|  |
| --- |
|  |
| 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 (Incomplete)Introduction 5](#_Toc354831641)

[1.1 (Incomplete) Team Introduction 5](#_Toc354831642)

[1.1.1 Team Name 5](#_Toc354831643)

[1.1.2 (Incomplete)Team Members and Duties 5](#_Toc354831644)

[1.2 (Incomplete)Project Description 5](#_Toc354831645)

[1.2.2 (Incomplete)Screenshots 5](#_Toc354831646)

[2. (Incomplete)Feasibility Study 6](#_Toc354831647)

[2.1 Client 6](#_Toc354831648)

[2.2 Task to be undertaken 6](#_Toc354831649)

[2.3 (Incomplete) Domain Analysis 6](#_Toc354831650)

[2.4 Benefits 6](#_Toc354831651)

[2.5 Preliminary Requirements Analysis 7](#_Toc354831652)

[2.5.1 Interface 7](#_Toc354831653)

[2.5.2 Backend 7](#_Toc354831654)

[2.5.3 Drawing 7](#_Toc354831655)

[2.6 Suggested Deliverables 7](#_Toc354831656)

[2.6.1 Management 7](#_Toc354831657)

[2.6.2 Technical 7](#_Toc354831658)

[2.7 Process to be followed 8](#_Toc354831659)

[2.8 (Incomplete) Development Schedule 8](#_Toc354831660)

[2.9 Outline Plan 8](#_Toc354831661)

[2.9.1 Requirements Gathering 9](#_Toc354831662)

[2.9.2 Design 9](#_Toc354831663)

[2.9.3 Customer Feedback 9](#_Toc354831664)

[2.9.4 Development 9](#_Toc354831665)

[2.9.5 Testing 9](#_Toc354831666)

[2.9.6 Release 10](#_Toc354831667)

[2.9.7 Maintenance 10](#_Toc354831668)

[2.10 Visibility Plan 10](#_Toc354831669)

[2.10.1 External 10](#_Toc354831670)

[2.10.2 Internal 10](#_Toc354831671)

[2.11 Risk Analysis 11](#_Toc354831672)

[2.11.1 Time Risk 11](#_Toc354831673)

[2.11.2 Existing Software Risk 11](#_Toc354831674)

[2.11.3 Resource Risks 11](#_Toc354831675)

[2.11.4 Changing Requirements 12](#_Toc354831676)

[2.12 Probable Technical Requirements 12](#_Toc354831677)

[2.13 Conclusion 12](#_Toc354831678)

[2.14 Revisions 13](#_Toc354831679)

[3. (Incomplete)Requirements Documents 14](#_Toc354831680)

[3.1 Overall Description 14](#_Toc354831681)

[3.1.1 Product Perspective 14](#_Toc354831682)

[3.1.2 User Interfaces 14](#_Toc354831683)

[3.1.3 Software Interfaces 15](#_Toc354831684)

[3.1.4 Communication Interfaces 15](#_Toc354831685)

[3.1.5 Memory Constraints 15](#_Toc354831686)

[3.1.6 Operations 15](#_Toc354831687)

[3.1.7 Site Adaption Requirements 16](#_Toc354831688)

[3.2 Production Functions 16](#_Toc354831689)

[3.3 User Characteristics 17](#_Toc354831690)

[3.4 Constraints 18](#_Toc354831691)

[3.4.1 Hardware Limitations 18](#_Toc354831692)

[3.4.2 Interface to other applications 18](#_Toc354831693)

[3.4.3 Parallel operation 18](#_Toc354831694)

[3.4.4 Audit functions 18](#_Toc354831695)

[3.4.5 Higher-order language requirements 18](#_Toc354831696)

[3.4.6 Reliability Requirements 18](#_Toc354831697)

[3.4.7 Criticality of the application 19](#_Toc354831698)

[3.4.8 Safety and security considerations 19](#_Toc354831699)

[3.5 Assumptions and Dependencies 19](#_Toc354831700)

[3.6 Apportioning the Requirements 19](#_Toc354831701)

[3.7 Specific Requirements 19](#_Toc354831702)

[3.8 External Interfaces 20](#_Toc354831703)

[3.9 Functions 21](#_Toc354831704)

[3.10 Performance Requirements 21](#_Toc354831705)

[3.11 Design Constraints 21](#_Toc354831706)

[3.11.1 Standards Compliance 22](#_Toc354831707)

[3.12 Software System Attributes 22](#_Toc354831708)

[3.12.1 Reliability 22](#_Toc354831709)

[3.12.2 Availability 23](#_Toc354831710)

[3.12.3 Security 23](#_Toc354831711)

[3.12.4 Maintainability 23](#_Toc354831712)

[3.12.5 Portability 24](#_Toc354831713)

[3.13 (Incomplete)Use Cases 24](#_Toc354831714)

[3.14 (Incomplete)Prototypes 25](#_Toc354831715)

[3.15 Revisions 25](#_Toc354831716)

[4. (Incomplete)Project Schedule 26](#_Toc354831717)

[4.1 Timeline 27](#_Toc354831718)

[4.2 Work Breakdown Structure 28](#_Toc354831719)

[4.3 (Incomplete)PERT Diagram 29](#_Toc354831720)

[4.4 (Incomplete)Lines of Code Estimation 29](#_Toc354831721)

[4.5 (Incomplete)Revisions 29](#_Toc354831722)

[5. (Incomplete)Design Documents 30](#_Toc354831723)

[5.1 UML Diagrams 30](#_Toc354831724)

[5.2 (Incomplete) Use Case Diagrams 32](#_Toc354831725)

[5.3 (Incomplete) Prototype 32](#_Toc354831726)

[5.4 (Incomplete)Revisions 32](#_Toc354831727)

[6. (Incomplete)SQA Plan 33](#_Toc354831728)

[6.1 (Incomplete)SQA Checklists 33](#_Toc354831729)

[6.1.1 (Incomplete)Requirements 33](#_Toc354831730)

[6.1.2 (Incomplete)Design 33](#_Toc354831731)

[6.1.3 (Incomplete)Implementation 33](#_Toc354831732)

[6.1.4 (Incomplete)Testing 33](#_Toc354831733)

[6.1.5 (Incomplete)Installation 33](#_Toc354831734)

[6.1.6 (Incomplete)Maintenance 33](#_Toc354831735)

[6.2 (Incomplete)Deliverables 33](#_Toc354831736)

[6.3 (Incomplete)Revisions 33](#_Toc354831737)

[7. (Incomplete)Test Plan 34](#_Toc354831738)

