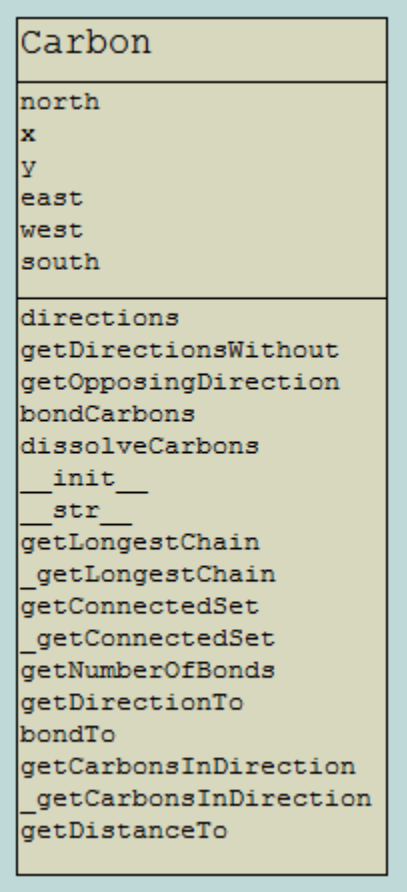
5.1 UML Diagrams

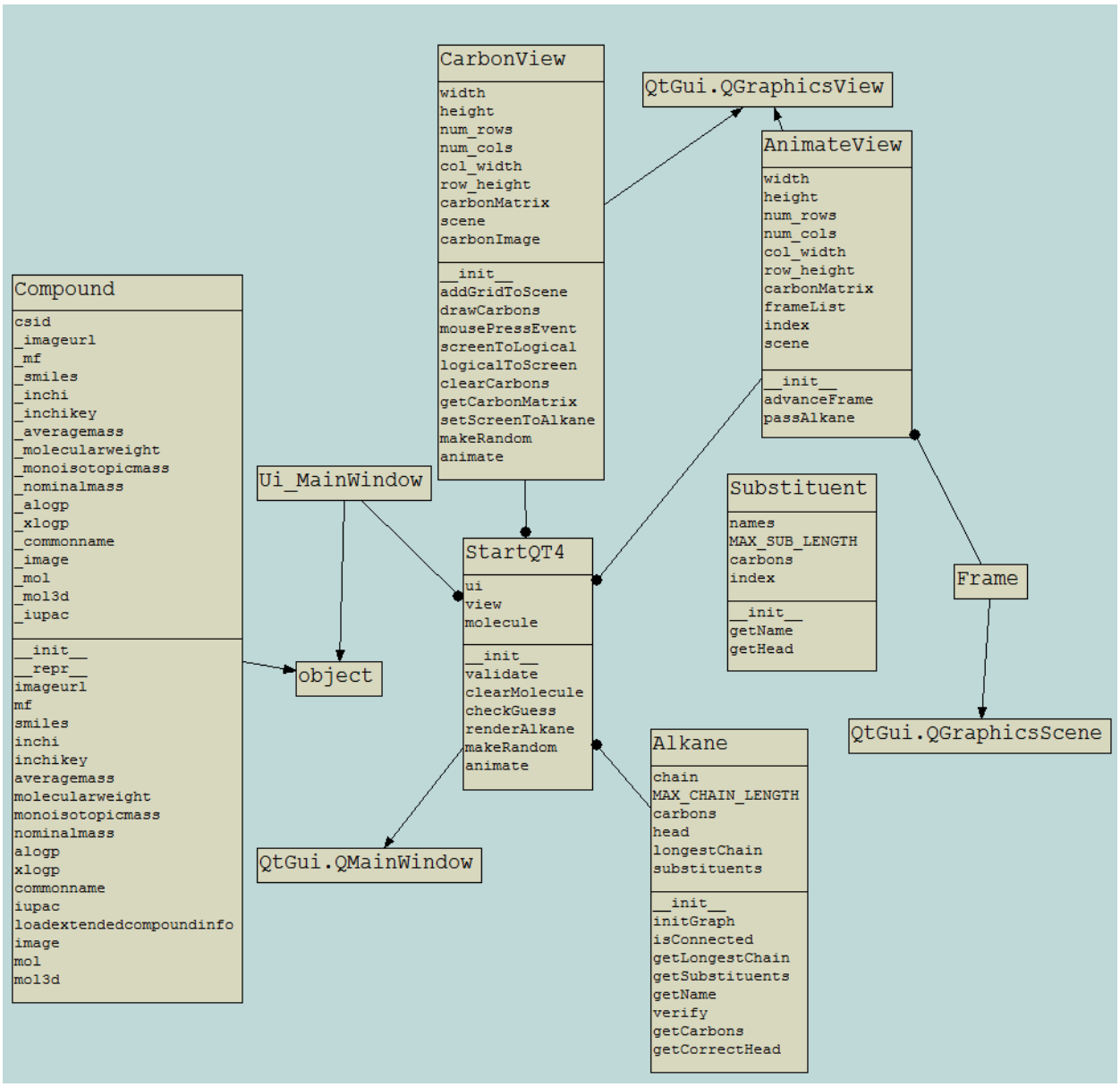
Original UML

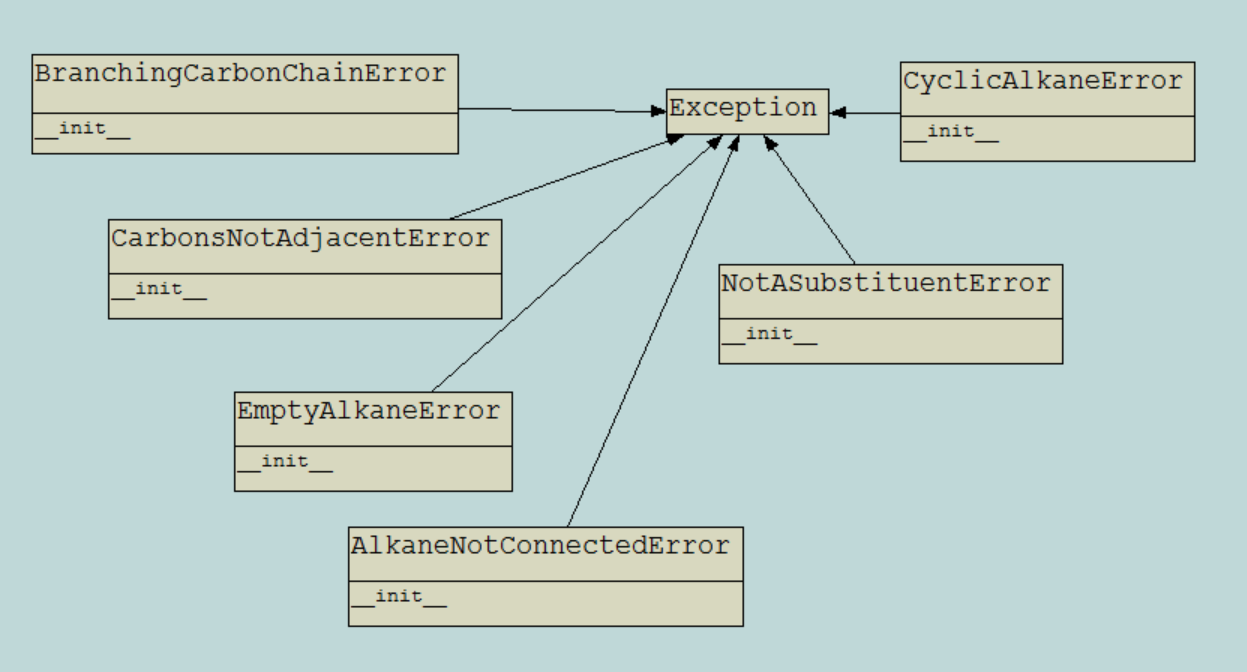


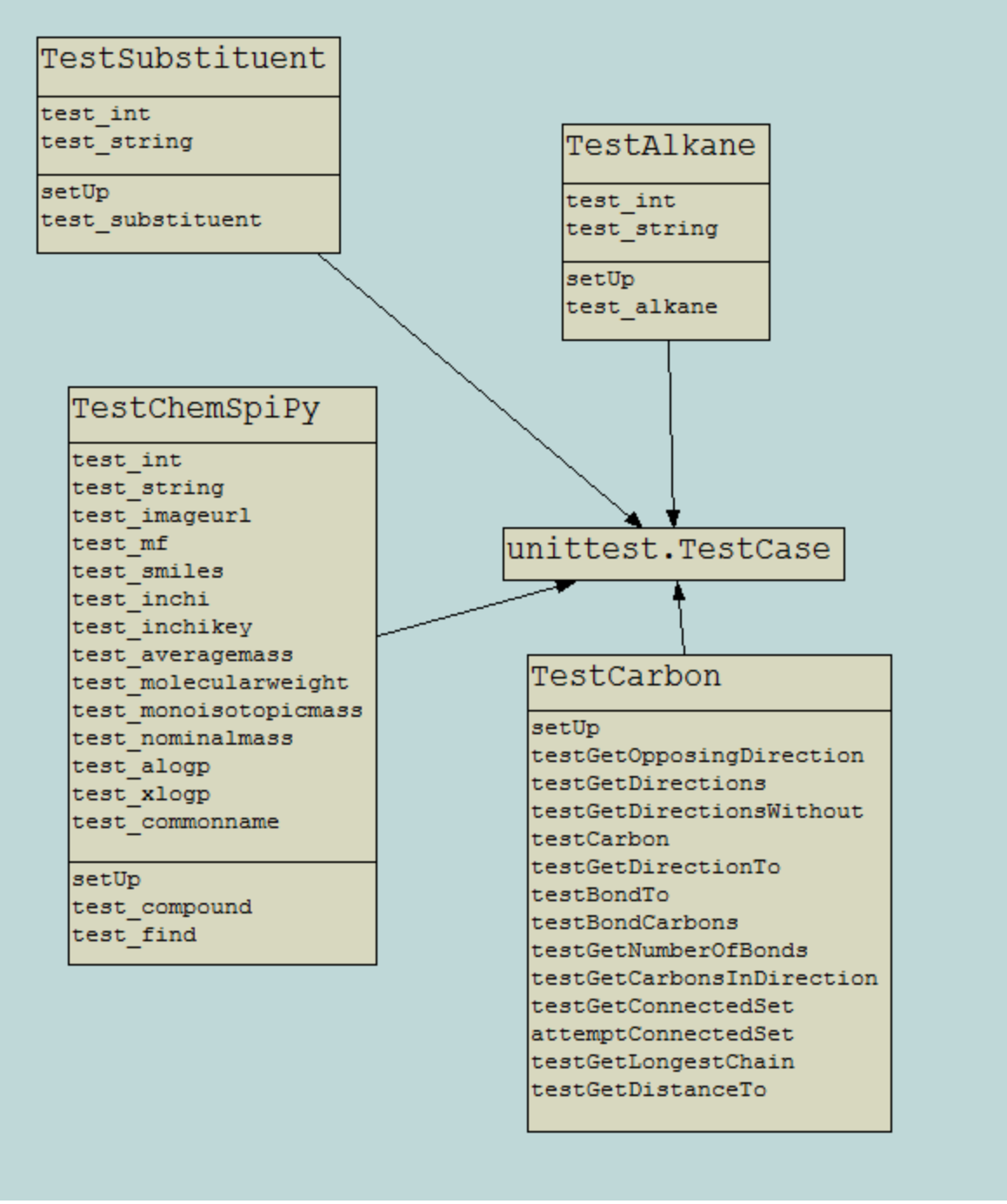

Compound
csid
_imageurl
_mf
_smiles
_inchi
_inchikey
_averagemass
_molecularweight
_monoisotopicmass
_nominalmass
_alogp
_xlogp
_commonname
_image
_mol
_mol3d
_iupac
__init__
__repr__
imageurl
mf
smiles
inchi
inchikey
averagemass
molecularweight
monoisotopicmass
nominalmass
alogp
xlogp
commonname
iupac
loadextendedcompoundinfo
image
mol
mol3d
Ui_MainWindow
object
QtGui.QMainWindow
Alkane
chain
substituent
carbons
head
__init__
_getSubstituents
initGraph
isConnectedHasCycles
getLongestChain
StartQT4
ui
view
molecule
__init__
validate
generate
makeCarbon
DrawView
width
height
num_rows
num_cols
col_width
row_height
carbonMatrix
scene
carbonImage
__init__
addGridToScene
drawCarbons
mousePressEvent
screenToLogical
logicalToScreen
clearCarbons
getCarbonMatrix
QtGui.QGraphicsView

Updated UML

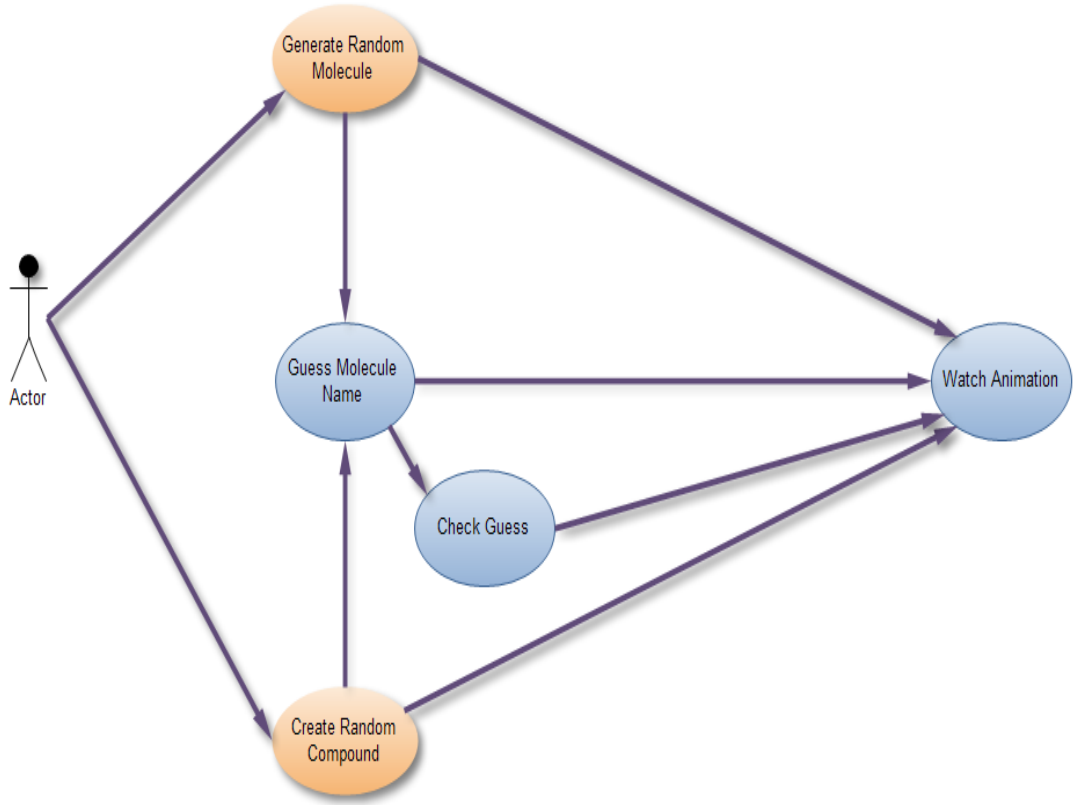

Carbon
north
x
y
east
west
south
directions
getDirectionsWithout
getOpposingDirection
bondCarbons
dissolveCarbons
__init__
__str__
getLongestChain
_getLongestChain
getConnectedSet
_getConnectedSet
getNumberOfBonds
getDirectionTo
bondTo
getCarbonsInDirection
_getCarbonsInDirection
getDistanceTo


CarbonView
width
height
num_rows
num_cols
col_width
row_height
carbonMatrix
scene
carbonImage
__init__
addGridToScene
drawCarbons
mousePressEvent
screenToLogical
logicalToScreen
clearCarbons
getCarbonMatrix
setScreenToAlkane
makeRandom
animate
QtGui.QGraphicsView
AnimateView
width
height
num_rows
num_cols
col_width
row_height
carbonMatrix
frameList
index
scene
__init__
advanceFrame
passAlkane
Compound
csid
_imageurl
_mf
_smiles
_inchi
_inchikey
_averagemass
_molecularweight
_monoisotopicmass
_nominalmass
_alogp
_xlogp
_commonname
_image
_mol
_mol3d
_iupac
__init__
__repr__
imageurl
mf
smiles
inchi
inchikey
averagemass
molecularweight
monoisotopicmass
nominalmass
alogp
xlogp
commonname
iupac
loadextendedcompoundinfo
image
mol
mol3d
Ui_MainWindow
object
StartQT4
ui
view
molecule
__init__
validate
clearMolecule
checkGuess
renderAlkane
makeRandom
animate
Substituent
names
MAX_SUB_LENGTH
carbons
index
__init__
getName
getHead
Frame
QtGui.QGraphicsScene
Alkane
chain
MAX_CHAIN_LENGTH
carbons
head
longestChain
substituents
__init__
initGraph
isConnected
getLongestChain
getSubstituents
getName
verify
getCarbons
getCorrectHead
QtGui.QMainWindow

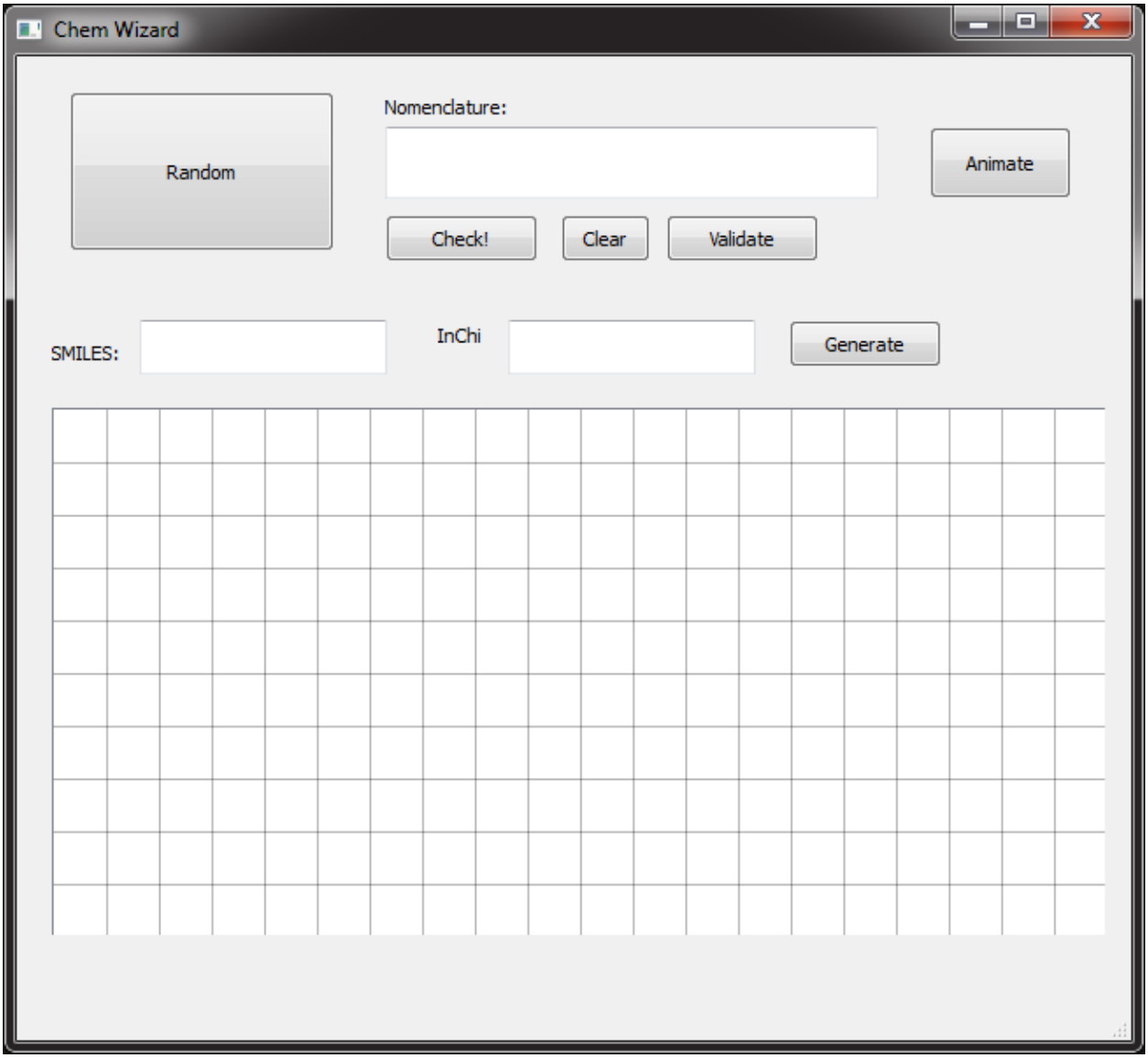

BranchingCarbonChainError
__init__
Exception
CyclicAlkaneError
__init__
CarbonsNotAdjacentError
__init__
NotASubstituentError
__init__
EmptyAlkaneError
__init__
AlkaneNotConnectedError
__init__

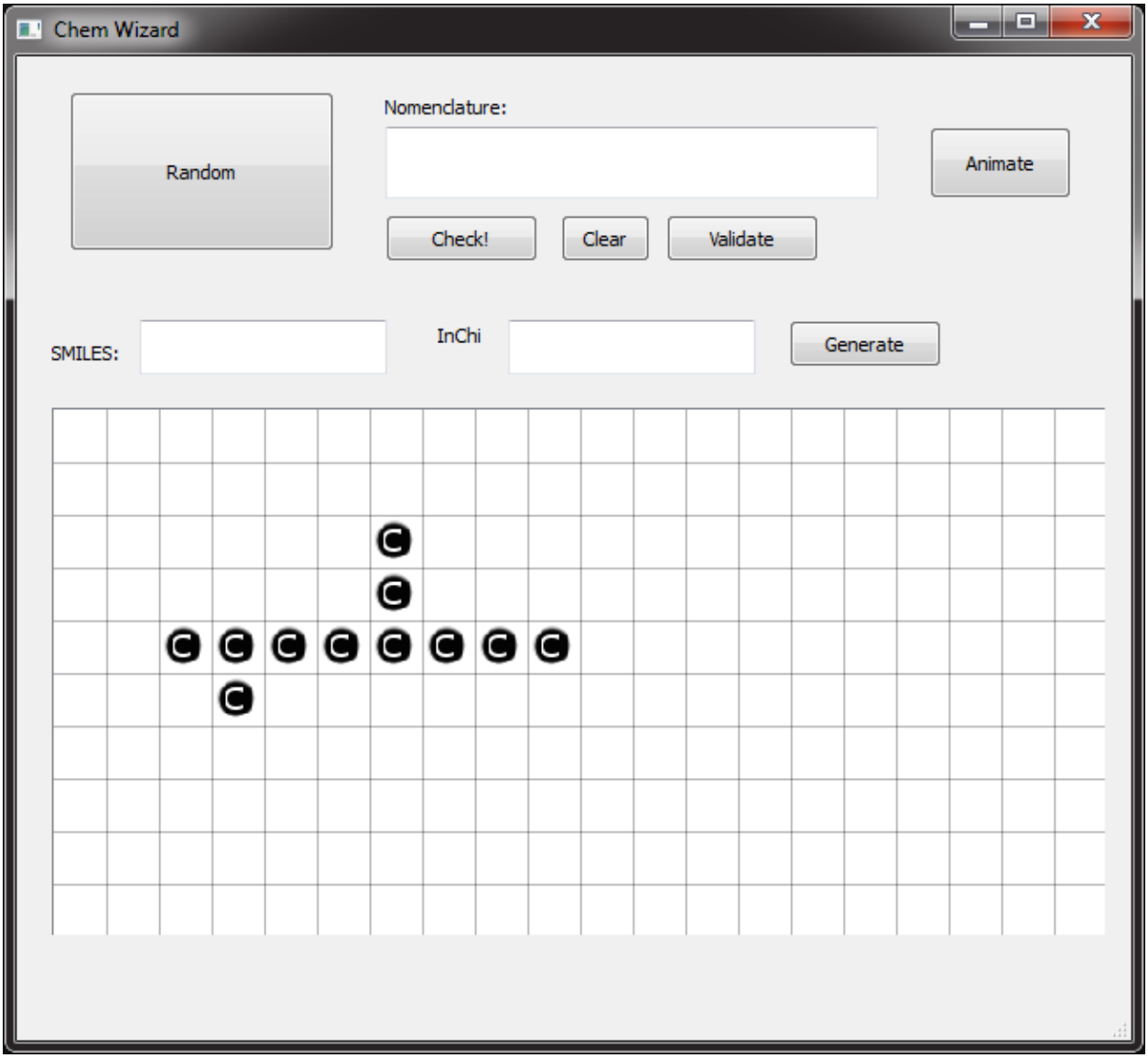

TestSubstituent
test_int
test_string
setUp
test_substituent
TestAlkane
test_int
test_string
setUp
test_alkane
TestChemSpiPy
test_int
test_string
test_imageurl
test_mf
test_smiles
test_inchi
test_inchikey
test_averagemass
test_molecularweight
test_monoisotopicmass
test_nominalmass
test_alogp
test_xlogp
test_commonname
setUp
test_compound
test_find
unittest.TestCase
TestCarbon
setUp
testGetOpposingDirection
testGetDirections
testGetDirectionsWithout
testCarbon
testGetDirectionTo
testBondTo
testBondCarbons
testGetNumberOfBonds
testGetCarbonsInDirection
testGetConnectedSet
attemptConnectedSet
testGetLongestChain
testGetDistanceTo

5.2 State Diagram



5.3 Prototype




Chem Wizard
Random
Nomenclature:
Animate
Check!
Clear
Validate
SMILES:
InChi
Generate

5.4 Revisions

Revision Number:	Revision Date:	Author:	Summary of Changes:
1	3-20-13	Chris Lansing	Created UML Diagrams
2	3-22-13	Chris Lansing	Created Use Case Diagram

6. SQA Plan

6.1 Requirements

6.1.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Documentation Standards:

- Were the documents prepared in accordance with Object Oriented Design principles?
- Is there a cover sheet and table of contents?
- Is there revision control?

