# **Общие сведения**

В рамках курса «Функциональное программирование» разработана программа «Симулятор лесного пожара».

## Функциональное назначение

Разработанная программа предназначена для имитации лесного пожара с учетом некоторых природных факторов, таких как:

* относительная влажность;
* ветер;
* вероятность возгорания дерева;
* вероятность появления нового дерева.

Программа разработана в первую очередь для демонстрации возможностей функционального программирования на Python. В перспективе, разработанная программа может иметь и практическое применение: структура программы предполагает расширение функциональности, благодаря чему появится возможность прогнозировать распространение пожара в лесном массиве.

## Программное и аппаратное обеспечение, необходимое для функционирования программы

«Симулятор лесного пожара» может быть запущен на ОС Windows 32/64. Запускать программу можно двумя способами:

1. С помощью готового exe-файла: необходимо скачать и распаковать архив main.zip (<https://github.com/patrikeevairina/fire_simulation>), и затем запустить файл main.exe
2. С помощью исходных файлов: необходимо следовать инструкциям из Readme.md (<https://github.com/patrikeevairina/fire_simulation>)

## Используемый язык программирования

Программа написана на языке Python версии 3.11, графический интерфейс реализован с использованием модуля PySide6, позволяющим использовать библиотеки Qt для разработки.

В первую очередь, было принято решение использовать для реализации графического интерфейса библиотеки Qt по нескольким причинам:

1. Я имею опыт разработки графических интерфейсов с помощью фреймворка Qt.
2. Qt предоставляет удобные инструменты для визуализации графических элементов, а также механизм сигналов/слотов, упрощающий обработку взаимодействия пользователя с графическим интерфейсом.

После выбора инструментов для реализации графического интерфейса в качестве языка программирования был выбран Python, так как:

1. Python хорошо адаптирован для использования библиотек Qt.
2. Python предоставляет возможность использования функциональной парадигмы.

# Структура программного обеспечения

## Алгоритмы работы программы

При разработке программы были отделены друг от друга графический интерфейс и модуль, отвечающий за определение нового состояния поля, в связи с чем ниже будут описаны алгоритмы работы графического интерфейса и алгоритмы работы вычислительного модуля.

Графический интерфейс программы предназначен для определения параметров симуляции, настройки среды симуляции, визуализации распространения деревьев в лесу.

1. Алгоритмы изменения параметров симуляции

Программа предоставляет возможность пользователю задать:

* новые параметры симуляции, такие как: количество деревьев по горизонтали, количество деревьев по вертикали, таймаут обновления графической сцены, размер дерева, тип симуляции (чистая симуляция или нет);
* новые параметры среды симуляции: вероятность появления нового дерева, вероятность случайного возгорания дерева, относительную влажность, направление ветра.

На рисунке 1 представлена диаграмма последовательности, отображающая алгоритм изменения параметров среды симуляции.

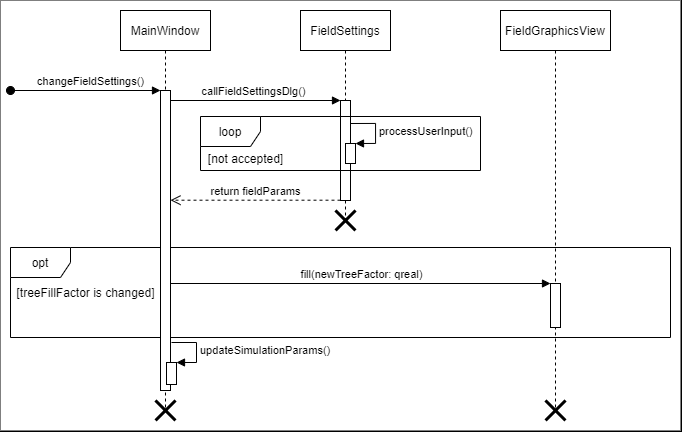


Рис. 1. Изменение параметров среды симуляции

С помощью клика на определенную кнопку в главном окне программы (MainWindow) пользователь инициирует вызов диалогового окна (FieldSettings) для настроек среды симуляции. После того, как пользователь ввел новые параметры симуляции, главное окно получает эти параметры и обновляет соответствующие параметры. Если пользователь задал новую начальную заполненность графической сцены деревьями, то главное окно посылает графической сцене сигнал на обновление.

