Content

1. **Personal information**

   Protein biosynthesis; Patrick James, 905325, kemian tekniikka, 6.5.2022.

2. **General description**

   The program has a and is able to:

- graphical interface
- Read a DNA-sequence from a file and draw its protein biosynthesis
- The simulation is animated
- unittests for the Reader functions
- DNA-> RNA transcription has a possibility for mistakes, in the translation of DNA there is a probability of 10^-4 for a mistake
- splicing (removing introns from DNA)
    - intron always starts with GU and ends with AG (usual for introns)

   The program has buttons to display different RNA sequences, to make my program unique. The program has been implemented with the hard requirements.

3. **Instructions for the user**

   The program is launched using the main.py. The program reads a .txt file in the path of the program with the given file name. The name of the file is asked using a prompt that opens when the program is launched.

   The program automatically generates appropriate DNA,RNA and peptide chains from the given .txt file. Some user input, such as scrolling the view to see the whole chain and buttons are available. By pressing the buttons, the user can see different phases of the transcription.

   After pressing the TATA-button, the RNA-chain in the top view is shown after each TATA-box. After pressing the TAC button the RNA-chain is shown after each TATA-box+after the first TAC codon has been reached. After pressing Introns-exons button, the same restrictions apply as after the TAC button, but also the RNA-chain only begins after the first GU is reached and the final AG before the next TATA-box is reached.

   Additionally, if one wants to change the probability in a mistake during the transcription, it can easily be changed in the _init_ of the Reader class, by changing the value in DNA_to_RNA().

4. **External libraries**

PyQT5 used for the GUI and graphics classes.

5. **Structure of the program**

The main classes of the program are the Reader, GUI, Amino acid graphics, Nucleotide graphics and Codon classes.

Initially the GUI asks for a filename, but then GUI needs to wait for the Reader to finish. Most of the logic of the program is handled by the Reader class. The Reader class is the class that does its required tasks first, such as constructing appropriate lists for the RNA, DNA and peptide chains from the given .txt file. These lists include information about TATA-boxes, TAC-start codons, and introns.

After the Reader class has constructed these lists, through the main function, the lists are fed to the GUI class, which creates the nucleotide and amino acid graphics objects from the items in the list. In addition, the GUI class also creates the window, separate views for translation and transcription, text labels and such. GUI is a subclass of QMainWindow.

The Codon class is used to build codon objects from given nucleotides in a list. The codon objects are fed through the GUI into the Amino acid graphics class to create the appropriate graphics objects.

the Nucleotide and Amino acid graphics classes create polygons with specific shapes, colors and labels for the given nucleotide or amino acid. These classes are subclasses of the QGraphicsPolygonItem class.

6. **Algorithms**

The program opens the file->reads its contents character at a time ->creates an ordered list from these characters in the same order as in the file->objects are created from every translatable nucleotide base in the list->objects are put in an ordered list.

In addition, the characters are checked individually to find the tata-boxes, starting tac codon after each tata-box, and introns between 2 tata-tac combo areas. Simple “for i in range(x,y)” loops are mostly used throughout the program, sometimes nested. The way I programmed these was to check every character or group of characters in the list. If the correct character(s) was found in list[i] for example, the program would insert the index “i” into a dictionary or list to be used later when building the RNA-chain.

There are surely more efficient ways to go through the lists, like using list comprehension, but this was a simple and easy way of doing it that I knew.

7. **Data structures**

The locations of the TATAs, TACS and introns are put into a dictionary marking their index, where the key of the dictionary item is the starting location in the chain and the value of the item is the ending location in the chain. For example in a nucleotide chain that is 1000 bases long, the area between one tac start-codon and the next tata box could be expressed as 234:295. These are the nucleotides that are expressed in the RNA chain and subsequently expressed in the peptide chain.

The RNA lists are built on the index dictionaries. If there should not be an RNA nucleotide at that location because it is not expressed, the list will contain a " " blank space. If there should be an RNA nucleotide there, it will be the complementary nucleotide to the one in the DNA chain at that location.

For example a DNA sequence and its complementary RNA sequence could be expressed like:

['T','A','T','A','T','A','C','T','A']

[' ',' ',' ',' ',' ',' ','U','G','A','U']

8. **Files**

The program reads a .txt file containing the nucleotide bases (for example: "AAGCTGAAACGT"). The program should handle files with a string of 500-1000 bases in length. The file contains no commas, spaces, or other characters except AGCTU. "\n" newlines can be handled by the program, and they are just skipped when reading the file.

