SP Mini Project

Ivik Lau Dalgas Hostrup
ihostr16@student.aau.dk

June 4, 2023

Contents

0.1 src

0.1.1 src/ChemicalSystem.h

Listing 1: src/ChemicalSystem.h

```
//
// Created by Ivik Hostrup.
// Chemical system class that represents the stochastic simulation algorithm
// Requirement 4.
//

#ifndef STOCHASTICSIMULATION_CHEMICALSYSTEM_H
#define STOCHASTICSIMULATION_CHEMICALSYSTEM_H

#include <chrono>
#include <vector>
#include "Reaction.h"
#include "SymbolTable.h"
#include "Monitor.h"

class ChemicalSystem {
public:
    ChemicalSystem() : m_gen(m_rd()) {};

    template <typename CallBackType>
    void Simulate(size_t endTime, Monitor<CallBackType>& monitor) {
        double startTime = 0.0;

        while (startTime <= endTime){
            ComputeDelay();

            auto reaction_map = m_symbolTable_reactions.GetAllSymbols();
            auto reaction_with_min_delay = reaction_map.begin()->second;

            for (const auto& [_, reaction] : reaction_map) {
                if (reaction->GetDelay() < reaction_with_min_delay->GetDelay()) {
                    reaction_with_min_delay = reaction;
                }
            }

            startTime += reaction_with_min_delay->GetDelay();
            //std::cout << startTime << std::endl;
            auto combinedSpecies = reaction_with_min_delay->GetReactants().GetCombinedSpecies();

            bool reactantsSufficient = true;
            for (const auto& reactant : combinedSpecies) {
                if(reactant->GetQuantity() < 1) { // If amount agents is less than 1, then the
↪reaction cannot proceed
                    reactantsSufficient = false;
                    break;
                }
            }

            if(reactantsSufficient) {
                for (const auto& reactant : combinedSpecies) {
                    reactant->SetQuantity(reactant->GetQuantity() - 1);
                }

                for (const auto& product :
↪reaction_with_min_delay->GetProducts().GetCombinedSpecies()) {
                    product->SetQuantity(product->GetQuantity() + 1);
```

```
                }
            }

            monitor.OnStateChange(startTime, *this);
        }
    }

    void ComputeDelay();
    void Reset();

    std::shared_ptr<Species> AddSpecies(const std::string& name, const size_t& initial_amount);
    std::shared_ptr<Species> GetSpecies(const std::string& name) const;
    std::shared_ptr<Reaction> AddReaction(const Reaction& reaction, const double& rate_constant);
    [[nodiscard]] std::vector<std::shared_ptr<Reaction>> GetReactions() const;


    friend std::ostream& operator<<(std::ostream& os, const ChemicalSystem& system);
private:
    std::vector<std::shared_ptr<Species>> m_species;
    std::vector<std::shared_ptr<Reaction>> m_reactions;
    SymbolTable<Species> m_symbolTable_species;
    SymbolTable<Reaction> m_symbolTable_reactions;
    std::unordered_map<std::string, size_t> m_initial_quantities;
    std::random_device m_rd;
    std::mt19937 m_gen;
};



#endif //STOCHASTICSIMULATION_CHEMICALSYSTEM_H
```

0.1.2 src/CircadianSimulator.h

Listing 2: src/CircadianSimulator.h

```
//
// Created by Ivik Hostrup.
// Circadian simulator class that simulates the circadian rhythm of a cell
// Subtask A.1
//

#ifndef STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H
#define STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H


#include "ChemicalSystem.h"
#include "plot.hpp"


class CircadianSimulator {
public:
    CircadianSimulator(size_t endTime) : m_endTime(endTime) {}

    template<typename CallBackType>
    void RunSimulation(Monitor<CallBackType>& monitor) {
        ChemicalSystem system;

        auto alphaA = 50.0;
        auto alpha_A = 500.0;
        auto alphaR = 0.01;
        auto alpha_R = 50.0;
        auto betaA = 50.0;
        auto betaR = 5.0;
        auto gammaA = 1.0;
        auto gammaR = 1.0;
        auto gammaC = 2.0;
        auto deltaA = 1.0;
        auto deltaR = 0.2;
        auto deltaMA = 10.0;
        auto deltaMR = 0.5;
        auto thetaA = 50.0;
        auto thetaR = 100.0;

        auto DA = system.AddSpecies("DA", 1);
        auto D_A = system.AddSpecies("D_A", 0);
        auto DR = system.AddSpecies("DR", 1);
        auto D_R = system.AddSpecies("D_R", 0);
        auto MA = system.AddSpecies("MA", 0);
        auto MR = system.AddSpecies("MR", 0);
        auto A = system.AddSpecies("A", 0);
        auto R = system.AddSpecies("R", 0);
        auto C = system.AddSpecies("C", 0);
        auto env = system.AddSpecies("env", 0);

        system.AddReaction(A + DA >>= D_A, gammaA);
        system.AddReaction(D_A >>= DA + A, thetaA);
        system.AddReaction(A + DR >>= D_R, gammaR);
        system.AddReaction(D_R >>= DR + A, thetaR);
        system.AddReaction(D_A >>= MA + D_A, alpha_A);
        system.AddReaction(DA >>= MA + DA, alphaA);
        system.AddReaction(D_R >>= MR + D_R, alpha_R);
        system.AddReaction(DR >>= MR + DR, alphaR);
        system.AddReaction(MA >>= MA + A, betaA);
```

```
59          system.AddReaction(MR >>= MR + R, betaR);
60          system.AddReaction(A + R >>= C, gammaC);
61          system.AddReaction(C >>= R, deltaA);
62          system.AddReaction(A >>= env, deltaA);
63          system.AddReaction(R >>= env, deltaR);
64          system.AddReaction(MA >>= env, deltaMA);
65          system.AddReaction(MR >>= env, deltaMR);
66
67          system.Simulate(m_endTime, monitor); // Run simulation
68      }
69
70  private:
71      size_t m_endTime;
72      std::vector<std::vector<double>> m_signals;
73  };
74
75
76  #endif //STOCHASTICSIMULATION_CIRCADIANSIMULATOR_H
```

