Лабораторная работа №7

Дискретно-событийное моделирование

Андрей Геннадьевич Софич

Содержание

1 Цель работы **4**

2 Задание **5**

3 Выполнение лабораторной работы **6**

4 Выводы **46**

Список иллюстраций

3.1 Инициализация проекта 6
3.2 Подключение необходимых пакетов 6
3.3 Создание файла для реализации M/M/c 7
3.4 Код в стиле jupyter notebook 7
3.5 Файл с графиками 8
3.6 Диграмма обслуживания 8
3.7 Количество клиентов в системе 9
3.8 Время ожидания по клиентам 9
3.9 Код в стиле jupyter notebook 10
3.10 Первый прогон модели Росса 10
3.11 Код в стиле jupyter notebook 11
3.12 Результат анимации 11
3.13 Код в стиле jupyter notebook 12
3.14 Код в стиле jupyter notebook 12
3.15 Код в стиле jupyter notebook 13
3.16 Загрузка ремонтника 13
3.17 Код в стиле jupyter notebook 13
3.18 Добавление графика 14
3.19 График 14
3.20 Код в стиле jupyter notebook 15

1 Цель работы

Изучить основные модели теории массового обслуживания

2 Задание

Применить модели на задачах, выполнить несколько различных симуляций.

3 Выполнение лабораторной работы

Создаем рабочий каталог и инициализируем проект в julia (рис. 3.1).

Рисунок 3.1: Инициализация проекта

Создаем файл, где прописываем пакеты, необходимые для выполнения работы и запускаем его (рис. 3.2).

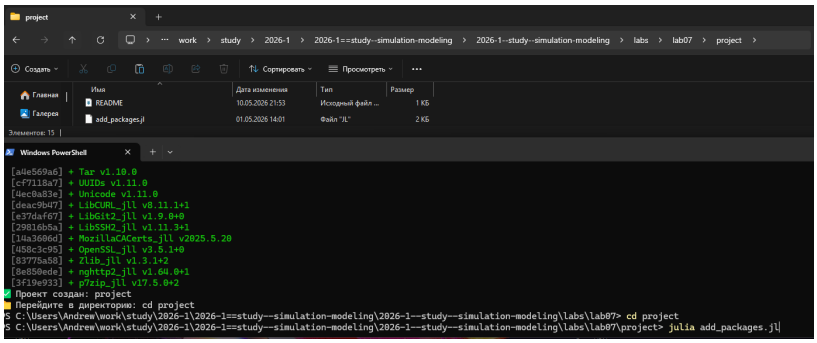

Рисунок 3.2: Подключение необходимых пакетов

Создаем файл в папке scripts, реализуем в нем модель M/M/c, которая будет симулировать и анализировать входящие потоки, обслуживания их интенсивности

и занятость приборов(каналов) на 10 клиентах(customers)и 2 серверах(каналах) (рис. 3.3).

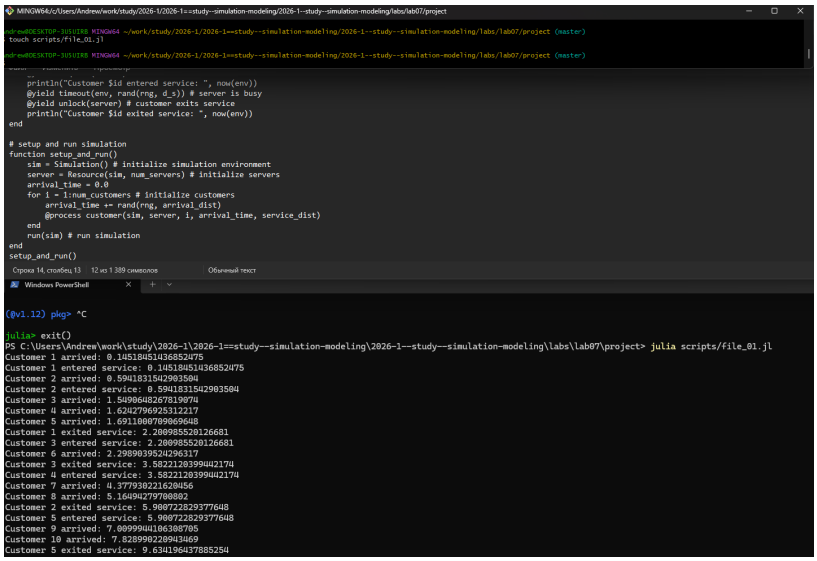

Рисунок 3.3: Создание файла для реализации M/M/c

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.4).

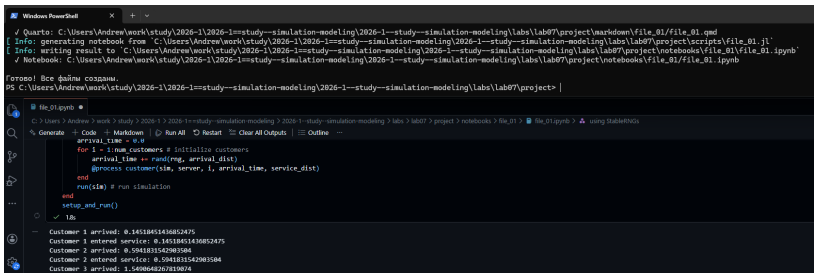

Рисунок 3.4: Код в стиле jupyter notebook

Создаем файл, в котором добавляем графики: диаграмма обслуживания, кол-во клиентов в системе, время ожидания по клиентам (рис. 3.5).

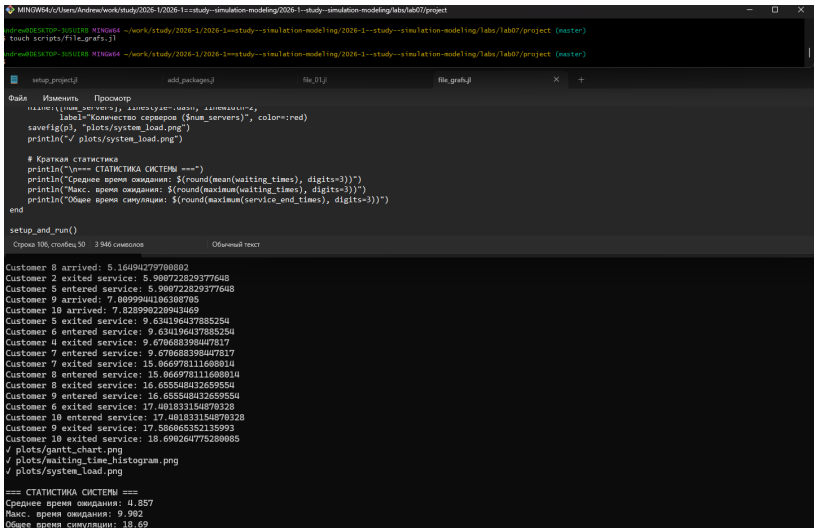

Рисунок 3.5: Файл с графиками

Первый график:диаграмма обслуживания - мы видим как система принимает клиентов и насколько долго обрабатывается каждый клиент (рис. 3.6).