I copied all the sequences from this website straight into a blank .txt file. I only deleted the text "Random DNA Sequence results

>random sequence 1 consisting of 1000 bases." from the top of the given sequence.

https://www.bioinformatics.org/sms2/random_dna.html

9. **Testing**

   During the testing of the project, I mainly tested the program with print commands and graphically, to make sure it was putting the correct objects in the correct order in the list of GUI etc.

   More specifically:

- Test for multiple TATA-boxes in a .txt file.
- Test for valid/invalid codon triplets
- Test that objects are created correctly and placed in the correct order

   Some unit testing was done on the Reader class to make sure it properly read the file and created appropriated lists when given different .txt files. Specifically it tested that the Reader would always create a proper DNA-list from the file.

   I also made tags that show the nth position of nucleotide object in the chain above the graphical object. This was very useful for testing and debugging.

10. **The known shortcomings and flaws in the program**

    The peptide chain and translation section might have some mistakes, but mainly I think they're biological and not programming mistakes. I'm not sure if the TAC starting codon and GU and AG codons are supposed to be included in the transcription RNA chain, so I left it kind of vague in the program. Also it wasn't specified what to do with the stop codons in the instructions, so I left them in the peptide chain without addressing them.

    There are some methods and attributes in the program that realistically could be combined because they serve very similar purposes, or separated, because there are many tasks handled in the method that should be their own individual method.

    The program is designed to be used in fullscreen, so the graphics don't scale well for a small window. There must be some easy way to fix this in PyQt, like **resizeEvent**(), but for this task it didn't seem that important.

    In the GUI class, on line 125, when adding the RNA graphics item, I had to set the x-coordinate to be 1 ahead of the x-coordinate of the DNA graphics items for them to line up properly. This bug has been here since close to the beginning of the project and I have no idea where it comes from.

11. **3 best and 3 worst areas**

The best areas in the program in my opinion are the Nucleotide graphics class and the GUI class's ability to update the RNA chain with the buttons.

The Reader class is very heavy and bloated, it serves many tasks in reading the file and creating appropriate lists for the nucleotides and amino acids. Changing one thing in the Reader class or its methods will often lead to many other things needing to be tuned accordingly, since they are all very closely connected in complex ways. The way I implemented the ability to find the tata boxes and subsequent TACs and introns was very messy, but I couldn't come up with a better way of doing it. Some of it is so messy, that even if I add many comments to explain a certain method, it may still be unclear to the examiner. If I did this project again, I think I would have separated the reading functions into one class, and the list analysis loops into another.

12. **Changes to the original plan**

I realized it didn't make sense to make a separate class for all the nucleotides, since they share practically all the same attributes and have very little difference. Also, there was little benefit in creating Nucleotide objects and Amino Acid objects for each element in the lists. I could easily just make the graphics objects of them straight away, because they had such simple and few attributes, and after generating the graphics once, they did not need to be moved or changed in anyway, in contrast to the Robotworld program. This change also made debugging and testing much easier.

I planned on making some 3D graphics for the amino acid graphics objects to make my project unique, but from trying to read how to do that, I realized it would be very difficult to combine 2D and 3D graphics into a single view and it would've meant rewriting most of my graphics code, and I didn't think it was worth it, because the 3D graphics didn't really add anything useful or nice to the program.

Instead I decided to make some buttons that allow the user to see the RNA chain unedited, after tata boxes have been located, after tac+tata boxes have been located, and after introns and exons have been managed. I think this was a good decision, because these buttons are a useful and cool feature.

13. **Realized order and scheduled**

-Initially I defined the Nucleotide class and Reader class according to my schedule, did some testing just using print commands.

-I created and finished the Codon class

-created the GUI class somewhat

-created the Nucleotide graphics class

-More work on the GUI and Reader classes

-I Realized I didn't need some classes I created, like the Nucleotide class, so I removed them. Overall, it wasn't a big mistake or waste of time luckily.

-Unittesting

I only did unittesting much later in the project, because I felt I couldn't get much use out of it early on. I spent much more time on the Reader class than I initially planned in my schedule. Getting the TATA-box -and TAC-indexes using the Reader class' methods took much longer than expected.

I was only planning on reaching the medium requirements for the project, but because I had more time and it didn't seem like the hard requirements took much time, I added them at the end.

14. **Assessment of the final result**

    All together I think the program is good. It meets all of the medium requirements and hard requirements. I think the graphics and GUI are good and serve their purpose. The weakest area is still the Reader class in my opinion. It does everything it should and it works fine, but it is a bit messy and inelegant. Given the instructions, I don't think there are any major short comings or bugs or issues.

