Topic: A Programming Language for Modeling and Simulating Biological Systems

G31 Members:

1. Aditya Hriday Sahu (2020A7PS0144G)	4. Shreyas Santosh Pawar (2019B1A70994G)
2. Yogesh Metukuru (2020A7PS0098G)	5. Aaryan Marar (2020A7PS0987G)
3. Subramanian V (2020A7PS1371G)	6. Monit Agrawal (2020A7PS0120G)

The two target domains that we initially considered as potential target domains were:

- A programming language or a framework for a high-level language for the simulation and modeling of hardware components and processors
    - This language would aim to be different from typical low-level hardware description languages (HDL) such as Verilog or VHDL
    - It would be rather a high-level programming language with greater ease of use and implementation similar to Python
    - It would aim to use the object-oriented programming and functional programming paradigms mostly for modeling and simulation purposes
- A programming language for modeling and simulating biological systems including a standard template library of some essential systems such as Neurons, Generic Amoeba/Paramecium, or similar unicellular protozoans
    - This language would again aim to be a high-level language that will be a lot easy and quick to learn by aspiring biologists or even professional researchers for computationally simulating simple biological systems or modeling complex structures
    - It would dominantly use the two paradigms throughout the standard library implementation:
        - Object Oriented Programming Paradigm
        - Structural Programming Paradigm
    - It would aim to include efficient algorithmic implementation using relevant data structures and predefined custom methods specifically designed for the purpose of biological modeling and simulation

We arrived at the final target domain, “A Programming Language for Modeling and Simulating Biological Systems” due to the following reasons:

- Implementing hardware design, modeling, and simulation should generally be done with the help of a low-level programming language because low-level programming languages are much closer to the hardware level when compared with high-level languages
- One of us on our team has already worked on a similar project which aims to simulate parasitic cells computationally using the object-oriented programming paradigm
- The idea of implementing neurons and simulating them using efficient logical and bit operations in a structural programming paradigm sounds promising

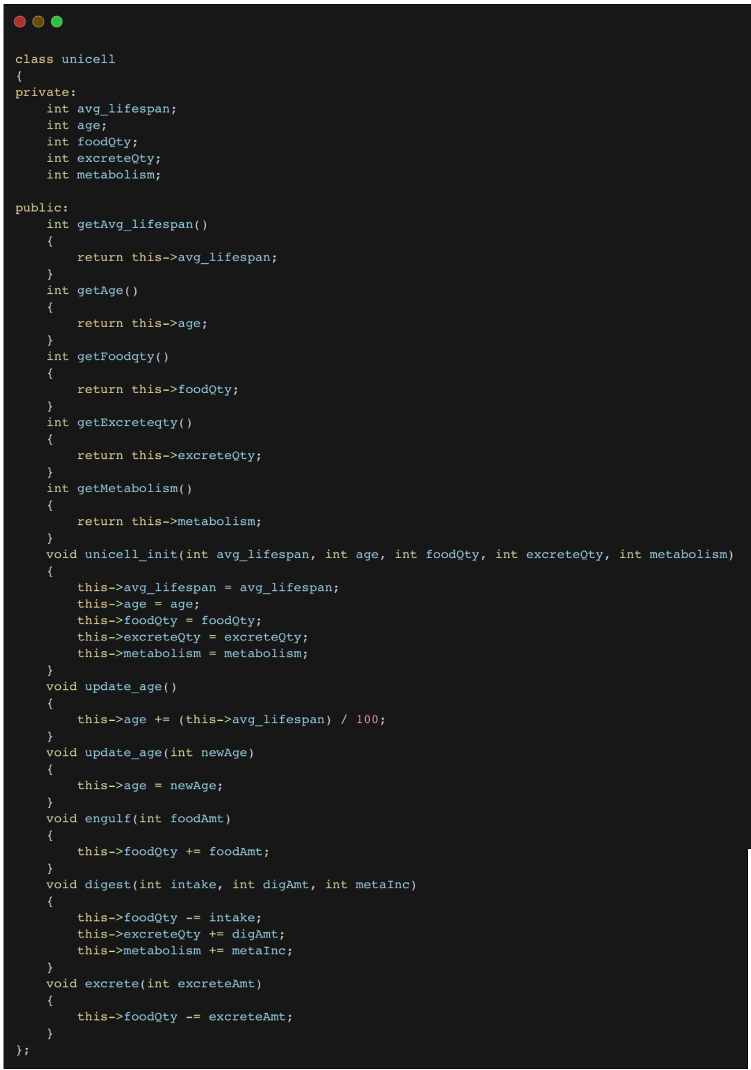
Implementation and Optimization Strategies:

- A typical unicellular organism:
    - The standard template library will have “unicell_sexual.h” and “unicell_asexual.h” importable header packages (the .h file extension is just for representational purposes in this context for header files)

- The most common type of sexual reproduction that occurs in unicellular bacterium and algae is conjugation
- Conjugation involves two cells, one is called an F^- cell or the recipient cell, and the other is called an F^+ cell or the donor cell
- After conjugation, the F^- cell becomes an F^+ cell and is ready to sexually reproduce through conjugation with another F^- cell
- One can instantiate unicell_sexual or unicell_asexual objects and take advantage of the built-in instance variables and methods such as:

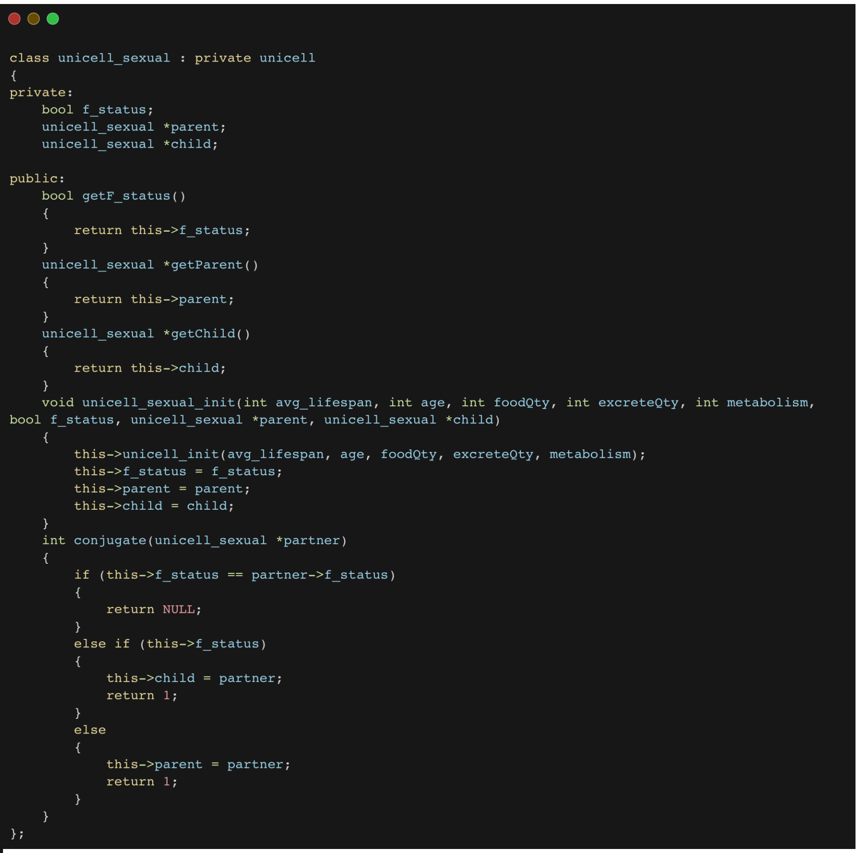
- unicell_sexual.f_status
  - Data type: boolean
  - f_status == true indicates that the cell is F^+ otherwise F^-
- unicell_sexual.parent
  - Data type: unicell_sexual *
- unicell_sexual.child
  - Data type: unicell_sexual *
- unicell_sexual.avg_lifespan
  - Data type: int
- unicell_sexual.age
  - Data type: int
- unicell_sexual.update_age()
  - Increments unicell_sexual.age by 1% of unicell_sexual.avg.lifespan
- unicell_sexual.update_age(int newAge)
  - Updates unicell_sexual.age to newAge
- unicell_sexual.foodQty
  - Data type: int
- unicell_sexual.excreteQty
  - Data type: int
- unicell_sexual.metabolism
  - Data type: int
- unicell_sexual.engulf(int foodAmt)
  - unicell_sexual.food += foodAmt
- unicell_sexual.digest(int intake, int digAmt, int metaInc)
  - unicell_sexual.foodQty -= intake
  - unicell_sexual.excreteQty += digAmt
  - unicell_sexual.metabolism += metaInc
- unicell_sexual.excrete(int excreteAmt)
  - unicell_sexual.excreteQty -= excreteAmt
- unicell_sexual.conjugate(unicell_sexual * partner)
  - Will return the “NULL” object if both the cells are F^+ or both the cells are F^-
  - Else, if this.f_status == true, then this.child = partner and will return non-null object
  - Else, this.parent = partner and will return non-null object