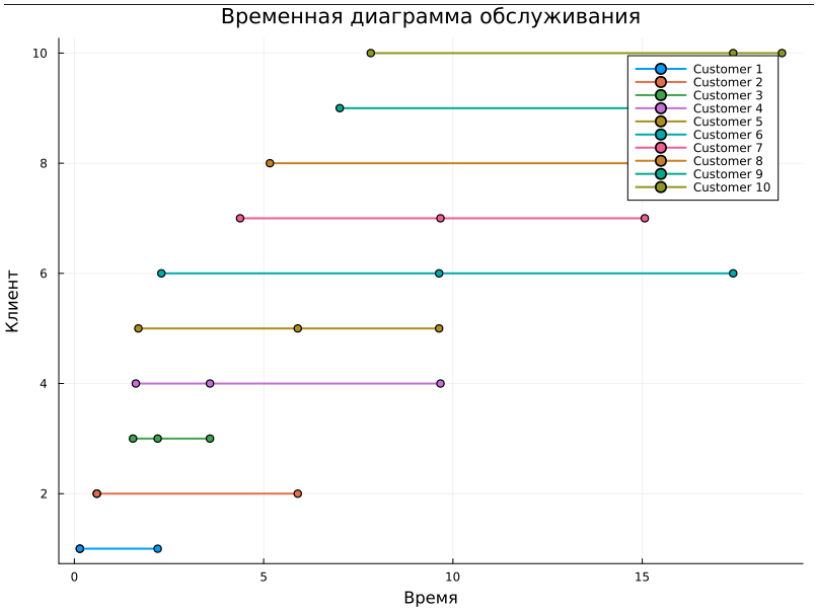

Рисунок 3.6: Диграмма обслуживания

Второй график: кол-во клиентов в системе - мы видим как со временем меняется количество клиентов в системе, кто-то ждет, кто то уже обрабатывается сервером (рис. 3.7).

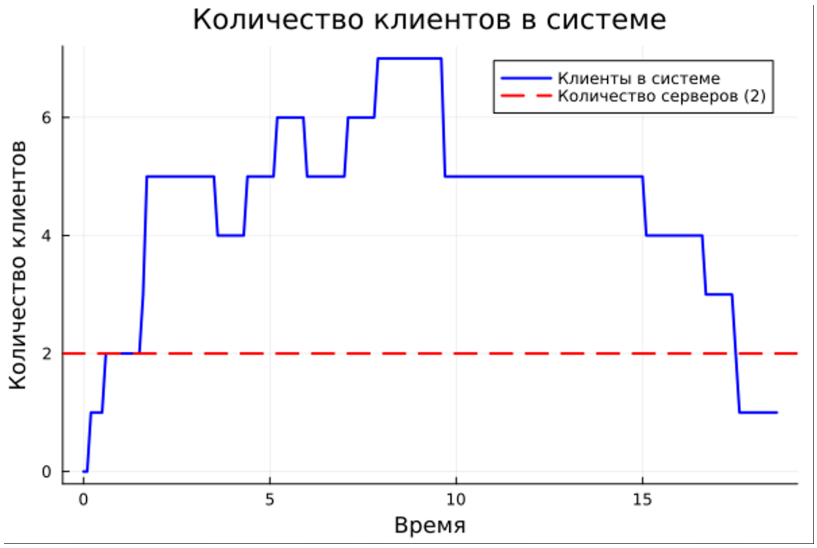

Рисунок 3.7: Количество клиентов в системе

Третий график: время ожидания по клиентам - мы видим, как скапливаются клиенты, образуются очереди и сколько customers приходится ждать пока освободится сервер (рис. 3.8).

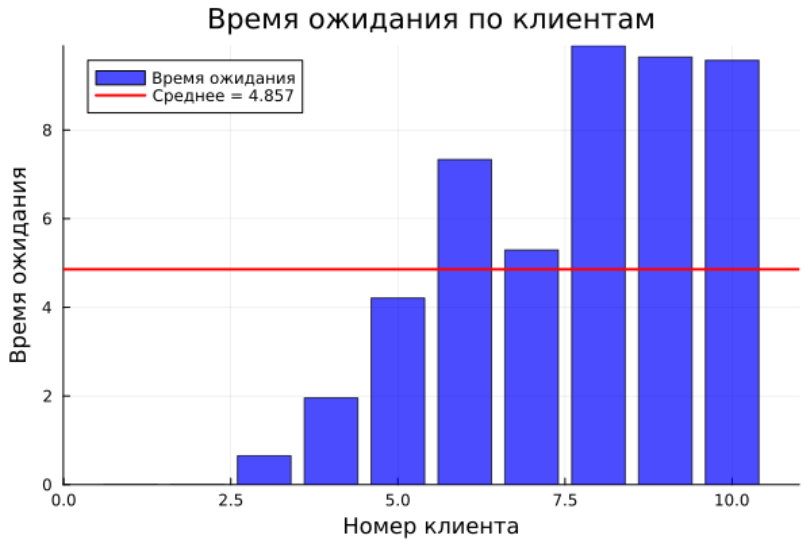

Рисунок 3.8: Время ожидания по клиентам

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.9).

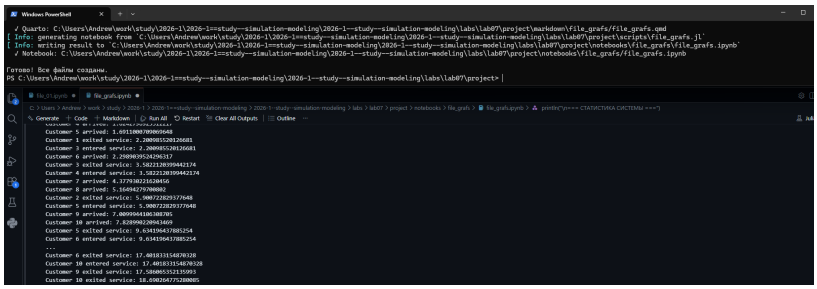

Рисунок 3.9: Код в стиле jupyter notebook

Переходим в модель Росса, для начала реализуем обычные симуляции с параметрами (рабочие машины, запасные, время починки, время поломки и тд.) (рис. 3.10).