[7.1 (Incomplete)Deliverables 34](#_Toc354831739)

[7.2 (Incomplete)Revisions 34](#_Toc354831740)

[8. (Incomplete)Post-mortem 35](#_Toc354831741)

[8.1 (Incomplete)Roland Heintze 35](#_Toc354831742)

[8.2 (Incomplete)Tim Elam 35](#_Toc354831743)

[8.3 (Incomplete)Chris Lansing 35](#_Toc354831744)

[8.4 (Incomplete)John Gibbons 35](#_Toc354831745)

[9. (Incomplete)Appendix 36](#_Toc354831746)

[9.1 Source Code 36](#_Toc354831747)

[\_\_init\_\_.py 36](#_Toc354831748)

[alkane.py 37](#_Toc354831749)

[alkane\_test.py 40](#_Toc354831750)

[cactusAPI.py 41](#_Toc354831751)

[carbon.py 42](#_Toc354831752)

[carbon\_test.py 44](#_Toc354831753)

[chemspipy.py 45](#_Toc354831754)

[chemspipy\_test.py 50](#_Toc354831755)

[chemspipy\_LICENSE.txt 51](#_Toc354831756)

[custom\_exceptions.py 52](#_Toc354831757)

[draw\_view.py 53](#_Toc354831758)

[launch.ui 55](#_Toc354831759)

[main.py 60](#_Toc354831760)

[substituent.py 62](#_Toc354831761)

[substituent\_test.py 63](#_Toc354831762)

[ui.py 64](#_Toc354831763)

[9.2 (Incomplete)User Manual 66](#_Toc354831764)

# (Incomplete)Introduction

## 1.1 (Incomplete) Team Introduction

### Team Name

Chemistry Visualization Software

### (Incomplete)Team Members and Duties

## (Incomplete)Project Description

### (Incomplete)Screenshots

# (Incomplete)Feasibility Study

## 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.

## 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.

## (Incomplete) Domain Analysis

## 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

## Preliminary Requirements Analysis

### 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.

### 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.

### Drawing

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

## Suggested Deliverables

### Management

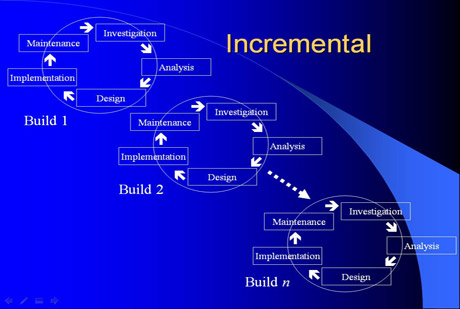
* + Requirements Analysis - A formal requirement. Will contain all requirements delivered from the client, minimum specifications the machines need to run the software, design constraints and user characteristics.
  + Status Reports - The customer will be kept updated with ongoing status reports that 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.

### Technical

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

## 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.



## (Incomplete) Development Schedule

## Outline Plan

### 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.

### 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.

### 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 are 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.

### 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.

### 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.

### 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.

### 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.

## Visibility Plan

### 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.

### 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.

## Risk Analysis

### 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.

### 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.

### 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.

### 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.

## 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 a .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.

## 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.

## 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. |
| 4 | 4-9-13 | John Gibbons | Fourth revision. |
| 5 | 4-25-13 | John Gibbons | Changed requirements and platform information. |
| 6 |  | Roland Heintze | Finish domain analysis and final revision. |
| 7 | 4-27-13 | Chris Lansing | Grammar corrections |

# (Incomplete)Requirements Documents

## 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.

### 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.

### 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.

### 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.

## 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 My.GeorgiaSouthern emails.

### Memory Constraints

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

### 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 names 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 so choose to. 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 a 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.

### Site Adaption Requirements

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

## Production Functions

This program will allow any user to generate a random organic compound and view an animation on the procedures of the scientific nomenclatures for carbons up to and including pentadecane. 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 field where the name can be entered does not have to contain anything or contain the correct answer to begin. 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.

## 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:

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

Students:

* + 1. 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.
    2. Experience.
       1. They possess no experience with this software.
       2. Assumed to possess basic to high level experience with common desktop applications.
    3. Technical Expertise.
       1. Ability to navigate common desktop environments.
       2. Understanding of virtual form controls.
       3. Understanding of the mouse clicking interaction metaphor.

## Constraints

### 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.

### Interface to other applications

The program will not interface with any external applications.

### 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.

### Audit functions

Does not apply

### Higher-order language requirements

We are 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.

### Reliability Requirements

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

### Criticality of the application

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

### Safety and security considerations

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

## Assumptions and Dependencies

Our team assumes that the users will mainly be 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.

## 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.

## 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/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.

## 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.

## 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.

## Performance Requirements

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

## 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 Python Interpreter installed to run the file. The system shall also have Internet Explorer 7.0 or newer, Firefox 3.6 or newer or Chrome to download the Py2exe software if it is not currently 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.

### 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.

## Software System Attributes

### 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.

### Availability

This program will be accessible on all machines. If some machines have issues with the exported python file to a .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.

### 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.

### 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.

### 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 run it.

d) Use of a particular compiler or language subset.

Because our team wanted this to 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.