    Because the data structures used are mutable and easily modifiable, like the dictionaries, I think it can be expanded upon relatively easily. I've tested the program and it can handle generating a chain at least 500 thousand bases long, so it should be optimized enough to be expanded upon into a bigger program.

    If the program should be expanded for more features or abilities, I think creating Nucleotide and Amino Acid objects would be useful, instead of just creating their graphic objects from a list. For example, attributes such as charge and size could be defined for the objects, and these could be useful for a future expansion of the project.

    The program could be improved in the future by adding some more user control. For example, adding the possibility to change the size of graphics objects, being able to export the peptide chain and RNA as a .txt file and such features.

15. **References**

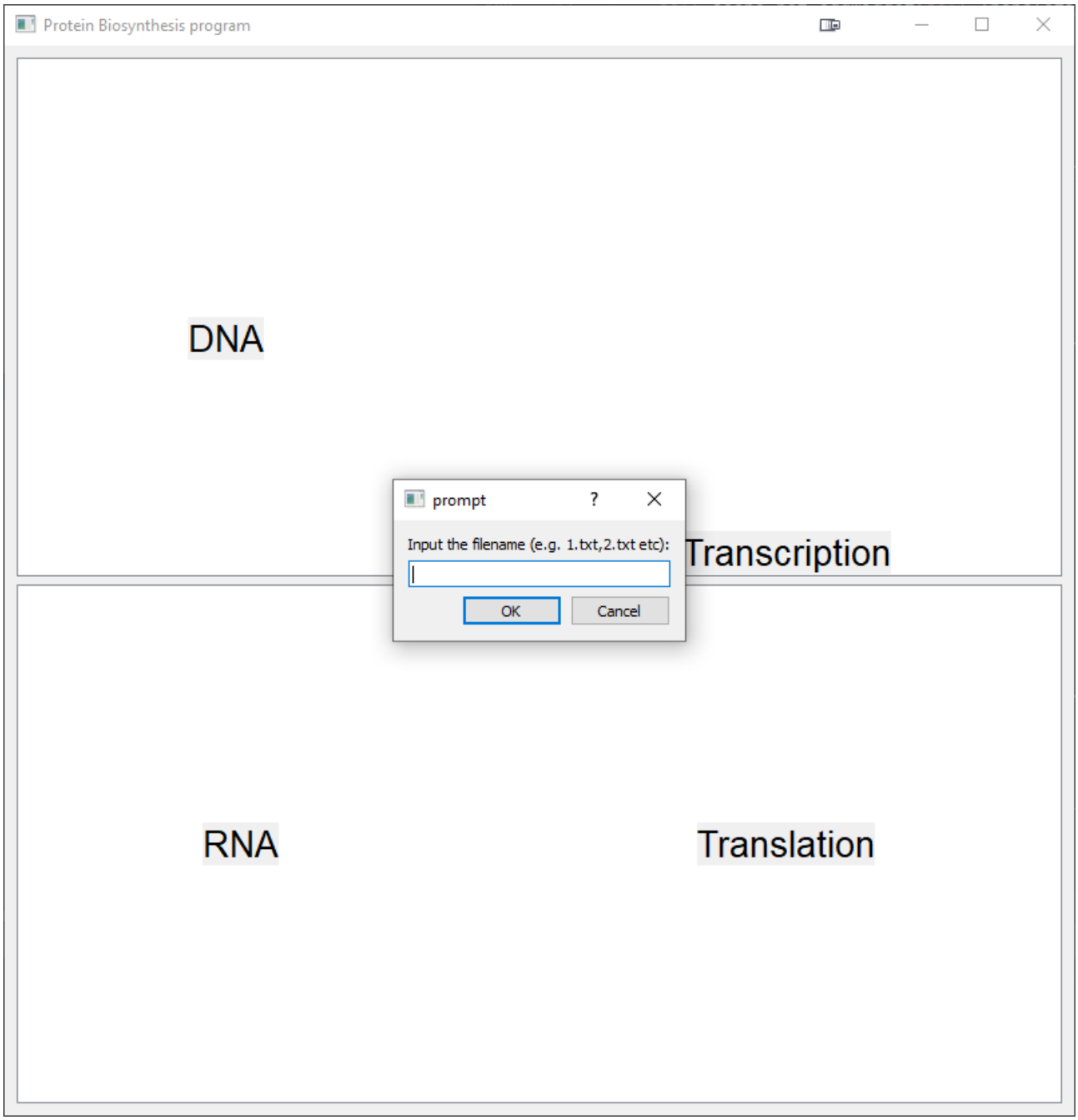
    I will mostly use various PyQT5 documentations and the course webpages(https://doc.qt.io/qtforpython/).