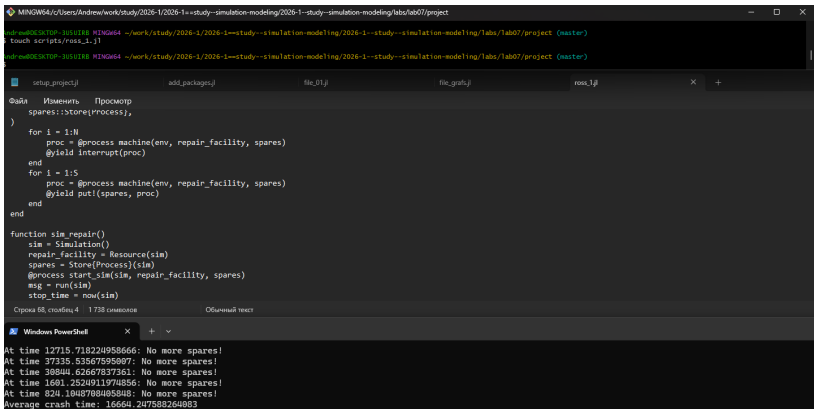

Рисунок 3.10: Первый прогон модели Росса

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.11).

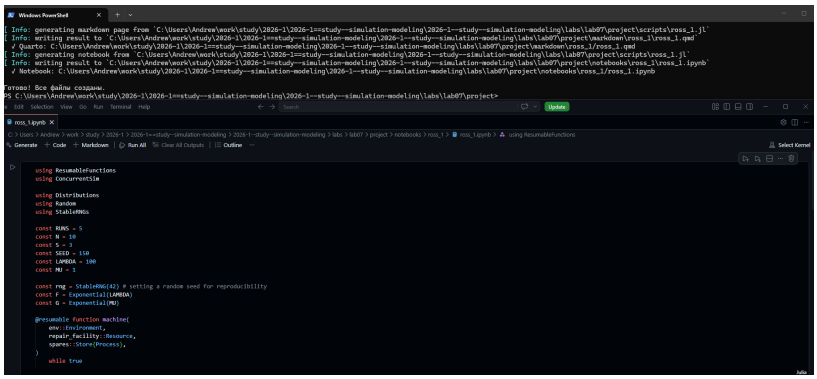

Рисунок 3.11: Код в стиле jupyter notebook

Преобразовываем код, добавляем ремонтников, я решил взять 5, как мы видим, время работы системы сильно увеличилось (рис. 3.12).

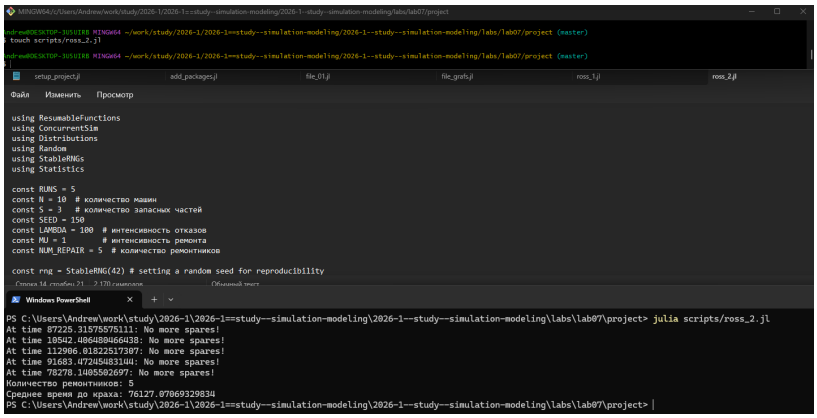

Рисунок 3.12: Результат анимации

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.13).

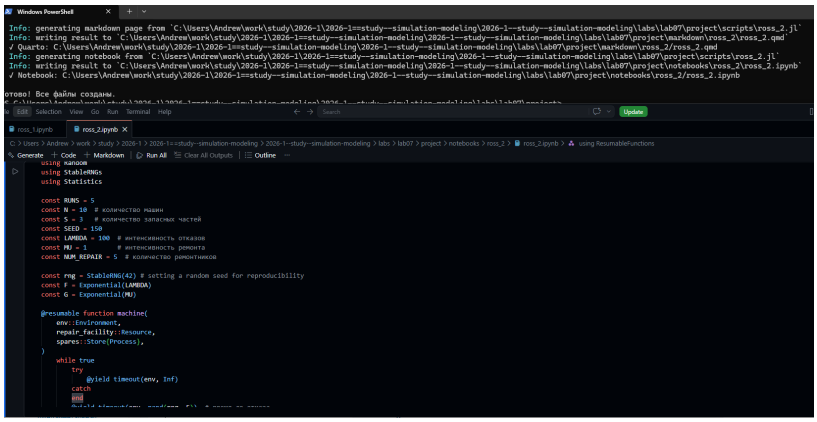

Рисунок 3.13: Код в стиле jupyter notebook

Теперь проводим симуляцию с разным количество работающих машин(N=30), оставляем резерв таким же, как мы видим, система довольно быстро вышла из строя, что не удивительно (рис. 3.14).

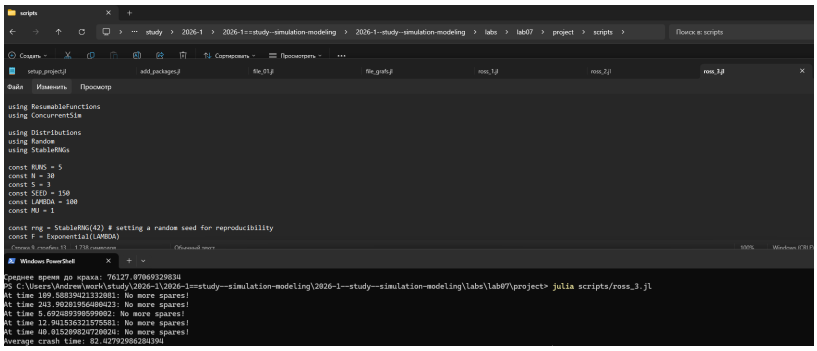

Рисунок 3.14: Код в стиле jupyter notebook

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.15).

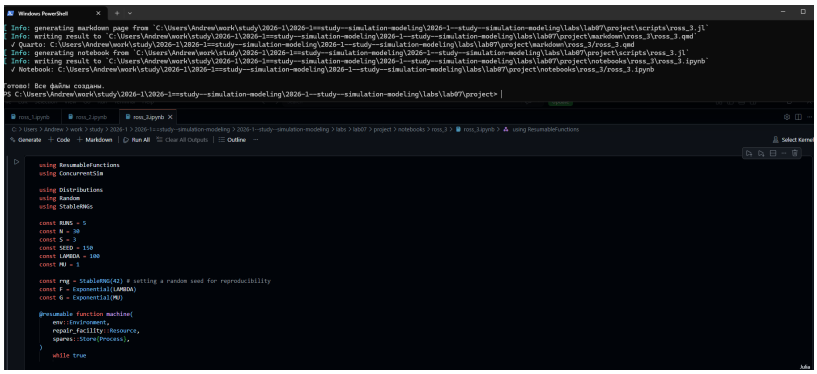

Рисунок 3.15: Код в стиле jupyter notebook

Проводим мониторинг загрузки ремонтиника,средней длины очереди на ремонт (рис. 3.16).