Feasibility:

- Have the client been met with and proper contact information exchanged?
- Have a domain analysis been done?
- Have features from similar software been identified?
- Has an appropriate development cycle been decided upon?
- Are proposed milestones realistic?
- Has a list of deliverables been made?
- Has the customer signed off on the list of deliverables and it satisfied with them?
- Is the list of deliverables reasonable for the time constraints?

SRS:

- Has all requirements in the SRS requirements document been fully documented?
- Has the customer signed off on all requirements in the SRS document?
- Are all requirements properly incorporated into the program design?
- Were there any models submitted with the requirements from the customer?
- Were those models complete and relatable?
- Do all requirements function according to the customers' models?
- Does the domain analysis show other programs with these requirements?

- Is the work breakdown structure and timeline reasonable?
- Are there any risks with the requirements being changed?
- Are there any risks with the deliverables being changed?
- Was research done into the machines this program would be running on?
- Was research done on the operating systems this program will be running on?
- Was research done on the minimum requirements needed to run this program?
- Was research done in I/O devices needed for this program to function properly with?
- Was research done on what is needed to upgrade the software/hardware on the machines if there were unsatisfactory?
- Does the program run properly on systems containing the minimum set requirements?
- Was storing results of the program address?
- Was security of the data generated by the program addressed?

Management:

- Is there a plan in place to ensure requirements are developed to the customers' requirements?
- Is there a plan in place to handle changes in requirements/deliverables during the project?
- Can the development team change requirements or deliverables without consulting the client?

Support Material:

- Are all use cases accounted for and adequately represent the requirements?
- Does each use case have at least one use case scenario?
- Does the code in Python 3 generate the animation as intended?
- Does the UML clearly show the relationships between each class?
- Has the general user population been determined?
- Has the design of the GUI and features taken this into account?
- Has any and all text seen by the user been written in terms the user will easily understand?
- Is the system reliable?
- Does the program meet the requirements set by the client?

6.2 Design

6.2.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Support Material:

The following material should be available to person(s) completing this check.

- A full and completed UML diagram should be provided.
- All state diagrams should be properly designed as well as logically flow and should be provided.
- All main uses cases involving this program's requirements should be provided.
- All use cases must contain at least one use case scenario that should be provided.
- All documents must fully reflect all the customer's requirements.
- The clients signature of approved must be provided showing approval of design.
- All documents should adhere to Object Oriented Design principles.
- Any change or update requests for documentation should be provided.
- All classes, interfaces, and objects should be provided in the UML documentation.

Inspection Section:

- Have all classes been named appropriately for the code segments and processes they contain?
- Are all class relations properly connected?
- Have the functions been named appropriately for the processes they will be committing?
- Have the methods been named appropriately for the processes they will be committing?
- Have the variables been named appropriately for the information they are storing?
- Have professional procedures been followed?
- Are all the relationships and number of possible instances between entities correct?
- Has any core functionality been left out that was outlined in the requirements documentation?

Variables:

- Are there any redundant functions, methods, or variables?

- Are any variables hard-coded?
- If hard-coded variables exist, if they are changed will they deter the program from functioning properly?
- Has any and all variables and arrays been properly set up and instantiated?
- Are all method return types properly type casted?
- Are all Data types properly typed and/or casted?

Structure:

- Will changes to the class diagrams alter the functionality of the program?
- Will changes to data types within the diagrams cause problems elsewhere?
- Will changes in method parameters cause problems elsewhere?
- Will any changes possibly result in pointer exceptions?

Finalization:

- Does the design contain appropriate error catching methods?
- Do the error catching methods provide adequate information to the user on how to resolve the error?
- Is the design modular enough such that further requirements and design changes are possible?
- Is the design realistic enough such that the development team can complete the tasks in the allotted timeframe?
- Does the design take into account all constraints in which may hinder the functionality of the program?
- Does the design ensure optimized usage and flow for the user?
- Does the design logically flow from one process/step to the next?
- Does the design and layout of the animations match the requirements?
- Are all options for the user to perform from the SRS document accounted for?

6.3 Implementation

6.3.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Documentation:

- Have all standards and guidelines been identified?
- Were any changes made to the implementation document?
- Have any rules been added for the initial implementation document?
- Is there a cover sheet and table of contents?
- Is there revision control?
- Has the implementation strategy been approved?
- Have all requirements been confirmed with the client?
- Has the design documentation been approved?
- Has the design checklist been performed?
- Does the design solution meet the requirements?
- Is the set of deficiencies provided?
- Was this checklist conducted on schedule?
- Were all needed resources available and properly and fully perform this checklist at time of completion?

Schedule:

- Has the workload been appropriately divided upon members in the development team?
- Has the work breakdown structure been established?
- Has the work breakdown structure been approved?
- Has proper deadlines been set for the coding phase?
- Were those deadlines met within reason?

Support Material:

- Was all source code provided?

- Was all documentation provided?
- Was analysis report provided?
- Is the analysis report in the correct format?

Structure:

- Is the user interface user friendly?
- Is the coding style consistent?
- Has professional comments been provided?
- Is there code that could be condensed?
- Is there code that could be re-written for higher optimization?
- Is there repetitive code?
- Does the code properly implement all requirements from the analysis report?
- File Opening/Closing: Will the proposed requirements cause additional files to be opened and/or closed from the previous version?
- Data Values: Will proposed changes to data values within the scope of the changes cause problems elsewhere?
- Parameter Change: Will proposed changes affect the subroutines?
- File Position change: Will proposed changes affect the files sensitive to record positioning?
- Invalid Pointer: Will proposed changes affect existing linked data structures, possible causing pointer exceptions?
- Record Layout Change: Will proposed changes affect record layouts?
- Is storage use efficient?
- Is the code consistent in style?

Variables:

- Do all classes follow proper naming conventions?
- Do all methods follow proper naming conventions?
- Do all functions follow proper naming conventions?
- Do all variables follow proper naming conventions?
- Are there any unused variables?
- Are variables of the correct types?
- Are 'Accessors' and 'Mutators' named properly and similarly?
- Are all array references in bounds?

Loops and Branches:

- Do loops properly initialize and increment/decrement the variables properly?
- Are loops too nested?
- Are bested loops properly nested?
- Are all cases covered in IF/ELSEIF/ELSE statements or CASE blocks?
- Does every switch statement have a default?
- Are loop termination conditions achievable?
- Does the code in the loop avoid manipulating the index variable or using it upon termination of the loop?
- Are case statements properly set up and contain breaks?
- Does the default case give an error if not reached?

Defensive Programming:

- Do processes occur in the right order throughout the program?
- Has unit testing been done to ensure functions and methods provide the correct solutions?
- Has the functions/methods been made modular such that further expansions are easier?
- Are there any unused functions or methods?
- Do methods/functions contain appropriate error catching capabilities?
- Are the error messages direct and easy to understand?
- Does the correct form get forefront attention?
- Is the correct data being operated upon in each statement?

Additional:

- Does the code properly send the constructed molecule structure to Python 3to generate the design?
- Does the code ensure that if the user makes their own version that it is indeed only up to a pentadecane molecule?
- Does Python 3properly generate the map?
- Does the animation follow the process in exact accordance to the customers' models?
- After the user watches the animation and wishes to go back to the first window, does the animation window close properly?
- If the user chooses to enter in the name of the constructed molecule, does the program appropriately inform the user if they are correct or incorrect?

- If the user chooses to enter in the name of the constructed molecule and does so incorrectly, does the program appropriately inform them of their mistake and how to fix it?
- Is there code that will inform the user when additional updates or requirements have been added?
- Is there code that will allow the user to leave messages for the software engineers/developers?

6.4 Testing

6.4.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Requirements Testing:

- Have all requirements from the SRS requirements document been satisfied?
- Do all the requirements in the SRS document fully meet the customers' requirements?
- Does the code properly and fully implement those requirements?
- Can the requirements' code be tested?
- Are the requirements clear and concise?
- Has the customer signed off on all requirements for verification?
- Does the program execute each step of the naming process and animation process according to the customers' models?
- Does the software meet security requirements?
- Does the software meet privacy requirements?
- Does the documentation have a description of higher architecture?

Functional Design:

- Does the design allow the user to perform any and all tasks promised?
- Does the design incorporate future software updates and expansions?

- Does the software allow for the user to create their own pentadecane molecule?
- Does the software allow for the user to enter in what they believe to be the name for the pentadecane molecule?
- Does the software allow for some actions to be performed multiple times?
- Does the design handle well under stress?
- Does the software inform the user when a mistake has been made and how to resolve it/
- Does the design give the user a simple interface to work with?
- Does the design allow the user to backtrack to previous forms?
- Does the design allow testers to add comments?
- Does the design allow the users to add comments?
- Does the design address techniques and tools that shall be used to assure software quality assurance?
- Does the design catch all exceptions correctly?

Testing Techniques:

- Does the software pass Black Box testing?
- Does the software pass White Box testing?
- Does the software pass Unit testing?
- Does the software pass Integration testing?
- Does the software pass System testing?
- Does the software pass Alpha testing?
- Does the software pass Beta testing?
- Does the software pass Acceptance testing?

Environmental Testing:

- Does the program run correctly on all operating systems specified in the SRS document?
- Does the program work correctly with all drivers?
- Does the program work smoothly in these environments? (not choppy)
- Does any aspect of the program crash or freeze during any step in these different environments?
- Is any functionality hindered by running in any of the operating systems specified in the SRS document?

Acceptance Testing:

- Are all previous issues and bugs fixed?
- Is the client satisfied with the finished project?
- Is the finished project ready for release?

6.5 Installation

6.5.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Support:

- In case of minimum software not installed, Internet Explorer 7.0 or higher, Firefox 3.6 or higher or Google Chrome should be installed.
- Windows Systems:
- Ram: 128mb (64mb for Windows XP 32 bit)
- Disc space 128mb.
- Mac OS X:
- Mac OS X 10.7.3 (Lion) or later
- Linux:
- Oracle Linux 5.5+
- Oracle Linux 6.x (32 bit)
- Oracle Linux 6.x (64 bit)
- Red Hat Enterprise Linux 5.5+
- Oracle Linux 6.x (32 bit)
- Oracle Linux 6.x (64 bit)
- Ubuntu Linux 10.04 and above
- Suse Linux Enterprise Server 10 sp2

- Ram 64mb
- Disc space 58 MB

Software Requirements:

- Does the system have Py2exe installed?

Additional:

- Is the program easy to install?
- Is the install process fully documented in the manual?
- Does the installation manual contain useful screen shots?
- Is the installation manual easy to use and navigate?
- Are trouble shooting options addresses as best as possible?
- Was this assessment done when scheduled?

Documentation:

- Is there a cover page and table of contents?
- Is there revision control?

6.6 Maintenance

6.6.1 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-17-2013	John Gibbons	Initial creation of document and draft.
2	4-19-2013	John Gibbons	Second revision of draft and added additional items.
3	5-2-2013	John Gibbons	Final revision.

Documentation:

- Is there a generated error report or a properly filed support ticket?
- Is there a customer support solutions manual provided?
- Is there an installation manual provided?
- Is there a user manual provided?
- Jas the support ticket been filled out properly?
- Has the issue been properly stated?
- Did the user provide evidence with the support ticket?

- Was the date and version number of the software properly documented on the support ticket?
- Do the errors generate error codes?

Resolution:

- Was the error able to be regenerated?
- Was the error related to the program or the users' machine?
- Was the maintenance done in a timely manner?
- Was the customer/user satisfied with the maintenance?

Future Additions:

- Will intended future additions affect the current functionality of the program?
- Will intended future additions add to the current functionality of the program?
- Does the program include ways to inform the users of new features and additions?
- Has the client fully approved of any additions?
- Is the code modular and well commented enough to easily modify?

Further Contact:

- Does the client have the ability to reach the development team should any issue arise?

6.7 Checklists

6.7.1 Requirements

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial creation of document and first draft.
2	3-14-2013	John Gibbons	Second draft and added new items.
3	4-27-2013	John Gibbons	Completed checklist.
4	4-30-2013	John Gibbons	Revision of checklist.
5	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments
	Documentation Standards		
1	Were the documents prepared in accordance with Object Oriented	Y	

	Design principles?		
2	Is there a cover sheet and table of contents?	Y	
3	Is there revision control?	Y	
	Feasibility	Y, N, NA	Comments
1	Has the client been met with and proper contact information exchanged?	Y	Roland, team leader, has been maintaining constant contact with the client.
2	Has a domain analysis been done?	Y	
3	Has features from similar software been identified?	Y	However, none including animations of the naming procedure.
4	Has an appropriate development cycle been decided upon?	Y	Yes but was never used nor has the first cycle been completed.
5	Are proposed milestones realistic?	Y	Was not created until 4-27-2013. The group never met in person.
6	Has a list of deliverables been made?	Y	
7	Has the customer signed off on the list of deliverables and is satisfied with them?	Y	
8	Is the list of deliverables reasonable for the time constraints?	Y	
	SRS	Y, N, NA	Comments
1	Has all requirements in the SRS requirements document been fully documented?	Y	Upgraded from decane (10 strand longest chain) to pentadecane (15 strand longest chain)
2	Has the customer signed off on all requirements in the SRS document?	Y	
3	Are all requirements properly incorporated into the program design?	Y	
4	Were there any models submitted with the requirements from the customer?	Y	
5	Were those models complete and relatable?	Y	These models are the basis for the design and flow of the animations.
6	Do all requirements function according to the customers' models?	Y	
7	Does the domain analysis show other programs with these	N	Not pertaining to animation, however other organic compound builders that display the

	requirements?		names have been found.
8	Is the work breakdown structure and timeline reasonable?	Y	
9	Are there any risks with the requirements being changed?	Y	A few changes were made. However, they were changes that could be implemented within the given timeframe.
10	Are there any risks with the deliverables being changed?	Y	
11	Was research done into the machines this program would run on?	Y	
12	Was research done on the operating systems this program will run on?	Y	Windows 7, Mac OS and Linux.
13	Was research done on the minimum requirements needed to run this program?	Y	Listed in the requirements documents.
14	Was research done on I/O devices needed for this program to function properly with?	Y	Standard mouse, keyboard and monitor.
15	Was research done on what is needed to upgrade the software/hardware on the machines if they were unsatisfactory?	Y	
16	Does the program run properly on systems containing the minimum set requirements?	NA	4-30-13 have yet to run alpha tests.
17	Was storing results of the program addressed?	Y	No information generated by the program is stored.
18	Was security of the data generated by the program addressed?	Y	No security is needed for this program as per current requirements.
	Management	Y, N, NA	Comments
1	Is there a plan in place to ensure requirements are developed to the customers' requirements?	Y	
2	Is there a plan in place to handle changes in requirements/deliverables during the project?	Y	
3	Can the development team change requirements or deliverables without consulting the client?	N	All changes have to be pre-approved by the client before implemented.

	Support Material	Y, N, NA	Comments
1	Are all use cases accounted for and adequately represent the requirements?	Y	
2	Does each use case have at least one use case scenario?	Y	
3	Does the code in Python 3 generate the animation as intended?	NA	No animation currently implemented enough to test at 4-30-13.
4	Does the UML clearly show the relationships between each class?	Y	
5	Has the general user population been determined?	Y	Organic Chemistry professors and students.
6	Has the design of the GUI and features taken this into account?	Y	GUI is simplistic and in context is in laymen terms.
7	Has any and all text seen by the user been written in terms the user will easily understand?	Y	
8	Is the system reliable?	Y	
9	Does the program meet the requirements set by the client?	Y	

6.7.2 Design

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial creation of document and draft.
2	3-14-2013	John Gibbons	Second revision of draft and added additional items.
3	4-27-2013	John Gibbons	Performed checklist.
4	4-30-2013	John Gibbons	Revision of checklist and changes.
5	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments
	Support Material		
1	Has the UML been completed and fully designed?	Y	Done 4-26-2013
2	Have the state diagrams been completed and fully designed?	Y	Completed 5-2-2013
3	Have use cases been made and fully represent all requirements?	Y	

4	Have at least one use case scenarios been made for each use case?	Y	They each contain one.
5	Has all the above documents been properly designed to reflect all the customers' requirements?	Y	
6	Has the client signed off on the design?	NA	
7	Do all documents conform to Object Oriented Design principles?	Y	
8	Do any documents need to be updated or changed?	Y	Due to changes during the term of the project, some documents were needed to be updated pertaining to requirements and platform changes.
9	Are all classes, interfaces and objects shown in the design?	y	
	Inspection Checklist	Y, N, NA	Comments
1	Has the classes been named appropriately for the code segments and processes they contain?	Y	
2	Are all class relations properly connected?	Y	
3	Has the functions been named appropriately for the processes they will be doing?	Y	They have been named after the actions they perform.
4	Has the methods been named appropriately for the processes they will be doing?	Y	They have been named after the actions they perform.
5	Has the variables been named appropriately for the information they are storing?	Y	They have been named after the data they store.
6	Have professional commenting procedures been followed?	Y	Some commenting is lacking. However, all necessary commenting exists.
7	Are all the relationships and number of possible instances between entities correct?	Y	Number of relations between entities not shown.
8	Has any core functionality been left out that was outlined in the requirements documentation?	N	All functionality is accounted for. 5-2-2013 currently no animation.
	Variables	Y, N, NA	Comments
1	Are there any redundant functions, methods, or variables?	N	
2	Are any variables hard-coded?	N	
3	If hard-coded variables exist, if they are	N	

	changed will they deter the program from functioning properly?		
4	Has any and all variables and arrays been properly set up and instantiated?	Y	
5	Are all method return types properly type casted?	Y	
6	Are all data types properly typed and/or casted?	Y	
	Structure	Y, N, NA	Comments
1	Will changes to the class diagrams alter the functionality of the program?	Y	Classes rely on the names listed as functions/methods in the UML. Changing the names would cause other code to fail.
2	Will changes to data types within the diagrams cause problems elsewhere?	Y	There is no failsafe explicit type casting because none was necessary.
3	Will changes in method parameters cause problems elsewhere?	Y	
4	Will any changes possibly result in pointer exceptions?	Y	
	Finalization	Y, N, NA	Comments
1	Does the design contain appropriate error catching methods?	Y	
2	Do the error catching methods provide adequate information to the user on how to resolve the error?	Y	All explained in laymen terms.
3	Is the design modular enough such that further requirements and design changes are possible?	Y	The program was designed to be later expanded on.
4	Is the design realistic enough such that the development team can complete the tasks in the allotted timeframe?	Y	
5	Does the design take into account all constraints in which may hinder the functionality of the program?	Y	
6	Does the design ensure optimized usage and flow for the user?	Y	Everything was kept as simple as possible.
7	Does the design logically flow from one process/step to the next?	Y	
8	Does the design and layout of the animations match the requirements?	N	4-30-13 Animation currently not functioning.
9	Are all options for the user to perform from	N	4-30-13 Animation

	the SRS document accounted for?		currently not functioning.