Аналогичным образом выстроено взаимодействие главного окна программы, диалогового окна настроек симуляции и графической сцены для реализации возможности изменения параметров симуляции.

1. Алгоритм обновления графической сцены

Программа предоставляет возможность запуска симуляции по заданному периоду. Алгоритм обновления графической сцены по заданному периоду представлен в виде диаграммы последовательности, изображенной на рисунке 2.

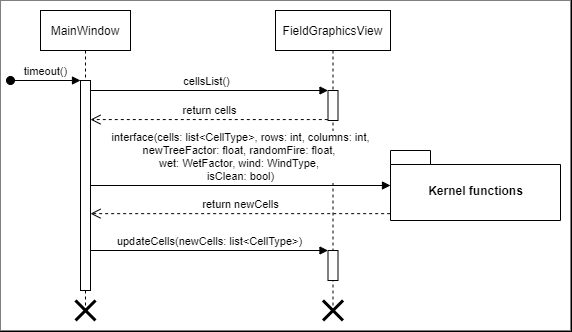


Рис. 2. Алгоритм обновления графической сцены по заданному периоду

По срабатыванию таймера главное окно программы запрашивает у графической сцены список текущих состояний элементов поля в виде списка. Каждый элемент поля может находиться в одном из состояний:

* занят деревом;
* занят горящим деревом;
* ничем не занят;
* занят сгоревшим деревом.

После получения текущих состояний элементов поля главное окно вызывает внешнюю функцию вычислительного модуля «Kernel functions». В результате работы этого модуля главное окно получает список новых состояний элементов поля в виде списка, и отправляет этот список графической сцене для её обновления.

1. Алгоритм работы внешней функции модуля «Kernel functions»

Алгоритм работы внешней функции модуля «Kernel functions» представлен в виде блок-схемы, изображенной на рисунке 3.

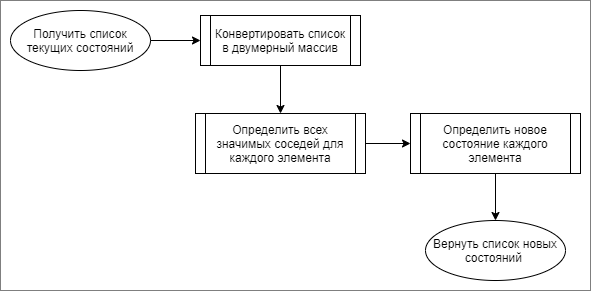


Рис. 3. Алгоритм работы внешней функции вычислительного модуля

На вход эта функция получает список текущих состояний элементов поля от главного окна программы. На следующем шаге создается копия этого списка в виде двумерной матрицы. Так как для определения нового состояния элемента поля требуется знать состояния соседей этого элемента, эта матрица подается на вход в функцию для определения значимых соседей. После этого для каждого элемента поля и его соседей вызывается функция, определяющая новое состояние этого элемента. Все результаты работы предыдущей функции объединяются в список и возвращаются в главное окно программы.

1. Алгоритм работы функции конвертации списка в двумерный массив

На рисунке 4 отображена блок-схема, иллюстрирующая алгоритм работы функции конвертации списка в двумерный массив.

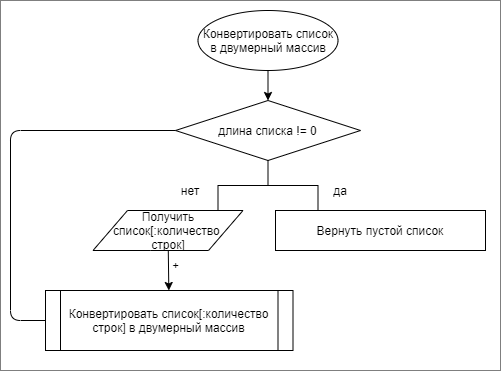


Рис. 4. Алгоритм работы функции конвертации списка в двумерный массив

## Используемые методы

Графический интерфейс: для обработки пользовательских действий, а также для обновления графической сцены по заданному периоду используется механизм сигналов/слотов, предоставляемый фреймворком Qt.