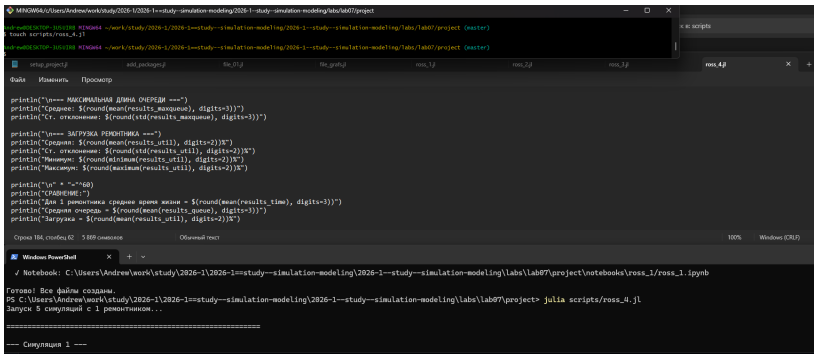

Рисунок 3.16: Загрузка ремонтника

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.17).

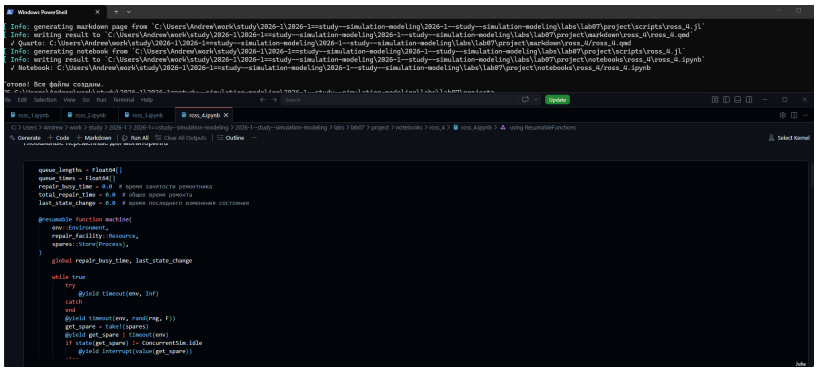

Рисунок 3.17: Код в стиле jupyter notebook

Создаем график изменения числа исправных машин во времени (рис. 3.18).

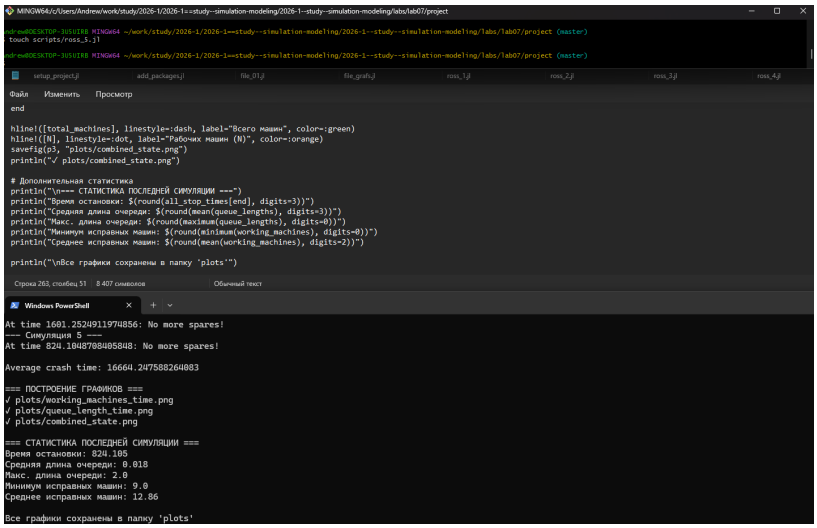

Рисунок 3.18: Добавление графика

Как мы видим, очередь на ремонт большая и рано или поздно система встает (машина сломалась но её нельзя заменить резервной) (рис. 3.19).

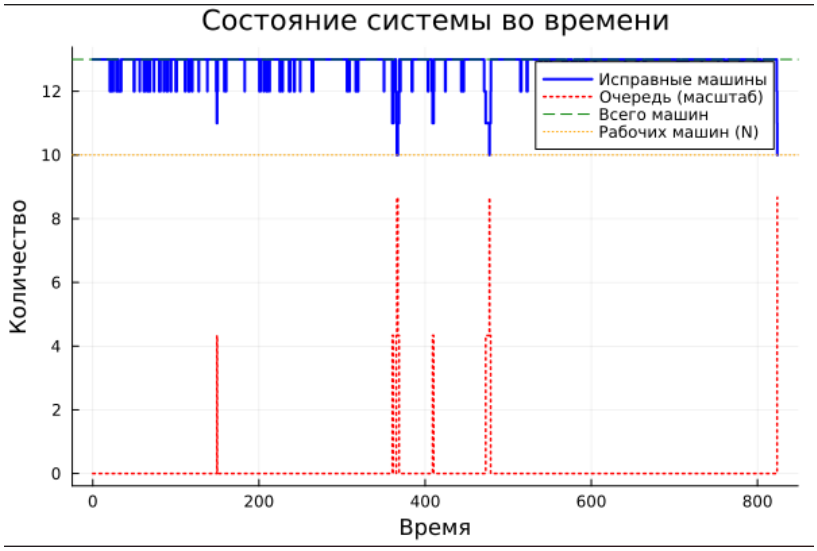

Рисунок 3.19: График

Преобразовываем код в произвольные стили и открываем один из них, чтобы убедиться в правильности компиляции (рис. 3.20).

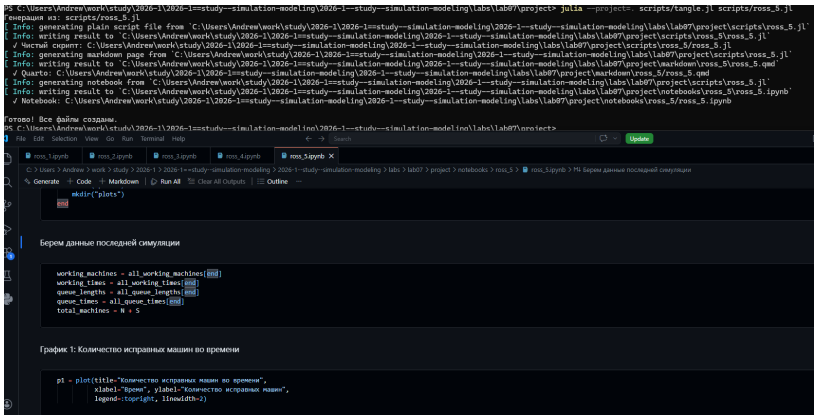

Рисунок 3.20: Код в стиле jupyter notebook

Для первого прогона модели M/M/c

```
using StableRNGs
using Distributions
using ConcurrentSim
using ResumableFunctions

#set simulation parameters
rng = StableRNG(123)
num_customers = 10 # total number of customers generated
```