6.7.3 Implementation

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial creation of document and first draft.
2	3-14-2013	John Gibbons	Second draft and additional items added.
3	4-27-2013	John Gibbons	Checklists filled out
4	4-30-2013	John Gibbons	Checklist changes and updates.
5	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments
	Documentation		
1	Has all standards and guidelines been identified?	Y	IEEE and Object Oriented Design standards have been followed.
2	Were any changes made to the implementation document?	Y	Few minor requirements changes.
3	Have any rules been added for initial implementation document?	N	
4	Is there a cover sheet and table of contents?	Y	
5	Is there revision control?	Y	
6	Has the implementation strategy been approved?	Y	
7	Has all requirements been confirmed with the client?	Y	
8	Has the Design documentation been approved?	Y	
9	Has the Design checklist been performed?	Y	Passed but some sections left unanswered due to insufficient code.
11	Does the design solution meet the requirements?	Y	Yes, if the design solution is fully completed, all requirements would be satisfied.
12	Is the set of deficiencies provided?	Y	
13	Was this checklist conducted on schedule?	N	Was conducted 4-27-13. Animation code still not functioning as well. Re-conducted on 5-2-13 and still no animation.
14	Were all needed resources	Y	All documentation was provided.

	available to properly and fully perform this checklist at time of completion?		
	Schedule	Y, N, NA	Comments
1	Has the workload been appropriately divided upon members in the development team?	N	Workload was left up to members to complete at will.
2	Has the work breakdown structure been established?	Y	Was made 4-26-13.
3	Has the work breakdown structure been approved?	Y	
4	Has proper deadlines been set for the coding phase?	N	This is the reason the coding is so far behind.
5	Were those deadlines met within reason?	N	No. Still no animation code as of 5-2-2013.
	Support Material	Y, N, NA	Comments
1	Was all source code provided?	Y	All hosted on a private GitHub.
2	Was all documentation provided?	Y	All provided in Google Docs.
3	Was Analysis report provided?	Y	
4	Is the Analysis report in the correct format?	Y	
	Structure	Y, N, NA	Comments
1	Is the user interface user friendly?	Y	
2	Is the coding style consistent?	Y	
3	Has professional comments been provided?	Y	Some comments are lacking. However, necessary comments are all accounted for.
4	Is there code that could be condensed?	N	
5	Is there code that could be re-written for higher optimization?	N	
6	Is there repetitive code?	N	
7	Does the code properly implement all requirements from the Analysis report?	N	Animation is still yet to be completed as of 4-30-13. Animation code still non-existent as of 5-2-2013.
8	File Opening/Closing: Will the proposed requirements cause additional files to be opened and/or closed from the previous version?	N	
9	Data Values: Will proposed changes to data values within the scope of the changes cause problems elsewhere?	N	
10	Parameter Change: Will proposed	Y	

	changes affect the subroutines?		
11	File Position Change: Will proposed changes affect the files sensitive to record positioning?	N	
12	Invalid Pointer: Will proposed changes affect existing linked data structures, possibly causing pointer exceptions?	N	
13	Record Layout Change: Will proposed changes affect record layouts?	N	
14	Is storage use efficient?	Y	
15	Is the code consistent in style?	Y	
	Variables	Y, N, NA	Comments
1	Do all classes follow proper naming conventions?	Y	
2	Do all methods follow proper naming conventions?	Y	
3	Do all functions follow proper naming conventions?	Y	
4	Do all variables follow proper naming conventions?	Y	
5	Are there any unused variables?	N	
6	Are variables of the correct types?	Y	
7	Are accessors and mutators named properly and similarly?	Y	
8	Are all array references in bounds?	Y	
	Loops and Branches	Y, N, NA	Comments
1	Do loops properly initialize and increment/decrement the variables properly?	Y	
2	Are loops too nested?	N	
3	Are nested loops properly nested?	Y	
4	Are all cases covered in IF/ELSEIF/ELSE statements or CASE blocks?	Y	
5	Does every switch statement have a default?	Y	
6	Are loop termination conditions achievable?	Y	
7	Does the code in the loop avoid manipulating the index variable	Y	

	or using it upon termination of the loop?		
8	Are case statements properly set up and contain breaks?	Y	
9	Does the default case give an error if not reached?	Y	
	Defensive Programming	Y, N, NA	Comments
1	Do processes occur in the right order throughout the program?	Y	
2	Has unit testing been done to ensure functions and methods provide the correct solutions?	Y	Unit tests were conducted throughout the entire implementation process using PyUnit.
3	Has the functions/methods been made modular such that further expansions are easier?	Y	Code was implemented with future expansions in mind.
4	Are there any unused functions or methods?	N	
5	Do methods/functions contain appropriate error catching capabilities?	Y	
6	Are the error messages direct and easy to understand?	Y	They are direct and are in laymen terms.
7	Does the correct form get forefront attention?	Y	Although, currently there is no additional form for the animation to test this with.
8	Is the correct data being operated upon in each statement?	Y	Fully tested in unit testing.
	Additional	Y, N, NA	Comments
1	Does the Python 3 properly construct the animation according to the design?	Y	Python 3 utilizes built in libraries for graphics generation. However, as of 4-30-13 animation is currently not functioning.
2	Does the code ensure that if the user makes their own organic molecule, that it is indeed only up to a pentadecane molecule?	Y	
3	Does Python 3 properly generate the layout for the animation?	N	Animation currently not functioning 4-30-13.
4	Does the animation follow the process in exact accordance to the customers' models?	N	No animation code as of 5-2-2013.
5	After the user watches the animation and wishes to go back to first window, does the animation window close properly?	N	No animation code as of 5-2-2013.
6	If the user chooses to enter in the name of the constructed	Y	

	molecule, does the program appropriately inform the user if they are correct or incorrect?		
7	If the user chooses to enter in the name of the constructed molecule and does so incorrectly, does the program appropriately inform them of their mistake and how to correct it?	Y	
8	Is there code that will inform the user when additional updates or requirements have been added?	N	Not available in current functionality.
9	Is there code that will allow the user to leave messages for the software engineers/developers?	N	Not available in current functionality.

6.7.4 Testing

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial creation of document and first draft.
2	3-14-2013	John Gibbons	Second draft and additional items added.
3	4-30-2013	John Gibbons	Testing yet to be done. Unable to complete checklist at this time.
4	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments
	Requirements Testing		
1	Have all requirements from the SRS requirements document been satisfied?	N	Currently animation code is non-existent as of 5-2-2013.
2	Do all the requirements in the SRS document fully meet the customers' requirements?	Y	The requirements in the SRS document were approved by the customer.
3	Does the code properly and fully implement those requirements?	N	No animation code 5-2-2013.
4	Can the requirements' code be tested?	Y	Extensive amount of Unit testing has been performed. However, cannot test animation code as it has yet to be written as of 5-2-2013.
5	Are the requirements clear and concise?	Y	

6	Has the customer signed off on all requirements for verification?	Y	
7	Does the program execute each step of the naming process and animation process according to the customers' models?	NA	No animation as of 5-2-2013.
8	Does the software meet security requirements?	Y	None were required as per current requirements.
9	Does the software meet privacy requirements?	y	None were required as per current requirements.
10	Does the documentation have a description of higher architecture?		
	Functional Design	Y, N, NA	Comments
1	Does the design allow the user to perform any and all tasks promised?		
2	Does the design incorporate future software updates and expansions?		
3	Does the software allow for the user to create their own pentadecane molecule?	Y	Through a series of mouse clicks in boxes laid out in a grid format.
4	Does the software allow for the user to enter in what they believe to be the name for the pentadecane molecule?	Y	
5	Does the software allow for some actions to be performed multiple times?	Y	Such as the user entering in guesses for the name of the molecule and checking the name.
6	Does the design handle well under stress load?		
7	Does the software inform the user when a mistake has been made and how to resolve it?	Y	
8	Does the design give the user a simple interface to work with?	Y	
9	Does the design allow the user to backtrack to previous forms?		
10	Does the design allow testers to add comments?		
11	Does the design allow the users to add comments?	N	Not in current functionality.
12	Does the design address techniques and tools that shall be used to assure software quality assurance?	Y	
13	Does the design catch all exceptions correctly?	Y	All except the animation code which is yet to be written as of 5-2-2013.
	Testing Techniques	Y, N, NA	Comments
1	Does the software pass Black box testing?		

2	Does the software pass White box testing?		
3	Does the software pass Unit testing?	Y	Unit testing performed 5-1-2013.
4	Does the software pass Integration testing?		
5	Does the software pass System testing?		
6	Does the software pass Alpha testing?		
7	Does the software pass Beta testing?		
8	Does the software pass Acceptance testing?		
	Environmental Testing	Y, N, NA	Comments
1	Does the program run correctly on all operating systems specified in the SRS document?		
2	Does the program work correctly with all drivers?		
3	Does the program work smoothly in these environments? (not choppy)		
4	Does any aspect of the program crash or freeze during any step in these different environments?		
5	Is any functionality hindered by running in any of the operating systems specified in the SRS document?		
	Acceptance Testing	Y, N, NA	Comments
1	Are all previous issues and bugs fixed?		
2	Is the client satisfied with the finished project?		
3	Is the finished project ready for release?		

6.7.5 Installation

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial document creation and first draft.
2	3-14-2013	John Gibbons	Second draft and additional items added.
3	4-27-2013	John Gibbons	Completed checklist for professor's machine and lab machines.
4	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments

	Support		Incase system does not have the necessary software.
1	Internet Explorer 7.0 or higher installed	Y	All machines possess one or more of these web browsers.
	Firefox 3.6 or higher installed	Y	
	Google Chrome	Y	
	Windows Systems		For machines which use Windows
2A			
	RAM - 128mb (64mb for Windows XP 32bit)	Y	All machines researched possess these characteristics or higher.
	Disc Space - 128mb	Y	
	Mac OS X		For machines which use MAC OS
2B			
	Mac OS X 10.7.3 (Lion) or later	Y	All machines researched possess these characteristics or higher.
	Linux		For machines which use Linux
2C			
	Oracle Linux 5.5+	Y	All machines researched possess these characteristics or higher.
	- Oracle Linux 6.x (32bit)	Y	
	- Oracle Linux 6.x (64 bit)	Y	
	Red Hat Enterprise Linux 5.5+	Y	
	- Oracle Linux 6.x (32bit)	Y	
	- Oracle Linux 6.x (64 bit)	Y	
	Ubuntu Linux 10.04 and above	Y	
	Suse Linux Enterprise Server 10 sp2	Y	
2C.1			
	Ram - 64 MB	Y	
	Disk Space - 58 MB	Y	
	Software Requirements	Y, N, NA	Comments
1	Does the system have Py2exe installed?	NA	Linux/Unix systems already have Python Interpreters installed. MAC and Windows may need it if the exported .exe file does not function properly.
	Additional	Y, N, NA	Comments
1	Is the program easy to install?	Y	Just copy and paste the .exe file to desktop and double click to run.
2	Is the install process fully documented in the manual?	Y	
3	Does the installation manual contain useful screen shots?	Y	Only need one.
4	Is the installation manual easy to use and navigate?	Y	
5	Are trouble shooting options addressed as best as possible?	Y	

6	Was this assessment done when scheduled?	Y	
	Documentation	Y, N, NA	Comments
1	Is there a cover page and table of contents?	Y	
2	Is there revision control?	Y	

6.7.6 Maintenance

Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	3-13-2013	John Gibbons	Initial creation of document and first draft.
2	3-14-2013	John Gibbons	Second draft and additional items added.
3	4-27-2013	John Gibbons	Checklist filled out.
4	5-2-2013	John Gibbons	Final revision.

No.		Y, N, NA	Comments
	Documentation		
1	Is there a generated error report or a properly filed support ticket?	Y	Properly filed support ticket.
2	Is there a customer support solutions manual provided?	N	A user manual is provided however no customer support sections currently exist at this time.
3	Is there an installation manual provided?	Y	Installation is transferring the file from a medium onto the desktop.
4	Is there a user manual provided?	Y	
5	Has the support ticket been filled out properly?	NA	
6	Has the issue been properly stated?	NA	
7	Did the user provide evidence with the support ticket?	NA	
8	Was the date and version number of the software properly documented on the support ticket?	NA	
9	Do the errors generate error codes?	N	Not in current functionality.
	Resolution	Y, N, NA	Comments
1	Was the error able to be	NA	

	regenerated?		
2	Was the error related to the program or the users' machine?	NA	
3	Was the maintenance done in a timely manner?	NA	
4	Was the customer/user satisfied with the maintenance?	NA	
	Future Additions	Y, N, NA	Comments
1	Will intended future additions affect the current functionality of the program?	N	It will simply add on to current functionality.
2	Will intended future additions add to the current functionality of the program?	Y	Future additions will add additional organic compound structures.
3	Does the program include ways to inform the users of new features and additions?	N	Not currently.
4	Has the client fully approved of any additions?	Y	
5	Is the code modular and well commented enough to easily modify?	Y	The code was implemented with future expansion in mind.
	Further Contact	Y, N, NA	Comments
1	Does the client have the ability to reach the development team should any issue arise?	Y	The client possesses each team member's full name and Georgia Southern email addresses.

7 Test Plan

7.9 Revisions

Revision Number	Revision Date	Author	Summary of Changes
1	4-21-2013	John Gibbons	Initial creation of document and first draft.
2	4-24-2013	John Gibbons	Second draft and added additional items.
3	4-26-2013	John Gibbons	Third draft and added additional items.
4	4-27-2013	Roland Heintze	Final revision and modified Risk Assessment

This document will lay out how our development team will conduct all testing pertaining to the Chemistry Molecular Naming and Designing with Animation project. These tests will ensure Dr. Landge's requirements are fully implemented and fully functional. The tests will begin with ensuring each method and function operates and returns the desired results. Following those tests, our team will ensure the entire program as a whole functions properly and as intended on the systems promised. Finally, we will transition into the customer testing the software to ensure all functionality and designs meet or exceed expectations.

7.1 Risk Assessment

1: Low Risk - 2: Medium Risk - 3: High Risk

- Algorithm to randomly create bonded molecule up to pentadecane chain: 1
- Algorithm to compare custom made molecule to standard: 3
- GUI interface to custom design algorithm: 2
- Queries from database: 1
- Check optionally entered name by user: 3
- Generate the animation: 3

7.2 Team Member's Responsibilities

- Roland Heintze - Team Leader and Primary Tester
  - Will oversee testing and conduct system, alpha, beta, and acceptance testing. Will ensure product is in release condition within schedule.
- John Gibbons - Quality Assurance Officer
  - Will ensure that testing is properly and fully conducted. All aspects will be run through and all results, especially issues, will be fully documented.
- Tim Elam - Tester
  - Will conduct unit, integration and functional testing to ensure product is fully operational. Will ensure all of the customers' requirements are implemented and fully functional as well as the program will properly function on all promised operating systems.
- Chris Lansing - Testing Assistant
  - Will ensure all paperwork is properly filled out and all reports are generated. Will ensure all steps and results are properly and fully conducted and results are recorded.

7.3 Test Items

Important Application Areas:

- Graphical User Interface
  - User friendly and in laymen terms.
  - Well designed and laid out.
- Molecule Design Window
  - Contains all necessary components.
  - Easy to understand functionality.
- Database Queries
  - Retrieves correct molecule names and structures.
- Notifications/Error Messages
  - Gives concise and direct error messages
  - Gives users useful information on how to resolve issues
- Proper Animations
  - Animation follows nomenclature rules.
  - Animation meets customers' requirements.
- Proper Naming
  - All bonded molecule's names are concise, correct, and are in correct format.

7.4 Features to be tested

Feature: Generating a Random Organic Molecule and Watching the Animation.

I. Input
   a. First button click.
   b. Second button click.

II. True Black Box Testing
   a. Description
      i. Open program.
      ii. Click 'Generate Random Compound' button.
      iii. Click 'View Animation' button.
      iv. Exit program.
   b. Expected Outcome
      i. Program will open.
      ii. A random organic compound up to and including pentadecane is generated.
      iii. The compound is passed to the animation generator and the animation plays according to the naming process.

iv. Program exits.

III. False Black Box Testing
   a. Description
      i. Open program.
      ii. Click “Generate Random Compound’ button.
   b. Expected Outcome
      i. Program will open.
      ii. Error message displays stating an error has occurred while trying to generate the organic compound.

IV. True White Box Testing (Same as black as all inputs are automated button clicks)

V. False White Box Testing
   a. Description
      i. Open program.
      ii. Click “View Animation” button
   b. Expected Outcome
      i. Program opens.
      ii. Error message is displayed stating that there is no current organic compound to base the animation from.

Feature: Generating a Random Organic Molecule, Entering in what is believed to be the Name and Watching the Animation.

I. Input:
   a. First button click.
   b. Name of organic molecule.
   c. Second button click.
   d. Third button click.

II. True Black Box Testing
   a. Description
      i. Open Program.
      ii. Click “Generate Random Compound’ button.
      iii. Enter in Supposed name of organic compound.
      iv. Click ‘Check Name’ button.
      v. Click ‘View Animation’ button.
      vi. Exit program.
   b. Expected Outcome
      i. Program opens.
      ii. Random organic molecule up to and including pentadecane is generated.
      iii. Program informs user their guess is correct.
      iv. Program displays correct animation for naming process.

v. Program Exits.

III. False Black Box Testing
   a. Description
      i. User clicks ‘Generate Random Compound’ button.
      ii. User enters in guess for name of molecule.
      iii. User clicks ‘Check Name’ button.
   b. Expected Outcome
      i. Dialog box will appear informing the user their guess for the name is incorrect.

IV. True White Box Testing
   a. Description
      i. User clicks ‘Generate Random Compound’ button.
         1. Makes sure compound generated is up to and including pentadecane and correct format compared to pre gathered data.
      ii. User enters in guess for name of molecule.
      iii. User clicks ‘Check Name’ button.
         1. Makes sure the guess is correct compared to pre gathered data.
      iv. User clicks “View Animation” button.
         1. Makes sure animation follows correct paths according to scientific nomenclature.
      v. Exit program.
   b. Expected Outcome
      i. A correct random organic compound is generated.
      ii. User is informed their guess is correct.
      iii. Correct animation is displayed.
      iv. Program exits.

V. False White Box Testing
   a. Description
      i. User clicks “Generate Random Compound’ button.
         1. Makes sure compound generated is up to and including a pentadecane and correct format compared to pre gathered data.
   b. Expected Outcome
      i. The generated organic compound is not in the allowed compound list.

Feature: Custom Creates an Organic Compound and Viewing the Animation.

I. Input
   a. User goes through a series of mouse clicks in a grid.
   b. First button click.
   c. Second button click.

II. True Black Box Testing

a. Description
    i. Open program.
    ii. User clicks in a series of boxes laid out in a grid.
    iii. User clicks ‘Validate Compound’ button.
    iv. User clicks ‘View Animation’ button.
    v. Exit program.
b. Expected Outcome
    i. Program opens.
    ii. User forms an acceptable organic compound.
    iii. User is informed their custom compound is indeed correct.
    iv. User views animation for organic compound.
    v. Exits program.

III. False Black Box Testing
a. Description
    i. Open program
    ii. User clicks in a series of boxes laid out in a grid.
    iii. User clicks ‘Validate Compound’ button.
b. Expected Outcome
    i. Program opens.
    ii. A message box is displayed stating that the user created a compound not allowed by the current functionality of the program.

IV. True White Box Testing
a. Description.
    i. Open program.
    ii. User clicks in a series of boxes laid out in a grid.
    iii. User clicks ‘Validate Compound’ button.
        1. Checks pre gathered compound structures to ensure compound structure is indeed correct.
    iv. User clicks ‘View Animation’ button.
        1. Checks pre gathered compound information to ensure animation follows the exact nomenclature procedure.
    v. Exit program.
b. Expected Outcome
    i. Program opens.
    ii. User constructed a valid organic compound within the current functionality of the program.
    iii. User is informed their custom compound is correct.
    iv. User views animation of naming process.
    v. Program closes.

V. False White Box Testing
a. Description
    i. Open Program.

ii. User clicks in a series of boxes laid out in a grid.
iii. User clicks 'Validate Compound' button.

b. Expected Outcome
i. Program opens.
ii. Error message displays the currently constructed compound is not within current constraints.

Feature: Custom Creates Organic Compound, Guesses Name and Views Animation.

I. Input
a. User goes through a series of mouse clicks in boxes laid out in a grid.
b. First button click.
c. User enters in guess of name for the compound.
d. Second button click.
e. Third button click.

II. True Black Box Testing
a. Description
i. Open program.
ii. User clicks in a series of boxes laid out in a grid.
iii. User clicks 'Validate Compound' button.
iv. User enters in a guess for the name of the compound.
v. User clicks 'Check Name' button.
vi. User clicks 'View Animation' button.
vii. Closes program.
b. Expected Outcome
i. Program opens.
ii. User constructs valid organic compound.
iii. User is informed their compound is indeed valid.
iv. User enters in the correct name for compound.
v. User is informed their guess was correct.
vi. User views animation for the compound.
vii. Exits program.

III. False Black Box Testing
a. Description
i. Open program.
ii. User clicks in a series of boxes laid out in a grid.
iii. User clicks 'Validate Compound' button.
iv. User enters in a guess for the name of the compound.
v. User clicks 'Check Name' button.
b. Expected Outcome

i. Program opens.
ii. User creates a valid organic compound.
iii. User enters in weird characters and numbers.
iv. Error message is displayed informing the user their guess name contained invalid characters.

IV. True White Box Testing
   a. Description
      i. Open program.
      ii. User clicks in a series of boxes laid out in a grid.
      iii. User clicks 'Validate Compound' button.
         1. Checks pre gathered data to ensure the custom organic compound is indeed allowed in current functionality and is indeed a compound.
      iv. User enters in a guess for the name of the compound.
      v. User clicks 'Check Name' button.
         1. Checks pre gathered data to ensure the name of the organic compound and format is indeed correct.
      vi. User clicks 'View Animation' button.
      vii. Exits program.
   b. Expected Outcome
      i. Program opens.
      ii. User indeed creates an acceptable organic compound.
      iii. User is informed their custom compound is acceptable.
      iv. User enters in a guess for the name of the compound.
      v. User is informed their guess is indeed correct.
      vi. User views animation on naming procedure of the compound.
      vii. Exits program.

V. False White Box Testing
   a. Description
      i. Open program.
      ii. User clicks in a series of boxes laid out in a grid.
      iii. User clicks 'Validate Compound' button.
      iv. User enters in guess for the name of the compound.
      v. User clicks 'Check Name' button.
   b. Expected Outcome
      i. Program opens.
      ii. User creates a valid organic compound within the functionality of the program (pentadecane).
      iii. User is informed their custom compound is indeed acceptable.
      iv. User enters in guess for the name of the compound.
      v. Message box appears informing the user the format of their guess is incorrect.

7.5 System Functions

Interfaces

- System functions properly on Windows XP, Vista, 7 and 8.
- System functions properly on Linux, UNIX and Mac OS.
- System functions properly with standard mouse and keyboard.
- System functions properly with standard monitors and resolutions.

7.6 Testing Machine's System Requirements

Recommended System Requirements

I. System Hardware
   a. Windows Systems
      i. RAM - 128MB (64MB for Windows XP 32bit)
      ii. Disc space 124MB.
   b. Linux Systems
      i. RAM - 64MB.
      ii. Disc space 58MB.
   c. MAC Systems
      i. RAM - 128MB
      ii. Disc space 124MB.
   d. Input/output Devices
      i. Contain at least 50mb of persistent memory.
      ii. Standard monitor that can support 4:3 and 16:10 aspect ratios.
      iii. Standard mouse and keyboard.

II. Additional Software
   a. Py2exe Python Interpreter
   b. Browsers:
      i. Internet Explorer 7.0 or higher.
      ii. Firefox 3.6 for higher.
      iii. Google Chrome.

III. Transfer Medium
   a. Internet access or USB drive port.

7.7 Approach

7.7.1 Unit Testing

Unit testing will be conducted to ensure all methods and functions execute properly and return the desired results. This step will be conducted as White-Box testing since it takes knowledge of how the source code executes and the desired results. This step is being done during implementation to ensure gradual and positive progress. Our development team will be using PyUnit for unit testing in Python. There will be sample code implemented and later disregarded to tests queries from the database to ensure the correct and desired data is being pulled.

7.7.2 Integration Testing

This session of testing will be conducted to ensure all classes and methods/functions come together to form one fully functioning program. This level of testing will be conducted as Black-Box testing as it requires zero knowledge of how the individual classes/methods/functions are executed in source or pseudo code. If issues occur during this stage, additional White-Box testing will be used to flesh out the issues that arose. After they are resolved the team will resume integration testing to ensure complete functionality for the end product. Our testing team will conduct integration testing by using a set of predefined test cases to see if the results match the predetermined outcomes.

7.7.3 Functional Testing

This stage of testing will be conducted to ensure all requirements set by the customer are fully implemented in the code. Each use case will be executed and the results carefully monitored to see execution style and results. If results do not match the scenarios set forth by the customer, corrections will be made to the code and testing will resume after corrections are completed.

7.7.4 System Testing

This stage of testing will be conducted to ensure the program functions properly and fluidly on all systems and environments containing the minimum specifications listed in the requirements documents. Our team will be transporting the software to the computer labs and classrooms which contain computers within the chemistry building in order to test functionality for the purpose of ensuring desired results in the environment the software will primarily be used in. Next our team will test functionality on laptops similar to or exactly like the machines used by the majority of students and professors to ensure functionality holds across other commonly used hardware.

7.7.5 Alpha Testing

This stage of testing will be conducted by having the customer test the software on one of the developer's machines at our team's location or the customer's office. This test will be conducted by Roland who is team leader of our software development group. In this process, the customer will identify any issues that may exist with the current GUI design or functionality and report them directly to the development team. All issues will be fully documented and fixed according to the customers' wishes and designs.

7.7.6 Beta Testing

This stage of testing will be conducted at the customers' environment of choice, which will be the chemistry buildings' computer labs or classrooms containing computers. This test will be conducted by Roland. In this process, our team will ensure all issues uncovered during Alpha testing have been resolved and the program is functioning to the customers' wishes. If any other issues arise, they will be fully documented and corrected before release.

7.7.7 Acceptance Testing

During this final stage of testing our team will deliver the final product to the customer. Along with the finished program, all paperwork and designs pertaining to this project will be handed over to the client. The customer will check to ensure all issues have been resolved and all supporting documentation and designs are accounted for. This process will be conducted at the customers' location and the software will be placed on all machines that the customer requests. If the computers have deep freeze installed, we will leave a hardcopy of the program with the customer on a flash drive or any other device the customer wishes. Finally, our team will ensure any and all questions and/or concerns the customer may have pertaining to this project are addressed.