## (Incomplete)Use Cases

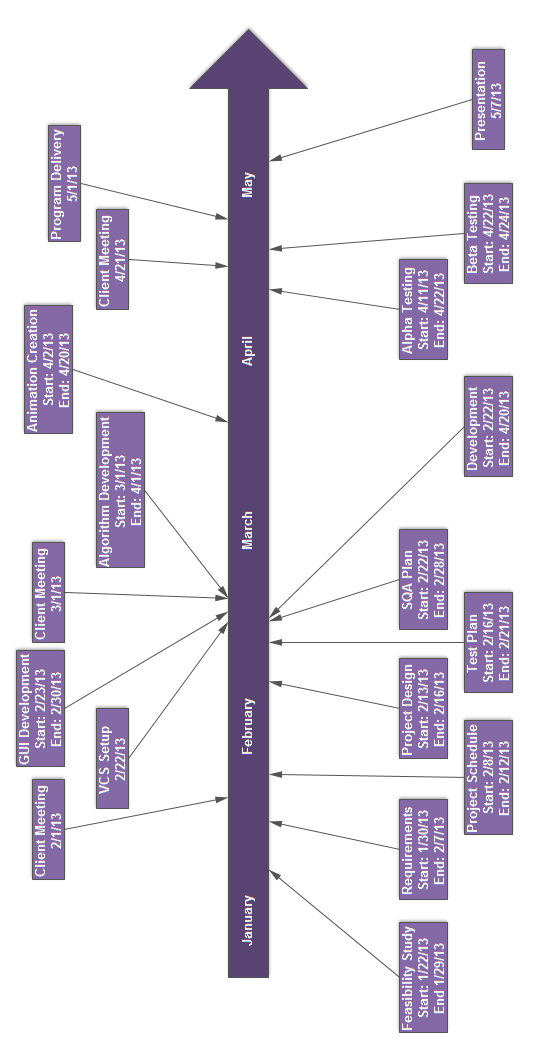
## (Incomplete)Prototypes

## Revisions

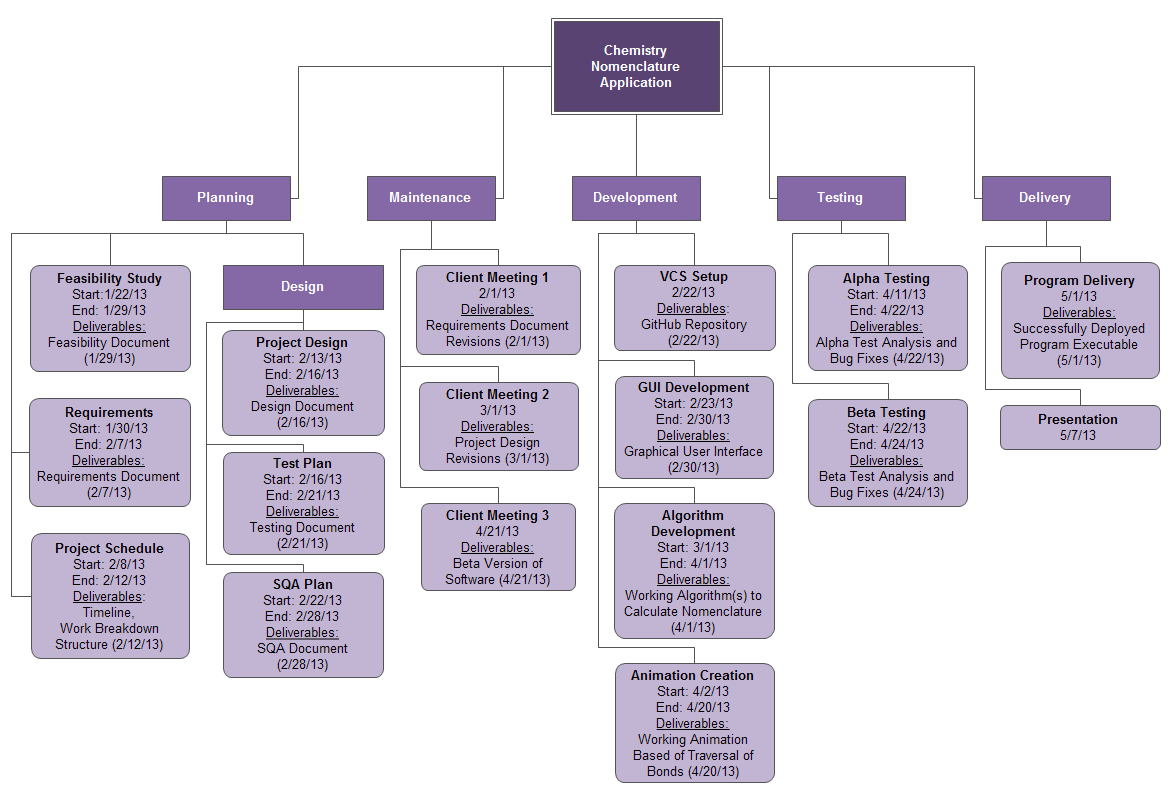
|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number: | Revision Date: | Author: | Summary of Changes: |
| 1 | 2-28-13 | John Gibbons | First revision of draft. Reworded sentences and corrected grammatical errors. |
| 2 | 3-4-13 | Roland Heintze and Chris Lansing | Second revision of draft. Moved document into Word. Added use cases, cover page, formatting and a table of contents. |
| 3 | 4-9-13 | John Gibbons | Third revision. |
| 4 | 4-25-13 | John Gibbons | Fourth revision while updating requirements and platform changes. |
| 5 |  | Roland Heintze | Final revision. |
| 6 | 4-27-13 | Chris Lansing | Grammar corrections |