set queue parameters

```
num_servers = 2 # number of servers
mu = 1.0 / 2 # service rate
lam = 0.9 # arrival rate
arrival_dist = Exponential(1 / lam) # interarrival time distributio
service_dist = Exponential(1 / mu) # service time distribution
```

define customer behavior

```
@resumable function customer(
    env::Environment,
    server::Resource,
    id::Integer,
    t_a::Float64,
    d_s::Distribution,
)
    @yield timeout(env, t_a) # customer arrives
    println("Customer $id arrived: ", now(env))
    @yield request(server) # customer starts service
    println("Customer $id entered service: ", now(env))
    @yield timeout(env, rand(rng, d_s)) # server is busy
    @yield unlock(server) # customer exits service
    println("Customer $id exited service: ", now(env))
end
```

setup and run simulation

```
function setup_and_run()
    sim = Simulation() # initialize simulation environment
    server = Resource(sim, num_servers) # initialize servers
    arrival_time = 0.0
    for i = 1:num_customers # initialize customers
        arrival_time += rand(rng, arrival_dist)
        @process customer(sim, server, i, arrival_time, service_dis
    end
    run(sim) # run simulation
end
setup_and_run()
```

Построения графиков M/M/c

```
using StableRNGs
using Distributions
using ConcurrentSim
using ResumableFunctions
using Plots
using Statistics

#set simulation parameters
rng = StableRNG(123)
num_customers = 10 # total number of customers generated
```

set queue parameters

```
num_servers = 2 # number of servers
mu = 1.0 / 2 # service rate
lam = 0.9 # arrival rate
arrival_dist = Exponential(1 / lam) # interarrival time distributio
service_dist = Exponential(1 / mu) # service time distribution
```

Массивы для сбора данных

```
arrival_times = Vector{Float64}(undef, num_customers)
service_start_times = Vector{Float64}(undef, num_customers)
service_end_times = Vector{Float64}(undef, num_customers)
```

define customer behavior

```julia
@resumable function customer(
    env::Environment,
    server::Resource,
    id::Integer,
    t_a::Float64,
    d_s::Distribution,
)
    @yield timeout(env, t_a) # customer arrives
    arrival_times[id] = now(env)
    println("Customer $id arrived: ", now(env))
    @yield request(server) # customer starts service
    service_start_times[id] = now(env)
    println("Customer $id entered service: ", now(env))
    service_time = rand(rng, d_s)
    @yield timeout(env, service_time) # server is busy
    service_end_times[id] = now(env)
    println("Customer $id exited service: ", now(env))
    @yield unlock(server) # customer exits service
end
```

setup and run simulation

```julia
function setup_and_run()
    sim = Simulation() # initialize simulation environment
    server = Resource(sim, num_servers) # initialize servers
    arrival_time = 0.0
    for i = 1:num_customers # initialize customers
        arrival_time += rand(rng, arrival_dist)
        @process customer(sim, server, i, arrival_time, service_dis
```

```
    end
    run(sim) # run simulation
```

Создание графиков

```
    waiting_times = service_start_times .- arrival_times
```

1. Диаграмма Ганта

```
    p1 = plot(title="□□□□□□□□□ □□□□□□□□□ □□□□□□□□□□□□",
              xlabel="□□□□□", ylabel="□□□□□□", legend=:topright, size=
    for i in 1:num_customers
        plot!([arrival_times[i], service_start_times[i], service_en
              [i, i, i], marker=:circle, linewidth=2, label="Custom
    end
    savefig(p1, "plots/gantt_chart.png")
    println("□ plots/gantt_chart.png")
```

2. Гистограмма времени ожидания

```
    p2 = bar(1:length(waiting_times), waiting_times,
             title="□□□□□ □□□□□□□□ □□ □□□□□□□□",
             xlabel="□□□□□ □□□□□□□", ylabel="□□□□□ □□□□□□□□",
             label="□□□□□ □□□□□□□□", color=:blue, alpha=0.7)
    hline!([mean(waiting_times)], label="□□□□□□□ = $(round(mean(waiti
           color=:red, linewidth=2)
    savefig(p2, "plots/waiting_time_histogram.png")
    println("□ plots/waiting_time_histogram.png")
```

3. Динамика количества клиентов

```
time_points = 0:0.1:maximum(service_end_times)
n_clients = zeros(length(time_points))

for j in 1:length(time_points)
    t = time_points[j]
    count = 0
    for i in 1:num_customers
        if arrival_times[i] <= t && service_end_times[i] > t
            count += 1
        end
    end
    n_clients[j] = count
end

p3 = plot(time_points, n_clients,
          title="□□□□□□□□□□ □□□□□□□□ □ □□□□□□□",
          xlabel="□□□□□", ylabel="□□□□□□□□□□ □□□□□□□□",
          label="□□□□□□□ □ □□□□□□□", linewidth=2, color=:blue)
hline!([num_servers], linestyle=:dash, linewidth=2,
       label="□□□□□□□□□□ □□□□□□□□ ($num_servers)", color=:red)
savefig(p3, "plots/system_load.png")
println("□ plots/system_load.png")
```

Краткая статистика

```
println("\n=== □□□□□□□□□□ □□□□□□□ ===")
println("□□□□□□□ □□□□□ □□□□□□□□: $(round(mean(waiting_times), digits
println("□□□□. □□□□□ □□□□□□□□: $(round(maximum(waiting_times), digi
println("□□□□□ □□□□□ □□□□□□□□□: $(round(maximum(service_end_times),
```

```
end

setup_and_run()
```

Первый прогон модели Росса

```
using ResumableFunctions
using ConcurrentSim

using Distributions
using Random
using StableRNGs

const RUNS = 5
const N = 10
const S = 3
const SEED = 150
const LAMBDA = 100
const MU = 1

const rng = StableRNG(42) # setting a random seed for reproducibili
const F = Exponential(LAMBDA)
const G = Exponential(MU)

@resumable function machine(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
```

```
    while true
        try
            @yield timeout(env, Inf)
        catch
        end
        @yield timeout(env, rand(rng, F))
        get_spare = take!(spares)
        @yield get_spare | timeout(env)
        if state(get_spare) != ConcurrentSim.idle
            @yield interrupt(value(get_spare))
        else
            throw(StopSimulation("No more spares!"))
        end
        @yield request(repair_facility)
        @yield timeout(env, rand(rng, G))
        @yield unlock(repair_facility)
        @yield put!(spares, active_process(env))
    end
end

@resumable function start_sim(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
    for i = 1:N
        proc = @process machine(env, repair_facility, spares)
        @yield interrupt(proc)
```

```
    end
    for i = 1:S
        proc = @process machine(env, repair_facility, spares)
        @yield put!(spares, proc)
    end
end

function sim_repair()
    sim = Simulation()
    repair_facility = Resource(sim)
    spares = Store{Process}(sim)
    @process start_sim(sim, repair_facility, spares)
    msg = run(sim)
    stop_time = now(sim)
    println("At time $stop_time: $msg")
    stop_time
end

results = Float64[]
for i = 1:RUNS
    push!(results, sim_repair())
end
println("Average crash time: ", sum(results)/RUNS)
```