7.8 Testing Deliverables

7.8.1 Unit Testing

For unit testing, our team's tester will include any functions/methods created as well as all information generated from PyUnit in the notebook. Some examples will be chosen of functions/methods containing the tests and results will also be included.

7.8.2 Integration Testing

All previously tested modules will be combined to test cohesion. A simple form will be filled out stating any issues found and where those issues may originate. The form shall include the date being performed, the version number of the software, and a list of all key modules and important sub modules being tested in unison. The document shall also include a list of test cases performed with their results and the name of the member who performed those tests.

7.8.3 Alpha Testing

Forms will be filled out by the customer stating any issues that may exist with the software from GUI design to functional issues. The issue(s) will be clearly stated by the customer along with how they wish it to be resolved. It will be accompanied by comments from Roland on how these issues will be resolved in accordance with the customers' wishes, on the technical side, as well as date performed and estimated date of issues being resolved. Before finishing the session, the customer and Roland will sign off on the forms stating the customer found the issues and stated how they should be resolved.

7.8.4 Beta Testing

Additional forms will be filled out by the customer stating any new issues and unresolved issues from GUI design to functional issues. The issue(s) will be clearly stated by the customer along with instructions on how they wish them to be resolved. It will be accompanied by comments by Roland on how these issues will be resolved in accordance with the customers' wishes, on the technical side, as well as date performed and estimated date of issues being resolved. Before finishing the session, the customer and Roland will sign off on the forms stating the customer found the issues and gave instructions on how they wish for them to be resolved.

7.8.5 Acceptance Testing

This form will be filled out by the customer stating the program meets all requirements set forth by them. The client will also verify all documentation and designs are included within the files given to them. Along with physicals items, the customer will confirm that all issues and concerns have been addressed and resolved to the best of the development team's capabilities.

8 Testing Reports:

8.1 Unit Testing:

8.1.1 Alkane Test

```
1.  # -*- coding: utf-8 -*-
2.  __author__ = 'rheintze'
3.
4.  """
5.  Unit tests for alkane.py
6.  """
7.
8.
9.  import unittest
10.
11. from chemistry.alkane import *
12. from chemistry.chem_exceptions import *
13.
14. class TestAlkane(unittest.TestCase):
15.     def setUp(self):
16.         self.test_int = 50
17.         self.test_string = '50'
18.
19.     def test_alkane(self):
20.         """ Unit tests for the Alkane class """
21.
22.
23.         #Matrix for empty Alkane
24.         #F F F
25.         #F F F
26.         #F F F
27.         emptyMatrix = [[False,False,False],[False,False,False],[False,False,False],]
28.         self.assertRaises(EmptyAlkaneError, Alkane, emptyMatrix)
29.
30.         #Matrix for disconnected Alkane
31.         #T T T
32.         #F F F
33.         #F F T
34.         disconnectedMatrix = [[True,False,False,],[True,False,False,],[True,False,True,],]
35.         self.assertRaises(AlkaneNotConnectedError, Alkane, disconnectedMatrix)
36.
37.         #Matrix for large-loop Alkane
```

```
        #T T T
        #T F T
        #T T T
        largeLoopMatrix = [[True,True,True,],[True,False,True,],[True,True,True,],]
        self.assertRaises(CyclicAlkaneError, Alkane, largeLoopMatrix)

        #Matrix for small-loop Alkane
        #T T T
        #F T T
        #F F F
        smallLoopMatrix = [[True,True,True,],[False,True,True,],[False,False,False,],]
        self.assertRaises(CyclicAlkaneError, Alkane, smallLoopMatrix)

        #Malformed substituent Carbon Matrix
        #T T T T T T T T T
        #T F T F T F F F F
        #T F F T T T F F F
        malformedSubMatrix = [
            [True, True, True, True, True, True, True, True, True,],
            [True, False, True, False, True, False, False, False, False,],
            [True, False, False, True, True, True, False, False, False,],
        ]
        self.assertRaises(BranchingCarbonChainError, Alkane, malformedSubMatrix)

        #Typical valid Carbon Matrix
        #F F T F F F T F F
        #T T T T T T T T T
        #T F T F T F F F F
        #T F F F T T F F F
        typicalValidMatrix = [
            [False, False, True, False, False, False, True, False, False,],
            [True, True, True, True, True, True, True, True, True,],
            [True, False, True, False, True, False, False, False, False,],
            [True, False, False, False, True, True, False, False, False,],
        ]



if __name__ == '__main__':
    unittest.main()
```

8.1.2 Carbon Test

```
1.  # -*- coding: utf-8 -*-
2.  __author__ = 'rheintze'
3.
4.  """
5.  Unit tests for carbon.py
6.  """
7.
8.
9.  import unittest
10.
11. from chemistry.carbon import *
12.
13. class TestCarbon(unittest.TestCase):
14.     def setUp(self):
15.         pass
16.
17.     def testGetOpposingDirection(self):
18.         self.assertEqual('north', Carbon.getOpposingDirection('south'), "The opposite of south was
    not north!")
19.         self.assertEqual('south', Carbon.getOpposingDirection('north'), "The opposite of north was
    not south!")
20.         self.assertEqual('east', Carbon.getOpposingDirection('west'), "The opposite of east was not
    west!")
21.         self.assertEqual('west', Carbon.getOpposingDirection('east'), "The opposite of west was not
    east!")
22.
23.     def testGetDirections(self):
24.         directions = Carbon.directions()
25.         self.assertIn('north', directions, "North not in directions!")
26.         self.assertIn('south', directions, "South not in directions!")
27.         self.assertIn('east', directions, "East not in directions!")
28.         self.assertIn('west', directions, "West not in directions!")
29.
30.     def testGetDirectionsWithout(self):
31.         for excludeDir in Carbon.directions():
32.             actual = Carbon.getDirectionsWithout(excludeDir)
33.             for direction in Carbon.directions():
34.                 if direction == excludeDir:
35.                     self.assertNotIn(direction, actual, "Direction: %s was wrongfully included!" %
    excludeDir)
36.                 else:
```

```
37.                 self.assertIn(direction, actual, "Direction: %s was wrongfully excluded!" % direction)
38.
39.     def testCarbon(self):
40.         carbon = Carbon(1,2)
41.         self.assertEqual(1, carbon.x, "X was wrong!")
42.         self.assertEqual(2, carbon.y, "Y was wrong!")
43.         for direction in Carbon.directions():
44.             self.assertEqual(getattr(carbon, direction, None), None, "Direciton: "+direction+" was not
    None!")
45.
46.     def testGetDirectionTo(self):
47.         left_carbon = Carbon(1, 3)
48.         right_carbon = Carbon(2, 3)
49.         left_carbon.east = right_carbon
50.         right_carbon.west = left_carbon
51.         self.assertEqual(left_carbon.getDirectionTo(right_carbon), "east", "getDirectionTo returned
    the wrong direction!")
52.
53.     def testBondTo(self):
54.         #    Top
55.         #Left Middle Right
56.         #    Bottom
57.         top = Carbon(1,0)
58.         middle = Carbon(1,1)
59.         right = Carbon(2,1)
60.         bottom = Carbon(1,2)
61.         left = Carbon(0,1)
62.         #Try all valid combinations to yield total success branch coverage
63.         try:
64.             top.bondTo(middle)
65.         except Exception as error:
66.             self.fail("bondTo from top to middle failed!\n"+str(error))
67.
68.         top = Carbon(1,0)
69.         middle = Carbon(1,1)
70.         try:
71.             middle.bondTo(top)
72.         except Exception as error:
73.             self.fail("bondTo from middle to top failed!\n"+str(error))
74.
75.         try:
76.             middle.bondTo(right)
```

```
77.         except Exception as error:
78.             self.fail("bondTo from middle to right failed!\n"+str(error))
79.
80.         middle = Carbon(1,1)
81.         right = Carbon(2,1)
82.
83.         try:
84.             right.bondTo(middle)
85.         except Exception as error:
86.             self.fail("bondTo from right to middle failed!\n"+str(error))
87.
88.         #All combinations of outer carbons bonding together will yield total failure branch coverage
89.         outerCarbons = [top, right, bottom, left]
90.         for i in range(len(outerCarbons)):
91.             for j in range(i, len(outerCarbons)):
92.                 self.assertRaises(CarbonsNotAdjacentError, outerCarbons[i].bondTo, outerCarbons[j])
93.
94.
95.     def testBondCarbons(self):
96.         #Create a set of connected carbons according to this map
97.         # 0
98.         #1 2 3
99.         # 4
100.                 carbons  = [
101.                         Carbon(1,0),
102.                 Carbon(0,1), Carbon(1,1), Carbon(2,1),
103.                         Carbon(1,2),
104.                 ]
105.                 Carbon.bondCarbons(carbons)
106.                 self.assertEqual(carbons[0].north,None)
107.                 self.assertEqual(carbons[0].south,carbons[2])
108.                 self.assertEqual(carbons[0].east,None)
109.                 self.assertEqual(carbons[0].west,None)
110.
111.                 self.assertEqual(carbons[1].north,None)
112.                 self.assertEqual(carbons[1].south,None)
113.                 self.assertEqual(carbons[1].east,carbons[2])
114.                 self.assertEqual(carbons[1].west,None)
115.
116.                 self.assertEqual(carbons[2].north,carbons[0])
117.                 self.assertEqual(carbons[2].south,carbons[4])
118.                 self.assertEqual(carbons[2].east,carbons[3])
```

```
119.                self.assertEqual(carbons[2].west,carbons[1])
120.
121.
122.                self.assertEqual(carbons[3].north,None)
123.                self.assertEqual(carbons[3].south,None)
124.                self.assertEqual(carbons[3].east,None)
125.                self.assertEqual(carbons[3].west,carbons[2])
126.
127.                self.assertEqual(carbons[4].north,carbons[2])
128.                self.assertEqual(carbons[4].south,None)
129.                self.assertEqual(carbons[4].east,None)
130.                self.assertEqual(carbons[4].west,None)
131.
132.
133.
134.            def testGetNumberOfBonds(self):
135.                #Create a set of connected carbons according to this map
136.                # 0
137.                #1 2 3 4
138.                #5 6 7   8
139.                carbons  = [
140.                        Carbon(1,0),
141.                Carbon(0,1), Carbon(1,1), Carbon(2,1), Carbon(3,1),
142.                Carbon(0,2), Carbon(1,2), Carbon(2,2), Carbon(4,2)]
143.                expectedNumBonds = [1, 2, 4, 3, 1, 2, 3, 2, 0]
144.                #Bond all carbons together
145.                Carbon.bondCarbons(carbons)
146.                #Check number of bonds against expected number
147.                for i in range(len(carbons)):
148.
    self.assertEqual(carbons[i].getNumberOfBonds(), expectedNumBonds[i], "Expected
  "+str(expectedNumBonds[i])+" bonds, found "+str(carbons[i].getNumberOfBonds)+" instead!")
149.
150.
151.            def testGetCarbonsInDirection(self):
152.                #A map of the carbons to test with
153.                #0 1
154.                # 2
155.                #3 4 5
156.                #    6
157.                carbons = [
158.                    Carbon(0,0), Carbon(1,0),
```

```
159.                     Carbon(1,1),
160.                 Carbon(0,2), Carbon(1,2), Carbon(2,2),
161.                                     Carbon(3,3),
162.             ]
163.             #Bond the carbons together
164.             Carbon.bondCarbons(carbons)
165.             #Mapping of params to method (index in carbons, direction) to expected result
166.             paramsToExpected = {
167.                 (carbons[4], 'west') : [carbons[3]],
168.                 (carbons[4], 'north') : [carbons[2],carbons[1],carbons[0]],
169.
170.             }
171.             for inputTuple, expected in paramsToExpected.items():
172.                 actual = inputTuple[0].getCarbonsInDirection(inputTuple[1])
173.                 if actual != expected:
174.                     message = "Carbons didn't match!\n"
175.                     message += "First Carbon: %s\n" % str(inputTuple[0])
176.                     message += "Direction: %s\n" % inputTuple[1]
177.                     message += "  Actual:\n"
178.                     for act in actual:
179.                         message += str(act)+"\n"
180.                     message += "  Expected:\n"
181.                     for exp in expected:
182.                         message += str(exp)+"\n"
183.                     self.fail(message)
184.
185.
    self.assertRaises(BranchingCarbonChainError, carbons[0].getCarbonsInDirection, 'east')
186.             self.assertEqual(carbons[4].getCarbonsInDirection('south'), [], "Returned non-
    empty list when called in empty direction!")
187.
188.
189.         def testGetConnectedSet(self):
190.             #A connected map with no cycles
191.             #0 1
192.             #  2
193.             #3 4 5
194.             #   6
195.             validMap = [
196.                 Carbon(0,0), Carbon(1,0),
197.                     Carbon(1,1),
198.                 Carbon(0,2), Carbon(1,2), Carbon(2,2),
```

```
                                Carbon(2,3),
            ]
            #Bond the carbons together
            Carbon.bondCarbons(validMap)
            self.attemptConnectedSet(validMap, "validMap", False)

            #A connected map with a cycle with a hole in the middle
            #0 1 2
            #3   4
            #5 6 7
            looseCycledMap = [
              Carbon(0,0), Carbon(1,0), Carbon(2,0),
              Carbon(0,1),            Carbon(2,1),
              Carbon(0,2), Carbon(1,2), Carbon(2,2),
            ]
            self.attemptConnectedSet(looseCycledMap, "looseCycledMap", True)

            #A connected map with tight cycle
            #0 1 2
            #3 4
            tightCycledMap = [
              Carbon(0,0), Carbon(1,0), Carbon(2,0),
              Carbon(0,1), Carbon(1,1),
            ]
            self.attemptConnectedSet(tightCycledMap, "tightCycledMap", True)

        def attemptConnectedSet(self, carbonList, name, shouldFail):
            #Bond the carbons together
            Carbon.bondCarbons(carbonList)
            failed=False
            try:
              for c in carbonList:
                c.getConnectedSet()
              failed=False
            except:
              failed=True
            if shouldFail and not failed:
              self.fail("Carbon list named %s should have failed, but didn't" % name)
            elif not shouldFail and failed:
              self.fail("Carbon list named %s shouldn't failed, but did" % name)

        def testGetLongestChain(self):
```

```
241.            #A connected map with no cycles
242.            #0 1
243.            # 2
244.            #3 4 5
245.            #   6
246.            carbons = [
247.              Carbon(0,0), Carbon(1,0),
248.                      Carbon(1,1),
249.              Carbon(0,2), Carbon(1,2), Carbon(2,2),
250.                              Carbon(2,3),
251.            ]
252.            carbonToExpected = {
253.
    carbons[0] : [carbons[0], carbons[1], carbons[2], carbons[4], carbons[5], carbons[6],],
254.              carbons[1] : [carbons[1], carbons[2], carbons[4], carbons[5], carbons[6],],
255.              carbons[2] : [carbons[2], carbons[4], carbons[5], carbons[6],],
256.              carbons[3] : [carbons[3], carbons[4], carbons[2], carbons[1], carbons[0],],
257.              carbons[4] : [carbons[4], carbons[2], carbons[1], carbons[0],],
258.              carbons[5] : [carbons[5], carbons[4], carbons[2], carbons[1], carbons[0],],
259.
    carbons[6] : [carbons[6], carbons[5], carbons[4], carbons[2], carbons[1], carbons[0],],
260.            }
261.            Carbon.bondCarbons(carbons)
262.            for testCarbon, expected in carbonToExpected.items():
263.              actual = testCarbon.getLongestChain()
264.              if actual != expected:
265.                message = "Carbons didn't match!\n"
266.                message += "First Carbon: %s\n" % str(testCarbon)
267.                message += "  Actual:\n"
268.                for act in actual:
269.                  message += str(act)+"\n"
270.                message += "  Expected:\n"
271.                for exp in expected:
272.                  message += str(exp)+"\n"
273.                self.fail(message)
274.
275.          def testGetDistanceTo(self):
276.            #A connected map with no cycles
277.            #0
278.            # 1
279.            #2 3 4
280.            #   5
```

```
281.                carbons = [
282.                    Carbon(0,0),
283.                            Carbon(1,1),
284.                    Carbon(0,2), Carbon(1,2), Carbon(2,2),
285.                                    Carbon(2,3),
286.                ]
287.                Carbon.bondCarbons(carbons)
288.                inputsToExpected = {
289.                    (carbons[5], carbons[0]) : -1,
290.                    (carbons[5], carbons[1]) : 3,
291.                    (carbons[5], carbons[2]) : 3,
292.                    (carbons[5], carbons[3]) : 2,
293.                    (carbons[5], carbons[4]) : 1,
294.                    (carbons[5], carbons[5]) : 0,
295.                }
296.                for inputs, expected in inputsToExpected.items():
297.                    actual = inputs[0].getDistanceTo(inputs[1])
298.                    self.assertEqual(expected, actual, "getDistanceTo
    Failed!\nInitial:"+str(inputs[0])+"\nTarget:\n"+str(inputs[1])+"\nActual:"+str(actual)+"\nExpecte
    d:"+str(expected))
299.
300.            if __name__ == '__main__':
301.                unittest.main()
```

8.1.3 Substituent Test

```
1.  # -*- coding: utf-8 -*-
2.  __author__ = 'rheintze'
3.
4.  """
5.  Unit tests for alkane.py
6.  """
7.
8.
9.  import unittest
10.
11. from chemistry.substituent import *
12.
13. class TestSubstituent(unittest.TestCase):
14.     def setUp(self):
15.         self.test_int = 50
16.         self.test_string = '50'
17.
18.     def test_substituent(self):
19.         """ Unit tests for the Alkane class """
20.         self.assertRaises(TypeError, Substituent, 1.2)
21.         self.assertRaises(TypeError, Substituent, 'notadigit')
22.         self.assertEqual(Substituent(self.test_int).int, self.test_string)
23.
24. if __name__ == '__main__':
25.     unittest.main()
```

8.1.3 Chemspipy Test

```
# -*- coding: utf-8 -*-
""" Unit tests for chemspipy.py
https://github.com/mcs07/ChemSpiPy

Forked from ChemSpiPy by Cameron Neylon
https://github.com/cameronneylon/ChemSpiPy
"""


import unittest

from chemspipy import *

class TestChemSpiPy(unittest.TestCase):

    def setUp(self):
        self.test_int = 236
        self.test_string = '236'
        self.test_imageurl = 'http://www.chemspider.com/ImagesHandler.ashx?id=236'
        self.test_mf = 'C_{6}H_{6}'
        self.test_smiles = 'c1ccccc1'
        self.test_inchi = 'InChI=1/C6H6/c1-2-4-6-5-3-1/h1-6H'
        self.test_inchikey = 'UHOVQNZJYSORNB-UHFFFAOYAH'
        self.test_averagemass = 78.112
        self.test_molecularweight = 78.1118
        self.test_monoisotopicmass = 78.0469970703125
        self.test_nominalmass = 78
        self.test_alogp = 0
        self.test_xlogp = 0
        self.test_commonname = 'Benzene'

    def test_compound(self):
        """ Unit tests for the Compound class """
        self.assertRaises(TypeError, Compound, 1.2)
        self.assertRaises(TypeError, Compound, 'notadigit')
        self.assertEqual(Compound(self.test_int).csid, self.test_string)
        self.assertEqual(Compound(self.test_string).csid, self.test_string)
        self.assertEqual(Compound(self.test_string).imageurl, self.test_imageurl)
        self.assertEqual(Compound(self.test_string).mf, self.test_mf)
        self.assertEqual(Compound(self.test_string).smiles, self.test_smiles)
        self.assertEqual(Compound(self.test_string).inchi, self.test_inchi)
```

```
42.         self.assertEqual(Compound(self.test_string).inchikey, self.test_inchikey)
43.         self.assertEqual(Compound(self.test_string).averagemass, self.test_averagemass)
44.         self.assertEqual(Compound(self.test_string).molecularweight, self.test_molecularweight)
45.
        self.assertEqual(Compound(self.test_string).monoisotopicmass, self.test_monoisotopicmass)
46.         self.assertEqual(Compound(self.test_string).nominalmass, self.test_nominalmass)
47.         self.assertEqual(Compound(self.test_string).alogp, self.test_alogp)
48.         self.assertEqual(Compound(self.test_string).xlogp, self.test_xlogp)
49.         self.assertEqual(Compound(self.test_string).commonname, self.test_commonname)
50.
51.     def test_find(self):
52.         self.assertEqual(find_one('zxbdfgbfgb'), None)
53.         self.assertEqual(find('zxbdfgbfgb'), None)
54.         self.assertEqual(find_one(self.test_commonname).csid, self.test_string)
55.         self.assertEqual(find(self.test_commonname)[0].csid, self.test_string)
56.
57.
58. if __name__ == '__main__':
59.     unittest.main()
```

9 Post-mortem

9.9 Roland Heintze

This project was an interesting experience in that it provided exposure to a very large domain of which very little was known. The complexity of the project was such that much of the development was in providing a reliable platform, as opposed to a feature-rich system. By this token, I found it to be an interesting take on the software design process. Throughout this project we strived to ensure proper testing and integration. As a result, I believe that the application serves well as an extendable platform for future development. In leading the team, there were some difficulties common to large projects in a domain which none is familiar with. One of the lessons I will draw from this experience is the need for a dedicated period in which all group members can thoroughly digest the scope of the domain and how it relates to the application's development. Another lesson garnered was that of clear responsibilities for the team members. By allowing free commitment to pieces of the project, I witnessed the difficulties that this caused in both the initiative of group members as well as the danger of stepping on one another's commits. Because of this, I have a new appreciation for project managers and the difficulties outside of programming that they must face. I believe myself capable of the position in a professional capacity only once I have received a much larger amount of experience in professional collaboration between programmers.

9.10 Tim Elam

This project was very challenging. From an algorithms point of view, writing code to traverse and perform operations across a custom graph was difficult to do correctly. I learned that adding unit tests to verify program correctness can save you a lot of headaches and let you confidently move to other sections of the code that need more attention without worrying about how correct your program is. From a client relations perspective, mitigating a client's unrealistic expectations and persuading them to agree with your vision of a product is a hard challenge. Although programmers may not have been the tool for the job that the client was looking for, I believe the software that we created has the potential to have a positive impact on students learning organic chemistry and will increase their engagement in learning.

9.11 Chris Lansing

During this project I learned a lot about all of the different types of planning and documentation that go into creating successful software and how much work it actually takes. I also found that working with a client, especially one that didn't understand what exactly a programmer does, was an enlightening experience for me. Both the client and our team had a set of expectations for the project and we had to work hard to find a common ground that would please the client. Working on a project with such a large scale and trying to distribute tasks amongst team members with different strengths was difficult at times, but ultimately turned out well because of the fact that everyone had at least some input at every part of the project. However, most everybody on the team wanted to be a part of the coding which just wasn't feasible with the lack of branches and tasks in our version control software. I hope that when I have a job developing in the future, I'll be able to pay someone who enjoys documentation and design to do that sort of work for me. I would much prefer being in the code base and actively developing features.

9.12 John Gibbons

During this project I learned many aspects of software development from design to documentation. I was unaware of how strict and at times repetitive IEEE documentation could be. However, fully and correctly filling out said paperwork fleshes out all aspects of the project and thus is highly beneficial. In solo projects before, design never mattered to this extent because if a change needed to be made, I could make it without affecting other aspects of the projects. In a team based environment, one change can bring the whole development process to a halt since the other team members are relying and implementing code of the previous design. As for dealing with actual clients, it was my first experience with a customer when developing software. I was completely surprised at the unrealistic requirements our team was first handed at the beginning of the semester. Customers tend to not fully understand the computer science field and that within it exists many different areas and specialties. In this case, the client thought we would be able to generate graphics similar to that of The Simpsons. However, we had

to work with the client to meet on a common ground for the extent of the graphics and functionality of the program. While meeting with the client, she showed us programs that allow the user to build complex compounds with extreme depth and customization. We had to explain this was impossible for such few programs with limited time and software. Finally, the last factor of projects like this is group members. Some group members would be 'otherwise occupied' for times or even until the end of the semester. Also, some members will assume something will be easy to do and then when the deadline is approaching they will realize it is more complicated than expected. The two biggest lessons I learned from this project is to keep working with the client to meet on common ground with requirements and expectations while having them sign off on stages and to start sections of projects as early as possible since there is always unforeseen issues that arise during development and implementation.