# (Incomplete)Project Schedule

## Timeline



## Work Breakdown Structure



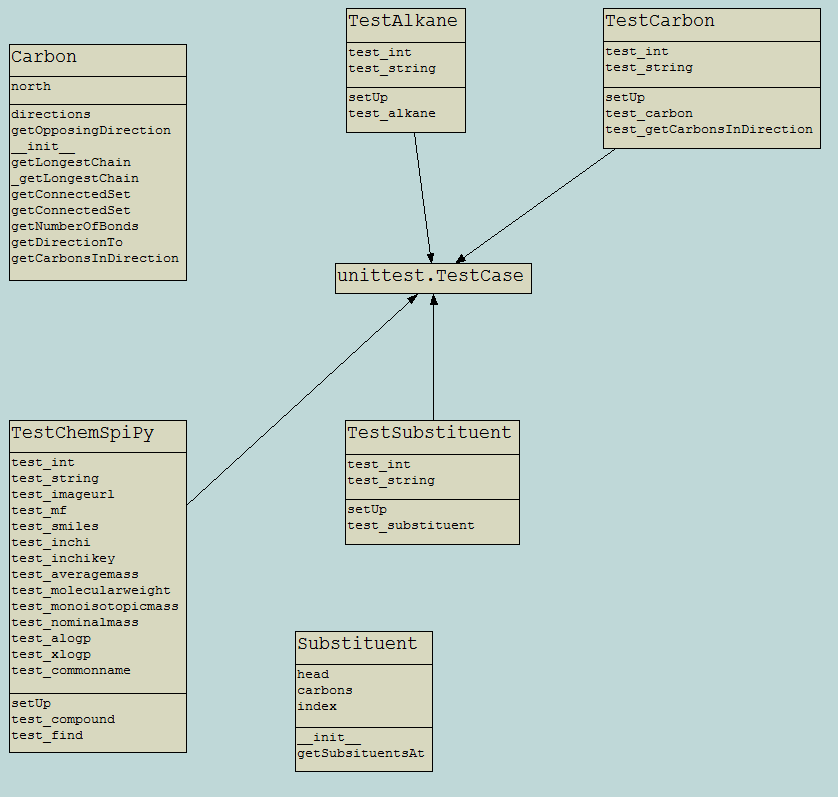
## (Incomplete)PERT Diagram

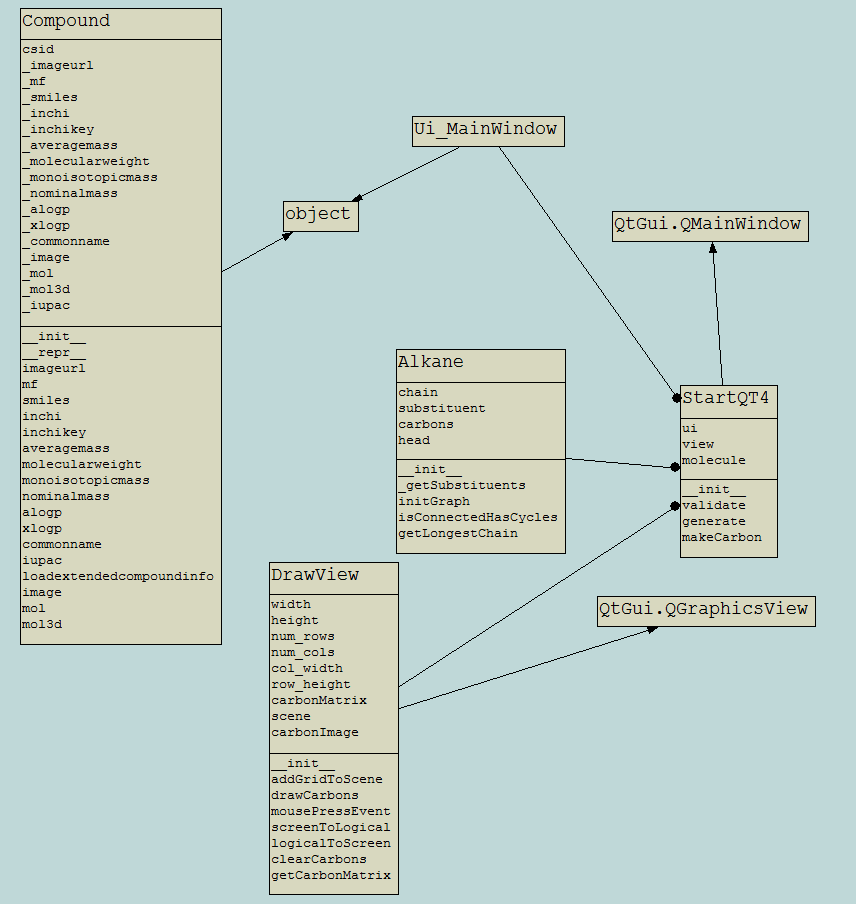
## (Incomplete)Lines of Code Estimation

## (Incomplete)Revisions

# (Incomplete)Design Documents

## UML Diagrams





## (Incomplete) Use Case Diagrams

## (Incomplete) Prototype

## (Incomplete)Revisions

# (Incomplete)SQA Plan

## (Incomplete)SQA Checklists

### (Incomplete)Requirements

### (Incomplete)Design

### (Incomplete)Implementation

### (Incomplete)Testing

### (Incomplete)Installation

### (Incomplete)Maintenance

## (Incomplete)Deliverables

## (Incomplete)Revisions

# (Incomplete)Test Plan

## (Incomplete)Deliverables

## (Incomplete)Revisions

# (Incomplete)Post-mortem

## (Incomplete)Roland Heintze

## (Incomplete)Tim Elam

## (Incomplete)Chris Lansing

## (Incomplete)John Gibbons

# (Incomplete)Appendix

## Source Code

### \_\_init\_\_.py

(empty, but required)

### alkane.py

1. \_\_author\_\_ = 'rheintze'
3. #from chemspipy import \*
4. #from cactusAPI import get\_csid
5. **from** carbon **import** Carbon
6. **from** substituent **import** Substituent
7. **from** custom\_exceptions **import** \*
8. **class** Alkane:
9. """
10. A class used to implement an organic compound.
11. """
13. chain = ('methane','ethane','propane','butane','pentane','hexane','heptane','octane','nonane','decane',
14. 'undecane','dodecane','tridecane','tetradecane','pentadecane')
15. substituent = ('methyl','ethyl','propyl','butyl','pentyl')
17. **def** \_\_init\_\_(self, matrix):
18. self.carbons = []
19. self.head = None
20. self.initGraph(matrix)
21. **if** self.head == None:
22. **raise** EmptyAlkaneError()
23. isConnected, hasCycles = self.isConnectedHasCycles()
24. **if** **not** isConnected:
25. **raise** AlkaneNotConnectedError()
26. **if** hasCycles:
27. **raise** CyclicAlkaneError()
28. #        self.longestChain = self.getLongestChain()
29. #        #Reset head to first element in longest chain. This is the cannonical head.
30. #        self.head = self.longestChain[0]
31. #        try:
32. #            self.substituents = self.\_getSubstituents()
33. #        except BranchingCarbonChainError as error:
34. #            raise error
36. #Throws BranchingCarbonChainError for malformed substituents
37. **def** \_getSubstituents(self):
38. subs = []
39. **for** i **in** range(len(self.longestChain)):
40. **try**:
41. subs.extend(Substituent.getSubsituentsAt(self.longestChain, i))
42. **except** BranchingCarbonChainError as error:
43. **raise** error
44. **return** subs
46. #Initialize graph and carbon list based on carbon matrix
47. **def** initGraph(self, graphicMatrix):
48. width = len(graphicMatrix)
49. height = len(graphicMatrix[0])
50. #Dict of position tuples to carbon objects
51. #Used to remember where we have already found carbons
52. carbonDict = {}
53. **for** col\_index **in** range(0, width):
54. **for** row\_index **in** range(0, height):
55. #Skip empty spaces
56. **if** **not** graphicMatrix[col\_index][row\_index]:
57. **continue**
58. thisCarbon = Carbon()
59. self.carbons.append(thisCarbon)
60. #Remember that this carbon is at this location for when we make our double-links
61. carbonDict[(col\_index, row\_index)] = thisCarbon
62. otherCarbon = None
63. #Have we found our first carbon?
64. **if** self.head == None:
65. self.head = thisCarbon
67. #Is there a valid, occupied space to the left?
68. **if** col\_index-1>0 **and** graphicMatrix[col\_index-1][row\_index]:
69. #Remember which carbon was in this space
70. otherCarbon = carbonDict[(col\_index-1,row\_index)]
71. #Create double-link between our Carbon nodes
72. thisCarbon.west = otherCarbon
73. otherCarbon.east = thisCarbon
75. #Is there a valid, occupied space to the top?
76. **if** row\_index-1>0 **and** graphicMatrix[col\_index][row\_index-1]:
77. #Remember which carbon was in this space
78. otherCarbon = carbonDict[(col\_index,row\_index-1)]
79. #Create double-link between our Carbon nodes
80. thisCarbon.north = otherCarbon
81. otherCarbon.south = thisCarbon

