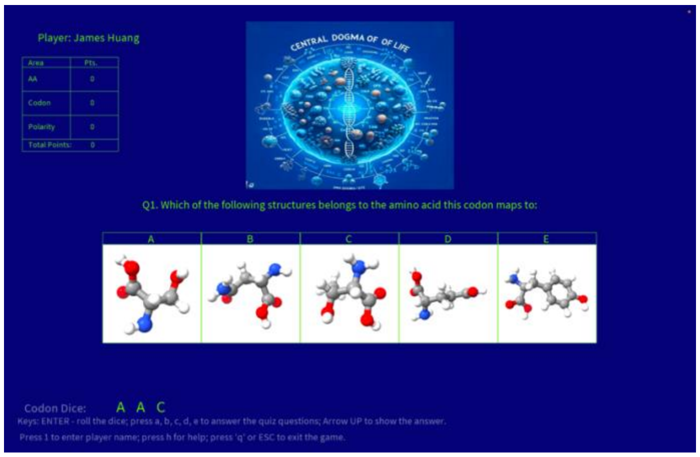
CSCI-UA. 101

INTRODUCTION TO COMPUTER SCIENCE

FALL 2023 PROJECT REPORT

The Central Dogma of Life

James Huang



Player: James Huang

Area	Pts.
AA	0
Codon	0
Polarity	0
Total Points:	0

Q1. Which of the following structures belongs to the amino acid this codon maps to:

Codon Dice: A A C

Keys: ENTER - roll the dice; press a, b, c, d, e to answer the quiz questions; Arrow UP to show the answer.

Press 1 to enter player name; press h for help; press 'q' or ESC to exit the game.

Instructor: Gizem Kayar Ceylan, PhD.

December'23

SECTION 1 - INTRODUCTION

Game Idea

A serious player rolls a “Codon Dice” to get dynamic quiz questions to test and build his/her knowledge on the Central Dogma of Life, aka, the genetic information mapping between codons, amino acids, amino acid molecular structures and polarity properties. A soothing piece of piano music accompanies throughout the game to help alleviate the pain and brain damage inflicted by the quiz questions.

Background and Justification

The Central Dogma of Life outlines the process how DNA contains the information needed to make all the proteins within a cell, and how that information moves through the stages of transcription to RNA in units of codons and translation to amino acids to futher assemble into proteins (Crick, 1958). A RNA codon is a triplet of nucleotides (4 kinds: A,G,C,U) that codes for 1 of 20 amino acids found on earth, the basic building blocks of proteins. Technically we have a 4x4x4 combinations to map to the 20 total amino acids. For example, Figure 1 shows the mapping, and Figure 2 illustrates 2 amino acids’ structures.

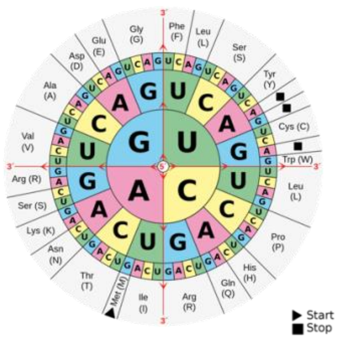


Figure 1. Codon ~ Amino Acid (Wikipedia)

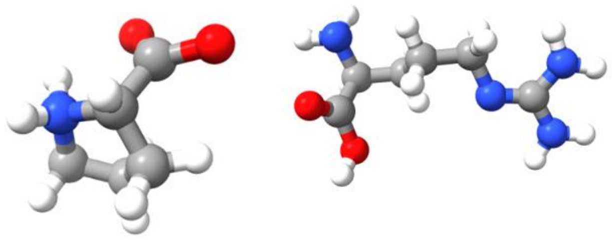


Figure 2. Molecular structure of amino acids HIS and ARG

For those aspiring to have a serious career in the biomed field, mastery of the Central Dogma of Life is a must. Given a codon, immediate recognition of what amino acid it encodes is expected; and vice versa. However, pain and frustration amount around this topic, as such mappings and the 20 amino acids' unique structures and properties are challenging to memorize.