Вычислительный модуль «Kernel functions»:

* конвертация списка в двумерный массив реализована с использованием хвостовой рекурсии;
* функция создания массива с соседями для всех элементов реализована с использованием генераторов;
* функция определения соседей одного элемента реализована с использованием функции filter и анонимных функций.

## Структура программы

На рисунке 5 представлена структура разработанной программы в виде диаграммы пакетов.

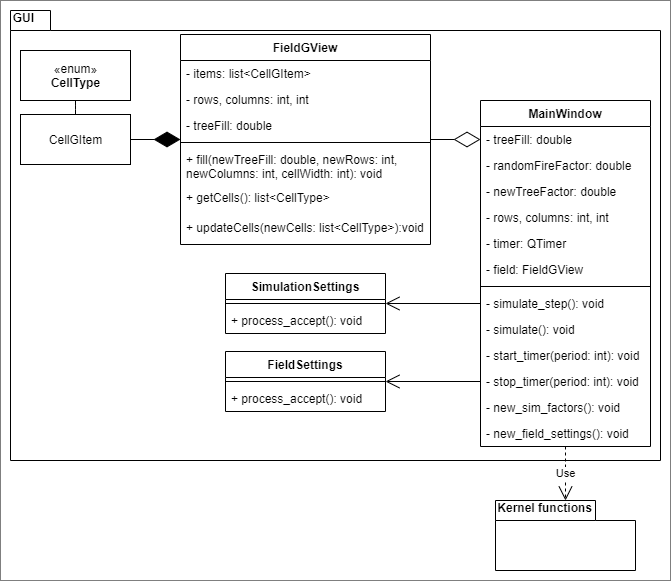


Рис. 5. Структура программы

Пакет графического интерфейса содержит в себе все классы, отвечающие за графический интерфейс программы. Эти классы являются наследниками классов Qt: MainWindow наследуется от QMainWindow, FieldGView наследуется от QGraphicsView, SimulationSettings и FieldSettings наследуются от QDialog, CellGItem наследуется от QGraphicsItem.

Пакет «Kernel functions» в себе не содержит классов, так как вся обработка в нем реализована с использованием функциональной парадигмы.

# Структуры данных

## Входные и выходные данные

Все входные данные программа получает посредством взаимодействия пользователя с графическим интерфейсом. Форматирование и кодирование входных данных реализовано на стороне используемых библиотек Qt.

В качестве выходных данных пользователь получает визуализацию распространения лесного пожара. Визуализацией является обновление состояния графической сцены по заданному пользователем периоду. Благодаря такому способу отображения выходных данных достигается удобство в восприятии динамически обновляющихся результатов. Также реализована возможность управлять процессом получения выходных данных: с помощью элементов графического интерфейса пользователь может запустить получение выходных данных, поставить получение выходных данных на паузу и выполнить шаг получения выходных данных вручную.

# Методика и результаты тестирования

## Состав и структура технических средств

Требования к аппаратным характеристикам компьютера сильно зависят от настроек симуляции: чем большее количество элементов необходимо обрабатывать, тем выше требования к аппаратному обеспечению. По умолчанию выставлена обработка графической сцены с 450 элементами: в таком случае для стабильной работы программы достаточно 60мб свободной оперативной памяти. Для стабильной работы программы с обработкой 17000 элементов требуется уже 160мб свободной оперативной памяти.

Для работы программы не требуется подключение к сети Интернет.

Программа протестирована на устройстве с ОС Windows 64-bit, 16гб свободной оперативной памяти и 8-ядерным процессором.

## Последовательность тестирования

Тестирование программы, как и её разработка, проводилось в два этапа: на первом этапе был разработан, реализован и протестирован графический интерфейс программы; на втором этапе был разработан, реализован и протестирован вычислительный модуль программы.