0.1.3 src/CombinedElements.h

Listing 3: src/CombinedElements.h

```
//
// Created by Ivik Hostrup.
// Combined elements are used to represent the reactants and products of a reaction.
// Baseline to satisfy all requirements.
//

#ifndef STOCHASTICSIMULATION_COMBINEDELEMENTS_H
#define STOCHASTICSIMULATION_COMBINEDELEMENTS_H



#include <vector>
#include <memory>
#include "Species.h"
#include <iostream>

class CombinedElements {
public:
    void Add(const std::shared_ptr<Species>& species);
    [[nodiscard]] std::vector<std::shared_ptr<Species>> GetCombinedSpecies() const;

    friend std::ostream& operator<<(std::ostream& os, const CombinedElements& combinedReactants);
private:
    std::vector<std::shared_ptr<Species>> m_combined_species;
};

CombinedElements operator+(const std::shared_ptr<Species>& leftElement, const
↪std::shared_ptr<Species>& rightElement);


#endif //STOCHASTICSIMULATION_COMBINEDELEMENTS_H
```

0.1.4 src/CovidSimulator.h

Listing 4: src/CovidSimulator.h

```
//
// Created by Ivik Hostrup.
// Covid simulator class that can be used to simulate the spread of covid-19
// Subtask A.2
//

#ifndef STOCHASTICSIMULATION_COVIDSIMULATOR_H
#define STOCHASTICSIMULATION_COVIDSIMULATOR_H


#include <cstddef>
#include "Monitor.h"
#include "ChemicalSystem.h"
#include "CsvWriter.h"
#include "plot.hpp"

class CovidSimulator {
public:
    CovidSimulator(size_t N, size_t endTime) : m_N(N), m_endTime(endTime) {}

    template<typename CallBackType>
    void RunCovidSimulator(Monitor<CallBackType>& monitor){
        ChemicalSystem system;

        const auto eps = 0.0009;
        const auto I0 = size_t(std::round(eps*m_N));
        const auto E0 = size_t(std::round(eps*m_N*15));
        const auto S0 = m_N-I0-E0;
        const auto R0 = 2.4;
        const auto alpha = 1.0 / 5.1;
        const auto gamma = 1.0 / 3.1;
        const auto beta = R0 * gamma;
        const auto P_H = 0.9e-3;
        const auto kappa = gamma * P_H*(1.0-P_H);
        const auto tau = 1.0/10.12;

        auto S = system.AddSpecies("S", S0);
        auto E = system.AddSpecies("E", E0);
        auto I = system.AddSpecies("I", I0);
        auto H = system.AddSpecies("H", 0);
        auto R = system.AddSpecies("R", 0);

        system.AddReaction(S+I >>= E+I, beta/m_N);
        system.AddReaction(E >>= I, alpha);
        system.AddReaction(I >>= R, gamma);
        system.AddReaction(I >>= H, kappa);
        system.AddReaction(H >>= R, tau);

        system.Simulate(m_endTime, monitor);
    };
private:
    size_t m_N;
    size_t m_endTime;
    std::vector<std::vector<double>> m_signals;
};


#endif //STOCHASTICSIMULATION_COVIDSIMULATOR_H
```

0.1.5 src/CsvWriter.h

Listing 5: src/CsvWriter.h

```
//
// Created by Ivik Hostrup.
// CSV writer class that can be used to write the results of a simulation to a csv file
// Early implementation to satisfy requirement 6.
//

#ifndef STOCHASTICSIMULATION_CSVWRITER_H
#define STOCHASTICSIMULATION_CSVWRITER_H


#include <string>
#include <vector>

class CsvWriter {
public:
    CsvWriter(const std::string& filename, const std::vector<std::string>& speciesNames)
        : m_filename(filename), m_species_names(speciesNames) {}

    void WriteToCsv(const std::vector<double>& timepoints, const
↪std::vector<std::vector<double>>& signals) const;
private:
    const std::string m_filename;
    const std::vector<std::string> m_species_names;
};


#endif //STOCHASTICSIMULATION_CSVWRITER_H
```

0.1.6 src/Monitor.h

Listing 6: src/Monitor.h

```
//
// Created by Ivik Hostrup.
// Generic monitor class that can be used to monitor the state of a chemical system
// Part of requirement 7.
//

#ifndef STOCHASTICSIMULATION_MONITOR_H
#define STOCHASTICSIMULATION_MONITOR_H


#include <utility>

//forward declaration
class ChemicalSystem;

template<typename CallBackType>
class Monitor {
public:
    explicit Monitor(CallBackType& callback) : m_callback(callback) {}

    void OnStateChange(double time, const ChemicalSystem& chemicalSystem) {
        m_callback(time, chemicalSystem);
    }

    CallBackType& GetCallback() {
        return m_callback;
    }
private:
    CallBackType& m_callback;
};


#endif //STOCHASTICSIMULATION_MONITOR_H
```

0.1.7 src/MonitorCallBack.h

Listing 7: src/MonitorCallBack.h

```
//
// Created by Ivik Hostrup.
// Monitor callback interface
// Part of requirement 7
//

#include "ChemicalSystem.h"

#ifndef STOCHASTICSIMULATION_MONITORCALLBACK_H
#define STOCHASTICSIMULATION_MONITORCALLBACK_H

#endif //STOCHASTICSIMULATION_MONITORCALLBACK_H

struct MonitorCallBack {
    virtual void operator()(double time, const ChemicalSystem& chemicalSystem) = 0;
};
```