Прогон модели Росса для 5 ремонтников

```
using ResumableFunctions
using ConcurrentSim
using Distributions
```

```julia
using Random
using StableRNGs
using Statistics

const RUNS = 5
const N = 10  # □□□□□□□□□□ □□□□□
const S = 3   # □□□□□□□□□□ □□□□□□□□ □□□□□□
const SEED = 150
const LAMBDA = 100  # □□□□□□□□□□□□□ □□□□□□□
const MU = 1        # □□□□□□□□□□□□□ □□□□□□□
const NUM_REPAIR = 5  # □□□□□□□□□□ □□□□□□□□□□□

const rng = StableRNG(42) # setting a random seed for reproducibili
const F = Exponential(LAMBDA)
const G = Exponential(MU)

@resumable function machine(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
    while true
        try
            @yield timeout(env, Inf)
        catch
        end
        @yield timeout(env, rand(rng, F))  # □□□□□ □□ □□□□□□
        get_spare = take!(spares)
```

```
        @yield get_spare | timeout(env)
        if state(get_spare) != ConcurrentSim.idle
            @yield interrupt(value(get_spare))
        else
            throw(StopSimulation("No more spares!"))
        end
        @yield request(repair_facility)  # [illegible]
        @yield timeout(env, rand(rng, G))  # [illegible]
        @yield unlock(repair_facility)     # [illegible]
        @yield put!(spares, active_process(env))  # [illegible]
    end
end

@resumable function start_sim(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
```

Запускаем N машин

```
    for i = 1:N
        proc = @process machine(env, repair_facility, spares)
        @yield interrupt(proc)
    end
```

Добавляем S запасных частей

```
    for i = 1:S
        proc = @process machine(env, repair_facility, spares)
        @yield put!(spares, proc)
    end
end

function sim_repair()
    sim = Simulation()
    repair_facility = Resource(sim, NUM_REPAIR)  # NUM_REPAIR □□□□□□
    spares = Store{Process}(sim)
    @process start_sim(sim, repair_facility, spares)
    msg = run(sim)
    stop_time = now(sim)
    println("At time $stop_time: $msg")
    return stop_time
end

results = Float64[]
for i = 1:RUNS
    push!(results, sim_repair())
end

println("□□□□□□□□□ □□□□□□□□□□: $NUM_REPAIR")
println("□□□□□□□ □□□□□ □□ □□□□□: ", sum(results)/RUNS)
```

Прогон для 30 машин

```julia
using ResumableFunctions
using ConcurrentSim

using Distributions
using Random
using StableRNGs

const RUNS = 5
const N = 30
const S = 3
const SEED = 150
const LAMBDA = 100
const MU = 1

const rng = StableRNG(42) # setting a random seed for reproducibili
const F = Exponential(LAMBDA)
const G = Exponential(MU)

@resumable function machine(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
    while true
        try
            @yield timeout(env, Inf)
        catch
        end
```

```
        @yield timeout(env, rand(rng, F))
        get_spare = take!(spares)
        @yield get_spare | timeout(env)
        if state(get_spare) != ConcurrentSim.idle
            @yield interrupt(value(get_spare))
        else
            throw(StopSimulation("No more spares!"))
        end
        @yield request(repair_facility)
        @yield timeout(env, rand(rng, G))
        @yield unlock(repair_facility)
        @yield put!(spares, active_process(env))
    end
end

@resumable function start_sim(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
    for i = 1:N
        proc = @process machine(env, repair_facility, spares)
        @yield interrupt(proc)
    end
    for i = 1:S
        proc = @process machine(env, repair_facility, spares)
        @yield put!(spares, proc)
    end
```

```
end

function sim_repair()
    sim = Simulation()
    repair_facility = Resource(sim)
    spares = Store{Process}(sim)
    @process start_sim(sim, repair_facility, spares)
    msg = run(sim)
    stop_time = now(sim)
    println("At time $stop_time: $msg")
    stop_time
end

results = Float64[]
for i = 1:RUNS
    push!(results, sim_repair())
end
println("Average crash time: ", sum(results)/RUNS)
```

Прогон для загрузки ремонтника

```
using ResumableFunctions
using ConcurrentSim
using Distributions
using Random
using StableRNGs
using Statistics

const RUNS = 5
```

```julia
const N = 10
const S = 3
const SEED = 150
const LAMBDA = 100
const MU = 1
const NUM_REPAIR = 1  # [illegible] [illegible]

const rng = StableRNG(42)
const F = Exponential(LAMBDA)
const G = Exponential(MU)
```

Глобальные переменные для мониторинга

```julia
queue_lengths = Float64[]
queue_times = Float64[]
repair_busy_time = 0.0  # [illegible] [illegible] [illegible]
total_repair_time = 0.0  # [illegible] [illegible] [illegible]
last_state_change = 0.0  # [illegible] [illegible] [illegible] [illegible]

@resumable function machine(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
    global repair_busy_time, last_state_change

    while true
        try
            @yield timeout(env, Inf)
```

```
        catch
        end
        @yield timeout(env, rand(rng, F))
        get_spare = take!(spares)
        @yield get_spare | timeout(env)
        if state(get_spare) != ConcurrentSim.idle
            @yield interrupt(value(get_spare))
        else
            throw(StopSimulation("No more spares!"))
        end
```

Запрос ремонтника и запись длины очереди

```
        current_queue = length(repair_facility.queue)
        push!(queue_times, now(env))
        push!(queue_lengths, current_queue)

        @yield request(repair_facility)
```

Ремонт

```
        repair_time = rand(rng, G)
        @yield timeout(env, repair_time)

        @yield unlock(repair_facility)
        @yield put!(spares, active_process(env))
    end
end

@resumable function monitor_system(
```

```
    env::Environment,
    repair_facility::Resource
)
    while true
        @yield timeout(env, 0.1)  # [illegible] [illegible] 0.1 [illegible]
        current_queue = length(repair_facility.queue)
        push!(queue_times, now(env))
        push!(queue_lengths, current_queue)
    end
end

@resumable function start_sim(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
)
```