To help reduce all future scientists' and MDs' headache and pain, I have decided to apply what I have learned in CS101 to develop a serious game for them.

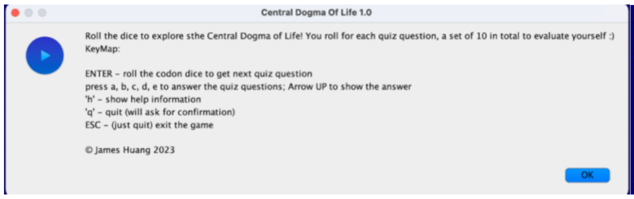
Game Scenario

My game will help reinforce this knowledge by having the player "roll" a virtual "codon dice" to get a new quiz question that tests the mapping between codon and amino acid's name, molecular structures, and polarity properties. Correct answers earn points, with a running score displayed on screen.

After rolling the dice and answering 10 quiz questions, the game round ends, and the players sees summary of his final scores. The game then offers the option to play again and continue practicing and learning.

Gameplay Details

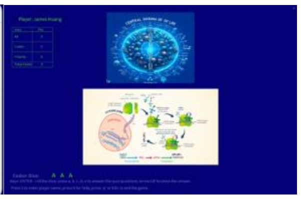
- Game Controls:


Central Dogma Of Life 1.0

Roll the dice to explore sthe Central Dogma of Life! You roll for each quiz question, a set of 10 in total to evaluate yourself :)
KeyMap:

ENTER – roll the codon dice to get next quiz question
press a, b, c, d, e to answer the quiz questions; Arrow UP to show the answer
'h' – show help information
'q' – quit (will ask for confirmation)
ESC – (just quit) exit the game

© James Huang 2023



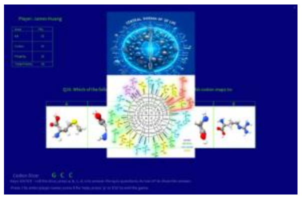
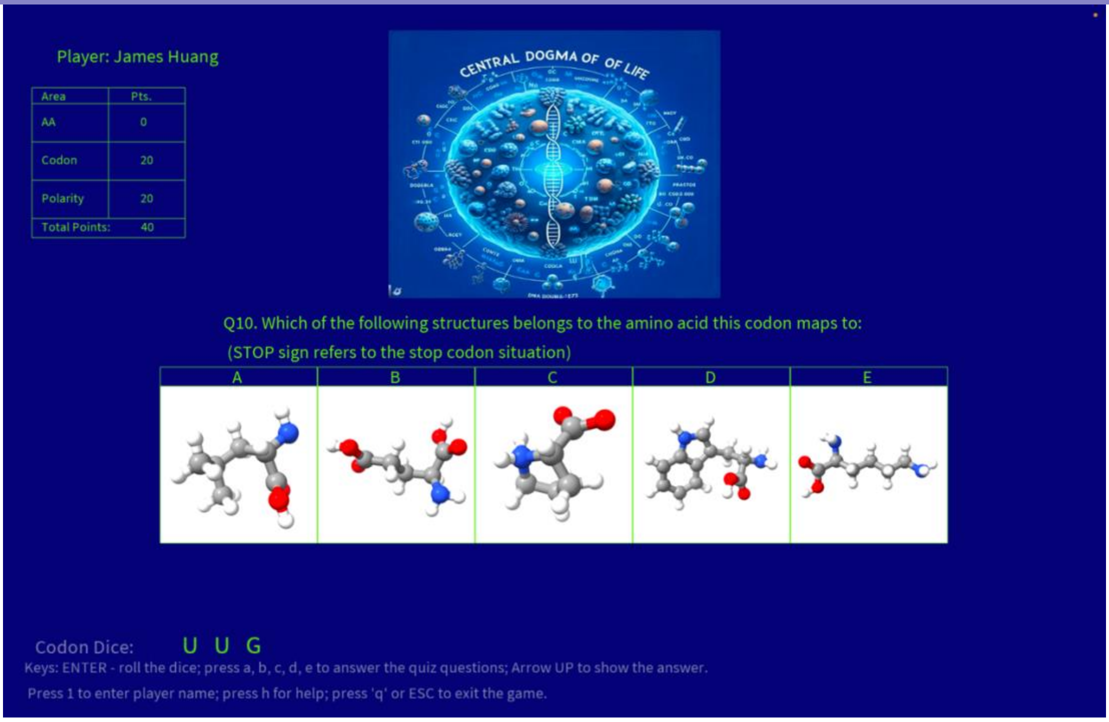