84. #Check if carbons are connected and graph has no cycles
85. **def** isConnectedHasCycles(self):
86. isConnected = True
87. hasCycles = False
88. totalSet = set(self.carbons)
89. **try**:
90. connectedSet = self.head.getConnectedSet()
91. **except** CyclicAlkaneError:
92. hasCycles = True
93. **if** totalSet != connectedSet:
94. isConnected = False
95. **return** (isConnected, hasCycles)

98. **def** getLongestChain(self):
99. chains = []
100. **for** carbon **in** self.carbons:
101. chains.append(carbon.getLongestChain())
102. **return** max(chains, key=len)

105. #    def \_\_init\_\_(self, query):  #query will generally be in smiles or inchi
106. #        self.query = query
107. #        self.compound\_maybe = Compound(get\_csid(query))  #uses cactus search to find csid & constructs compound with it
108. #
109. #        pass
110. #
111. #if \_\_name\_\_ == '\_\_main\_\_':
112. #    #testing
113. #    a = Alkane('C')                 #C Methane
114. #    b = Alkane('CC(C)CC')           #CC(C)CC 2-Methylbutane
115. #    c = Alkane('CC(C)(C)C')         #CC(C)(C)C 2,2-Dimethylpropane
116. #
117. #    print(a.compound\_maybe.smiles) #a
118. #    print(a.compound\_maybe.iupac)
119. #    print(b.compound\_maybe.smiles) #b
120. #    print(b.compound\_maybe.iupac)
121. #    print(c.compound\_maybe.smiles) #c
122. #    print(c.compound\_maybe.iupac)
123. #    pass

### alkane\_test.py

1. # -\*- coding: utf-8 -\*-
2. \_\_author\_\_ = 'rheintze'
4. """
5. Unit tests for alkane.py
6. """

9. **import** unittest
11. **from** alkane **import** \*
13. **class** TestAlkane(unittest.TestCase):
14. **def** setUp(self):
15. self.test\_int = 50
16. self.test\_string = '50'
18. **def** test\_alkane(self):
19. """ Unit tests for the Alkane class """
20. self.assertRaises(TypeError, Alkane, 1.2)
21. self.assertRaises(TypeError, Alkane, 'notadigit')
22. self.assertEqual(Carbon(self.test\_int).int, self.test\_string)

25. **if** \_\_name\_\_ == '\_\_main\_\_':
26. unittest.main()

### cactusAPI.py

1. # -\*- coding: utf-8 -\*-
3. """
4. CactusAPI
6. Simple Python wrapper for the Cactus API.
7. Used ChemSpiPy for inspiration/guidance
9. """
11. **import** urllib.request
13. \_\_author\_\_ = 'Roland Heintze'
14. \_\_email\_\_ = 'rheintze@linux.com'
15. \_\_version\_\_ = '1.0'
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.quote(query))
20. response = str(urllib.request.urlopen(searchurl).readline().strip())[2:-1]
22. **return** response
23. **def** get\_iupac(smiles\_query):
24. """  uses smiles to pull compound's IUPAC from cactus """
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]
30. **return** response

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'))

### carbon.py

1. **from** custom\_exceptions **import** \*
3. **class** Carbon:
4. """
5. A class used to represent a carbon in an alkane molecule
6. """
7. #Ordered such that directions[i] opposes directions[3-i]
8. @staticmethod
9. **def** directions():
10. **return** ('north', 'east', 'west', 'south')
11. @staticmethod
12. **def** getOpposingDirection(direction):
13. **return** Carbon.directions()[3-Carbon.directions().index(direction)]
15. **def** \_\_init\_\_(self):
16. self.north = self.south = self.east = self.west = None
18. **def** getLongestChain(self):
19. **return** self.\_getLongestChain()
21. **def** \_getLongestChain(self, ignoreDirection=None):
22. #Append this carbon to the chain
23. chain = [self]
24. #List of directions that have carbons and are not ignoreDirection
25. validChains = []
26. **for** dir **in** Carbon.directions():
27. **if** dir == ignoreDirection:
28. **continue**
29. **if** getattr(self, dir):
30. opposingDirection = Carbon.getOpposingDirection(dir)
31. nextCarbon = getattr(self, dir)
32. validChains.append(nextCarbon.\_getLongestChain(opposingDirection))
33. **if** any(validChains):
34. #Extend our chain with the longest chain we found
35. chain.extend(max(validChains, key=len))
36. **return** chain
38. **def** getConnectedSet(self):
39. **return** self.\_getConnectedCarbonSet()
41. #Helper recursive method for isConnectedHasCycles
42. **def** getConnectedSet(self, carbonSet=set(), ignoreDirection=None):
43. #Ordered such that directions[i] opposes directions[3-i]
44. directions = ('north', 'east', 'west', 'south')
45. **if** self **in** carbonSet:
46. **raise** CyclicAlkeneError()
47. **else**:
48. carbonSet.add(self)
49. **for** dir **in** directions:
50. **if** dir == ignoreDirection:
51. **continue**
52. nextCarbon = getattr(self, dir)
53. **if** nextCarbon:
54. opposingDirection = Carbon.getOpposingDirection(dir)
55. nextCarbon.getConnectedSet(carbonSet, opposingDirection)
56. **return** carbonSet
58. **def** getNumberOfBonds(self):
59. num = 0
60. **for** dir **in** Carbon.directions():
61. **if** getattr(self,dir):
62. num+=1
63. **return** num
65. **def** getDirectionTo(self, carbon):
66. **for** direction **in** Carbon.directions():
67. **if** getattr(self, direction) == carbon:
68. **return** direction
69. **return** None
71. #Throws AmbiguousBranchingPath if any Carbon in the sequence has
72. #more than two connected Carbons
73. **def** getCarbonsInDirection(self, direction, ignoreDirection=None):
74. chain = [self]
75. **if** direction:
76. nextCarbon = getattr(self, direction)
77. **if** nextCarbon:
78. opposingDirection = Carbon.getOpposingDirection(direction)
79. chain.extend(nextCarbon.getCarbonsInDirection(direction=None, ignoreDirection=opposingDirection))
80. **else**:
81. #To check for branching paths
82. hasChain = False
83. **for** dir **in** Carbon.directions():
84. **if** dir == ignoreDirection:
85. **continue**
86. nextCarbon = getattr(self, dir)
87. **if** nextCarbon:
88. #If we find a second chain, raise an error. The chain would be ambiguous.
89. **if** hasChain:
90. **raise** BranchingCarbonChainError()
91. opposingDirection = Carbon.getOpposingDirection(direction)
92. #Search for more Carbons and add them to our list
93. chain.extend(nextCarbon.getCarbonsInDirection(direction=None, ignoreDirection=opposingDirection))
94. hasChain=True
95. **return** chain