0.1.8 src/plot.hpp

Listing 8: src/plot.hpp

```
// Plotting class to create graphs of the simulation results
// Taken from class lectures
// Requirement 6.

#ifndef PLOT_HPP
#define PLOT_HPP

#include <memory>
#include <vector>
#include <unordered_map>
#include <string>

class Plot
{
    std::string title;
    struct app_t;
    struct chart_t;
    std::unique_ptr<app_t> app;      // pimpl of the application
    std::unique_ptr<chart_t> chart;  // pimpl of the chart
    std::string x_axis_label;
    std::string y_axis_label;

public:
    Plot(std::string title, std::string x_axis_label, std::string y_axis_label, int width, int height);
    Plot(const Plot&) = delete;
    Plot& operator=(const Plot&) = delete;
    Plot(Plot&&) noexcept = default;
    Plot& operator=(Plot&&) = default;
    ~Plot() noexcept;
    void save_to_png(const std::string &filename);
    void scatter(const std::string& name, const std::vector<double>& x, const std::vector<double>& y);
    void lines(const std::string& name, const std::vector<double>& x, const std::vector<double>& y);
    void plot_data(const std::vector<double>& time, const std::unordered_map<std::string, std::vector<double>>& species_quantities);

    void process();
};

#endif //PLOT_HPP
```

0.1.9 src/Reaction.h

Listing 9: src/Reaction.h

```
//
// Created by Ivik Hostrup.
// Reaction class is used to represent a chemical reaction.
// Baseline to satisfy all requirements.
//

#ifndef STOCHASTICSIMULATION_REACTION_H
#define STOCHASTICSIMULATION_REACTION_H



#include <memory>
#include <vector>
#include "Species.h"
#include "CombinedElements.h"
#include <iostream>
#include <random>

class Reaction {
public:
    void AddReactant(const std::shared_ptr<Species>& species);
    void AddProduct(const std::shared_ptr<Species>& species);
    void SetRateConstant(const double& rate_constant);
    [[nodiscard]] double GetDelay() const;
    void ComputeDelay(std::mt19937& gen);
    [[nodiscard]] const CombinedElements& GetReactants() const;
    [[nodiscard]] const CombinedElements& GetProducts() const;
    std::string to_string() const;

    friend std::ostream& operator<<(std::ostream& os, const Reaction& reaction);
private:
    CombinedElements m_reactants;
    CombinedElements m_products;
    double m_lambda;
    double m_delay = std::numeric_limits<double>::max();

    void PrintReaction(std::ostream& os) const;
};

Reaction operator>>=(const CombinedElements& combinedReactants, const std::shared_ptr<Species>&
↪product);
Reaction operator>>=(const std::shared_ptr<Species>& reactant, const CombinedElements&
↪combinedProducts);
Reaction operator>>=(const CombinedElements& combinedReactants, const CombinedElements&
↪combinedProducts);
Reaction operator>>=(const std::shared_ptr<Species>& reactant, const std::shared_ptr<Species>&
↪product);

#endif //STOCHASTICSIMULATION_REACTION_H
```

0.1.10 src/SimpleSimulator.h

Listing 10: src/SimpleSimulator.h

```cpp
//
// Created by Ivik Hostrup.
// Simple simulation class that can be used to run a simulation with a simple chemical system.
// Requirement 4.
//

#ifndef STOCHASTICSIMULATION_SIMPLESIMULATOR_H
#define STOCHASTICSIMULATION_SIMPLESIMULATOR_H

#include <cstddef>
#include "Monitor.h"
#include "ChemicalSystem.h"
#include "CsvWriter.h"
#include "plot.hpp"

class SimpleSimulator {
public:
    SimpleSimulator(size_t endTime) : m_endTime(endTime) {}

    template<typename CallBackType>
    void RunFirstSimulation(Monitor<CallBackType>& monitor) {
        ChemicalSystem system;

        auto A = system.AddSpecies("A", 100);
        auto B = system.AddSpecies("B", 0);
        auto C = system.AddSpecies("C", 1);

        system.AddReaction(A + C >>= B + C, 0.001);

        std::vector<std::string> speciesToMonitor = monitor.GetCallback().GetMonitoredSpecies();
        system.Simulate(m_endTime, monitor);
    }

    template<typename CallBackType>
    void RunSecondSimulation(Monitor<CallBackType>& monitor) {
        ChemicalSystem system;

        auto A = system.AddSpecies("A", 100);
        auto B = system.AddSpecies("B", 0);
        auto C = system.AddSpecies("C", 2);

        system.AddReaction(A + C >>= B + C, 0.001);

        system.Simulate(m_endTime, monitor, false);
    }

    template<typename CallBackType>
    void RunThirdSimulation(Monitor<CallBackType>& monitor) {
        ChemicalSystem system;

        auto A = system.AddSpecies("A", 50);
        auto B = system.AddSpecies("B", 50);
        auto C = system.AddSpecies("C", 1);

        system.AddReaction(A + C >>= B + C, 0.001);

        system.Simulate(m_endTime, monitor, false);
    }
```

```
59
60  private:
61      size_t m_endTime;
62      std::vector<std::vector<double>> m_signals;
63  };
64
65
66  #endif //STOCHASTICSIMULATION_SIMPLESIMULATOR_H
```