1. Тестирование графического интерфейса

Последовательно протестированы все формы для взаимодействия с пользователем: окно настроек симуляции, окно настроек среды симуляции, главное окно программы. Тестирование осуществлено вручную.

1. Тестирование вычислительного модуля

При тестировании вычислительного модуля протестированы все функции вычислительного модуля по отдельности. Так как реализация вычислительного модуля соответствует функциональной парадигме, то нет необходимости проверять работу функций в совокупности.

## Результаты тестирования

В этом разделе приведены некоторые сценарии тестирования графического интерфейса и вычислительного модуля программы.

1. Запуск программы

Ожидаемый результат: главное окно программы, содержащее в себе элементы графического интерфейса:

* кнопка для запуска симуляции;
* кнопка для остановки симуляции;
* кнопка для запуска выполнения шага симуляции;
* кнопка для вызова диалогового окна настроек симуляции;
* кнопка для вызова диалогового окна настроек параметров симуляции;
* графическая сцена с объектами;
* элементы для управления главным окном программы (свернуть, развернуть или закрыть главное окно программы).

Результат запуска программы отображен на рисунке 6. Полученный результат совпадает с ожидаемым.

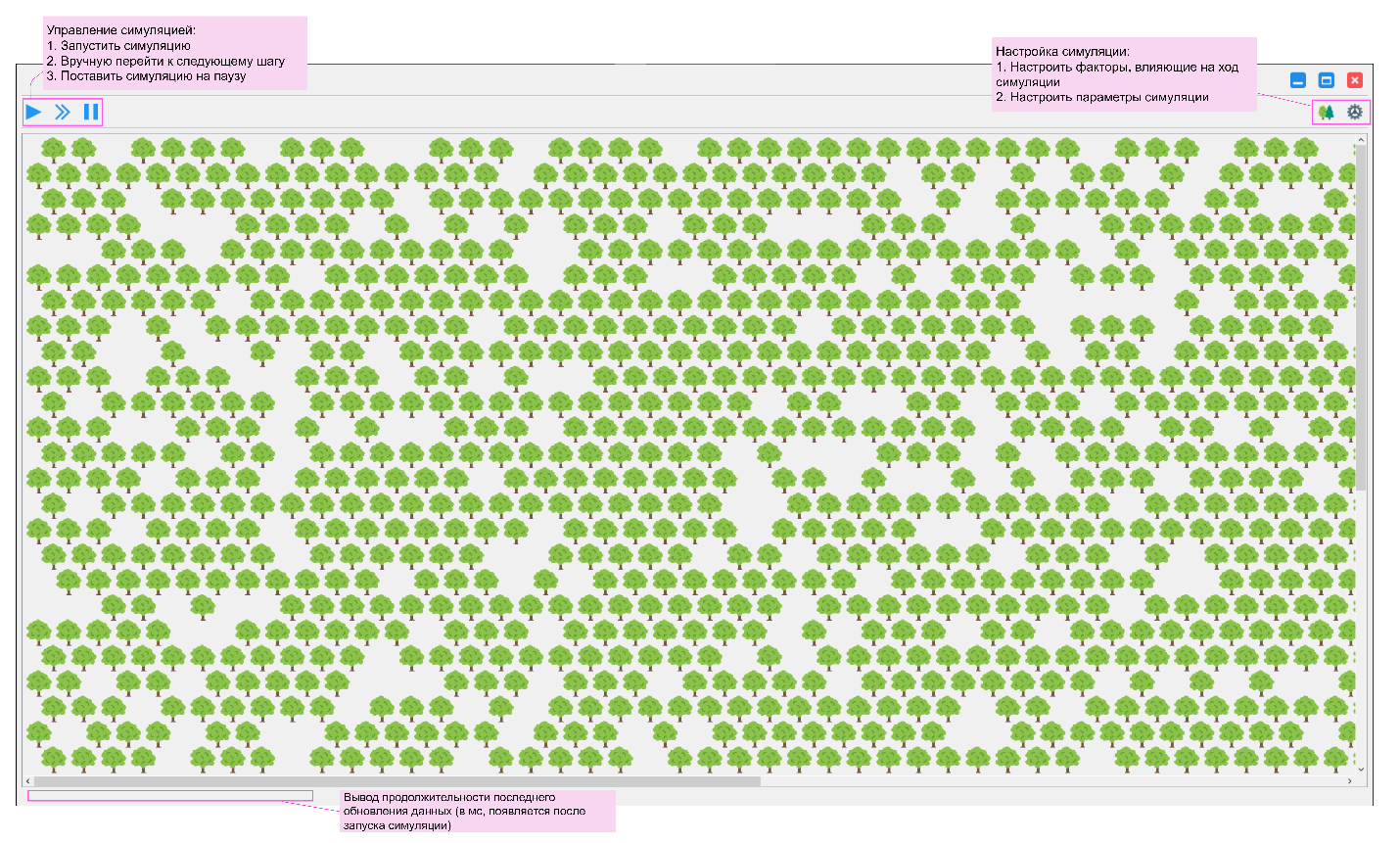


Рис. 6. Пример отображения главного окна после запуска программы

1. Запуск симуляции

Настройки симуляции: период срабатывания таймера – 0 мс (минимальный возможный); размер поля – 100 \* 55 элементов; размер элемента – 15 px; чистая симуляция -- нет.

Настройки среды симуляции: процент поля, заполненного деревьями – 90%; вероятность случайного возгорания дерева – 0.001%; вероятность появления нового дерева – 0.15%; ветер – северный; влажность – нормальная.

Пример отображения главного окна при данных настройках до запуска симуляции приведен на рисунке 7.



Рис.7. Пример отображения главного окна программы

Ожидаемый результат: визуализация распространения пожара

На рисунке 8 приведен пример отображения главного окна программы спустя 10 секунд после запуска симуляции. На приведенном рисунке отображено одно из состояний леса во время распространения пожара с заданными параметрами. Полученный результат совпал с ожидаемым: на графической сцене отображается динамика распространения пожара.