### carbon\_test.py

1. # -\*- coding: utf-8 -\*-
2. \_\_author\_\_ = 'rheintze'
4. """
5. Unit tests for carbon.py
6. """

9. **import** unittest
11. **from** carbon **import** \*
13. **class** TestCarbon(unittest.TestCase):
14. **def** setUp(self):
15. self.test\_int = 50
16. self.test\_string = '50'
18. **def** test\_carbon(self):
19. """ Unit tests for the Carbon class """
20. self.assertRaises(TypeError, Carbon, 1.2)
21. self.assertRaises(TypeError, Carbon, 'notadigit')
22. self.assertEqual(Carbon(self.test\_int).int, self.test\_string)
24. **def** test\_getCarbonsInDirection(self):
25. self.assertEqual(False, None)
27. **if** \_\_name\_\_ == '\_\_main\_\_':
28. unittest.main()

### chemspipy.py

1. # -\*- coding: utf-8 -\*-
2. #modified by 2to3.py
3. """
4. ChemSpiPy
6. Python wrapper for the ChemSpider API.
7. https://github.com/mcs07/ChemSpiPy
9. Forked from ChemSpiPy by Cameron Neylon
10. https://github.com/cameronneylon/ChemSpiPy
11. """
13. **import** urllib.request, urllib.error, urllib.parse
14. **from** xml.etree **import** ElementTree as ET
16. # adds IUPAC Nomenclature as an attribute of the Compound class
17. # pulled from http://cactus.nci.nih.gov/chemical/structure API
18. **from** cactusAPI **import** get\_iupac

21. \_\_author\_\_ = 'Matt Swain'
22. \_\_email\_\_ = 'm.swain@me.com'
23. \_\_version\_\_ = '1.0'
24. \_\_license\_\_ = 'MIT'

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

30. **class** Compound(object):
31. """ A class for retrieving record details about a compound by CSID.
33. The purpose of this class is to provide access to various parts of the
34. ChemSpider API that return information about a compound given its CSID.
35. Information is loaded lazily when requested, and cached for future access.
36. """
38. **def** \_\_init\_\_(self,csid):
39. """ Initialize with a CSID as an int or str """
40. **if** type(csid) **is** str **and** csid.isdigit():
41. 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')
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
64. **def** \_\_repr\_\_(self):
65. **return** "Compound(%r)" % self.csid
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
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
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
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
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
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
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
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
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
130. @property
131. **def** alogp(self):
132. """ Retrieve ALogP from ChemSpider """
133. **if** self.\_alogp **is** None:
134. self.loadextendedcompoundinfo()
135. **return** self.\_alogp
137. @property
138. **def** xlogp(self):
139. """ Retrieve XLogP from ChemSpider """
140. **if** self.\_xlogp **is** None:
141. self.loadextendedcompoundinfo()
142. **return** self.\_xlogp
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
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
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
186. #custom field for IUPAC Nomenclature format. Pulls from Cactus API
187. self.\_iupac = get\_iupac(self.smiles)
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
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
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
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' % (urllib.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
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

### chemspipy\_test.py

1. # -\*- coding: utf-8 -\*-
2. \_\_author\_\_ = 'rheintze'
4. """
5. Unit tests for carbon.py
6. """

9. **import** unittest
11. **from** carbon **import** \*
13. **class** TestCarbon(unittest.TestCase):
14. **def** setUp(self):
15. self.test\_int = 50
16. self.test\_string = '50'
18. **def** test\_carbon(self):
19. """ Unit tests for the Carbon class """
20. self.assertRaises(TypeError, Carbon, 1.2)
21. self.assertRaises(TypeError, Carbon, 'notadigit')
22. self.assertEqual(Carbon(self.test\_int).int, self.test\_string)
24. **def** test\_getCarbonsInDirection(self):
25. self.assertEqual(False, None)
27. **if** \_\_name\_\_ == '\_\_main\_\_':
28. unittest.main()

### 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.

### custom\_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\_\_()
8. **class** CyclicAlkaneError(Exception):
9. **def** \_\_init\_\_(self):
10. super(Exception, self).\_\_init\_\_()
12. **class** BranchingCarbonChainError(Exception):
13. **def** \_\_init\_\_(self):
14. super(Exception, self).\_\_init\_\_()
16. **class** NotASubstituentError(Exception):
17. **def** \_\_init\_\_(self):
18. super(Exception, self).\_\_init\_\_()