0.1.11 src/SimulationMethods.h

Listing 11: src/SimulationMethods.h

```
//
// Created by Ivik Hostrup.
// Container for all simulation methods
// Part of all requirements
//

#ifndef STOCHASTICSIMULATION_SIMULATIONMETHODS_H
#define STOCHASTICSIMULATION_SIMULATIONMETHODS_H

#include <string>
#include <vector>
#include <thread>
#include "SpeciesQuantityMonitorCallBack.h"
#include "CircadianSimulator.h"
#include "SimpleSimulator.h"
#include "CovidSimulator.h"
#include "plot.hpp"

void PlotSimple();
void PlotCircadian();
void PlotCovid(double N = 10000);
void MultithreadedCovid(size_t numSimulations = 20, size_t numThreads =
↪std::thread::hardware_concurrency());

#endif //STOCHASTICSIMULATION_SIMULATIONMETHODS_H
```

0.1.12 src/Species.h

Listing 12: src/Species.h

```
//
// Created by Ivik Hostrup.
// Species class for representing a species in a chemical reaction
// Baseline to satisfy all requirements.
//

#ifndef STOCHASTICSIMULATION_SPECIES_H
#define STOCHASTICSIMULATION_SPECIES_H


#include <string>
#include <iostream>

class Species {
public:
    Species(std::string name, int initialQuantity):
            m_name(std::move(name)), m_quantity(initialQuantity) {}

    const std::string& GetName() const;
    const size_t& GetQuantity() const;
    void SetQuantity(const size_t& quantity);

    friend std::ostream& operator<<(std::ostream& os, const Species& species);
private:
    std::string m_name;
    size_t m_quantity;
};


#endif //STOCHASTICSIMULATION_SPECIES_H
```

0.1.13 src/SpeciesQuantityMonitorCallBack.h

Listing 13: src/SpeciesQuantityMonitorCallBack.h

```
//
// Created by Ivik Hostrup.
// Specific monitor class that can be used to monitor the quantity of a species in a chemical system
// Part of requirement 7.
//

#ifndef STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H
#define STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H


#include <vector>
#include <mutex>
#include "Monitor.h"
#include "Species.h"
#include "ChemicalSystem.h"
#include "MonitorCallBack.h"

class SpeciesQuantityMonitorCallBack : public MonitorCallBack {
public:
    explicit SpeciesQuantityMonitorCallBack(const std::vector<std::string>& speciesName)
        : m_species_names(speciesName), m_signals_monitor(speciesName.size()), m_timepoints() {}

        void operator()(double time, const ChemicalSystem& chemicalSystem) override;

        const std::vector<std::string>& GetMonitoredSpecies() const;
        void CreatePlot(const std::string& plotName = "Covid Simulation",
                        const std::string& xAxisLabel = "Time",
                        const std::string& yAxisLabel = "Quantity",
                        int width = 800, int height = 600) const;
        std::pair<double, double> GetPeakAndMean(const std::string& speciesName) const;
private:
    std::vector<std::string> m_species_names;
    std::vector<std::vector<double>> m_signals_monitor;
    std::vector<double> m_timepoints;
    std::mutex m_mutex;
};


#endif //STOCHASTICSIMULATION_SPECIESQUANTITYMONITORCALLBACK_H
```

0.1.14 src/SymbolTable.h

Listing 14: src/SymbolTable.h

```cpp
//
// Created by Ivik Hostrup.
// Generic symbol table class that can be used to store any type of object
// Requirement 3.
//

#ifndef STOCHASTICSIMULATION_SYMBOLTABLE_H
#define STOCHASTICSIMULATION_SYMBOLTABLE_H

#include <string>
#include <memory>
#include <unordered_map>
#include <stdexcept>

template <typename T>
class SymbolTable {
public:
    void AddSymbol(const std::string& name, const std::shared_ptr<T>& object);
    std::shared_ptr<T> GetSymbol(const std::string& name) const;
    const std::unordered_map<std::string, std::shared_ptr<T>>& GetAllSymbols() const;

private:
    std::unordered_map<std::string, std::shared_ptr<T>> m_symbol_table;
};

template<typename T>
std::shared_ptr<T> SymbolTable<T>::GetSymbol(const std::string &name) const {
    auto item = m_symbol_table.find(name);
    if (item != m_symbol_table.end()) {
        return item->second;
    } else {
        throw std::runtime_error("Symbol not found");
    }
}

template<typename T>
const std::unordered_map<std::string, std::shared_ptr<T>>& SymbolTable<T>::GetAllSymbols() const {
    return m_symbol_table;
}

template<typename T>
void SymbolTable<T>::AddSymbol(const std::string &name, const std::shared_ptr<T> &object) {
    // Add check to see if symbol already exists
    if(m_symbol_table.find(name) != m_symbol_table.end())
        throw std::runtime_error("Symbol already exists");


    m_symbol_table[name] = object;
}


#endif //STOCHASTICSIMULATION_SYMBOLTABLE_H
```

0.2 benchmarks

0.2.1 benchmarks/benchmark_multithreaded_covid.cpp

Listing 15: benchmarks/benchmark_multithreaded_covid.cpp

```cpp
//
// Created by Ivik Hostrup on 6/3/2023.
//
#include <benchmark/benchmark.h>
#include "../src/SimulationMethods.h"

static void BM_multithreadedCovid(benchmark::State& state) {
    for (auto _ : state) {
        MultithreadedCovid(state.range(0));
    }
}

BENCHMARK(BM_multithreadedCovid)->Arg(std::thread::hardware_concurrency());
BENCHMARK(BM_multithreadedCovid)->Arg(1);
BENCHMARK(BM_multithreadedCovid)->Arg(5);
BENCHMARK(BM_multithreadedCovid)->Arg(10);
BENCHMARK(BM_multithreadedCovid)->Arg(20);
```