Figure 3. Help window activated by press 'h' key. Figure 4. Opening Screen. Figure 5. Close Screen

- Game features:
  - Interactive rolling your own quiz question, dynamically generated using the codon dice result
  - Nice view of 20 amino acids' 3d molecular structures
  - Opening screen presents the protein synthesis flow, a good learning guide
  - Closing screen presents the codon-amino acid mapping table
  - A healing piece of piano music accompanies throughout the game

SECTION 2 - IMPLEMENTATION



Player: James Huang

Area	Pts.
AA	0
Codon	20
Polarity	20
Total Points:	40

Q10. Which of the following structures belongs to the amino acid this codon maps to:

(STOP sign refers to the stop codon situation)

Codon Dice: U U G

Keys: ENTER - roll the dice; press a, b, c, d, e to answer the quiz questions; Arrow UP to show the answer.

Press 1 to enter player name; press h for help; press 'q' or ESC to exit the game.

Figure 4. Game screen

Class Diagram (UML)

Below is a class diagram of the game files. I have learnt to code up using plantUML, a popular UML modeling language with online references at ***https://plantuml.com/en/class-diagram.*** In the bottom right of Figure 5, there is a Legends class diagram shows public, private, protected, default access level data and methods' legends.

The main sketch is an unnamed class that inherits from PApplet. It has two main inherited methods: setup() and draw(). Code in setup() runs once unless explicitly called again, so I used this section to set up the full screen mode, initializes the player instance, reading and loading images files, read and play the mp3 file. Code in draw() refreshes 60 times per second, so I used this section to paint the on screen elements such as the logo, the question, the choices, the molecular structures, the rolling dice, the score report section, the on-screen help text, the welcome screen, etc.