- unicell_asexual.avg_lifespan
  - Data type: int
- unicell_asexual.age
  - Data type: int
- unicell_asexual.update_age()
  - Increments unicell_sexual.age by 1% of unicell_sexual.avg.lifespan
- unicell_asexual.update_age(int newAge)
  - Updates unicell_sexual.age to newAge
- unicell_asexual.foodQty
  - Data type: int
- unicell_asexual.excreteQty
  - Data type: int
- unicell_asexual.metabolism
  - Data type: int
- unicell_asexual.engulf(int foodAmt)
  - unicell_asexual.food += foodAmt
- unicell_asexual.digest(int intake, int digAmt, int metaInc)
  - unicell_asexual.foodQty -= intake
  - unicell_asexual.excreteQty += digAmt
  - unicell_asexual.metabolism += metaInc
- unicell_asexual.excrete(int excreteAmt)
  - unicell_asexual.excreteQty -= excreteAmt
- unicell_asexual.fission(vector<unicell_asexual *> * children, vector<int> metabolism, int n_ary)
  - Will push pointers to child unicell_asexual in the vector such that foodQty, and excreteQty are distributed uniformly among children
  - Metabolism of each child is fetched from the metabolism vector
  - Age of each child is made 0



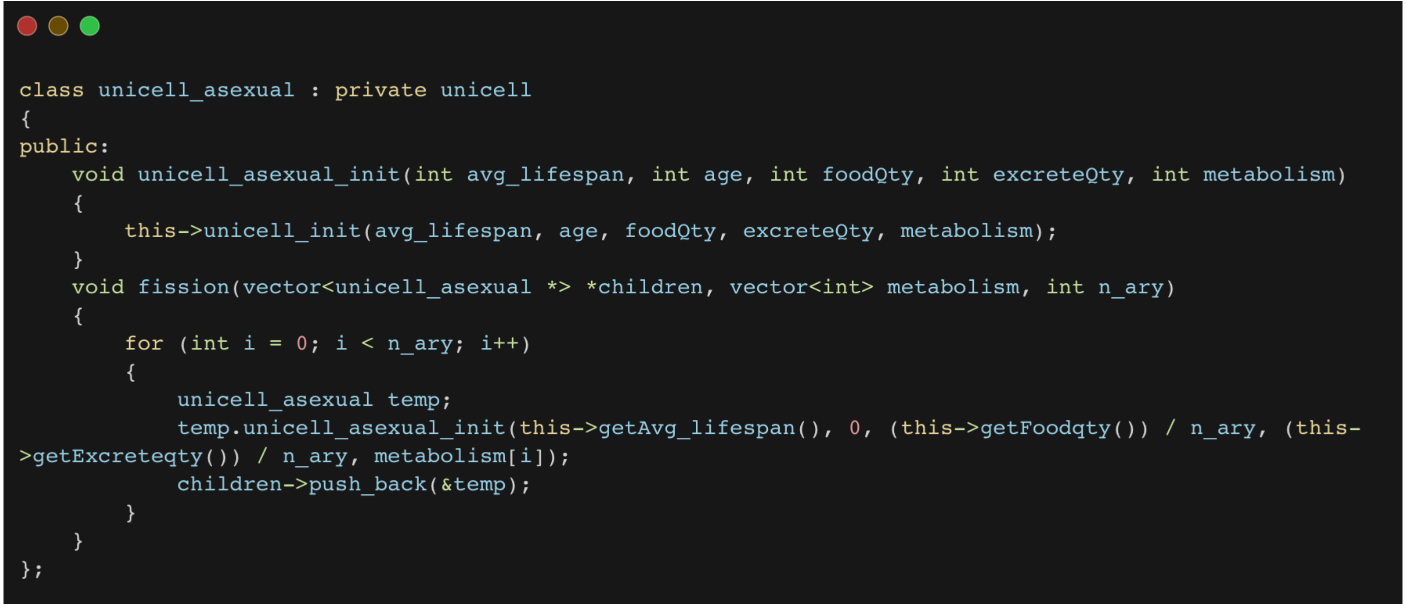
```cpp
class unicell
{
private:
    int avg_lifespan;
    int age;
    int foodQty;
    int excreteQty;
    int metabolism;

public:
    int getAvg_lifespan()
    {
        return this->avg_lifespan;
    }
    int getAge()
    {
        return this->age;
    }
    int getFoodqty()
    {
        return this->foodQty;
    }
    int getExcreteqty()
    {
        return this->excreteQty;
    }
    int getMetabolism()
    {
        return this->metabolism;
    }
    void unicell_init(int avg_lifespan, int age, int foodQty, int excreteQty, int metabolism)
    {
        this->avg_lifespan = avg_lifespan;
        this->age = age;
        this->foodQty = foodQty;
        this->excreteQty = excreteQty;
        this->metabolism = metabolism;
    }
    void update_age()
    {
        this->age += (this->avg_lifespan) / 100;
    }
    void update_age(int newAge)
    {
        this->age = newAge;
    }
    void engulf(int foodAmt)
    {
        this->foodQty += foodAmt;
    }
