Content

1. **Personal information**

   Protein biosynthesis;Patrick James, 905325, kemian tekniikka,17.2.2022.

2. **General description**

   The program will have a and be able to:

- graphical interface
- the program reads a DNA-sequence from a file and draws its protein biosynthesis
- the simulation is animated
- unittests for at least part of the program

  Ideally the program would reach the medium requirements shown above (or higher). If I have enough time, the program will also have the hard requirements shown below.

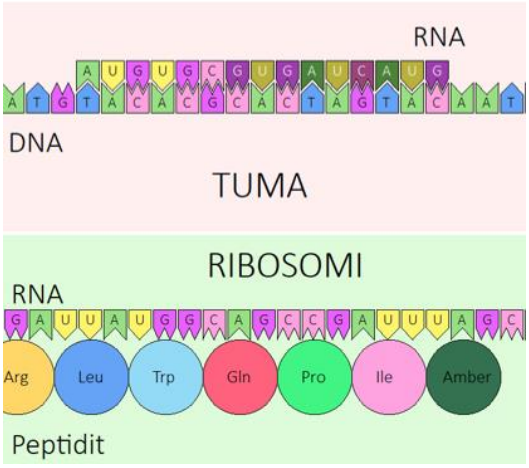
- DNA-> RNA translation has a possibility for mistakes, in the translation of DNA there is a probability of 10^-4 for a mistake, but you can pick the odds yourself
- splicing (removing introns from DNA)
    - introns can be separated from exons e.g. by realising that an intron always starts with GU and ends with AG (usual for introns)
    - introns don't have to be literally spliced. It's sufficient to remove them from the sequence in a way that is clearly shown in the program

  I will be using a mix of 3D and 2D- graphics to make my project unique.

3. **Instructions for the user**

   The program will be 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 some kind of prompt, graphical or just text based.

   The program should automatically graphically present the DNA/RNA chain without user input. Some kind of user input, like moving up and down the chain using the mouse scroll wheel and having that be animated would be ideal.

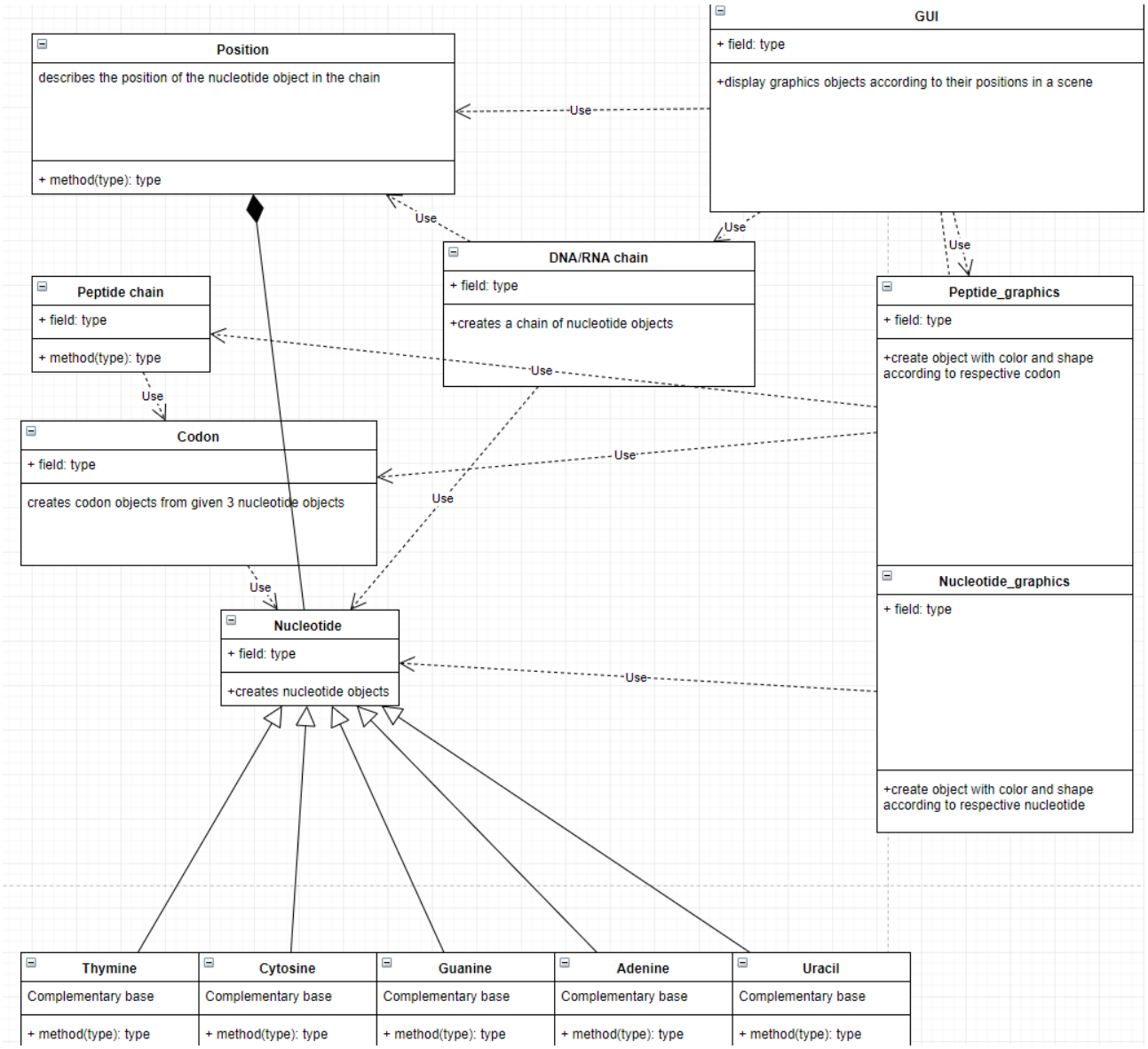


The program will look something like the example GUI in the instructions. It is an efficient design and conveys what it needs to.