Запускаем мониторинг

```
    @process monitor_system(env, repair_facility)

    for i = 1:N
        proc = @process machine(env, repair_facility, spares)
        @yield interrupt(proc)
    end
    for i = 1:S
        proc = @process machine(env, repair_facility, spares)
        @yield put!(spares, proc)
    end
```

```
end

function sim_repair()
    global queue_lengths, queue_times, repair_busy_time, last_state
```

Сбрасываем данные

```
    queue_lengths = Float64[]
    queue_times = Float64[]
    repair_busy_time = 0.0
    last_state_change = 0.0

    sim = Simulation()
    repair_facility = Resource(sim, NUM_REPAIR)
    spares = Store{Process}(sim)
    @process start_sim(sim, repair_facility, spares)
    msg = run(sim)
    stop_time = now(sim)
```

Расчет метрик

```
    avg_queue_length = length(queue_lengths) > 0 ? mean(queue_lengt
    max_queue_length = length(queue_lengths) > 0 ? maximum(queue_le
```

Расчет загрузки ремонтника через статистику ресурса ConcurrentSim не дает прямой доступ к времени занятости, оцениваем косвенно

```
    utilization = (1 - (length(repair_facility.queue) / (NUM_REPAIR
```

Альтернативный расчет загрузки через количество обращений

```
    total_services = length(repair_facility.server)  # [illegible]

    println("\n=== [illegible] [illegible] ===")
    println("[illegible]: $(round(stop_time, digits=3))")
    println("[illegible]: $msg")
    println("\n=== [illegible] [illegible] ===")
    println("[illegible] [illegible]: $(round(avg_queue_length, digits=3))
    println("[illegible] [illegible]: $max_queue_length")
    println("[illegible]: $(length(queue_lengths))")
    println("\n=== [illegible] [illegible] ===")
    println("[illegible]: $NUM_REPAIR")
    println("[illegible]: $(round(utilization, digits=2))%")

    return stop_time, avg_queue_length, max_queue_length, utilizati
end
```

Запуск нескольких симуляций

```
results_time = Float64[]
results_queue = Float64[]
results_maxqueue = Float64[]
results_util = Float64[]

println("[illegible] $(RUNS) [illegible] [illegible] 1 [illegible]...\n")
println("="^60)

for i = 1:RUNS
    println("\n--- [illegible] $i ---")
    stop_time, avg_q, max_q, util = sim_repair()
```

```
    push!(results_time, stop_time)
    push!(results_queue, avg_q)
    push!(results_maxqueue, max_q)
    push!(results_util, util)
end
```

Итоговая статистика

```
println("\n" * "="^60)
println("□□□□□□□□ □□□□□□□□□□ □□ $(RUNS) □□□□□□□□ (1 □□□□□□□□□)")
println("="^60)

println("\n=== □□□□□ □□ □□□□□ ===")
println("□□□□□□□□: $(round(mean(results_time), digits=3))")
println("□□. □□□□□□□□□□: $(round(std(results_time), digits=3))")
println("□□□□□□□□: $(round(minimum(results_time), digits=3))")
println("□□□□□□□□: $(round(maximum(results_time), digits=3))")

println("\n=== □□□□□□□□ □□□□□ □□□□□□□□ ===")
println("□□□□□□□□: $(round(mean(results_queue), digits=3))")
println("□□. □□□□□□□□□□: $(round(std(results_queue), digits=3))")
println("□□□□□□□□: $(round(minimum(results_queue), digits=3))")
println("□□□□□□□□: $(round(maximum(results_queue), digits=3))")

println("\n=== □□□□□□□□□□□□ □□□□□ □□□□□□□□ ===")
println("□□□□□□□□: $(round(mean(results_maxqueue), digits=3))")
println("□□. □□□□□□□□□□: $(round(std(results_maxqueue), digits=3))")

println("\n=== □□□□□□□□ □□□□□□□□□□ ===")
```

```
println("□□□□□□□: $(round(mean(results_util), digits=2))%")
println("□□. □□□□□□□□□□: $(round(std(results_util), digits=2))%")
println("□□□□□□□: $(round(minimum(results_util), digits=2))%")
println("□□□□□□□□: $(round(maximum(results_util), digits=2))%")

println("\n" * "="^60)
println("□□□□□□□□□:")
println("□□□ 1 □□□□□□□□□□ □□□□□□□ □□□□□ □□□□□ = $(round(mean(results_time)
println("□□□□□□□ □□□□□□□ = $(round(mean(results_queue), digits=3))")
println("□□□□□□□□ = $(round(mean(results_util), digits=2))%")
```

Добавление графика изменения числа машин во времени

```
using ResumableFunctions
using ConcurrentSim
using Distributions
using Random
using StableRNGs
using Plots
using Statistics

const RUNS = 5
const N = 10
const S = 3
const SEED = 150
const LAMBDA = 100
const MU = 1

const rng = StableRNG(42) # setting a random seed for reproducibili
```

```
const F = Exponential(LAMBDA)
const G = Exponential(MU)
```

Глобальные переменные для сбора данных

```
all_working_machines = Vector{Vector{Float64}}()
all_working_times = Vector{Vector{Float64}}()
all_queue_lengths = Vector{Vector{Float64}}()
all_queue_times = Vector{Vector{Float64}}()
```

Счетчик машин в очереди

```
queue_counter = 0

@resumable function machine(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
    working_machines::Vector{Float64},
    working_times::Vector{Float64},
    queue_lengths::Vector{Float64},
    queue_times::Vector{Float64},
    total_machines::Int,
)
    global queue_counter

    while true
        try
            @yield timeout(env, Inf)
```

```
        catch
        end
```

Машина работает

```
        @yield timeout(env, rand(rng, F))
```

Поломка - уменьшаем исправные машины

```
        push!(working_times, now(env))
        push!(working_machines, working_machines[end] - 1)

        get_spare = take!(spares)
        @yield get_spare | timeout(env)
        if state(get_spare) != ConcurrentSim.idle
            @yield interrupt(value(get_spare))
        else
            throw(StopSimulation("No more spares!"))
        end
```

Запись очереди перед запросом ремонтника

```
        push!(queue_times, now(env))
        push!(queue_lengths, queue_counter)

        queue_counter += 1  # [illegible] [illegible] [illegible]
        @yield request(repair_facility)
        queue_counter -= 1  # [illegible] [illegible] [illegible]

        @yield timeout(env, rand(rng, G))
        @yield unlock(repair_facility)
```

Ремонт завершен - увеличиваем исправные машины

```
        push!(working_times, now(env))
        push!(working_machines, working_machines[end] + 1)

        @yield put!(spares, active_process(env))
    end
end

@resumable function monitor_system(
    env::Environment,
    repair_facility::Resource,
    working_machines::Vector{Float64},
    working_times::Vector{Float64},
    queue_lengths::Vector{Float64},
    queue_times::Vector{Float64},
    total_machines::Int,
)
    global queue_counter

    while true
        @yield timeout(env, 0.1)
```