    void digest(int intake, int digAmt, int metaInc)
    {
        this->foodQty -= intake;
        this->excreteQty += digAmt;
        this->metabolism += metaInc;
    }
    void excrete(int excreteAmt)
    {
        this->foodQty -= excreteAmt;
    }
};
```



```cpp
class unicell_sexual : private unicell
{
private:
    bool f_status;
    unicell_sexual *parent;
    unicell_sexual *child;

public:
    bool getF_status()
    {
        return this->f_status;
    }
    unicell_sexual *getParent()
    {
        return this->parent;
    }
    unicell_sexual *getChild()
    {
        return this->child;
    }
    void unicell_sexual_init(int avg_lifespan, int age, int foodQty, int excreteQty, int metabolism,
bool f_status, unicell_sexual *parent, unicell_sexual *child)
    {
        this->unicell_init(avg_lifespan, age, foodQty, excreteQty, metabolism);
        this->f_status = f_status;
        this->parent = parent;
        this->child = child;
    }
    int conjugate(unicell_sexual *partner)
    {
        if (this->f_status == partner->f_status)
        {
            return NULL;
        }
        else if (this->f_status)
        {
            this->child = partner;
            return 1;
        }
        else
        {
            this->parent = partner;
            return 1;
        }
    }
};
```



```cpp
class unicell_asexual : private unicell
{
public:
    void unicell_asexual_init(int avg_lifespan, int age, int foodQty, int excreteQty, int metabolism)
    {
        this->unicell_init(avg_lifespan, age, foodQty, excreteQty, metabolism);
    }
    void fission(vector<unicell_asexual *> *children, vector<int> metabolism, int n_ary)
    {
        for (int i = 0; i < n_ary; i++)
        {
            unicell_asexual temp;
            temp.unicell_asexual_init(this->getAvg_lifespan(), 0, (this->getFoodqty()) / n_ary, (this-
>getExcreteqty()) / n_ary, metabolism[i]);
            children->push_back(&temp);
        }
    }
};
```

References:

- https://en.wikipedia.org/wiki/Bacterial_conjugation
- https://en.wikipedia.org/wiki/Fission_(biology)
- https://www.geeksforgeeks.org/inheritance-in-c/