10 Appendix

10.9 Source Code

chemistry/__init__.py

(empty file, but required)

chemistry/alkane.py

```
1.  __author__ = 'rheintze'
2.
3.  #from chemspipy import *
4.  #from cactusAPI import get_csid
5.  from chemistry.carbon import Carbon
6.  from chemistry.substituent import Substituent
7.  from chemistry.chem_exceptions import *
8.  class Alkane:
9.      """
10.   A class used to implement an organic compound.
11.   """
12.
13.
    chain = ('methane','ethane','propane','butane','pentane','hexane','heptane','octane','nonane','d
    ecane',
14.           'undecane','dodecane','tridecane','tetradecane','pentadecane')
15.   MAX_CHAIN_LENGTH = len(chain)
16. #   def __init__(self, longestChain):
17. #       self.carbons = longestChain[:]
18. #       self.longestChain = longestChain
19. #       self.head = longestChain[0]
20.
21.   #Raises EmptyAlkaneErorr, CyclicAlkaneError, AlkaneNotConnectedErrorr,
    BranchingCarbonChainError
22.   def __init__(self, matrix):
23.       self.carbons = [] #list of all nodes (carbons) in the graph (molecule)
24.       self.head = None
25.       self.initGraph(matrix)
26.       if self.head == None:
27.           raise EmptyAlkaneError()
28.       #Raises CyclicAlkaneError
29.       if not self.isConnected():
30.           raise AlkaneNotConnectedError()
31.       self.longestChain = self.getLongestChain()
32.       #Throws BranchingCarbonChainError for malformed substituents
33.       self.substituents = self.getSubstituents()
34.       self.head = self.getCorrectHead()
35. #       self.name = self.getName()
36. #           raise error
37.
38.
```

```
    #Initialize graph and carbon list based on carbon matrix
    def initGraph(self, graphicMatrix):
        width = len(graphicMatrix)
        height = len(graphicMatrix[0])
        for col_index in range(0, width):
            for row_index in range(0, height):
                #Skip empty spaces
                if not graphicMatrix[col_index][row_index]:
                    continue
                thisCarbon = Carbon(col_index, row_index)
                self.carbons.append(thisCarbon)
        if any(self.carbons):
            self.head = self.carbons[0]
        Carbon.bondCarbons(self.carbons)


    #Check if carbons are connected and graph has no cycles
    #Raises CyclicAlkaneError if this molecule contains any cycles
    def isConnected(self):
        totalSet = set(self.carbons)
        #Raises CyclicAlkane Error if any cycles exist
        connectedSet = self.head.getConnectedSet()
        return totalSet == connectedSet

    def getLongestChain(self):
        #The longest chain per carbon atom
        candidateLongestChains = []
        for carbon in self.carbons:
            candidateLongestChains.append(carbon.getLongestChain())
        #Return the longest chain
        return max(candidateLongestChains, key=len)

    def getSubstituents(self):
        subs = []
        for i in range(len(self.longestChain)):
            current = self.longestChain[i]
            validDirs = list(Carbon.directions())
            if i != 0:
                previousCarbon = self.longestChain[i-1]
                prevDir = current.getDirectionTo(previousCarbon)
                validDirs.remove(prevDir)
            if i != len(self.longestChain)-1:
```

```
81.             nextCarbon = self.longestChain[i+1]
82.             nextDir = current.getDirectionTo(nextCarbon)
83.             validDirs.remove(nextDir)
84.
85.         for subDirection in validDirs:
86.             subCarbon = getattr(current, subDirection)
87.             if subCarbon:
88.                 #Throws BranchingCarbonChainError for malformed substituents
89.                 subs.append(Substituent(current, subDirection, i))
90.     return subs
91.
92.   def getName(self):
93.     basename = self.chain[len(self.getLongestChain())-1]
94.
95.   def verify(self,guess):
96.     return guess == self.getName()
97.
98.   def getCarbons(self):
99.     return self.carbons
100.
101.            #Get the correct head. That is, the end of the longest chain that
102.            # is closest to a substituent
103.            def getCorrectHead(self):
104.                #Mapping of candidates for head (the ends of the longest chain) to distance of
    closest substituent
105.                candidates = [
106.                  self.longestChain[0],
107.                  self.longestChain[-1],
108.                ]
109.                closestDistances = [None, None]
110.                for sub in self.substituents:
111.                  for i in range(len(candidates)):
112.                    subDist = candidates[i].getDistanceTo(sub.getHead())
113.                    #Did we find this substituent's head?
114.                    # and is this distance smaller than what we have so far?
115.                    if subDist != -
    1 and (closestDistances[i] == None or subDist < closestDistances[i]):
116.                      #Then set the closest distance for this candidate to this distance
117.                      closestDistances[i] = subDist
118.              #Did the second candidate not find any substituents?
119.              if closestDistances[1] == None:
120.                return candidates[0]
```

```
121.            #Did the first candidate not find any substituents?
122.            elif closestDistances[0] == None:
123.              return candidates[1]
124.            #Both candidates found substituents
125.            else:
126.              #Get the index of the candidate with the closest substituent
127.              winningIndex = closestDistances.index(min(closestDistances))
128.              print(str(winningIndex))
129.              return candidates[winningIndex]
130.
131.
132.        #   def createRandomAlkane(self):
133.        #     chain_length = random.randInt(6,Alkane.MAX_CHAIN_LENGTH)
134.        #     max_sub_length = Alkane.MAX_CHAIN_LENGTH-chain_length
135.        #     carbons = []
136.        #     y = 5
137.        #     for i in range(chain_length):
138.        #       carbons.append(Carbon(2+i, y))
139.        #     alkane = Alkane(carbons)
140.        #     for i in range(chain_length):
141.        #       if not random.randInt(0,y-1):
142.        #         max_sub_length = max(0, min(i+1, chain_length-i)-2)
143.        #         sub_chain = []
144.        #         upNotDown = random.getrandbits(1)
145.        #         for s in range(max_sub_length):
146.        #           newCarbon = None
147.        #           if upNotDown:
148.        #             newCarbon = Carbon(x, y+s+1)
149.        #           else:
150.        #             newCarbon = Carbon(x, y-s-1)
151.        #           sub_chain.append(Carbon)
152.        #         alkane.addSubstituent(Substituent(sub_chain))
153.        #         alkane.carbons.extend(sub_chain)
```

chemistry/carbon.py

```
1.  from chemistry.chem_exceptions import *
2.
3.  class Carbon:
4.      """
5.      A class used to represent a carbon in an alkane molecule
6.      """
7.
8.      #Ordered such that directions[i] opposes directions[3-i]
9.      @staticmethod
10.     def directions():
11.         return ('north', 'east', 'west', 'south')
12.
13.     @staticmethod
14.     def getDirectionsWithout(ignoreDirection):
15.         dirs = ['north', 'east', 'west', 'south']
16.         if ignoreDirection:
17.             dirs.remove(ignoreDirection)
18.         return tuple(dirs)
19.
20.     @staticmethod
21.     def getOpposingDirection(direction):
22.         return Carbon.directions()[3-Carbon.directions().index(direction)]
23.
24.     @staticmethod
25.     def bondCarbons(carbons):
26.         for i in range(len(carbons)):
27.             for j in range(i+1, len(carbons)):
28.                 try:
29.                     carbons[i].bondTo(carbons[j])
30.                 except:
31.                     pass
32.
33.     @staticmethod
34.     def dissolveCarbons(carbons):
35.         for c in carbons:
36.             for direction in Carbon.directions():
37.                 setattr(c, direction, None)
38.
39.     def __init__(self, x, y):
40.         self.north = self.south = self.east = self.west = None
41.         self.x = x
```

```
42.         self.y = y
43.
44.
45.     def __str__(self):
46.         result = super().__str__()
47.         result += "\nX: %d Y: %d\n" % (self.x, self.y)
48.         for direction in Carbon.directions():
49.             result += " "+direction+": %s\n" % ("yes " if getattr(self, direction, None) else "no ")
50.         return result
51.
52.     #Returns longest chain of carbons that includes this carbon,
53.     #with this carbon as the first one in the list
54.     def getLongestChain(self):
55.         return self._getLongestChain()
56.
57.     #Recursive helper method
58.     def _getLongestChain(self, ignoreDirection=None):
59.         #Append this carbon to the chain
60.         chain = [self]
61.         #List of carbon chains that are valid candidates for longest chain
62.         validChains = []
63.         for direction in Carbon.getDirectionsWithout(ignoreDirection):
64.             nextCarbon = getattr(self, direction)
65.             if nextCarbon:
66.                 opposingDirection = Carbon.getOpposingDirection(direction)
67.                 validChains.append(nextCarbon._getLongestChain(opposingDirection))
68.         if any(validChains):
69.             #Extend our chain with the longest chain we found
70.             chain.extend(max(validChains, key=len))
71.         return chain
72.
73.     def getConnectedSet(self):
74.         return self._getConnectedSet(carbonSet = None)
75.
76.     #Helper recursive method for isConnectedHasCycles
77.     #TODO: Fix this method to clear the set if carbon is deleted
78.     def _getConnectedSet(self, carbonSet, ignoreDirection=None):
79.         if not carbonSet:
80.             carbonSet = set()
81.         if self in carbonSet:
82.             raise CyclicAlkaneError()
83.         else:
```

```
84.             carbonSet.add(self)
85.         for direction in Carbon.getDirectionsWithout(ignoreDirection):
86.             nextCarbon = getattr(self, direction, None)
87.             if nextCarbon:
88.                 opposingDirection = Carbon.getOpposingDirection(direction)
89.                 nextCarbon._getConnectedSet(carbonSet, opposingDirection)
90.         return carbonSet
91.
92.     def getNumberOfBonds(self):
93.         num = 0
94.         for direction in Carbon.directions():
95.             if getattr(self,direction, None):
96.                 num+=1
97.         return num
98.
99.     def getDirectionTo(self, carbon):
100.                 for direction in Carbon.directions():
101.                     if getattr(self, direction, None) == carbon:
102.                         return direction
103.                 return None
104.
105.
106.             def bondTo(self, other):
107.                 #Are these carbons the same or in the same position?
108.                 if self == other or self.x == other.x and self.y == other.y:
109.                     raise CarbonsNotAdjacentError
110.                 #Are we not on the same column and not on the same row?
111.                 if self.x != other.x and self.y != other.y:
112.                     raise CarbonsNotAdjacentError
113.                 else:
114.                     #Are we on the same row?
115.                     if self.y == other.y:
116.                         #Are we one space apart?
117.                         if abs(self.x - other.x) == 1:
118.                             #Am I on the left?
119.                             if self.x < other.x:
120.                                 #Bond the carbons together appropriately
121.                                 self.east = other
122.                                 other.west = self
123.                             #I am on the right
124.                             else:
125.                                 #Bond the carbons together appropriately
```

```
126.                     self.west = other
127.                     other.east = self
128.                 #We are too far apart
129.                 else:
130.                     raise CarbonsNotAdjacentError
131.             #We are on the same column
132.             else:
133.                 #Are we one space apart?
134.                 if abs(self.y - other.y) == 1:
135.                     #Am I on top?
136.                     if self.y < other.y:
137.                         #Bond the carbons together appropriately
138.                         self.south = other
139.                         other.north = self
140.                     #I am on the bottom
141.                     else:
142.                         #Bond the carbons together appropriately
143.                         self.north = other
144.                         other.south = self
145.                 #We are too far apart
146.                 else:
147.                     raise CarbonsNotAdjacentError
148.
149.         #Returns a list of carbons from self outward in direction, not including self
150.         #Throws AmbiguousBranchingPath if any Carbon in the sequence has
151.         #more than two connected Carbons
152.         def getCarbonsInDirection(self, direction):
153.             result = None
154.             try:
155.                 result = self._getCarbonsInDirection(direction)
156.             #Was there a branching, ambiguous path?
157.             except BranchingCarbonChainError as error:
158.                 #Pass the error along to the caller
159.                 raise error
160.             #Return the result from the helper method without the first element,
161.             #which will always equal this
162.             return result[1:]
163.
164.         #Recursive helper method that returns chain in direction, plus original carbon
165.         def _getCarbonsInDirection(self, direction, ignoreDirection=None):
166.             chain = [self]
167.             #Was this the top-level call?
```

```
168.                if direction:
169.                    nextCarbon = getattr(self, direction, None)
170.                    if nextCarbon:
171.                        opposingDirection = Carbon.getOpposingDirection(direction)
172.                        #search for more carbons and add them to our chain
173.
    chain.extend(nextCarbon._getCarbonsInDirection(None, ignoreDirection=opposingDirection))
174.                    return chain
175.                else:
176.                    numBonds = self.getNumberOfBonds()
177.                    #Is our only bond the one we came here on, hence a dead end?
178.                    if numBonds == 1:
179.                        return chain
180.                    #Do we have an extra bond to find more carbons on?
181.                    elif numBonds == 2:
182.                        for direction in Carbon.getDirectionsWithout(ignoreDirection):
183.                            nextCarbon = getattr(self, direction, None)
184.                            if nextCarbon:
185.                                opposingDirection = Carbon.getOpposingDirection(direction)
186.
    chain.extend(nextCarbon._getCarbonsInDirection(None, opposingDirection))
187.                    #We have too many branches
188.                    else:
189.                        #So throw an exception
190.                        raise BranchingCarbonChainError()
191.                return chain
192.
193.
194.            def getDistanceTo(self, target, ignoreDirection=None):
195.                count = 1
196.                if self == target:
197.                    return 0
198.                else:
199.                    #List of distance results from recursive calls
200.                    distances = []
201.                    for direction in Carbon.getDirectionsWithout(ignoreDirection):
202.                        nextCarbon = getattr(self, direction)
203.                        if nextCarbon:
204.                            opposingDirection = Carbon.getOpposingDirection(direction)
205.                            distance = nextCarbon.getDistanceTo(target, opposingDirection)
206.                            if distance > -1:
207.                                distances.append(distance)
```

```
208.                #Did we find our target?
209.                if len(distances):
210.                    #Add the shortest distance to the target to our count
211.                    count += min(distances)
212.                #We didn't find our target
213.                else:
214.                    #So return -1
215.                    count = -1
216.            return count
```

chemistry/chem_exceptions.py

```
1.  class AlkaneNotConnectedError(Exception):
2.      def __init__(self):
3.          super(Exception, self).__init__()
4.  class EmptyAlkaneError(Exception):
5.      def __init__(self):
6.          super(Exception, self).__init__()
7.
8.  class CyclicAlkaneError(Exception):
9.      def __init__(self):
10.         super(Exception, self).__init__()
11.
12. class BranchingCarbonChainError(Exception):
13.     def __init__(self):
14.         super(Exception, self).__init__()
15.
16. class NotASubstituentError(Exception):
17.     def __init__(self):
18.         super(Exception, self).__init__()
19.
20. class CarbonsNotAdjacentError(Exception):
21.     def __init__(self):
22.         super(Exception, self).__init__()
```

chemistry/substituent.py

```
1.  from chemistry.carbon import Carbon
2.  class Substituent():
3.
4.      names = ('methyl','ethyl','propyl','butyl','pentyl')
5.      MAX_SUB_LENGTH = len(names)
6.
7.      def __init__(self, carbon, direction, index):
8.          #Throws BranchingCarbonChainError for malformed substituents
9.          self.carbons=carbon.getCarbonsInDirection(direction)
10.         self.index=index
11.
12.     def getName(self):
13.         carbon_length = len(self.carbons)
14.         return "%d-%s" % (self.index, Substituent.names[carbon_length-1])
15.
16.     #Return the first carbon in the sequence, the one adjacent to the longest chain
17.     def getHead(self):
18.         return self.carbons[0]
```

data/launch.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>635</width>
    <height>560</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Chem Wizard</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="randomButton">
    <property name="geometry">
     <rect>
      <x>30</x>
      <y>20</y>
      <width>151</width>
      <height>91</height>
     </rect>
    </property>
    <property name="text">
     <string>Random</string>
    </property>
   </widget>
   <widget class="QLabel" name="smilesLabel">
    <property name="geometry">
     <rect>
      <x>20</x>
      <y>160</y>
      <width>51</width>
      <height>16</height>
     </rect>
    </property>
    <property name="text">
     <string>SMILES:</string>
    </property>
```

```
 </widget>
 <widget class="QLabel" name="inchiLabel">
  <property name="geometry">
   <rect>
    <x>240</x>
    <y>150</y>
    <width>41</width>
    <height>17</height>
   </rect>
  </property>
  <property name="text">
   <string>InChi</string>
  </property>
 </widget>
 <widget class="QPlainTextEdit" name="smilesBox">
  <property name="geometry">
   <rect>
    <x>70</x>
    <y>150</y>
    <width>141</width>
    <height>31</height>
   </rect>
  </property>
 </widget>
 <widget class="QPlainTextEdit" name="inchiBox">
  <property name="geometry">
   <rect>
    <x>280</x>
    <y>150</y>
    <width>141</width>
    <height>31</height>
   </rect>
  </property>
 </widget>
 <widget class="QPushButton" name="animateButton">
  <property name="geometry">
   <rect>
    <x>520</x>
    <y>40</y>
    <width>81</width>
    <height>41</height>
   </rect>
```

```
 </property>
 <property name="text">
  <string>Animate</string>
 </property>
</widget>
<widget class="QLabel" name="nomenclatureLabel">
 <property name="geometry">
  <rect>
   <x>210</x>
   <y>20</y>
   <width>111</width>
   <height>17</height>
  </rect>
 </property>
 <property name="text">
  <string>Nomenclature:</string>
 </property>
</widget>
<widget class="QPlainTextEdit" name="nomenclatureBox">
 <property name="geometry">
  <rect>
   <x>210</x>
   <y>40</y>
   <width>281</width>
   <height>41</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="checkButton">
 <property name="geometry">
  <rect>
   <x>210</x>
   <y>90</y>
   <width>87</width>
   <height>27</height>
  </rect>
 </property>
 <property name="text">
  <string>Check!</string>
 </property>
</widget>
<widget class="QLabel" name="checkLabel">
```

```
126.            <property name="geometry">
127.             <rect>
128.              <x>510</x>
129.              <y>50</y>
130.              <width>101</width>
131.              <height>31</height>
132.             </rect>
133.            </property>
134.            <property name="text">
135.             <string/>
136.            </property>
137.           </widget>
138.           <widget class="QGraphicsView" name="graphicsView">
139.            <property name="geometry">
140.             <rect>
141.              <x>20</x>
142.              <y>200</y>
143.              <width>600</width>
144.              <height>300</height>
145.             </rect>
146.            </property>
147.            <property name="sizePolicy">
148.             <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
149.              <horstretch>0</horstretch>
150.              <verstretch>0</verstretch>
151.             </sizepolicy>
152.            </property>
153.            <property name="minimumSize">
154.             <size>
155.              <width>600</width>
156.              <height>300</height>
157.             </size>
158.            </property>
159.            <property name="maximumSize">
160.             <size>
161.              <width>600</width>
162.              <height>300</height>
163.             </size>
164.            </property>
165.            <property name="verticalScrollBarPolicy">
166.             <enum>Qt::ScrollBarAlwaysOff</enum>
167.            </property>
```

```
168.            <property name="horizontalScrollBarPolicy">
169.             <enum>Qt::ScrollBarAlwaysOff</enum>
170.            </property>
171.            <property name="backgroundBrush">
172.             <brush brushstyle="NoBrush">
173.              <color alpha="255">
174.               <red>255</red>
175.               <green>255</green>
176.               <blue>255</blue>
177.              </color>
178.             </brush>
179.            </property>
180.            <property name="foregroundBrush">
181.             <brush brushstyle="SolidPattern">
182.              <color alpha="255">
183.               <red>0</red>
184.               <green>0</green>
185.               <blue>0</blue>
186.              </color>
187.             </brush>
188.            </property>
189.            <property name="sceneRect">
190.             <rectf>
191.              <x>0.000000000000000</x>
192.              <y>0.000000000000000</y>
193.              <width>600.000000000000000</width>
194.              <height>300.000000000000000</height>
195.             </rectf>
196.            </property>
197.           </widget>
198.           <widget class="QPushButton" name="clearButton">
199.            <property name="geometry">
200.             <rect>
201.              <x>310</x>
202.              <y>90</y>
203.              <width>51</width>
204.              <height>27</height>
205.             </rect>
206.            </property>
207.            <property name="text">
208.             <string>Clear</string>
209.            </property>
```

```
210.          </widget>
211.          <widget class="QPushButton" name="validateButton">
212.           <property name="geometry">
213.            <rect>
214.             <x>370</x>
215.             <y>90</y>
216.             <width>87</width>
217.             <height>27</height>
218.            </rect>
219.           </property>
220.           <property name="text">
221.            <string>Validate</string>
222.           </property>
223.          </widget>
224.          <widget class="QLabel" name="guessResultLabel">
225.           <property name="enabled">
226.            <bool>true</bool>
227.           </property>
228.           <property name="geometry">
229.            <rect>
230.             <x>210</x>
231.             <y>120</y>
232.             <width>111</width>
233.             <height>17</height>
234.            </rect>
235.           </property>
236.           <property name="visible">
237.            <bool>false</bool>
238.           </property>
239.           <property name="text">
240.            <string>Correct!/Wrong</string>
241.           </property>
242.          </widget>
243.          <widget class="QPushButton" name="renderButton">
244.           <property name="geometry">
245.            <rect>
246.             <x>470</x>
247.             <y>130</y>
248.             <width>98</width>
249.             <height>27</height>
250.            </rect>
251.           </property>
```

```
252.              <property name="text">
253.               <string>Render</string>
254.              </property>
255.             </widget>
256.            </widget>
257.            <widget class="QMenuBar" name="menubar">
258.             <property name="geometry">
259.              <rect>
260.               <x>0</x>
261.               <y>0</y>
262.               <width>635</width>
263.               <height>25</height>
264.              </rect>
265.             </property>
266.            </widget>
267.            <widget class="QStatusBar" name="statusbar"/>
268.           </widget>
269.           <resources/>
270.           <connections/>
271.          </ui>
```

network/__init__.py

(empty, but required)

network/cactusAPI.py

```
1.  # -*- coding: utf-8 -*-
2.
3.  """
4.  CactusAPI
5.
6.  Simple Python wrapper for the Cactus API.
7.  Used ChemSpiPy for inspiration/guidance
8.
9.  """
10.
11. import urllib.request
12.
13. __author__ = 'Roland Heintze'
14. __email__ = 'rheintze@linux.com'
15. __version__ = '1.0'
16.
17. def get_csid(query):
18.     """ searches for a csid based on search query, smiles/inchi preferred """
19.
    searchurl = 'http://cactus.nci.nih.gov/chemical/structure/%s/chemspider_id' % (urllib.parse.quo
    te(query))
20.     response = str(urllib.request.urlopen(searchurl).readline().strip())[2:-1]
21.
22.     return response
23. def get_iupac(smiles_query):
24.     """  uses smiles to pull compound's IUPAC from cactus """
25.
26.     assert type(smiles_query) == str or type(smiles_query) == str, 'query not a string object'
27.
    searchurl = 'http://cactus.nci.nih.gov/chemical/structure/%s/iupac_name' % (urllib.parse.quote(
    smiles_query))
28.     response = str(urllib.request.urlopen(searchurl).read())[2:-1]
29.
30.     return response
31.
32.
33. if __name__ == '__main__':
34.     print(get_csid('C'))
35.     print(get_iupac('C'))
36.     print(get_csid('CC(C)CC'))
37.     print(get_iupac('CC(C)CC'))
```

```
38.  print(get_csid('CC(C)(C)C'))
39.  print(get_iupac('CC(C)(C)C'))
```

network/chemspipy.apy

```
# -*- coding: utf-8 -*-
#modified by 2to3.py
"""
ChemSpiPy

Python wrapper for the ChemSpider API.
https://github.com/mcs07/ChemSpiPy

Forked from ChemSpiPy by Cameron Neylon
https://github.com/cameronneylon/ChemSpiPy
"""

import urllib.request, urllib.error, urllib.parse
from xml.etree import ElementTree as ET

# adds IUPAC Nomenclature as an attribute of the Compound class
# pulled from http://cactus.nci.nih.gov/chemical/structure API
from cactusAPI import get_iupac


__author__ = 'Matt Swain'
__email__ = 'm.swain@me.com'
__version__ = '1.0'
__license__ = 'MIT'


TOKEN = '78eed73d-ef86-4479-985a-463d7fba9d57'


class Compound(object):
    """ A class for retrieving record details about a compound by CSID.