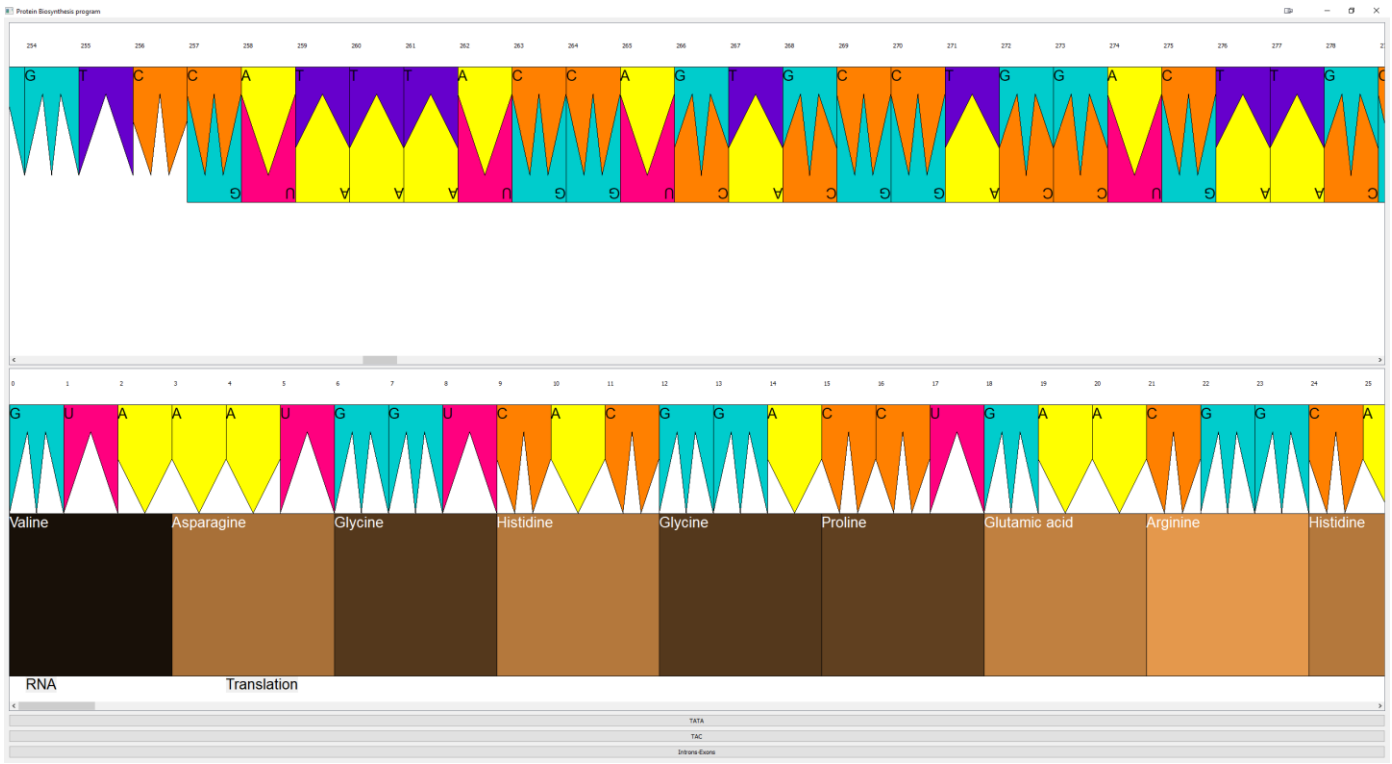
    I will use a website to generate random DNA sequences (https://www.bioinformatics.org/sms2/random_dna.html).

    Misc

    https://www.reddit.com/r/learnpython/comments/6gseko/pyqt5_qcolor_doesnt_accept_variable/

16. **Attachments**


Protein Biosynthesis program
DNA
prompt
Input the filename (e.g. 1.txt,2.txt etc):
OK
Cancel
Transcription
RNA
Translation


Protein Biosynthesis program
Valine
Asparagine
Glycine
Histidine
Glycine
Proline
Glutamic acid
Arginine
Histidine
RNA
Translation
TATA
TAC
Introns-Exons