Запись текущей длины очереди

```
        push!(queue_times, now(env))
        push!(queue_lengths, queue_counter)
```

Запись исправных машин

```
        push!(working_times, now(env))
        push!(working_machines, working_machines[end])
    end
end

@resumable function start_sim(
    env::Environment,
    repair_facility::Resource,
    spares::Store{Process},
    working_machines::Vector{Float64},
    working_times::Vector{Float64},
    queue_lengths::Vector{Float64},
    queue_times::Vector{Float64},
    total_machines::Int,
)
```

Запускаем мониторинг

```
    @process monitor_system(env, repair_facility, working_machines,
                            queue_lengths, queue_times, total_machin

    for i = 1:N
        proc = @process machine(env, repair_facility, spares, worki
                                working_times, queue_lengths, queue_
        @yield interrupt(proc)
    end
    for i = 1:S
        proc = @process machine(env, repair_facility, spares, worki
                                working_times, queue_lengths, queue_
```

```
        @yield put!(spares, proc)
    end
end

function sim_repair()
    global queue_counter
    queue_counter = 0

    total_machines = N + S
    working_machines = [Float64(total_machines)]
    working_times = [0.0]
    queue_lengths = Float64[]
    queue_times = Float64[]

    sim = Simulation()
    repair_facility = Resource(sim)  # 1 ⌧⌧⌧⌧⌧⌧⌧⌧
    spares = Store{Process}(sim)
    @process start_sim(sim, repair_facility, spares, working_machin
                       queue_lengths, queue_times, total_machines)
    msg = run(sim)
    stop_time = now(sim)
    println("At time $stop_time: $msg")
```

Сохраняем данные для графиков

```
    push!(all_working_machines, copy(working_machines))
    push!(all_working_times, copy(working_times))
    push!(all_queue_lengths, copy(queue_lengths))
    push!(all_queue_times, copy(queue_times))
```

```
    return stop_time, working_machines, working_times, queue_length
end
```

Запуск симуляций и сбор данных

```
results = Float64[]
all_stop_times = Float64[]

println("□□□□□□ $(RUNS) □□□□□□□□□...\n")

for i = 1:RUNS
    println("--- □□□□□□□□□ $i ---")
    stop_time, _, _, _, _ = sim_repair()
    push!(results, stop_time)
    push!(all_stop_times, stop_time)
end

println("\nAverage crash time: ", sum(results)/RUNS)
```

Построение графиков по последней симуляции

```
println("\n=== □□□□□□□□□□ □□□□□□□□ ===")
```

Создаем папку для графиков

```
if !isdir("plots")
    mkdir("plots")
end
```

Берем данные последней симуляции

```
working_machines = all_working_machines[end]
working_times = all_working_times[end]
queue_lengths = all_queue_lengths[end]
queue_times = all_queue_times[end]
total_machines = N + S
```

График 1: Количество исправных машин во времени

```
p1 = plot(title="□□□□□□□□□□ □□□□□□□□□□ □□□□□□ □□ □□□□□□□□",
          xlabel="□□□□□□", ylabel="□□□□□□□□□□ □□□□□□□□□□ □□□□□□",
          legend=:topright, linewidth=2)
```

Подготовка данных для ступенчатого графика

```
plot_times = []
plot_values = []
for i in 1:length(working_times)-1
    push!(plot_times, working_times[i])
    push!(plot_values, working_machines[i])
    push!(plot_times, working_times[i+1])
    push!(plot_values, working_machines[i])
end

plot!(plot_times, plot_values, label="□□□□□□□□□□ □□□□□□□□", color=:blue, s
hline!([total_machines], linestyle=:dash, label="□□□□□□ □□□□□□ ($total_n
hline!([N], linestyle=:dot, label="□□□□□□□□ □□□□□□ (N=$N)", color=:orang
savefig(p1, "plots/working_machines_time.png")
println("□ plots/working_machines_time.png")
```

График 2: Длина очереди на ремонт

```
p2 = plot(title="□□□□□ □□□□□□□ □□ □□□□□□",
          xlabel="□□□□□", ylabel="□□□□□ □□□□□□□",
          legend=:topright, linewidth=2)
```

Подготовка данных для ступенчатого графика очереди

```
if length(queue_times) > 0
    queue_times_plot = []
    queue_vals_plot = []
    for i in 1:length(queue_times)-1
        push!(queue_times_plot, queue_times[i])
        push!(queue_vals_plot, queue_lengths[i])
        push!(queue_times_plot, queue_times[i+1])
        push!(queue_vals_plot, queue_lengths[i])
    end

    plot!(queue_times_plot, queue_vals_plot, label="□□□□□ □□□□□□□", co
    if length(queue_lengths) > 0
        hline!([mean(queue_lengths)], linestyle=:dash,
               label="□□□□□□□ = $(round(mean(queue_lengths), digits=2
    end
end
savefig(p2, "plots/queue_length_time.png")
println("□ plots/queue_length_time.png")
```

График 3: Совмещенный график

```
p3 = plot(title="□□□□□□□□□ □□□□□□□ □□ □□□□□□□",
          xlabel="□□□□□", ylabel="□□□□□□□□□□",
          legend=:topright)
```

Исправные машины

```
plot!(plot_times, plot_values, label="□□□□□□□□□ □□□□□□", color=:blue, s
```

Очередь (масштабируем для наглядности)

```
if length(queue_lengths) > 0 && maximum(queue_lengths) > 0
    scale_factor = total_machines / (maximum(queue_lengths) + 1)
    queue_scaled = queue_vals_plot .* scale_factor
    plot!(queue_times_plot, queue_scaled, label="□□□□□□□ (□□□□□□□)", co
end

hline!([total_machines], linestyle=:dash, label="□□□□□ □□□□□", color=
hline!([N], linestyle=:dot, label="□□□□□□□ □□□□□ (N)", color=:orange)
savefig(p3, "plots/combined_state.png")
println("□ plots/combined_state.png")
```

Дополнительная статистика

```
println("\n=== □□□□□□□□□ □□□□□□□□□ □□□□□□□□□ ===")
println("□□□□□ □□□□□□□□□: $(round(all_stop_times[end], digits=3))")
println("□□□□□□□ □□□□□ □□□□□□□: $(round(mean(queue_lengths), digits=3))"
println("□□□□. □□□□□ □□□□□□□: $(round(maximum(queue_lengths), digits=0)
println("□□□□□□□ □□□□□□□□□ □□□□□: $(round(minimum(working_machines), dig
println("□□□□□□□ □□□□□□□□□ □□□□□: $(round(mean(working_machines), digits

println("\n□□□ □□□□□□□ □□□□□□□□□ □ □□□□□ 'plots'")
```

4 Выводы

В данной работе мы изучили и применили дискретно-событийное моделирование.