    The purpose of this class is to provide access to various parts of the
    ChemSpider API that return information about a compound given its CSID.
    Information is loaded lazily when requested, and cached for future access.
    """

    def __init__(self,csid):
        """ Initialize with a CSID as an int or str """
        if type(csid) is str and csid.isdigit():
            self.csid = csid
```

```
42.         elif type(csid) == int:
43.             self.csid = str(csid)
44.         else:
45.             raise TypeError('Compound must be initialised with a CSID as an int or str')
46.
47.         self._imageurl = None
48.         self._mf = None
49.         self._smiles = None
50.         self._inchi = None
51.         self._inchikey = None
52.         self._averagemass = None
53.         self._molecularweight = None
54.         self._monoisotopicmass = None
55.         self._nominalmass = None
56.         self._alogp = None
57.         self._xlogp = None
58.         self._commonname = None
59.         self._image = None
60.         self._mol = None
61.         self._mol3d = None
62.         self._iupac = None
63.
64.     def __repr__(self):
65.         return "Compound(%r)" % self.csid
66.
67.     @property
68.     def imageurl(self):
69.         """ Return the URL of a png image of the 2D structure """
70.         if self._imageurl is None:
71.             self._imageurl = 'http://www.chemspider.com/ImagesHandler.ashx?id=%s' % self.csid
72.         return self._imageurl
73.
74.     @property
75.     def mf(self):
76.         """ Retrieve molecular formula from ChemSpider """
77.         if self._mf is None:
78.             self.loadextendedcompoundinfo()
79.         return self._mf
80.
81.     @property
82.     def smiles(self):
83.         """ Retrieve SMILES string from ChemSpider """
```

```
84.         if self._smiles is None:
85.             self.loadextendedcompoundinfo()
86.         return self._smiles
87.
88.     @property
89.     def inchi(self):
90.         """ Retrieve InChi string from ChemSpider """
91.         if self._inchi is None:
92.             self.loadextendedcompoundinfo()
93.         return self._inchi
94.
95.     @property
96.     def inchikey(self):
97.         """ Retrieve InChi string from ChemSpider """
98.         if self._inchikey is None:
99.             self.loadextendedcompoundinfo()
100.                return self._inchikey
101.
102.            @property
103.            def averagemass(self):
104.                """ Retrieve average mass from ChemSpider """
105.                if self._averagemass is None:
106.                    self.loadextendedcompoundinfo()
107.                return self._averagemass
108.
109.            @property
110.            def molecularweight(self):
111.                """ Retrieve molecular weight from ChemSpider """
112.                if self._molecularweight is None:
113.                    self.loadextendedcompoundinfo()
114.                return self._molecularweight
115.
116.            @property
117.            def monoisotopicmass(self):
118.                """ Retrieve monoisotropic mass from ChemSpider """
119.                if self._monoisotopicmass is None:
120.                    self.loadextendedcompoundinfo()
121.                return self._monoisotopicmass
122.
123.            @property
124.            def nominalmass(self):
125.                """ Retrieve nominal mass from ChemSpider """
```

```
126.                 if self._nominalmass is None:
127.                     self.loadextendedcompoundinfo()
128.                 return self._nominalmass
129.
130.             @property
131.             def alogp(self):
132.                 """ Retrieve ALogP from ChemSpider """
133.                 if self._alogp is None:
134.                     self.loadextendedcompoundinfo()
135.                 return self._alogp
136.
137.             @property
138.             def xlogp(self):
139.                 """ Retrieve XLogP from ChemSpider """
140.                 if self._xlogp is None:
141.                     self.loadextendedcompoundinfo()
142.                 return self._xlogp
143.
144.             @property
145.             def commonname(self):
146.                 """ Retrieve common name from ChemSpider """
147.                 if self._commonname is None:
148.                     self.loadextendedcompoundinfo()
149.                 return self._commonname
150.
151.             @property
152.             def iupac(self):
153.                 """ Retrieve the IUPAC Nomenclature from the cactus API """
154.                 if self._iupac is None:
155.                     self.loadextendedcompoundinfo()
156.                 return self._iupac
157.
158.             def loadextendedcompoundinfo(self):
159.                 """ Load extended compound info from the Mass Spec API """
160.
    apiurl = 'http://www.chemspider.com/MassSpecAPI.asmx/GetExtendedCompoundInfo?CSID=%s
    &token=%s' %(self.csid,TOKEN)
161.                 response = urllib.request.urlopen(apiurl)
162.                 tree = ET.parse(response)
163.                 mf = tree.find('{http://www.chemspider.com/}MF')
164.                 self._mf = mf.text if mf is not None else None
165.                 smiles = tree.find('{http://www.chemspider.com/}SMILES')
```

```
166.                self._smiles = smiles.text if smiles is not None else None
167.                inchi = tree.find('{http://www.chemspider.com/}InChI')
168.                self._inchi = inchi.text if inchi is not None else None
169.                inchikey = tree.find('{http://www.chemspider.com/}InChIKey')
170.                self._inchikey = inchikey.text if inchikey is not None else None
171.                averagemass = tree.find('{http://www.chemspider.com/}AverageMass')
172.                self._averagemass = float(averagemass.text) if averagemass is not None else None
173.                molecularweight = tree.find('{http://www.chemspider.com/}MolecularWeight')
174.
      self._molecularweight = float(molecularweight.text) if molecularweight is not None else None
175.                monoisotopicmass = tree.find('{http://www.chemspider.com/}MonoisotopicMass')
176.
      self._monoisotopicmass = float(monoisotopicmass.text) if monoisotopicmass is not None else
    None
177.                nominalmass = tree.find('{http://www.chemspider.com/}NominalMass')
178.                self._nominalmass = float(nominalmass.text) if nominalmass is not None else None
179.                alogp = tree.find('{http://www.chemspider.com/}ALogP')
180.                self._alogp = float(alogp.text) if alogp is not None else None
181.                xlogp = tree.find('{http://www.chemspider.com/}XLogP')
182.                self._xlogp = float(xlogp.text) if xlogp is not None else None
183.                commonname = tree.find('{http://www.chemspider.com/}CommonName')
184.                self._commonname = commonname.text if commonname is not None else None
185.
186.                #custom field for IUPAC Nomenclature format. Pulls from Cactus API
187.                self._iupac = get_iupac(self.smiles)
188.
189.            @property
190.            def image(self):
191.                """ Return string containing PNG binary image data of 2D structure image """
192.                if self._image is None:
193.
    apiurl = 'http://www.chemspider.com/Search.asmx/GetCompoundThumbnail?id=%s&token=%s'
     %(self.csid,TOKEN)
194.                    response = urllib.request.urlopen(apiurl)
195.                    tree = ET.parse(response)
196.                    self._image = tree.getroot().text
197.                return self._image
198.
199.            @property
200.            def mol(self):
201.                """ Return record in MOL format """
202.                if self._mol is None:
```

```
203.
    apiurl = 'http://www.chemspider.com/MassSpecAPI.asmx/GetRecordMol?csid=%s&calc3d=false
    &token=%s' %(self.csid,TOKEN)
204.                 response = urllib.request.urlopen(apiurl)
205.                 tree = ET.parse(response)
206.                 self._mol = tree.getroot().text
207.             return self._mol
208.
209.         @property
210.         def mol3d(self):
211.             """ Return record in MOL format with 3D coordinates calculated """
212.             if self._mol3d is None:
213.
    apiurl = 'http://www.chemspider.com/MassSpecAPI.asmx/GetRecordMol?csid=%s&calc3d=true
    &token=%s' %(self.csid,TOKEN)
214.                 response = urllib.request.urlopen(apiurl)
215.                 tree = ET.parse(response)
216.                 self._mol3d = tree.getroot().text
217.             return self._mol3d
218.
219.     def find(query):
220.         """ Search by Name, SMILES, InChI, InChIKey, etc. Returns first 100 Compounds """
221.         assert type(query) == str or type(query) == str, 'query not a string object'
222.
    searchurl = 'http://www.chemspider.com/Search.asmx/SimpleSearch?query=%s&token=%s' %(u
    rllib.parse.quote(query), TOKEN)
223.         response = urllib.request.urlopen(searchurl)
224.         tree = ET.parse(response)
225.         elem = tree.getroot()
226.         csid_tags = elem.getiterator('{http://www.chemspider.com/}int')
227.         compoundlist = []
228.         for tag in csid_tags:
229.             compoundlist.append(Compound(tag.text))
230.         return compoundlist if compoundlist else None
231.
232.     def find_one(query):
233.         """ Search by Name, SMILES, InChI, InChIKey, etc. Returns a single Compound """
234.         compoundlist = find(query)
235.         return compoundlist[0] if compoundlist else None
```

network/chemspipy_LICENSE.txt

The MIT License

Copyright (c) 2013 Matt Swain <m.swain@me.com>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

ui/__init__.py

(empty, but required)

ui/carbon_view.py

```
1.  from PyQt4 import QtCore, QtGui
2.
3.  class CarbonView(QtGui.QGraphicsView):
4.      def __init__(self, parent):
5.          QtGui.QWidget.__init__(self,parent)
6.          self.width = 600
7.          self.height = 300
8.
9.          self.num_rows = 10
10.         self.num_cols = 20
11.
12.         #Size of each square
13.         self.col_width = self.width / self.num_cols
14.         self.row_height = self.height / self.num_rows
15.
16.         #2D Array to remember which squares have carbons in them
17.         self.carbonMatrix = []
18.         for col in range(0, self.num_cols):
19.             self.carbonMatrix.append([])
20.             for row in range(0, self.num_rows):
21.                 self.carbonMatrix[col].append(None)
22.
23.         self.setScene(QtGui.QGraphicsScene(self.parent()))
24.         self.scene = self.scene()
25.         self.scene.setSceneRect(0,0,600,300)
26.
27.         #Load and resize image to fit squarely in the grid
28.
        self.carbonImage = QtGui.QPixmap('../data/carbon.png').scaled(self.col_width, self.row_height
    )
29.         self.addGridToScene()
30.
31.     def addGridToScene(self):
32.         #Set up pen
33.         pen = QtGui.QPen(QtCore.Qt.black, .3, QtCore.Qt.SolidLine)
34.         #Draw Horizontal Lines
35.         for y in range(0, self.num_rows-1): # don't draw the two outside edge lines
36.             self.scene.addLine(0, (y+1)*self.row_height, self.width, (y+1)*self.row_height, pen)
37.         #Draw Vertical Lines
38.         for x in range(0, self.num_cols-1): # don't draw the two outside edge lines
39.             self.scene.addLine((x+1)*self.col_width, 0, (x+1)*self.col_width, self.height, pen)
```

```
40.
41.    def drawCarbons(self, qp):
42.        for row in range(0, self.num_rows):
43.            for col in range(0, self.num_cols):
44.                #Is there a carbon at this logical position to draw?
45.                if self.carbonMatrix[row][col]:
46.                    #Get the transformed graphical coordinates
47.                    coords = self.logicalToScreen((row, col))
48.                    #Draw the carbon image at
49.                    qp.drawImage(coords[0], coords[1], self.carbonImage)
50.
51.    #Add or remove carbons based on selected square status
52.    def mousePressEvent(self,event):
53.        #Convert global QT coordinates to logical coordinates
54.        x,y = self.screenToLogical(event.pos())
55.        #Is this square empty?
56.        if not self.carbonMatrix[x][y]:
57.            #Place a carbon in this square
58.            self.carbonMatrix[x][y]=self.scene.addPixmap(self.carbonImage)
59.            #Get top-left screen coordinates of this square
60.            screen_pos = self.logicalToScreen((x,y))
61.            #Add carbon image to screen
62.            self.carbonMatrix[x][y].setOffset(*screen_pos)
63.        else:
64.            #Remove the carbon from this square
65.            self.scene.removeItem(self.carbonMatrix[x][y])
66.            #Remove carbon image from scene
67.            self.carbonMatrix[x][y]=None
68.
69.    #Return (x,y) in logical, safe carbonMatrix coordinates
70.    def screenToLogical(self, position):
71.        x = int(position.x()/self.col_width)
72.        y = int(position.y()/self.row_height)
73.        #Clamp logical coordinates to logical boundaries
74.        x = max(x, 0)
75.        x = min(x, self.num_cols-1)
76.
77.        y = max(y, 0)
78.        y = min(y, self.num_rows-1)
79.
80.        return (x, y)
81.
```

```
82.    #Return (x,y) in top-left corner screen coordinates
83.    def logicalToScreen(self, tuple):
84.        return (int(tuple[0]*self.col_width), int(tuple[1]*self.row_height))
85.
86.    def clearCarbons(self):
87.        del self.carbonMatrix
88.        #Reset carbonMatrix to remove logical carbons
89.        self.carbonMatrix = []
90.        for col in range(0, self.num_cols):
91.            self.carbonMatrix.append([])
92.            for row in range(0, self.num_rows):
93.                self.carbonMatrix[col].append(None)
94.        #Remove graphical carbons
95.        for item in self.scene.items():
96.            if type(item) is QtGui.QGraphicsPixmapItem:
97.                self.scene.removeItem(item)
98.
99.    def getCarbonMatrix(self):
100.                return self.carbonMatrix
101.
102.            def setScreenToAlkane(self, alkane):
103.                self.clearCarbons()
104.                carbons = alkane.getCarbons()
105.                for element in carbons:
106.                    x = element.x
107.                    y = element.y
108.                    self.carbonMatrix[x][y]=self.scene.addPixmap(self.carbonImage)
109.                    #Get top-left screen coordinates of this square
110.                    screen_pos = self.logicalToScreen((x,y))
111.                    #Add carbon image to screen
112.                    self.carbonMatrix[x][y].setOffset(*screen_pos)
113.
114.            def makeRandom(self):
115.                #Make a random carbonMatrix
116.                pass
117.
118.            def animate(self):
119.                #Start the show
120.                pass
```

ui/main.py

```
1.  __author__ = 'rheintze'
2.
3.  import sys, traceback
4.  from PyQt4 import QtCore, QtGui
5.  from ui import Ui_MainWindow #pulls from ui.py which can be created from .ui xml file using
    the pyuic tool
6.
7.  from chemistry.alkane import Alkane
8.  from carbon_view import CarbonView
9.  from AnimateView import AnimateView
10.
11. class StartQT4(QtGui.QMainWindow):
12.
13.     def __init__(self, parent=None):
14.         QtGui.QWidget.__init__(self,parent)
15.         self.ui = Ui_MainWindow()
16.         self.ui.setupUi(self)
17.         self.view = CarbonView(self.ui.graphicsView)
18.
19.         self.molecule = None
20.
21.         #function bindings
22.         QtCore.QObject.connect(self.ui.validateButton,QtCore.SIGNAL("clicked()"),self.validate)
23.         QtCore.QObject.connect(self.ui.clearButton,QtCore.SIGNAL("clicked()"),self.clearMolecule)
24.         QtCore.QObject.connect(self.ui.checkButton,QtCore.SIGNAL("clicked()"),self.checkGuess)
25.         QtCore.QObject.connect(self.ui.animateButton,QtCore.SIGNAL("clicked()"),self.animate)
26.
    QtCore.QObject.connect(self.ui.randomButton,QtCore.SIGNAL("clicked()"),self.makeRandom)
27.
    QtCore.QObject.connect(self.ui.renderButton,QtCore.SIGNAL("clicked()"),self.renderAlkane)
28.
29.     def validate(self):
30.         carbonMatrix = self.view.getCarbonMatrix()
31.         try:
32.             self.molecule = Alkane(carbonMatrix)
33.         except Exception as error:
34.             print("Validation Error:")
35.             print(error)
36.             print(error.__class__)
37.             exc_traceback = sys.exc_info()[2]
38.             traceback.print_tb(exc_traceback, limit=100, file=sys.stdout)
```

```
39.
40.     def clearMolecule(self):
41.         self.view.clearCarbons()
42.         for carbon in self.molecule.carbons:
43.             del carbon
44. #       self.molecule = None
45.
46.     def checkGuess(self):
47.         try:
48.             self.ui.guessResultLabel.setVisible(True)
49.             self.molecule.verify(self.ui.nomenclatureBox.toPlainText)
50.             if self.molecule.verify(self.ui.nomenclatureBox.toPlainText):
51.                 self.ui.guessResultLabel.setText("Correct!")
52.             else:
53.                 self.ui.guessResultLabel.setText("Incorrect")
54.         except Exception as error:
55.             print("Guess not made or molecule does not exist")
56.             print(error)
57.             print(error.__class__)
58.             exc_type, exc_value, exc_traceback = sys.exc_info()
59.             traceback.print_tb(exc_traceback, limit=100, file=sys.stdout)
60.
61.     def renderAlkane(self):
62.         self.view.setScreenToAlkane(self.molecule)
63.
64.     def makeRandom(self):
65.         #TODO: Make and display random Alkane
66.         pass
67.
68.     def animate(self):
69.         self.view = AnimateView(self.ui.graphicsView)
70.         self.view.passAlkane(self.molecule)
71.
72. if __name__ == '__main__':
73.     import sys
74.     app = QtGui.QApplication(sys.argv)
75.     myapp = StartQT4()
76.
77.     myapp.show()
78.     sys.exit(app.exec_())
```

ui/ui.py

```
1.  # -*- coding: utf-8 -*-
2.
3.  # Form implementation generated from reading ui file 'launch.ui'
4.  #
5.  # Created: Wed May  1 11:19:46 2013
6.  #      by: PyQt4 UI code generator 4.9.1
7.  #
8.  # WARNING! All changes made in this file will be lost!
9.
10. from PyQt4 import QtCore, QtGui
11.
12. try:
13.     _fromUtf8 = QtCore.QString.fromUtf8
14. except AttributeError:
15.     _fromUtf8 = lambda s: s
16.
17. class Ui_MainWindow(object):
18.     def setupUi(self, MainWindow):
19.         MainWindow.setObjectName(_fromUtf8("MainWindow"))
20.         MainWindow.resize(635, 560)
21.         self.centralwidget = QtGui.QWidget(MainWindow)
22.         self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
23.         self.randomButton = QtGui.QPushButton(self.centralwidget)
24.         self.randomButton.setGeometry(QtCore.QRect(30, 20, 151, 91))
25.         self.randomButton.setObjectName(_fromUtf8("randomButton"))
26.         self.smilesLabel = QtGui.QLabel(self.centralwidget)
27.         self.smilesLabel.setGeometry(QtCore.QRect(20, 160, 51, 16))
28.         self.smilesLabel.setObjectName(_fromUtf8("smilesLabel"))
29.         self.inchiLabel = QtGui.QLabel(self.centralwidget)
30.         self.inchiLabel.setGeometry(QtCore.QRect(240, 150, 41, 17))
31.         self.inchiLabel.setObjectName(_fromUtf8("inchiLabel"))
32.         self.smilesBox = QtGui.QPlainTextEdit(self.centralwidget)
33.         self.smilesBox.setGeometry(QtCore.QRect(70, 150, 141, 31))
34.         self.smilesBox.setObjectName(_fromUtf8("smilesBox"))
35.         self.inchiBox = QtGui.QPlainTextEdit(self.centralwidget)
36.         self.inchiBox.setGeometry(QtCore.QRect(280, 150, 141, 31))
37.         self.inchiBox.setObjectName(_fromUtf8("inchiBox"))
38.         self.animateButton = QtGui.QPushButton(self.centralwidget)
39.         self.animateButton.setGeometry(QtCore.QRect(520, 40, 81, 41))
40.         self.animateButton.setObjectName(_fromUtf8("animateButton"))
41.         self.nomenclatureLabel = QtGui.QLabel(self.centralwidget)
```

```
42.         self.nomenclatureLabel.setGeometry(QtCore.QRect(210, 20, 111, 17))
43.         self.nomenclatureLabel.setObjectName(_fromUtf8("nomenclatureLabel"))
44.         self.nomenclatureBox = QtGui.QPlainTextEdit(self.centralwidget)
45.         self.nomenclatureBox.setGeometry(QtCore.QRect(210, 40, 281, 41))
46.         self.nomenclatureBox.setObjectName(_fromUtf8("nomenclatureBox"))
47.         self.checkButton = QtGui.QPushButton(self.centralwidget)
48.         self.checkButton.setGeometry(QtCore.QRect(210, 90, 87, 27))
49.         self.checkButton.setObjectName(_fromUtf8("checkButton"))
50.         self.checkLabel = QtGui.QLabel(self.centralwidget)
51.         self.checkLabel.setGeometry(QtCore.QRect(510, 50, 101, 31))
52.         self.checkLabel.setText(_fromUtf8(""))
53.         self.checkLabel.setObjectName(_fromUtf8("checkLabel"))
54.         self.graphicsView = QtGui.QGraphicsView(self.centralwidget)
55.         self.graphicsView.setGeometry(QtCore.QRect(20, 200, 600, 300))
56.         sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expanding)
57.         sizePolicy.setHorizontalStretch(0)
58.         sizePolicy.setVerticalStretch(0)
59.         sizePolicy.setHeightForWidth(self.graphicsView.sizePolicy().hasHeightForWidth())
60.         self.graphicsView.setSizePolicy(sizePolicy)
61.         self.graphicsView.setMinimumSize(QtCore.QSize(600, 300))
62.         self.graphicsView.setMaximumSize(QtCore.QSize(600, 300))
63.         self.graphicsView.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
64.         self.graphicsView.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
65.         brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
66.         brush.setStyle(QtCore.Qt.NoBrush)
67.         self.graphicsView.setBackgroundBrush(brush)
68.         brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
69.         brush.setStyle(QtCore.Qt.SolidPattern)
70.         self.graphicsView.setForegroundBrush(brush)
71.         self.graphicsView.setSceneRect(QtCore.QRectF(0.0, 0.0, 600.0, 300.0))
72.         self.graphicsView.setObjectName(_fromUtf8("graphicsView"))
73.         self.clearButton = QtGui.QPushButton(self.centralwidget)
74.         self.clearButton.setGeometry(QtCore.QRect(310, 90, 51, 27))
75.         self.clearButton.setObjectName(_fromUtf8("clearButton"))
76.         self.validateButton = QtGui.QPushButton(self.centralwidget)
77.         self.validateButton.setGeometry(QtCore.QRect(370, 90, 87, 27))
78.         self.validateButton.setObjectName(_fromUtf8("validateButton"))
79.         self.guessResultLabel = QtGui.QLabel(self.centralwidget)
80.         self.guessResultLabel.setEnabled(True)
81.         self.guessResultLabel.setGeometry(QtCore.QRect(210, 120, 111, 17))
82.         self.guessResultLabel.setVisible(False)
83.         self.guessResultLabel.setObjectName(_fromUtf8("guessResultLabel"))
```