### draw\_view.py

1. **from** PyQt4 **import** QtCore, QtGui
2. **from** PyQt4.QtGui **import** QImage
3. **from** ui **import** Ui\_MainWindow #pulls from ui.py which can be created from .ui xml file using the pyuic tool
5. **from** alkane **import** Alkane
7. **class** DrawView(QtGui.QGraphicsView):
8. **def** \_\_init\_\_(self,parent):
9. QtGui.QWidget.\_\_init\_\_(self,parent)
10. self.width = 600
11. self.height = 300
13. self.num\_rows = 10
14. self.num\_cols = 20
16. #Size of each square
17. self.col\_width = self.width / self.num\_cols
18. self.row\_height = self.height / self.num\_rows
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)
27. self.setScene(QtGui.QGraphicsScene(self.parent()))
28. self.scene = self.scene()
29. self.scene.setSceneRect(0,0,600,300)
31. #Load and resize image to fit squarely in the grid
32. self.carbonImage = QtGui.QPixmap('carbon.png').scaled(self.col\_width, self.row\_height)
33. self.addGridToScene()
35. **def** addGridToScene(self):
36. #Set up pen
37. pen = QtGui.QPen(QtCore.Qt.black, .3, QtCore.Qt.SolidLine)
38. #Draw Horizontal Lines
39. **for** y **in** range(0, self.num\_rows-1): # don't draw the two outside edge lines
40. self.scene.addLine(0, (y+1)\*self.row\_height, self.width, (y+1)\*self.row\_height, pen)
41. #Draw Vertical Lines
42. **for** x **in** range(0, self.num\_cols-1): # don't draw the two outside edge lines
43. self.scene.addLine((x+1)\*self.col\_width, 0, (x+1)\*self.col\_width, self.height, pen)
45. **def** drawCarbons(self, qp):
46. **for** row **in** range(0, self.num\_rows):
47. **for** col **in** range(0, self.num\_cols):
48. #Is there a carbon at this logical position to draw?
49. **if** self.carbonMatrix[row][col]:
50. #Get the transformed graphical coordinates
51. coords = logicalToScreen(self, (row, col))
52. #Draw the carbon image at
53. qp.drawImage(coords[0], coords[1], self.carbonImage)
55. #Add or remove carbons based on selected square status
56. **def** mousePressEvent(self,event):
57. #Convert global QT coordinates to logical coordinates
58. x,y = self.screenToLogical(event.pos())
59. #Is this square empty?
60. **if** **not** self.carbonMatrix[x][y]:
61. #Place a carbon in this square
62. self.carbonMatrix[x][y]=self.scene.addPixmap(self.carbonImage)
63. #Get top-left screen coordinates of this square
64. screen\_pos = self.logicalToScreen((x,y))
65. #Add carbon image to screen
66. self.carbonMatrix[x][y].setOffset(\*screen\_pos)
67. **else**:
68. #Remove the carbon from this square
69. self.scene.removeItem(self.carbonMatrix[x][y])
70. #Remove carbon image from scene
71. self.carbonMatrix[x][y]=None
73. #Return (x,y) in logical, safe carbonMatrix coordinates
74. **def** screenToLogical(self, position):
75. x = int(position.x()/self.col\_width)
76. y = int(position.y()/self.row\_height)
77. #Clamp logical coordinates to logical boundaries
78. x = max(x, 0)
79. x = min(x, self.num\_cols-1)
81. y = max(y, 0)
82. y = min(y, self.num\_rows-1)
84. **return** (x, y)
86. #Return (x,y) in top-left corner screen coordinates
87. **def** logicalToScreen(self, tuple):
88. **return** (int(tuple[0]\*self.col\_width), int(tuple[1]\*self.row\_height))
90. **def** clearCarbons(self):
91. #Reset carbonMatrix to remove logical carbons
92. self.carbonMatrix = []
93. **for** col **in** range(0, self.num\_cols):
94. self.carbonMatrix.append([])
95. **for** row **in** range(0, self.num\_rows):
96. self.carbonMatrix[col].append(None)
97. #Remove graphical carbons
98. **for** item **in** self.scene.items():
99. **if** type(item) **is** QtGui.QGraphicsPixmapItem:
100. self.scene.removeItem(item)
102. **def** getCarbonMatrix(self):
103. **return** self.carbonMatrix

### 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">

<property name="geometry">

<rect>

<x>510</x>

<y>50</y>

<width>101</width>

<height>31</height>

</rect>

</property>

<property name="text">

<string/>

</property>

</widget>

<widget class="QPushButton" name="generateButton">

<property name="geometry">

<rect>

<x>440</x>

<y>150</y>

<width>87</width>

<height>27</height>

</rect>

</property>

<property name="text">

<string>Generate</string>

</property>

</widget>

<widget class="QGraphicsView" name="graphicsView">

<property name="geometry">

<rect>

<x>20</x>

<y>200</y>

<width>600</width>

<height>300</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Expanding" vsizetype="Expanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="minimumSize">

<size>

<width>600</width>

<height>300</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>600</width>

<height>300</height>

</size>

</property>

<property name="verticalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

<property name="horizontalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

<property name="backgroundBrush">

<brush brushstyle="NoBrush">

<color alpha="255">

<red>255</red>

<green>255</green>

<blue>255</blue>

</color>

</brush>

</property>

<property name="foregroundBrush">

<brush brushstyle="SolidPattern">

<color alpha="255">

<red>0</red>

<green>0</green>

<blue>0</blue>

</color>

</brush>

</property>

<property name="sceneRect">

<rectf>

<x>0.000000000000000</x>

<y>0.000000000000000</y>

<width>600.000000000000000</width>

<height>300.000000000000000</height>

</rectf>

</property>

</widget>

<widget class="QPushButton" name="clearButton">

<property name="geometry">

<rect>

<x>310</x>

<y>90</y>

<width>51</width>

<height>27</height>

</rect>

</property>

<property name="text">

<string>Clear</string>

</property>

</widget>

<widget class="QPushButton" name="validateButton">

<property name="geometry">

<rect>

<x>370</x>

<y>90</y>

<width>87</width>

<height>27</height>

</rect>

</property>

<property name="text">

<string>Validate</string>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>635</width>

<height>25</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

### main.py

1. \_\_author\_\_ = 'rheintze'
3. **import** sys, traceback
4. **from** PyQt4 **import** QtCore, QtGui
5. **from** PyQt4.QtGui **import** QImage
6. **from** ui **import** Ui\_MainWindow #pulls from ui.py which can be created from .ui xml file using the pyuic tool
8. **from** alkane **import** Alkane
9. **from** draw\_view **import** DrawView
11. **class** StartQT4(QtGui.QMainWindow):
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 = DrawView(self.ui.graphicsView)
19. self.molecule = None
21. #function bindings
22. QtCore.QObject.connect(self.ui.generateButton,QtCore.SIGNAL("clicked()"),self.generate)
23. QtCore.QObject.connect(self.ui.validateButton,QtCore.SIGNAL("clicked()"),self.validate)
24. QtCore.QObject.connect(self.ui.clearButton,QtCore.SIGNAL("clicked()"),self.view.clearCarbons)
26. **def** validate(self):
27. carbonMatrix = self.view.getCarbonMatrix()
28. **try**:
29. self.molecule = Alkane(carbonMatrix)
30. **for** carbonList **in** carbonMatrix:
31. **print**(carbonList)
32. **print**()
33. **except** Exception as error:
34. **print**("Validation Error:")
35. **print**(error)
36. **print**(error.\_\_class\_\_)
37. exc\_type, exc\_value, exc\_traceback = sys.exc\_info()
38. traceback.print\_tb(exc\_traceback, limit=100, file=sys.stdout)
40. **def** generate(self):
41. **pass**
43. **def** makeCarbon(self):
44. **pass**

47. **if** \_\_name\_\_ == '\_\_main\_\_':
48. **import** sys
49. app = QtGui.QApplication(sys.argv)
50. myapp = StartQT4()
52. myapp.show()
53. sys.exit(app.exec\_())