4. **External libraries**

   PyQT5 used for the GUI. No other libraries planned for the time being, and I don't think will be needed.

5. Structure of the program



The central focus of the program is the Nucleotide-class. Most classes will either be subclasses of it or use it in some way. Methods for the nucleotides include creating nucleotide objects, and returning them and returning their positions.

The graphics classes aren't very planned out yet, but I plan on focusing on those later after I get the logic of the program sorted.

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.

   I'm not sure what else in the project could be considered an algorithm.

7. **Data structures**

   Nucleotide base objects will be stored in a list of some sort. A regular list may be too limiting for the hard requirements of the project, so a dictionary/nested dictionaries or linked list may be better. It could be useful to store codons in their own nested dictionary, with the key value being the name of the amino acid the codon codes for.

   Although dictionaries have the ability to be mutable and changed at a later date, I don't think I will need that for the medium requirements. If I choose to try to reach the hard requirements, having the dictionary be mutable will be useful for splicing introns out of the DNA.

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.

   Translation of the DNA chain to RNA does not start before the TATA box is reached in the string. Translation does not start at the TATA box, but the start codon after it (TAC in DNA, AUG in RNA).

9. **Testing**

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

   Testing using random DNA chains, and potentially some known real world ones as well, to compare that the program truly is outputting the correct DNA/RNA/Peptide-chains.

- 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

10. **The known shortcomings and flaws in the program (later)**

    Describe here all the shortcomings and flaws in your program that you are aware of. Explain how you would fix these problems if you would continue the project. *The less the assistant finds shortcomings in the parts that you claim are working, the better*. So be honest. In addition, well-explained development suggestions are a sign that you have thought about the problem and its solution, and that you are capable of critical evaluation of your own work.

11. **3 best and 3 worst areas (later)**

    The assistant uses a lot of time to get to know your program but does not necessarily see or feel your implementation the same way that you do. If there are some parts in the program that you think are brilliant, explain 1-3 of them here with brief arguments. If there are parts in your program that you know to be weak, you can mention these as well. By doing this the possibility that these weak parts dominate the grading will decrease significantly. Here you can also verbally explain how you would have fixed these problems.

12. **Changes to the original plan (later)**

    Did you do something differently than you had originally planned? Why? Was the time you scheduled for the project near correct? Why not? How about the implementation order?

13. **Schedule**

-planning some more = 5 hours

-reading the file appropriately and creating a list = 3 hours

-testing = 2 h

-creating a nucleotide object with some of its methods, class variables, instance variables, and attributes = 4 h

-testing = 2 h

-creating the subclasses from the nucleotide parent class and their methods etc. = 4 hours

-testing = 1 h

-Defining the position class and DNA/RNA-chain class =5 hours

-some unittesting = 3 h

-basic graphics = 10 h

-testing = 5 h

-defining the peptide chain class and codon class = 5 hours

-Graphics(drawing shapes, coloring, animating, text) = 25 hours

Trying to get the first 5-6 done before the first checkpoint. Basic graphics and testing by second checkpoint.

checkpoint1 by 25.3.2022:

checkpoint2 by 15.4.2022:

Explain here generally in which order the project was ultimately implemented (preferably the dates as well). Was there a change made in the plan?

14. **Assessment of the final result (later)**

    A summary and self-evaluation of your final project, which is allowed to repeat some of the above-mentioned things.

    Assess the program's quality, explain its strong and weak parts. Are there significant shortcomings in the project and what are the reasons for them (a good justification in the document can compensate for small shortcomings)? How could the program have been or could be enhanced in the future? Could the solution methods, data structures, or the class division have been implemented in a better way? Is the program's structure suitable for making changes and expanding? Why or why not?

15. **References**

    What books, web pages, or other material have you used? All sources must be reported, even if they included only the course textbook and API description of the base class libraries.

    I will mostly use various PyQT5 documentations and the course webpages.

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

16. **Attachments (later)**

    As attachments, you should include at least **a couple of demonstrating examples of running the program**, which are easy to make with a script program in unix environments. A script program starts by writing `script` and ends by pressing CTRL-D or exit in the unit shell. While the script is running, it saves into a file everything that appears on the screen and the input a user writes. In graphic project topics running examples are not required, but a couple of actual pictures of using the program as an independent attachment or with the user instructions would be appreciated. (Pictures can be snapped in Linux or Windows environments by pressing Alt+PrintScreen keys. Depending on the operating system, you can either save the file right away or onto the clipboard, from where it can be saved through some drawing software.)