```
84.         self.renderButton = QtGui.QPushButton(self.centralwidget)
85.         self.renderButton.setGeometry(QtCore.QRect(470, 130, 98, 27))
86.         self.renderButton.setObjectName(_fromUtf8("renderButton"))
87.         MainWindow.setCentralWidget(self.centralwidget)
88.         self.menubar = QtGui.QMenuBar(MainWindow)
89.         self.menubar.setGeometry(QtCore.QRect(0, 0, 635, 25))
90.         self.menubar.setObjectName(_fromUtf8("menubar"))
91.         MainWindow.setMenuBar(self.menubar)
92.         self.statusbar = QtGui.QStatusBar(MainWindow)
93.         self.statusbar.setObjectName(_fromUtf8("statusbar"))
94.         MainWindow.setStatusBar(self.statusbar)
95.
96.         self.retranslateUi(MainWindow)
97.         QtCore.QMetaObject.connectSlotsByName(MainWindow)
98.
99.     def retranslateUi(self, MainWindow):
100.                 MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow", "Chem
    Wizard", None,QtGui.QApplication.UnicodeUTF8))
101.
     self.randomButton.setText(QtGui.QApplication.translate("MainWindow", "Random", None,Qt
    Gui.QApplication.UnicodeUTF8))
102.
     self.smilesLabel.setText(QtGui.QApplication.translate("MainWindow", "SMILES:", None,QtGui.
    QApplication.UnicodeUTF8))
103.
     self.inchiLabel.setText(QtGui.QApplication.translate("MainWindow", "InChi", None,QtGui.QAp
    plication.UnicodeUTF8))
104.
     self.animateButton.setText(QtGui.QApplication.translate("MainWindow", "Animate", None,Qt
    Gui.QApplication.UnicodeUTF8))
105.
     self.nomenclatureLabel.setText(QtGui.QApplication.translate("MainWindow", "Nomenclature:
    ", None,QtGui.QApplication.UnicodeUTF8))
106.
     self.checkButton.setText(QtGui.QApplication.translate("MainWindow", "Check!", None,QtGui.
    QApplication.UnicodeUTF8))
107.
     self.clearButton.setText(QtGui.QApplication.translate("MainWindow", "Clear", None,QtGui.QA
    pplication.UnicodeUTF8))
108.
     self.validateButton.setText(QtGui.QApplication.translate("MainWindow", "Validate", None,QtG
    ui.QApplication.UnicodeUTF8))
```

```
109.
        self.guessResultLabel.setText(QtGui.QApplication.translate("MainWindow", "Correct!/Wrong",
    None,QtGui.QApplication.UnicodeUTF8))
110.
        self.renderButton.setText(QtGui.QApplication.translate("MainWindow", "Render", None,QtGui
    .QApplication.UnicodeUTF8))
```

ui/AnimateView.py

```
1.  __author__ = 'rheintze'
2.
3.  from PyQt4 import QtCore, QtGui
4.  from Frame import Frame
5.
6.  class AnimateView(QtGui.QGraphicsView):
7.      def __init__(self,parent):
8.          QtGui.QWidget.__init__(self,parent)
9.
10.         #Used to set view size & logically separate it into a grid
11.         self.width = 600
12.         self.height = 300
13.         self.num_rows = 10
14.         self.num_cols = 20
15.
16.         #Size of each square
17.         self.col_width = self.width / self.num_cols
18.         self.row_height = self.height / self.num_rows
19.
20.         #2D Array to remember which squares have carbons in them
21.         self.carbonMatrix = []
22.         for col in range(0, self.num_cols):
23.             self.carbonMatrix.append([])
24.             for row in range(0, self.num_rows):
25.                 self.carbonMatrix[col].append(None)
26.
27.         #Creates list of frames; starting with parent's scene
28.         self.frameList = []
29.         self.frameList.append(Frame(QtGui.QGraphicsScene(self.parent())))
30.         self.index = 0
31.         self.setScene(self.frameList[self.index])
32.
33.         self.scene = self.scene()
34.         self.scene.setSceneRect(0,0,600,300)
35.
36.         #Creates next frame; clearing the screen and drawing all carbons again
37.         #Need to figure out constructor which will make a clean scene
38.         #Alternatively copy frameList & delete all items in it
39.         self.frameList.append(Frame(QtGui.QGraphicsScene()))
40.         self.advanceFrame()
41.
```