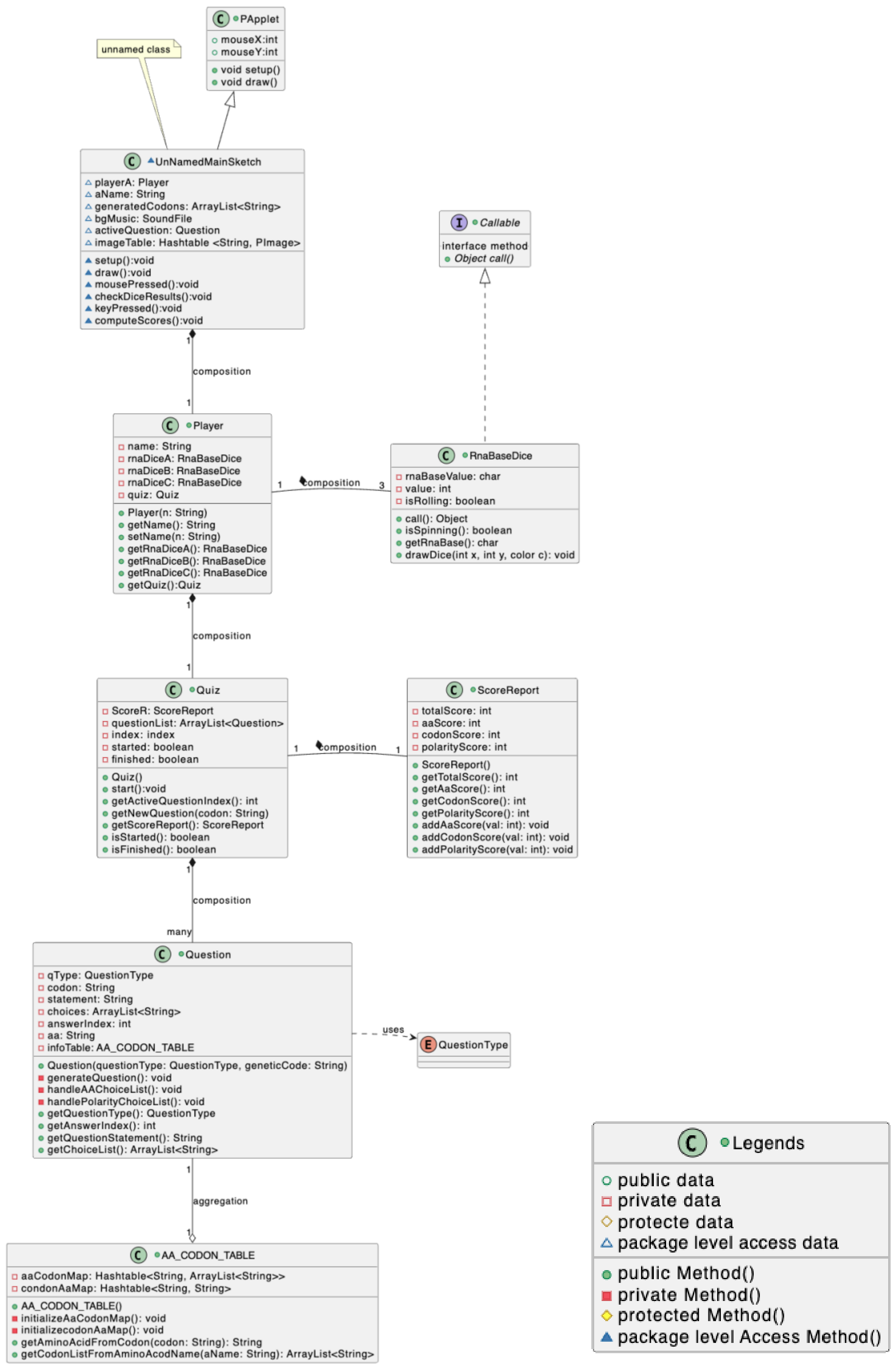


Figure 5. Class Diagram (left); Legends (right bottom corner)

The unnamed main sketch class declares and initializes an instance of class Player. So, the relationship here is 1:1 composition. The Player class declares and initializes 3 RnaBaseDice instances to form the rolling codon dice, with 1:3 composition. The Player class also has declaration and initialization of a Quiz class object, with 1:1 composition. The Quiz class declares and initializes an ArrayList<Question> instance, with 1:many to Question class and 1:1 composition to ScoreReport instance. The Question class uses information from class AA_CODON_TABLE, and enum QuestionType. The relationship between Question and AA_CODON_TABLE is aggregate.

A notable class is RnaBaseDice, the class that is responsible for generating a single base like 'A', 'G', 'C', or 'U' upon rolling. It implements Java's `Callable` Interface, which I will discuss in next section.

List of Techniques:

- **Data Structures**

`Array`, `ArrayList` and `Hashtable` are used extensively. `Array` is great for accessing by index if the size of the element list does not change. For example, I used the following array to keep track of the amino acids:

```
public final static String [] entries = {
ALA, ASN, ASP,  ARG, …
};
```

`ArrayList` is suitable for a list whose size changes dynamically. For example, when forming a question's multiple choices, I needed to grow the choices from 0 to 5, so I used `ArrayList`*:*

```
public class Quiz
{
   private ArrayList<Question> questionList =null;
   …
     questionList = new ArrayList<Question> ();
   …
           q = new Question(QuestionType.AA, codon);
           questionList.add(q);
```