### substituent.py

1. **from** carbon **import** Carbon
2. **class** Substituent():
4. **def** \_\_init\_\_(self, carbons, index):
5. self.head = carbons[0]
6. self.carbons=carbons
7. self.index=index
9. #Raises BranchingCarbonChainError for malformed substituents
10. @staticmethod
11. **def** getSubsituentsAt(longestChain, index):
12. subs = []
13. **for** i **in** range(len(longestChain)):
14. head = longestChain[i]
15. #All directions that don't link to longest chain are valid
16. #directions for a substituent
17. validSubDirections = list(Carbon.directions())
18. **print**(validSubDirections)
19. #Is this not the first one?
20. **if** i>0:
21. previousCarbon = longestChain[i-1]
22. invalidDirection = head.getDirectionTo(previousCarbon)
23. validSubDirections.remove(invalidDirection)
24. #Is this not the last one?
25. **if** i+1 < len(longestChain):
26. nextCarbon = longestChain[i+1]
27. invalidDirection = head.getDirectionTo(nextCarbon)
28. validSubDirections.remove(invalidDirection)
30. **for** dir **in** validSubDirections:
31. #Is there a carbon chain here?
32. subCandidate = getattr(head, dir)
33. **if** subCandidate:
34. **try**:
35. carbons = head.getCarbonsInDirection(direction = dir)
36. **except** BranchingCarbonChainError as error:
37. **raise** error
38. subs.append(Substituent(carbons = carbons, index = i))
39. **return** subs

### substituent\_test.py

1. # -\*- coding: utf-8 -\*-
2. \_\_author\_\_ = 'rheintze'
4. """
5. Unit tests for alkane.py
6. """

9. **import** unittest
11. **from** substituent **import** \*
13. **class** TestSubstituent(unittest.TestCase):
14. **def** setUp(self):
15. self.test\_int = 50
16. self.test\_string = '50'
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)
24. **if** \_\_name\_\_ == '\_\_main\_\_':
25. unittest.main()

### ui.py

1. # -\*- coding: utf-8 -\*-
3. # Form implementation generated from reading ui file 'launch.ui'
4. #
5. # Created: Tue Apr 16 21:44:08 2013
6. #      by: PyQt4 UI code generator 4.9.1
7. #
8. # WARNING! All changes made in this file will be lost!
10. **from** PyQt4 **import** QtCore, QtGui
12. **try**:
13. \_fromUtf8 = QtCore.QString.fromUtf8
14. **except** AttributeError:
15. \_fromUtf8 = **lambda** s: s
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.generateButton = QtGui.QPushButton(self.centralwidget)
55. self.generateButton.setGeometry(QtCore.QRect(440, 150, 87, 27))
56. self.generateButton.setObjectName(\_fromUtf8("generateButton"))
57. self.graphicsView = QtGui.QGraphicsView(self.centralwidget)
58. self.graphicsView.setGeometry(QtCore.QRect(20, 200, 600, 300))
59. sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expanding)
60. sizePolicy.setHorizontalStretch(0)
61. sizePolicy.setVerticalStretch(0)
62. sizePolicy.setHeightForWidth(self.graphicsView.sizePolicy().hasHeightForWidth())
63. self.graphicsView.setSizePolicy(sizePolicy)
64. self.graphicsView.setMinimumSize(QtCore.QSize(600, 300))
65. self.graphicsView.setMaximumSize(QtCore.QSize(600, 300))
66. self.graphicsView.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
67. self.graphicsView.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
68. brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
69. brush.setStyle(QtCore.Qt.NoBrush)
70. self.graphicsView.setBackgroundBrush(brush)
71. brush = QtGui.QBrush(QtGui.QColor(0, 0, 0))
72. brush.setStyle(QtCore.Qt.SolidPattern)
73. self.graphicsView.setForegroundBrush(brush)
74. self.graphicsView.setSceneRect(QtCore.QRectF(0.0, 0.0, 600.0, 300.0))
75. self.graphicsView.setObjectName(\_fromUtf8("graphicsView"))
76. self.clearButton = QtGui.QPushButton(self.centralwidget)
77. self.clearButton.setGeometry(QtCore.QRect(310, 90, 51, 27))
78. self.clearButton.setObjectName(\_fromUtf8("clearButton"))
79. self.validateButton = QtGui.QPushButton(self.centralwidget)
80. self.validateButton.setGeometry(QtCore.QRect(370, 90, 87, 27))
81. self.validateButton.setObjectName(\_fromUtf8("validateButton"))
82. MainWindow.setCentralWidget(self.centralwidget)
83. self.menubar = QtGui.QMenuBar(MainWindow)
84. self.menubar.setGeometry(QtCore.QRect(0, 0, 635, 25))
85. self.menubar.setObjectName(\_fromUtf8("menubar"))
86. MainWindow.setMenuBar(self.menubar)
87. self.statusbar = QtGui.QStatusBar(MainWindow)
88. self.statusbar.setObjectName(\_fromUtf8("statusbar"))
89. MainWindow.setStatusBar(self.statusbar)
91. self.retranslateUi(MainWindow)
92. QtCore.QMetaObject.connectSlotsByName(MainWindow)
94. **def** retranslateUi(self, MainWindow):
95. MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow", "Chem Wizard", None, QtGui.QApplication.UnicodeUTF8))
96. self.randomButton.setText(QtGui.QApplication.translate("MainWindow", "Random", None, QtGui.QApplication.UnicodeUTF8))
97. self.smilesLabel.setText(QtGui.QApplication.translate("MainWindow", "SMILES:", None, QtGui.QApplication.UnicodeUTF8))
98. self.inchiLabel.setText(QtGui.QApplication.translate("MainWindow", "InChi", None, QtGui.QApplication.UnicodeUTF8))
99. self.animateButton.setText(QtGui.QApplication.translate("MainWindow", "Animate", None, QtGui.QApplication.UnicodeUTF8))
100. self.nomenclatureLabel.setText(QtGui.QApplication.translate("MainWindow", "Nomenclature:", None, QtGui.QApplication.UnicodeUTF8))
101. self.checkButton.setText(QtGui.QApplication.translate("MainWindow", "Check!", None, QtGui.QApplication.UnicodeUTF8))
102. self.generateButton.setText(QtGui.QApplication.translate("MainWindow", "Generate", None, QtGui.QApplication.UnicodeUTF8))
103. self.clearButton.setText(QtGui.QApplication.translate("MainWindow", "Clear", None, QtGui.QApplication.UnicodeUTF8))
104. self.validateButton.setText(QtGui.QApplication.translate("MainWindow", "Validate", None, QtGui.QApplication.UnicodeUTF8))

## (Incomplete)User Manual