0.3 src

0.3.1 src/ChemicalSystem.cpp

Listing 16: src/ChemicalSystem.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include <iostream>
#include <algorithm>
#include <chrono>
#include "ChemicalSystem.h"
#include "Monitor.h"


std::shared_ptr<Species> ChemicalSystem::AddSpecies(const std::string& name, const size_t&
↪initial_amount) {
    auto new_species = std::make_shared<Species>(name, initial_amount);
    m_symbolTable_species.AddSymbol(name, new_species);

    m_initial_quantities[name] = initial_amount;
    return new_species;
}

std::shared_ptr<Species> ChemicalSystem::GetSpecies(const std::string& name) const {
    return m_symbolTable_species.GetSymbol(name);
}


std::shared_ptr<Reaction> ChemicalSystem::AddReaction(const Reaction &reaction, const double&
↪rate_constant) {
    auto new_reaction = std::make_shared<Reaction>(reaction);
    new_reaction->SetRateConstant(rate_constant);
    auto reaction_name = new_reaction->to_string();
    m_symbolTable_reactions.AddSymbol(reaction_name, new_reaction);

    return new_reaction;
}

std::vector<std::shared_ptr<Reaction>> ChemicalSystem::GetReactions() const {
    return m_reactions;
}

// Overload << for reactions in system
std::ostream& operator<<(std::ostream& os, const ChemicalSystem& system) {
    for (const auto& reaction : system.GetReactions()) {
        os << *reaction << std::endl;
    }
    return os;
}

void ChemicalSystem::ComputeDelay() {
    auto reaction_map = m_symbolTable_reactions.GetAllSymbols();

    for(const auto& [name, reaction] : reaction_map){
        reaction->ComputeDelay(m_gen);
    }
}

void ChemicalSystem::Reset() {
```

```
55      for (const auto& [name, species] : m_symbolTable_species.GetAllSymbols()) {
56          species->SetQuantity(m_initial_quantities[name]);
57      }
58  }
```

0.3.2 src/CombinedElements.cpp

Listing 17: src/CombinedElements.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include "CombinedElements.h"
#include <iostream>

void CombinedElements::Add(const std::shared_ptr<Species>& species) {
    if(species){
        m_combined_species.push_back(species);
    } else {
        std::cout << "Species is null" << std::endl;
    }
}

std::vector<std::shared_ptr<Species>> CombinedElements::GetCombinedSpecies() const{
    return m_combined_species;
}

std::ostream& operator<<(std::ostream& os, const CombinedElements& combinedReactants) {
    // Print the Species objects in the CombinedElements object
    for (const auto& species : combinedReactants.GetCombinedSpecies()) {
        os << *species;
    }
    return os;
}

CombinedElements operator+(const std::shared_ptr<Species>& leftElement, const
↪std::shared_ptr<Species>& rightElement) {
    CombinedElements combination;
    combination.Add(leftElement);
    combination.Add(rightElement);

    return std::move(combination);
}
```

0.3.3 src/CsvWriter.cpp

Listing 18: src/CsvWriter.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include <fstream>
#include <iostream>
#include "CsvWriter.h"

void CsvWriter::WriteToCsv(const std::vector<double>& timepoints, const std::vector<std::vector<double>>& signals) const {
    std::ofstream file;

    file.open(m_filename);
    if (file.fail()) {
        std::cerr << "Failed to open file: " << m_filename << "\n";
        return;
    }
    // Write species names as header
    file << "Time,";
    for (const auto& name : m_species_names) {
        file << name;
        if (&name != &m_species_names.back()) { // Check if not the last element to avoid trailing comma
            file << ',';
        }
    }
    file << '\n';

    // Write signals
    for (size_t t = 0; t < signals[0].size(); ++t) {
        file << timepoints[t] << ',';
        for (size_t j = 0; j < signals.size(); ++j) {
            file << signals[j][t];
            if (j < signals.size() - 1) { // Check if not the last element to avoid trailing comma
                file << ',';
            }
        }
        file << '\n';
    }

    if (file.bad()) {
        std::cerr << "Failed to write to file: " << m_filename << "\n";
    }
    file.close();
}
```

0.3.4 src/main.cpp

Listing 19: src/main.cpp

```
1
2
3
4  #include "SimulationMethods.h"
5  #include <benchmark/benchmark.h>
6
7  int main() {
8
9      //PlotSimple();
10     //PlotCircadian();
11     //PlotCovid();
12     //MultithreadedCovid();
13
14     return 0;
15 }
```

0.3.5 src/plot.cpp

Listing 20: src/plot.cpp

```
//
// Taken from class lectures
//

#include "plot.hpp"

#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>

#include <QtCharts/QChartView>
#include <QtCharts/QScatterSeries>
#include <QtCharts/QLineSeries>
#include <QtCharts/QBoxPlotSeries>
#include <QtCharts/QLegendMarker>
#include <QtGui/QImage>
#include <QtGui/QPainter>
#include <QtCore/QtMath>

#include <random>
#include <algorithm>

QT_CHARTS_USE_NAMESPACE

struct Plot::app_t
{
    int argc{0};
    char** args{nullptr};
    QApplication app{argc, args};
    QMainWindow window{nullptr};
};

class Plot::chart_t : public QChartView
{
public:
    chart_t(): QChartView{new QChart{}, nullptr} {}
    ~chart_t() noexcept {}
};

Plot::Plot(std::string title, std::string x_axis_label, std::string y_axis_label, int width, int height):
        title{std::move(title)}, app{std::make_unique<app_t>()}, chart{std::make_unique<chart_t>()}, x_axis_label{std::move(x_axis_label)}, y_axis_label{std::move(y_axis_label)}
{
    app->window.setCentralWidget(chart.get());
    app->window.setWindowTitle(this->title.c_str());
    app->window.resize(width, height);
    chart->setRenderHint(QPainter::Antialiasing);
}