`Hashtable` is suitable for (key, value) pair storage and fast retrieval by key. For example, I used a `Hashtable` to store the amino acids' image files with their name as key, this way the program can retrieve the image file directly using the amino acid's name, with a constant O(1) time.

```
Hashtable <String, PImage> imageTable;
imageTable = new Hashtable<String, PImage>();
…
PImage p;
for (int i=0; i< AA.entries.length; i++)
{
    p = loadImage(dataFolder + AA.entries[i] + ".png");
    imageTable.put(AA.entries[i], p);
}
```

- **Class Decomposition**

Using OOP design learned from Professor Gizem Kaya, I have managed to decompose the soltuion into different classes: `Player`, `RnaBaseDice`, `Quiz`, `Question`, `ScoreReport`, etc. This way the logic is quite clear, which greatly simplifies the design. Plus, I get to enjoy the benefits of OOP's encapsulation, inheritance and polymorphism.

- **Non-Blocking Client UI: Threading, Callable interface, FutureTask/ReturnTask**

I have learnt to use threading to enable parallel tasks to handle background jobs when the GUI thread ensures responsiveness to the users:

```
Thread threadC = new Thread(cReturnTask);

threadC.start();
```

I have also learnt to make the `RnaBaseDice` implements the `Callable` Interface,

```
public class RnaBaseDice implements Callable<Object>{

public Object call() throws Exception { … }
```

so that the rolled value can be fetched via `FutureTask`/`ReturnResult`

```
      if (aReturnTask.isDone() …

          Integer aResult = (Integer)aReturnTask.get();
```

- **Exception Handling**

When dealing with IO or threading executions, I used exception `try{ } catch{}` logic that I learned in CS101 to deal with error situations.

SECTION 3 - CONCLUSION

I have verified the game works as expected by carefully walking through the game scenario steps. I have also checked the game's runtime outputs to make sure there are no errors. As of now, the whole thing works fine as designed.

Implementation Related Difficulties

One of the main difficulties is to learn about all the codon and amino acid mapping, as well as digging out all the amino acids' structures. My high school science class has proven to be helpful, so has NYU library. A second difficulty is the time constraints during the last weeks of the semester when various final exams and projects join forces to deprive me of sleep time. However I have found this project to be so interesting that I am willing to sleep less ☺

Evaluation

Future improvements can be made to show the real 3D models of the amino acids and provide more interactive experience. It can also be extended to be multi-player mode to encourage competition and be more fun.

Throughout the project, I have learnt quite a few things like how to do processing programming, how to choose the right data structures, how to use OOP in real coding, how to do threading, how to do event-driven programming, how to do async calls and how to get the results, etc. Meanwhile, I have got myself a heavy dose of Molecular Genetics and Biochemistry, which is an amazing realm.

Overall, I have had a great time working on the project using what I have learned from Professor Gizem Kayar's excellent class. Merry Christmas!

REFERENCES

Callable and Future Tasks: https://www.javatpoint.com/callable-and-future-in-java

Crick, F. (1958). On protein synthesis. *Symposia of the Society for Experimental Biology*, 12, 138–163.

Codon-Amino Acid Mapping:
https://commons.wikimedia.org/wiki/File:Aminoacids_table.svg

Karen Steward. “Essential Amino Acids: Chart, Abbreviations and Structure.” *Applied Sciences from Technology Networks*, Technology Networks, 26 Sept. 2019, www.technologynetworks.com/applied-sciences/articles/essential-amino-acids-chart-abbreviations-and-structure-324357.

Mattick, J. S., & Makunin, I. V. (2006). Non-coding RNA. *Human molecular genetics,* 15(suppl 1), R17-R29.

Molecular 3D model pictures: https://3d.nih.gov/entries/3dpx-011342

mp3: http://tools.liumingye.cn/music_old/

PlantUML. "Class Diagram." https://plantuml.com/en/class-diagram