```
42.
43.     def advanceFrame(self):
44.         self.index+=1
45.         self.setScene(self.frameList[self.index])
46.         self.scene = self.scene()
47.         self.scene.setSceneRect(0,0,600,300)
48.
49.     def passAlkane(self,alkane):
50.         carbons = alkane.getCarbons()
51.         print(carbons)
```

ui/Frame.py

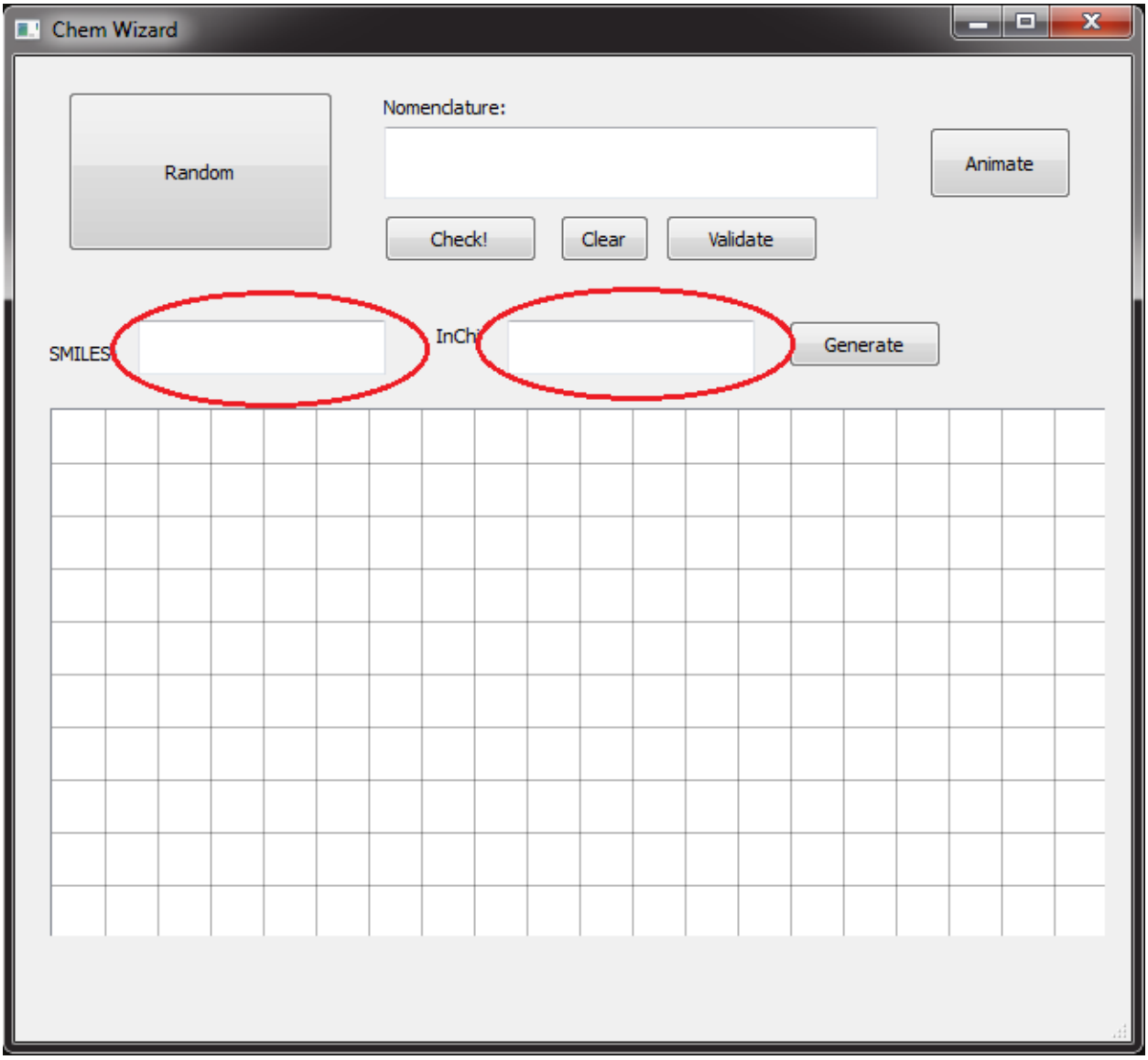
```
1.  __author__ = 'rheintze'
2.
3.  from PyQt4 import QtCore, QtGui
4.  from PyQt4.QtGui import QImage
5.  from chemistry.alkane import Alkane
6.
7.  class Frame(QtGui.QGraphicsScene):
8.      def __init__(self,parent):
9.          QtGui.QGraphicsScene.__init__(self,parent)
10.         def __init__(self):
11.             self.info = "This is help text to be displayed in the additional info box"
12.             self.type = 'start' #Used to create frames of a larger animation; ie start, intermediary, end
13.             self.items = []
14.             self.buildFrame()
15.
16.             self.carbonImage = QtGui.QPixmap('carbon.png').scaled(self.col_width, self.row_height)
17.             self.HouseImage = QtGui.QPixmap('house.png').scaled(self.col_width, self.row_height)
18.             self.RoadImage = QtGui.QPixmap('road.png').scaled(self.col_width, self.row_height)
19.
20.     def getExplanation(self):
21.         #This will build a string for the help text based on the molecule state
22.         return {
23.             'start':"The first step of the IUPAC naming convention is to find the longest path, note
    that this is x in our alkane",
24.             'road':"In our mailman analogy, this is replaced by a road",
25.         }.get(self.type,"Not a valid frame type")
26.
27.     def getDrawables(self):
28.         return self.items
29.
30.     def buildFrame(self):
31.         pass
```

10.10 User Manual

Creating a Custom Molecule

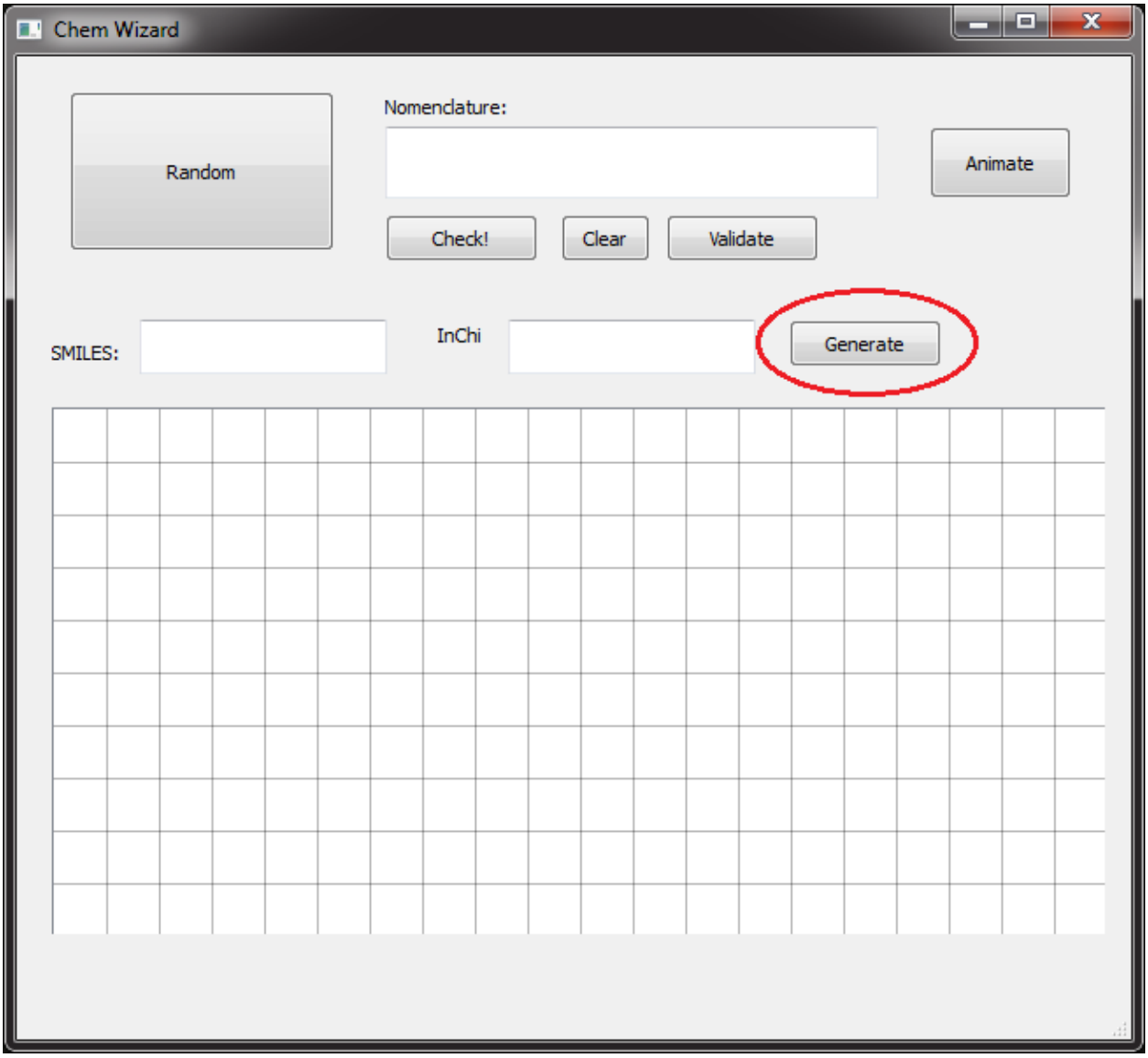
Step 1:

Option a) Enter either InChi or Smiles format in their respective boxes.



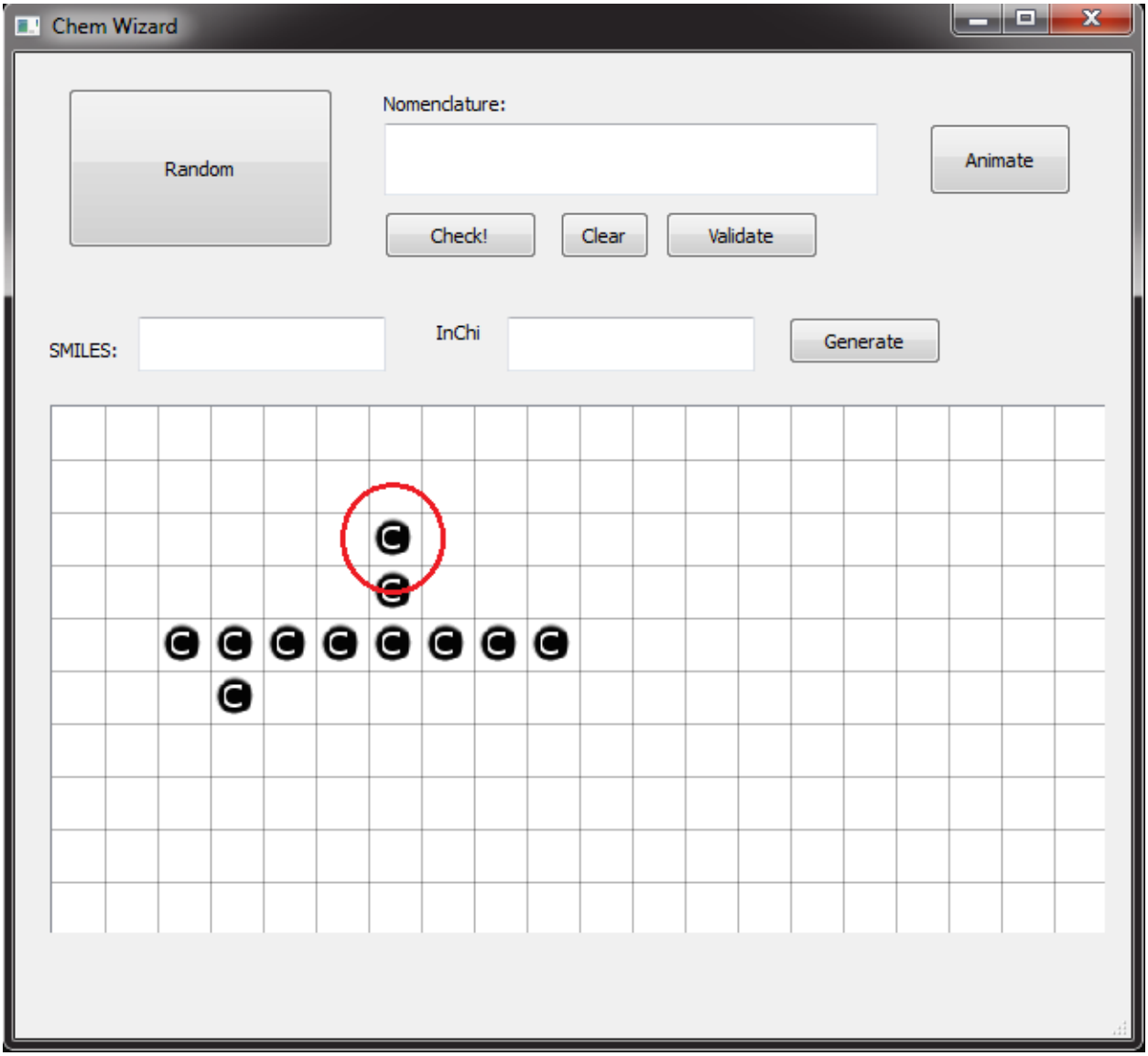
Step 2:

Click the Generate button to draw carbons.



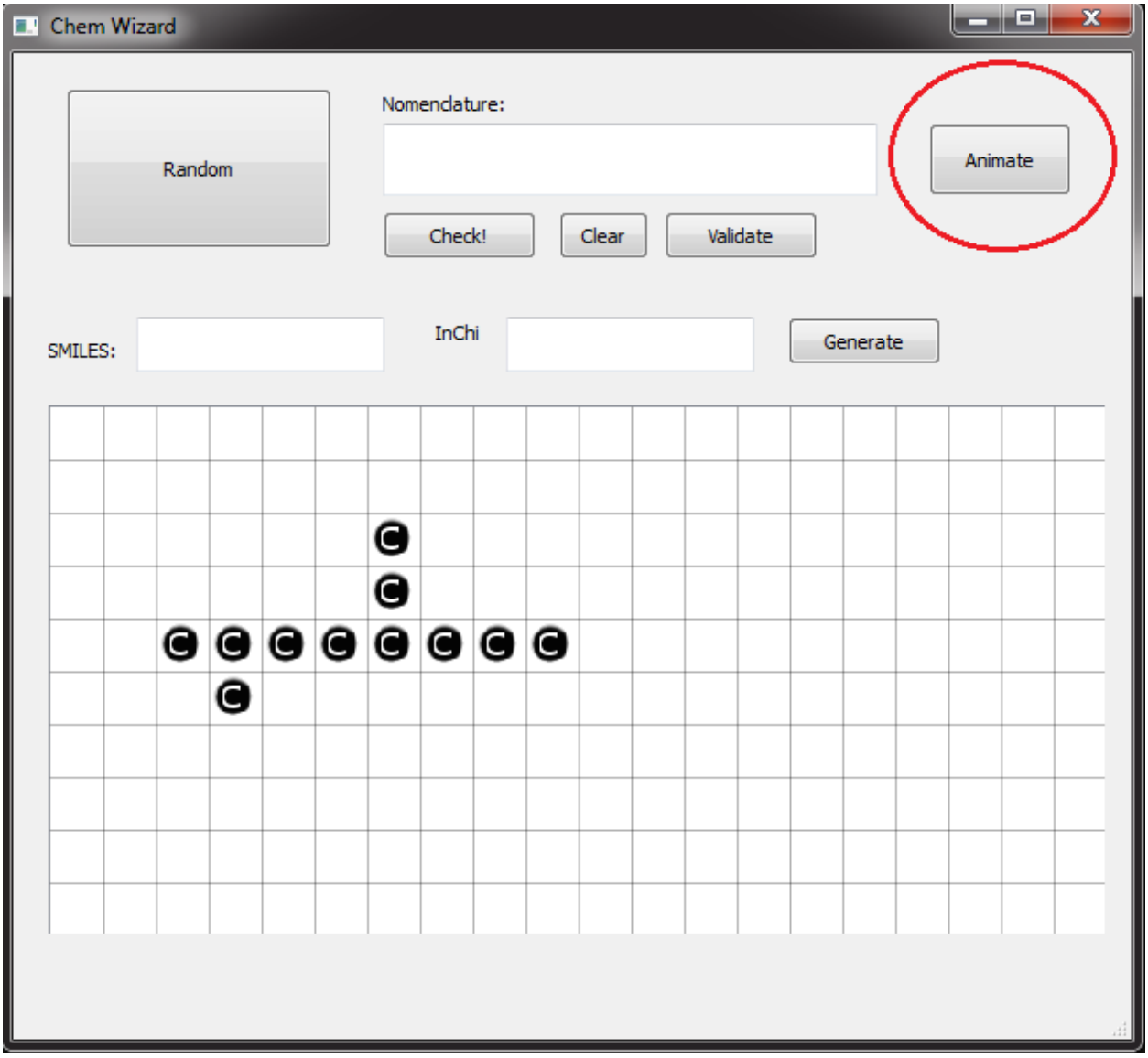
Step 3:

Carbons will appear on the Grid.



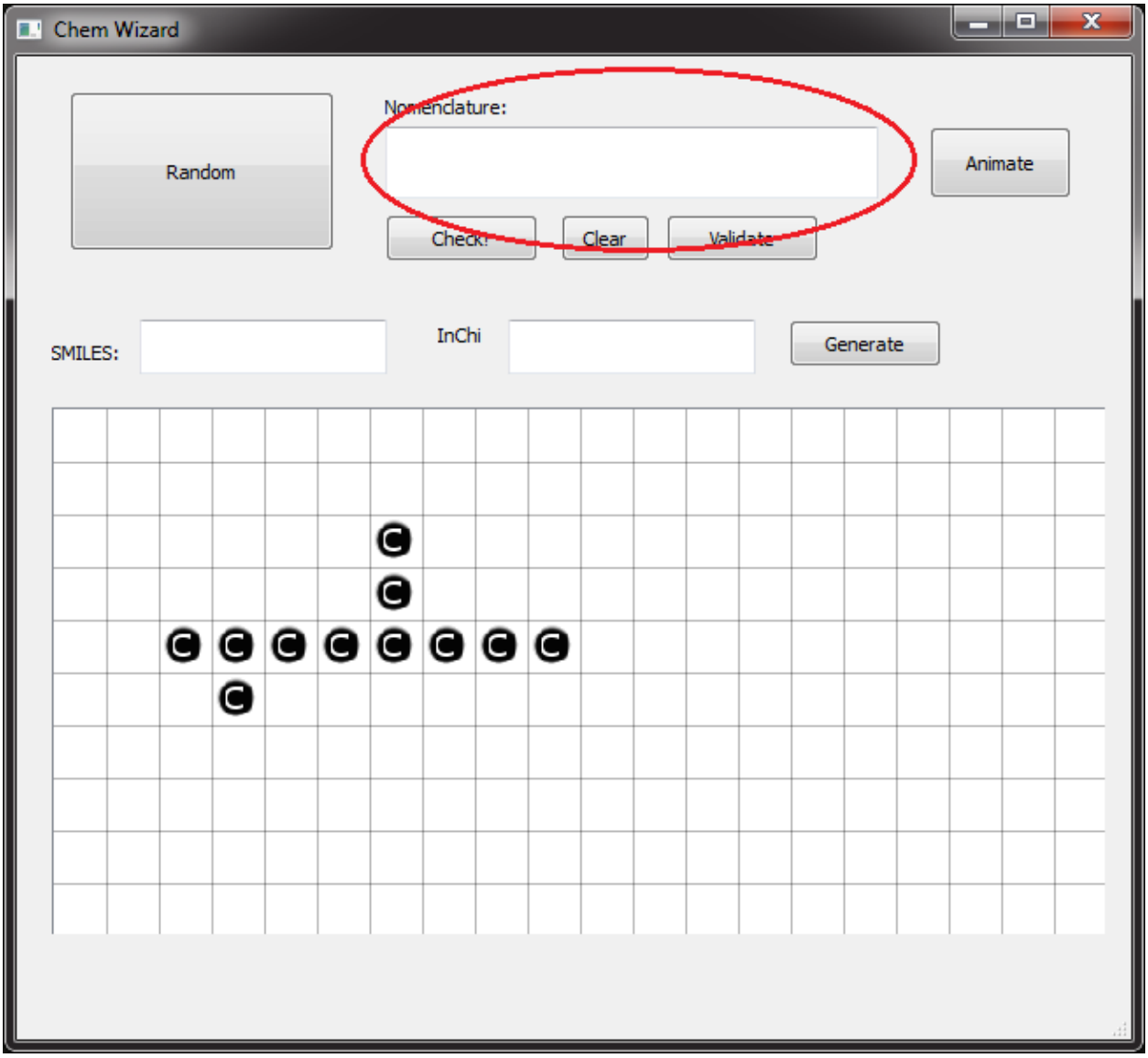
Step 4:

Click the Animate button to show the animation.



Step 5:

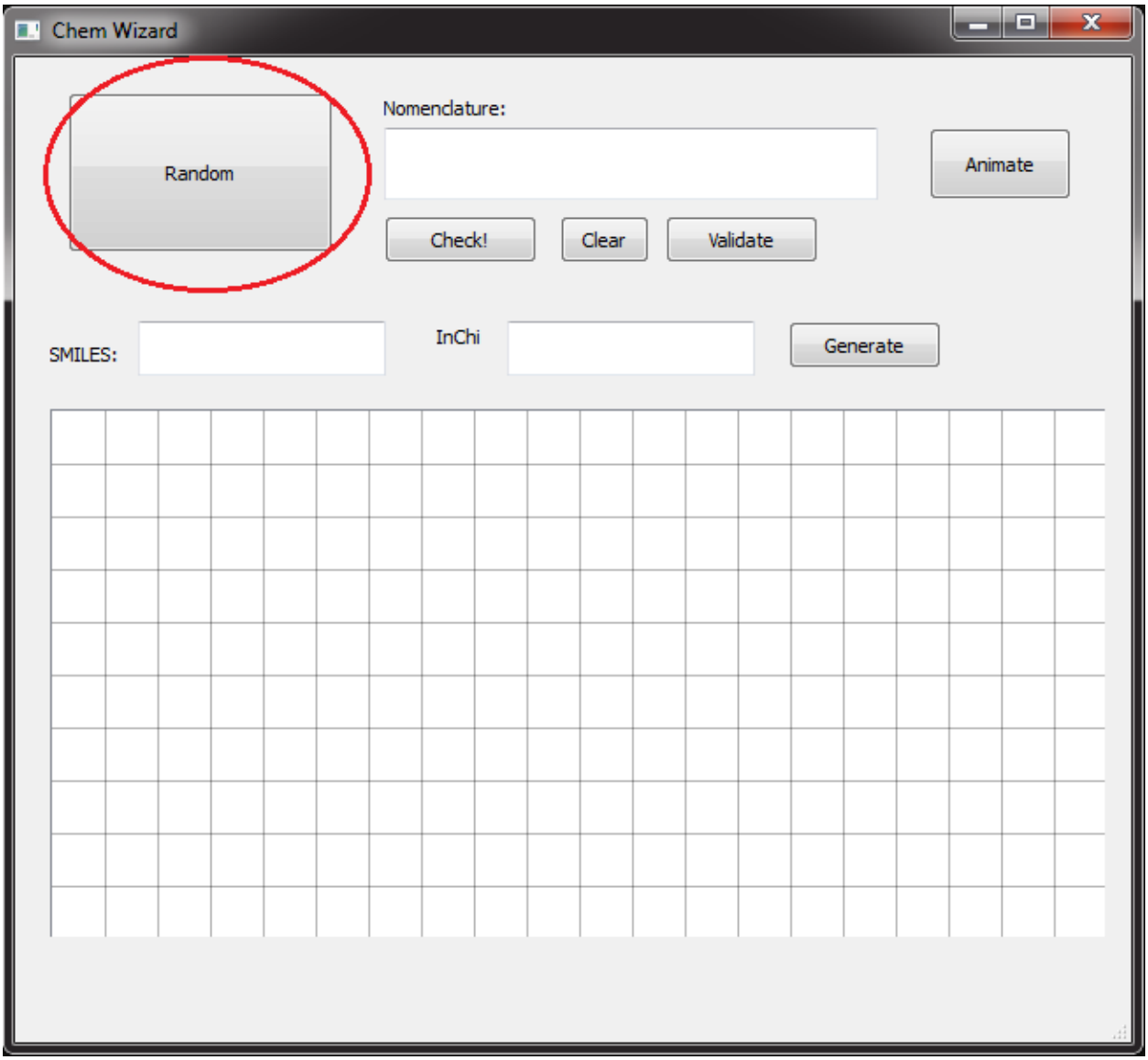
Calculated nomenclature will appear in the Nomenclature box.



Creating a Random Molecule

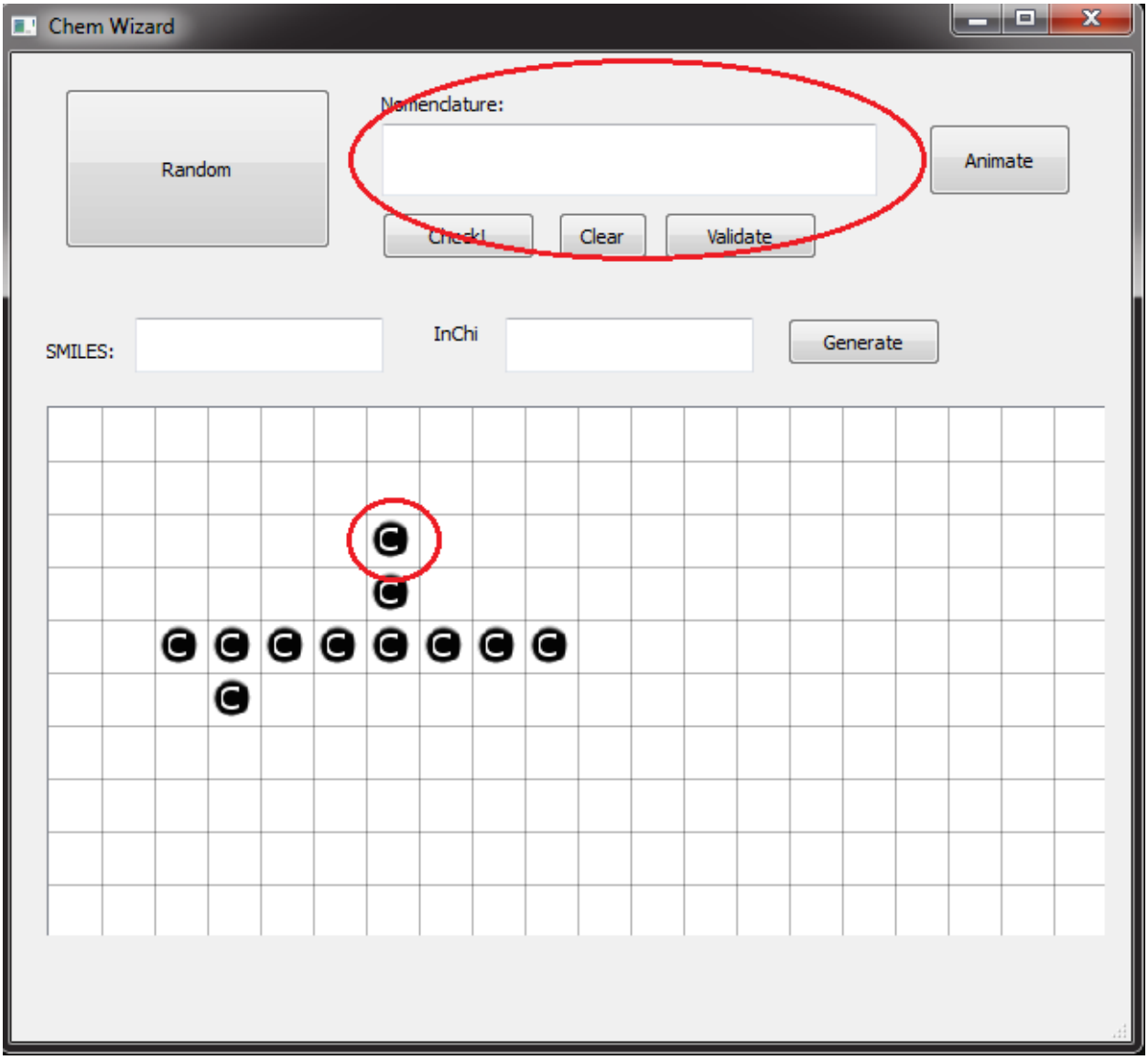
Step 1:

Click the random button to draw a random molecule.



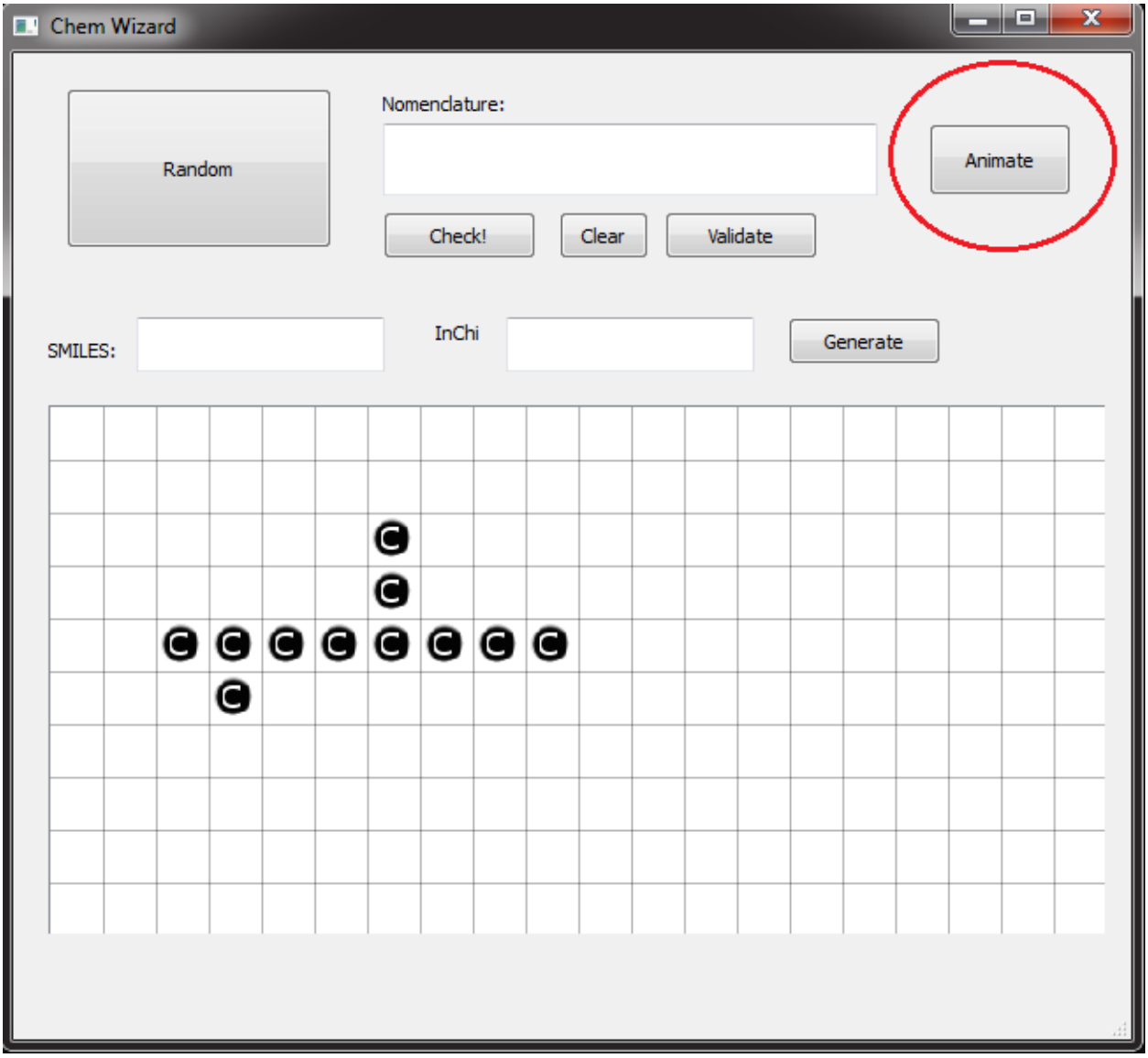
Step 2:

Nomenclature and Carbons will be displayed in their proper areas.



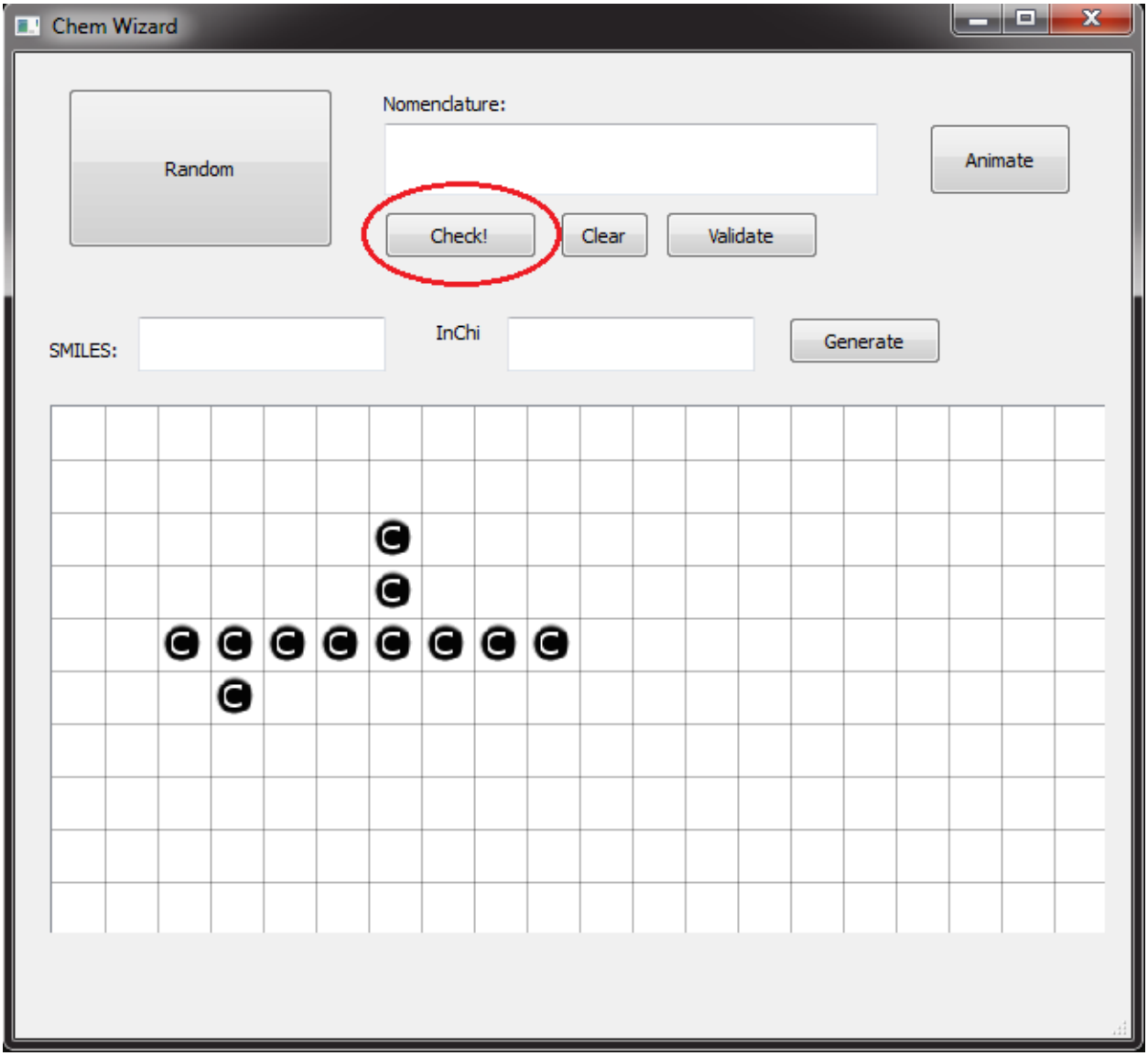
Step 3:

Click the Animate button to show the animation.



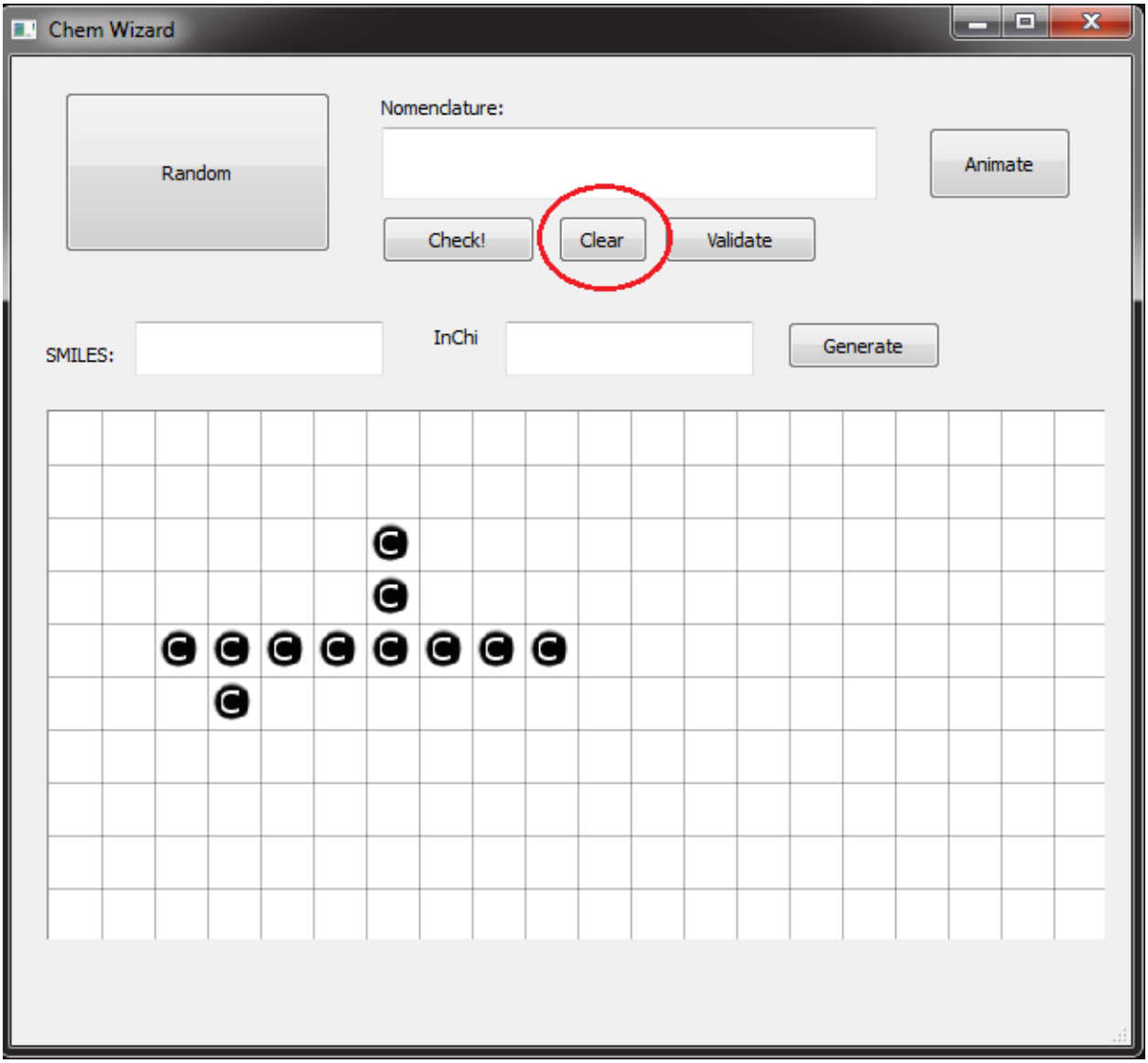
Checking your Nomenclature

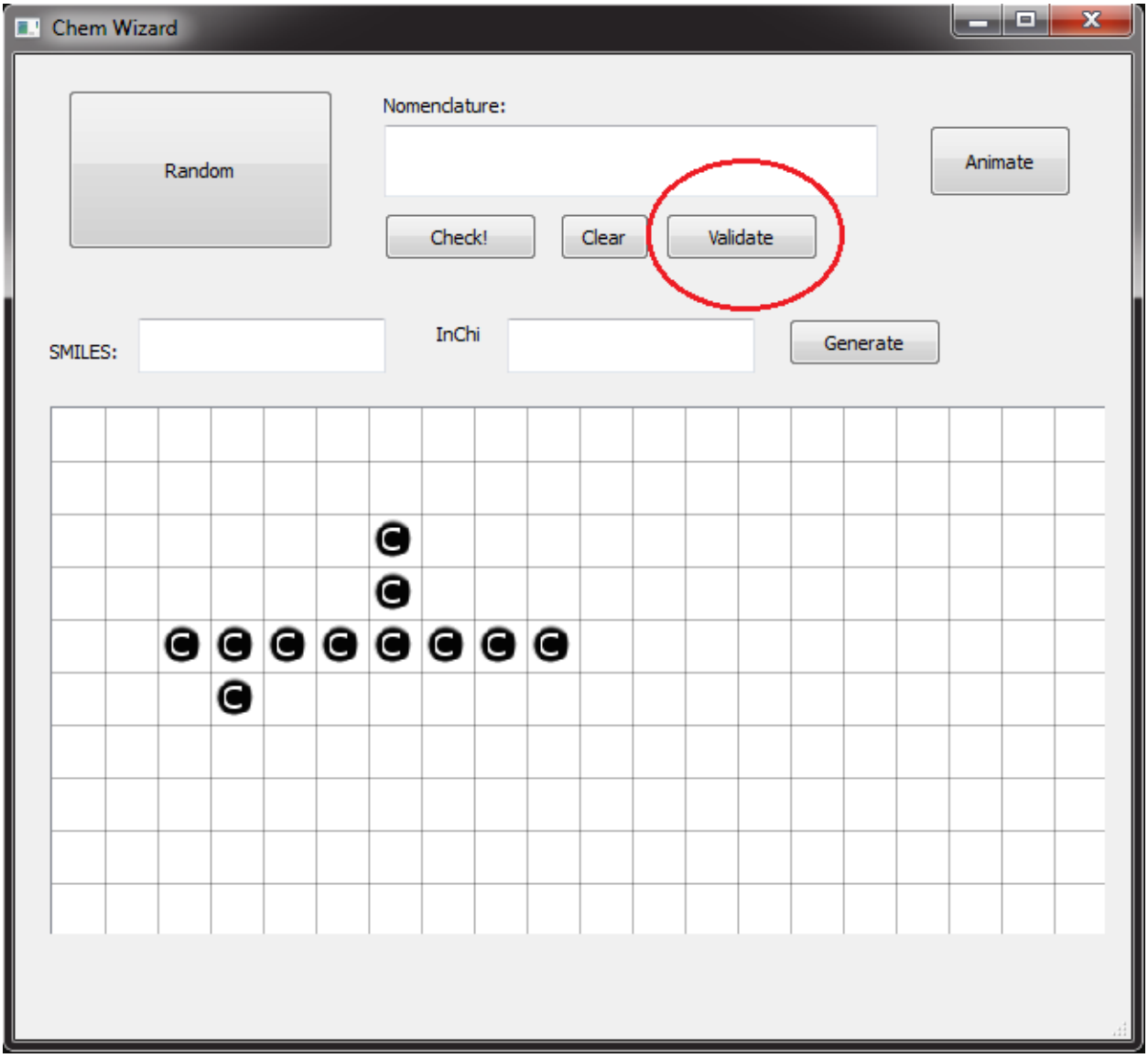
Click the Check! button to check the Nomenclature field for correctness.



Clearing the Nomenclature

Clicking the Clear button will empty the Nomenclature field and also remove the present carbons.



Validating the Nomenclature

Click the Validate button to determine the correctness of the nomenclature.