Plot::~Plot() noexcept = default;

void Plot::save_to_png(const std::string& filename)
{
    auto* ch = chart->chart();
    QPixmap pixmap(chart->size());
    pixmap.fill(Qt::white);
    QPainter painter(&pixmap);
```

```
56      chart->render(&painter);
57      pixmap.save(QString::fromStdString(filename), "PNG");
58  }
59
60  template <typename Series, typename Chart>
61  void add_xy(Chart& chart, const std::string& name, const std::vector<double>& x, const
    ↪std::vector<double>& y)
62  {
63      assert(x.size() <= y.size());
64      auto* series = new Series();
65      series->setName(name.c_str());
66      for (auto i = 0; i < x.size(); ++i)
67          series->append(x[i], y[i]);
68      chart.addSeries(series);
69  }
70
71  void Plot::scatter(const std::string& name, const std::vector<double>& x, const
    ↪std::vector<double>& y)
72  {
73      add_xy<QScatterSeries>(*chart->chart(), name, x, y);
74  }
75
76  void Plot::lines(const std::string& name, const std::vector<double>& x, const
    ↪std::vector<double>& y)
77  {
78      add_xy<QLineSeries>(*chart->chart(), name, x, y);
79  }
80
81  void Plot::plot_data(const std::vector<double>& time, const std::unordered_map<std::string,
    ↪std::vector<double>>& species_quantities)
82  {
83      for (const auto& [species, quantities] : species_quantities)
84      {
85          lines(species, time, quantities);
86      }
87  }
88
89  void Plot::process()
90  {
91      auto* ch = chart->chart();
92      ch->setTitle(title.c_str());
93      ch->createDefaultAxes();
94      ch->setDropShadowEnabled(false);
95
96      if (!ch->axes(Qt::Vertical).isEmpty())
97          ch->axes(Qt::Vertical).first()->setTitleText(y_axis_label.c_str());
98      if (!ch->axes(Qt::Horizontal).isEmpty())
99          ch->axes(Qt::Horizontal).first()->setTitleText(x_axis_label.c_str());
100
101     ch->legend()->setMarkerShape(QLegend::MarkerShapeFromSeries);
102     ch->legend()->setAlignment(Qt::AlignBottom);
103
104     app->window.show();
105     app->app.exec();
106 }
```

0.3.6 src/Reaction.cpp

Listing 21: src/Reaction.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include "Reaction.h"
#include <iostream>
#include <sstream>
#include <random>

void Reaction::AddReactant(const std::shared_ptr<Species>& species){
    m_reactants.Add(species);
}

void Reaction::AddProduct(const std::shared_ptr<Species>& species) {
    m_products.Add(species);
}

[[nodiscard]] const CombinedElements& Reaction::GetReactants() const {
    return m_reactants;
}

[[nodiscard]] const CombinedElements& Reaction::GetProducts() const {
    return m_products;
}


void Reaction::SetRateConstant(const double& rate_constant){
    m_lambda = rate_constant;
}

std::ostream& operator<<(std::ostream& os, const Reaction& reaction) {
    reaction.PrintReaction(os);
    return os;
}

std::string Reaction::to_string() const {
    std::ostringstream os;
    PrintReaction(os);
    return os.str();
}

void Reaction::PrintReaction(std::ostream &os) const {
    size_t count = 0;
    for (const auto& reactant : this->GetReactants().GetCombinedSpecies()) {
        os << *reactant;
        if(++count < this->GetReactants().GetCombinedSpecies().size()){
            os << " + ";
        }
    }
    os << " -> ";

    count = 0;
    for (const auto& product : this->GetProducts().GetCombinedSpecies()) {
        os << *product;
        if(++count < this->GetProducts().GetCombinedSpecies().size()){
            os << " + ";
        }
    }
```

```
}

double Reaction::GetDelay() const {
    return m_delay;
}

void Reaction::ComputeDelay(std::mt19937& gen) {
    auto lambda = m_lambda;

    for(const auto& reactant : m_reactants.GetCombinedSpecies()){
        lambda *= static_cast<double>(reactant->GetQuantity());
    }

    std::exponential_distribution distribution(lambda);

    m_delay = distribution(gen);
}

// Multiple reactants and single products
Reaction operator>>=(const CombinedElements& combinedReactants, const std::shared_ptr<Species>&
↪product){
    Reaction reaction;

    for(auto const& reactant : combinedReactants.GetCombinedSpecies()){
        reaction.AddReactant(reactant);
    }

    if(product->GetName() != "env"){
        reaction.AddProduct(product);
    }

    return std::move(reaction);
}

// single reactactant and multiple products
Reaction operator>>=(const std::shared_ptr<Species>& reactant, const CombinedElements&
↪combinedProducts){
    Reaction reaction;

    reaction.AddReactant(reactant);

    for(auto const& product : combinedProducts.GetCombinedSpecies()){
        if(product->GetName() != "env"){
            reaction.AddProduct(product);
        }
    }

    return std::move(reaction);
}

// Multiple reactants and multiple products
Reaction operator>>=(const CombinedElements& combinedReactants, const CombinedElements&
↪combinedProducts){
    Reaction reaction;

    for(auto const& reactant : combinedReactants.GetCombinedSpecies()){
        reaction.AddReactant(reactant);
    }

    for(auto const& product : combinedProducts.GetCombinedSpecies()){
        if(product->GetName() != "env"){
```

```
            reaction.AddProduct(product);
        }
    }

    return std::move(reaction);
}

// single reactactant and a single product
Reaction operator>>=(const std::shared_ptr<Species>& reactant, const std::shared_ptr<Species>&
↪product){
    Reaction reaction;

    reaction.AddReactant(reactant);
    if(product->GetName() != "env"){
        reaction.AddProduct(product);
    }

    return std::move(reaction);
}
```