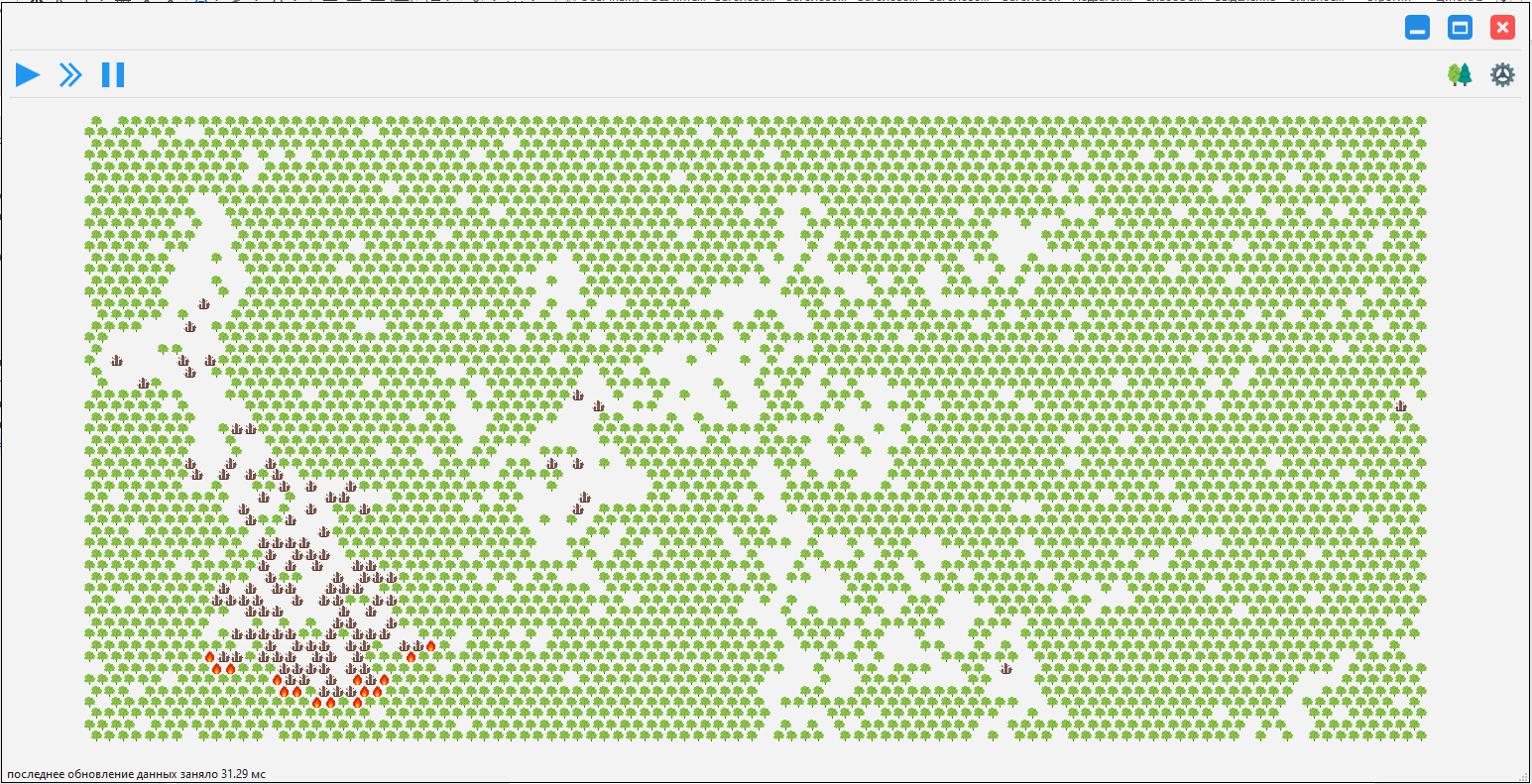


Рис. 8. Пример отображения главного окна программы спустя 10 секунд после запуска симуляции

1. Функции вычислительного модуля для получения значимых соседей всех элементов графической сцены

Функция вычислительного модуля get\_neighbours(row, column, column\_count, row\_count, wind, wet, is\_clean) отвечает за вычисление состояний значимых соседей элемента в матрице размером row\_count \* column\_count, расположенного на позиции (row, column).

Функция вычислительного модуля get\_dx\_dy(offset, wind, wet, is\_clean) отвечает за вычисление относительных позиций значимых соседей текущего элемента. Под относительными позициями подразумеваются сдвиги от позиции текущего элемента. Список всех возможных относительных позиций представляет все уникальные сочетания целочисленных пар из диапазона [-1, 1], за исключением пары (0, 0), так как нулевой сдвиг по обеим осям соответствует текущему элементу.

От правильности работы этих функции зависит, насколько правильно будет высчитано новое состояние текущего элемента, так как оно определяется именно по состояниям всех его значимых соседей.

Тестирование этих функций:

Входные данные: is\_clean == True (чистая симуляция; в противном случае добавляется фильтрация с использованием вероятностей, что осложняет процесс тестирования); wind == WindType.miss (отсутствие ветра); wet == WetType.normal (нормальная влажность)

Ожидаемый результат: распространение пожара по правильным шестиугольникам

Для наглядности используется отображение высчитанных состояний элементов в определенной окрестности на графической сцене. На рисунке 9 представлены фрагменты графической сцены, на которых отображена динамика распространения пожара при заданных параметрах.



Рис. 9. Динамика распространения пожара после самовозгорания дерева

По рисунку можно сделать вывод, что фактический результат совпадает с ожидаемым: пожар распространяется по правильному шестиугольнику, а значит, позиции значимых соседей высчитаны верно.

# Заключение

В результате выполнения данной курсовой работы была спроектирована, разработана и реализована программа «Симулятор пожаров». Программа имеет насыщенный графический интерфейс, а также отделенный от графического интерфейса вычислительный модуль, реализованный согласно функциональной парадигме. Благодаря такому подходу программа эффективно обрабатывает большое количество элементов, поддается тестированию, а также имеет точки расширения для добавления новой функциональности. Все поставленные в техническом задании задачи выполнены успешно.