0.3.7 src/SimulationMethods.cpp

Listing 22: src/SimulationMethods.cpp

```cpp
//
// Created by Ivik Hostrup.
//


#include <future>
#include "CovidSimulator.h"
#include "SpeciesQuantityMonitorCallBack.h"
#include "CircadianSimulator.h"
#include "SimpleSimulator.h"

void PlotSimple(){
    std::vector<std::string> speciesToMonitor = {"A", "B", "C"};
    SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
    Monitor monitor(speciesMonitorCallBack);

    SimpleSimulator simulator(1000);
    simulator.RunFirstSimulation(monitor);
    monitor.GetCallback().CreatePlot("Simple Simulation");
};

void PlotCircadian(){
    std::vector<std::string> speciesToMonitor = {"A", "R", "C"};
    SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
    Monitor monitor(speciesMonitorCallBack);

    CircadianSimulator simulator(100);
    simulator.RunSimulation(monitor);
    monitor.GetCallback().CreatePlot("Circadian Simulation");
};

void PlotCovid(double N = 10000, const std::string& title = "Covid Simulation"){
    std::vector<std::string> speciesToMonitor = {"S", "E", "I","H","R"};
    SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
    Monitor monitor(speciesMonitorCallBack);

    CovidSimulator simulator(N, 100);
    simulator.RunCovidSimulator(monitor);

    auto [peak, mean] = speciesMonitorCallBack.GetPeakAndMean("H");

    std::cout << "Peak hospitalized: " << peak << std::endl;
    std::cout << "Mean hospitalized: " << mean << std::endl;

    monitor.GetCallback().CreatePlot(title);
};

void MultithreadedCovid(size_t numSimulations = 20, size_t numThreads =
↪std::thread::hardware_concurrency()){
    std::vector<std::string> speciesToMonitor = {"S", "E", "I","H","R"};

    std::vector<std::future<std::pair<double, double>>> futures(numSimulations);

    for(size_t i = 0; i < numSimulations; ++i) {
        futures[i] = std::async(std::launch::async, [&speciesToMonitor] {
            // Create a new callback and monitor for each thread
            SpeciesQuantityMonitorCallBack speciesMonitorCallBack(speciesToMonitor);
            Monitor monitor(speciesMonitorCallBack);
```

```
58
59              // Create a new simulator for each thread
60              CovidSimulator simulator(10000, 100);
61              simulator.RunCovidSimulator(monitor);
62
63              // Compute and return the peak hospitalized number
64              return speciesMonitorCallBack.GetPeakAndMean("H");
65          });
66      }
67
68      std::vector<double> peakValues(numSimulations);
69      for (size_t i = 0; i < numSimulations; ++i) {
70          auto [peaks, _] = futures[i].get();
71          peakValues[i] = peaks;
72      }
73
74      double meanPeakAcrossSimulations = std::accumulate(peakValues.begin(), peakValues.end(),
    ↪0.0) / numSimulations;
75
76      std::cout << "Mean peak hospitalized number: " << meanPeakAcrossSimulations << std::endl;
77  }
```

0.3.8 src/Species.cpp

Listing 23: src/Species.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include <stdexcept>
#include "Species.h"
#include <iostream>

const std::string& Species::GetName() const {
    return m_name;
}

std::ostream& operator<<(std::ostream& os, const Species& species) {
    os << species.GetName();
    return os;
}

const size_t& Species::GetQuantity() const {
    return m_quantity;
}

void Species::SetQuantity(const size_t &quantity) {
    m_quantity = quantity;
}
```

0.3.9 src/SpeciesQuantityMonitorCallBack.cpp

Listing 24: src/SpeciesQuantityMonitorCallBack.cpp

```cpp
//
// Created by Ivik Hostrup.
//

#include <algorithm>
#include "SpeciesQuantityMonitorCallBack.h"
#include "plot.hpp"


void SpeciesQuantityMonitorCallBack::operator()(double time, const ChemicalSystem &chemicalSystem) {
    std::scoped_lock lock(m_mutex);

    m_timepoints.push_back(time);
    for(size_t i = 0; i < m_species_names.size(); ++i) {
        m_signals_monitor[i].push_back(chemicalSystem.GetSpecies(m_species_names[i])->GetQuantity());
    }

    // Mutex unlocks when going out of scope
}

const std::vector<std::string>& SpeciesQuantityMonitorCallBack::GetMonitoredSpecies() const {
    return m_species_names;
}

void SpeciesQuantityMonitorCallBack::CreatePlot(const std::string &plotName,
                                                const std::string &xAxisLabel,
                                                const std::string &yAxisLabel,
                                                int width, int height) const {

    // Create map for species and their quantities
    std::unordered_map<std::string, std::vector<double>> speciesQuantities;

    for(size_t i = 0; i < m_species_names.size(); ++i) {
        std::vector<double> signals = m_signals_monitor[i];
        if(m_species_names[i] == "H") {
            for(size_t j = 0; j < signals.size(); ++j) {
                signals[j] *= 1000;
            }
        }
        speciesQuantities[m_species_names[i]] = signals;
    }


    Plot plot(plotName, xAxisLabel, yAxisLabel, width, height);

    plot.plot_data(m_timepoints, speciesQuantities);

    plot.process();

    plot.save_to_png(plotName + ".png");
}

std::pair<double, double> SpeciesQuantityMonitorCallBack::GetPeakAndMean(const std::string& speciesName) const {

    auto it = std::find(m_species_names.begin(), m_species_names.end(), speciesName);

```

```
    if(it != m_species_names.end()) {
        size_t index = std::distance(m_species_names.begin(), it);

        auto peak = *std::max_element(m_signals_monitor[index].begin(),
↪m_signals_monitor[index].end());
        auto sum = std::accumulate(m_signals_monitor[index].begin(),
↪m_signals_monitor[index].end(), 0.0);
        auto mean = sum / m_signals_monitor[index].size();

        return std::make_pair(peak, mean);
    } else {
        throw std::invalid_argument("Species name not found");
    }
}
```

0.4 tests

0.4.1 tests/doctest.cpp

Listing 25: tests/doctest.cpp

```cpp
//
// Created by Ivik Hostrup on 6/3/2023.
//

#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include <doctest.h>
```

0.4.2 tests/test_reaction.cpp

Listing 26: tests/test_reaction.cpp

```cpp
//
// Created by Ivik Hostrup on 6/2/2023.
//


#include "doctest.h"
#include "../src/ChemicalSystem.h"

TEST_CASE("Testing Reaction class") {
    ChemicalSystem system;
    auto S = system.AddSpecies("S", 10);
    auto E = system.AddSpecies("E", 20);
    auto I = system.AddSpecies("I", 30);

    auto constructedReaction = system.AddReaction(S+E >>= I, 0.5);

    SUBCASE("Testing AddReactant and GetReactants") {
        Reaction reaction;
        reaction.AddReactant(S);
        CHECK(reaction.GetReactants().GetCombinedSpecies().size() == 1);
    }

    SUBCASE("Testing AddProduct and GetProducts") {
        Reaction reaction;
        reaction.AddProduct(E);
        CHECK(reaction.GetProducts().GetCombinedSpecies().size() == 1);
    }

    SUBCASE("Testing reaction pretty print") {
        CHECK(constructedReaction->to_string() == "S + E -> I");
    }
}
```

0.4.3 tests/test_symbolTable.cpp

Listing 27: tests/test_symbolTable.cpp

```cpp
//
// Created by Ivik Hostrup on 6/2/2023.
//

#include "doctest.h"
#include "../src/ChemicalSystem.h"

TEST_CASE("Testing SymbolTable class") {
    ChemicalSystem system;
    auto S = system.AddSpecies("S", 10);

    SUBCASE("Testing AddSymbol and GetSymbol") {
        SymbolTable<Species> table;
        table.AddSymbol("S", S);
        CHECK(table.GetSymbol("S")->GetQuantity() == 10);
    }

    SUBCASE("Testing AddSymbol with existing symbol") {
        SymbolTable<Species> table;
        table.AddSymbol("S", S);

        CHECK_THROWS_AS(table.AddSymbol("S", S), std::runtime_error);
    }

    SUBCASE("Testing GetAllSymbols") {
        SymbolTable<Species> table;
        table.AddSymbol("S", S);
        CHECK(table.GetAllSymbols().size() == 1);
    }
}
```

0.5 benchmarks

0.5.1 benchmarks/CMakeLists.txt

Listing 28: benchmarks/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.24)

# Google Benchmark requires at least C++11
set(CMAKE_CXX_STANDARD 11)

# List of files
set(BENCHMARK_FILES
        benchmark_multithreaded_covid.cpp
        )

add_executable(Benchmarks ${BENCHMARK_FILES})

target_link_libraries(Benchmarks benchmark::benchmark_main stochasticsimulation_lib)
```

0.6 src

0.6.1 CMakeLists.txt

Listing 29: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.24)
project(StochasticSimulation)

set(CMAKE_CXX_STANDARD 17)

# Qt5
find_package(Qt5 COMPONENTS Widgets Charts QUIET)
if (Qt5_FOUND)
    message(STATUS "Qt5 found, fantastic!")
    set(CMAKE_AUTOMOC ON)
    set(CMAKE_AUTORCC ON)
    set(CMAKE_AUTOUIC ON)
else(Qt5_FOUND)
    message(WARNING "Qt5 NOT found, test_qt5 will be disabled. Please install qt5charts development package.")
endif(Qt5_FOUND)

include_directories(include)

# Create a library from source files
add_library(stochasticsimulation_lib src/Species.cpp src/Reaction.cpp src/ChemicalSystem.cpp src/CombinedElements.cpp src/SpeciesQuantityMonitorCallBack.cpp src/CsvWriter.cpp src/plot.cpp src/SimulationMethods.cpp src/SimulationMethods.h)

# Link the library to the required Qt5 components
if(Qt5_FOUND)
    target_link_libraries(stochasticsimulation_lib Qt5::Widgets Qt5::Charts)
endif(Qt5_FOUND)

add_executable(StochasticSimulation src/main.cpp benchmarks/benchmark_multithreaded_covid.cpp)

target_link_libraries(StochasticSimulation stochasticsimulation_lib benchmark::benchmark)

set(BENCHMARK_ENABLE_GTEST_TESTS ON CACHE BOOL "Enable benchmark's tests" FORCE)
set(BENCHMARK_DOWNLOAD_DEPENDENCIES ON CACHE BOOL "Let benchmark download its dependencies" FORCE)
add_subdirectory(benchmark EXCLUDE_FROM_ALL)

add_subdirectory(tests)
add_subdirectory(benchmarks) # different from benchmark
```

0.7 tests

0.7.1 tests/CMakeLists.txt

Listing 30: tests/CMakeLists.txt

```
add_library(doctest OBJECT doctest.cpp)

add_executable(tests test_reaction.cpp test_symbolTable.cpp $<TARGET_OBJECTS:doctest>)
target_link_libraries(tests stochasticsimulation_lib benchmark::benchmark)

add_test(NAME tests COMMAND tests)
```